Advances in Engineering Software 82 (2015) 105–118

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Graphics processing unit (GPU) accelerated fast multipole BEM with level-skip M2L for 3D elasticity problems

Yingjun Wang [a], Qifu Wang [b], Xiaowei Deng [a,*], Zhaohui Xia [b], Jinhui Yan [a], Hua Xu [c]

[a] Department of Structural Engineering, University of California, San Diego, 9500 Gilman Drive, Mail Code 0085, La Jolla, CA 92093, USA
[b] National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, China
[c] School of Civil Engineering, Southwest Jiaotong University, Chengdu 610031, China

ARTICLE INFO

Article history:
Received 12 October 2014
Received in revised form 6 December 2014
Accepted 3 January 2015
Available online 28 January 2015

Keywords:
Fast multipole method
Boundary element method
Parallel computing
Graphics processing unit
3D elasticity
Level-skip M2L

ABSTRACT

In order to accelerate fast multipole boundary element method (FMBEM), in terms of the intrinsic parallelism of boundary elements and the FMBEM tree structure, a series of CUDA based GPU parallel algorithms for different parts of FMBEM with level-skip M2L for 3D elasticity are presented. A rigid body motion method (RBMM) for the FMBEM is proposed based on special displacement boundary conditions to deal with strongly singular integration and free term coefficients. The numerical example results show that our parallel algorithms obviously accelerates the FMBEM and can be used in large scale engineering problems with wide applications in the future.

1. Introduction

The boundary element method (BEM), due to its high accuracy and dimension reduction characteristics, has been widely applied in a variety of engineering areas, e.g. acoustics [1], flow [2] and elastostatics [3]. However, the advantages of the BEM are compensated by its dense and nonsymmetric coefficient matrix which needs $O(N^2)$ operations to compute the coefficients and $O(N^3)$ or $O(N^2)$ operations to solve the equation system by direct or iterative solvers (N is the number of degrees of freedom (DOFs)). It is necessary to develop the BEM of faster computing and less memory usage for the purpose of dealing with large scales problems. One of the most common strategies to achieve this is combining the BEM with the fast multipole method (FMM), which leads to fast multipole BEM (FMBEM) whose complexities of both computational time and memory space are $O(N)$.

The FMM was presented in the late 1980s by Rokhlin and Greengard [4] in the context of N-particle (N-body) simulations. Based on the hierarchical tree structure as the $O(N\log N)$ algorithm used in [5], two concepts are introduced in the FMM: (1) "multipole expansion (ME)" used to compute the moments of particle groups and (2) "local expansion (LE)" used to evaluate the contribution from particle groups. With the help of these two expansions, the FMM reduces the complexity of N-body problems to $O(N)$. At the beginning, the FMM utilized full tree structure to store data, but it was inefficient if the particles were distributed nonuniformly in the domain. In 1988, Carrier et al. [6] introduced adaptive tree structure into the FMM, and then presented the adaptive FMM which was efficient for both uniform and non-uniform particle distribution. In the FMM, the multipole-to-local (M2L) translation costs high computational expanse because of its high complexity. In order to optimize the M2L translation, the exponential expansion was used to convert the M2L translation into a series of low complexity translation and the new version FMM was proposed by Greengard and Rokhlin [7], but this method increases memory cost and results in acceleration only if the number of expansion terms is large. In recent years, Gumerov and Duraiswami [8] presented a child-to-parent level-skip M2L translation, which may apparently accelerate all expansion terms almost without memory increment. Later, Bapat and Liu [9] presented an more efficient parent-to-child level-skip M2L translation, but it loses the accuracy.

All the FMMs stated above can be introduced into BEMs to set up FMBEMs, and the advent of FMBEMs makes the BEMs obtain ability to efficiently solve large-scale engineering and scientific problems without losing its intrinsic advantages [10–13]. In the FMBEM, the final boundary integral equations are assembled from the unknowns at the nodes but the boundary integrals are element-based, while using high-order elements. Therefore, some approaches, e.g. "global-node approach" [14] and "node-patch approach" [15], need to be used if the tree structure of FMM only contains node information. An alternative better approach using high-order element in the FMBEM is modifying the tree structure

* Corresponding author. Tel.: +1 858 8223273.
E-mail address: x8deng@eng.ucsd.edu (X. Deng).

http://dx.doi.org/10.1016/j.advengsoft.2015.01.002
0965-9978/

to include information of both nodes and elements [16], which will be also used in this work. The major difference between the FMBEM and the BEM is that most of the coefficients in the matrix are not computed explicitly in the FMBEM, which makes the rigid body motion method (RBMM) [17] unavailable in the computation of the singular integrals and the free-term coefficients. Therefore, for 3D elastic problems, the $O(1/r^2)$ singular integrals need to be computed by analytical integral method [18] or Cauchy principal value integral method [19] and the free-term coefficients have to be computed by the solid angle method [20]. Compared to the RBMM, the latter methods are more complicated and cut off the relationship between the diagonal block coefficients and the non-diagonal coefficients, which may influence the behaviors of the coefficient matrix. Figuring out an treatment of the singular integrals that is suitable for FMBEM is meaningful and that is also an important aspect of this paper.

Although the FMBEM has successfully solved large scale problems within a relatively short time, serial performance has not been improving, and parallel computing on CPUs becomes one of the efficient approaches [21–23]. In recent years, with a more important role the graphics processing unit (GPU) parallel computing playing in scientific computing, the GPU parallel FMMs or FMBEMs have been studied by researchers. In 2008, Gumerov et al. [8] pioneered an GPU parallel FMM for 3D Laplace problems and achieved 30–60 times speedup. Darve et al. [24] discussed the parallel FMM on different parallel architectures including computer clusters, multicore processors and GPU. Lashuk et al. [25] researched parallel adaptive FMM on heterogeneous architectures, and obtained 30 times speedup over a single core CPU and 7 times speedup over a multicore CPU implementation. Yokota et al. [26] and Nguyen et al. [27] used GPU clusters to accelerate FMM to solve large-scale problems. Hamada [28] used two GTX295 GPUs to accelerate indirect FMBEM for voxel model analysis with double precision, compared to the Intel Core i7-975 CPU (4 cores), and the speedups achieved 5–8 times. Takahashi et al. [29] presented four M2L GPU parallel schemes and analyzed them in detail in the case of a uniform tree, and the speedups to multicore parallelized CPU are 4–8 times. Yokota et al. [30] implemented parallel FMBEM on a GPU cluster of 512 GPUs to solve billion-scale biomolecular electrostatics problem within one minute.

The outline of this paper is organized as follows: Section 2 describes the theory and procedures of FMBEM; Section 3 presents level-skip M2L and FMBEM-suitable RBMM approaches; Section 4 proposes our GPU parallel strategies of different parts in FMBEM; Section 5 analyzes and discusses large scale engineering examples; in Section 6 we present our conclusions.

2. FMBEM for 3D elasticity

2.1. Theory of FMBEM

For a 3D physical domain Ω with boundary Γ, in the absence of body force, the discrete form boundary integral equation (BIE) for linear elasticity [31] is

$$\begin{aligned} c_{ij}(\boldsymbol{P})u_j(\boldsymbol{P}) &= \sum_{e=1}^{M}\int_{\Gamma_e} U_{ij}(\boldsymbol{P},\boldsymbol{Q})t_j(\boldsymbol{Q})dS(\boldsymbol{Q}) \\ &\quad - \sum_{e=1}^{M}\int_{\Gamma_e} T_{ij}(\boldsymbol{P},\boldsymbol{Q})u_j(\boldsymbol{Q})dS(\boldsymbol{Q}) \\ &= \sum_{e=1}^{M}\left\{\sum_{\alpha=1}^{N} t_j^{\alpha}(\boldsymbol{Q})\int_{\Gamma_e} U_{ij}(\boldsymbol{P},\boldsymbol{Q})N_\alpha(\boldsymbol{Q})dS(\boldsymbol{Q})\right\} \\ &\quad - \sum_{e=1}^{M}\left\{\sum_{\alpha=1}^{N} u_j^{\alpha}(\boldsymbol{Q})\int_{\Gamma_e} T_{ij}(\boldsymbol{P},\boldsymbol{Q})N_\alpha(\boldsymbol{Q})dS(\boldsymbol{Q})\right\}, \end{aligned} \tag{1}$$

where $\boldsymbol{P}$ is the source point, $\boldsymbol{Q}$ is the field point, M is the number of elements, Γ_e is the area of the element e, $N_\alpha(\boldsymbol{Q})$ is the shape function of α-th node at point $\boldsymbol{Q}$, $u_j^{\alpha}(\boldsymbol{Q})$ and $t_j^{\alpha}(\boldsymbol{Q})$ are the j-th component of displacements and tractions at the α-th node of the element where $\boldsymbol{Q}$ is, $C_{ij}(\boldsymbol{P})$ is free term coefficient depending on the boundary geometry at point $\boldsymbol{P}$, and $U_{ij}(\boldsymbol{P},\boldsymbol{Q})$ and $T_{ij}(\boldsymbol{P},\boldsymbol{Q})$ are the fundamental solution kernels for 3D elasticity which are given as follows

$$U_{ij}(\boldsymbol{P},\boldsymbol{Q}) = \frac{1}{16\pi\mu(1-\nu)r}[(3-4\nu)\delta_{ij} + r_{,i}r_{,j}], \tag{2}$$

$$T_{ij}(\boldsymbol{P},\boldsymbol{Q}) = -\frac{1}{8\pi(1-\nu)r^2}\left\{\frac{\partial r}{\partial \mathbf{n}}[(1-2\nu)\delta_{ij} + 3r_{,i}r_{,j}] - (1-2\nu)(r_{,i}n_j - r_{,j}n_i)\right\}, \tag{3}$$

where ν is Poisson's ratio, μ is shear modulus, δ_{ij} is Kronecker delta, n_i is the i-th component of the unit outward normal, r is the distance between the point $\boldsymbol{P}$ and point $\boldsymbol{Q}$, and $r_{,i}$ is partial derivative in the direction i of coordinates.

Collocating source points on the boundary and applying Eq. (1) at each source point, the equation system is assembled as

$$\mathbf{Hu} = \mathbf{Gt}. \tag{4}$$

Reassembling Eq. (4) by assigning all unknown displacement/traction variables and their coefficients to the left hand side and other known ones to the right hand side, the equation system may be written as

$$\mathbf{Ax} = \mathbf{b}, \tag{5}$$

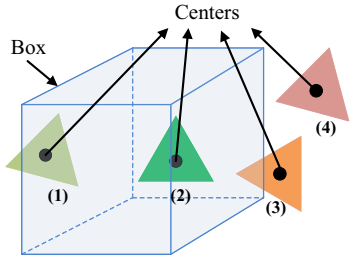
and all the unknown variables will be obtained by solving Eq. (5).

The biggest bottleneck limiting application of conventional BEM in large scale problems is the dense and non-symmetric matrix $\mathbf{A}$ in Eq. (5), which requires $O(N^2)$ operations to compute the coefficients and solve the equation system. To solve this problem, the FMM is introduced into the BEM to construct the FMBEM. The formulations of FMBEM for 3D elasticity are listed briefly here and the detail formulation derivation could refer to [32,33]. The formulations are divided into five parts: (1) Multipole Expansions (ME), (2) Multipole-to-Multipole (M2M) translation, (3) Local Expansion (LE), (4) Multipole-to-Local (M2L) translation, and (5) Local-to-Local (L2L) translation.

ME uses a expansion point $\boldsymbol{Q}_c$ near the field point to converts the integrals of $U_{ij}(\boldsymbol{P},\boldsymbol{Q})$ and $T_{ij}(\boldsymbol{P},\boldsymbol{Q})$ in Eq. (1) as follows

$$\begin{aligned} \int_{\Gamma_e} U_{ij}(\boldsymbol{P},\boldsymbol{Q})t_j(\boldsymbol{Q})d\Gamma(\boldsymbol{Q}) &\cong \frac{1}{8\pi\mu}\sum_{n=0}^{p}\sum_{m=-n}^{n}[F^S_{ij,n,m}(\overrightarrow{\boldsymbol{Q}_c\boldsymbol{P}})\overline{M^1_{j,n,m}}(\boldsymbol{Q}_c) \\ &\quad + G^S_{i,n,m}(\overrightarrow{\boldsymbol{Q}_c\boldsymbol{P}})\overline{M^1_{n,m}}(\boldsymbol{Q}_c)], \end{aligned} \tag{6}$$

$$\begin{aligned} \int_{\Gamma_e} T_{ij}(\boldsymbol{P},\boldsymbol{Q})u_j(\boldsymbol{Q})d\Gamma(\boldsymbol{Q}) &\cong \frac{1}{8\pi\mu}\sum_{n=0}^{p}\sum_{m=-n}^{n}[F^S_{ij,n,m}(\overrightarrow{\boldsymbol{Q}_c\boldsymbol{P}})\overline{M^2_{j,n,m}}(\boldsymbol{Q}_c) \\ &\quad + G^S_{i,n,m}(\overrightarrow{\boldsymbol{Q}_c\boldsymbol{P}})\overline{M^2_{n,m}}(\boldsymbol{Q}_c)], \end{aligned} \tag{7}$$



(1) (2) Belong to Box (3) (4) Not belong to Box

Fig. 1. An element classification example.

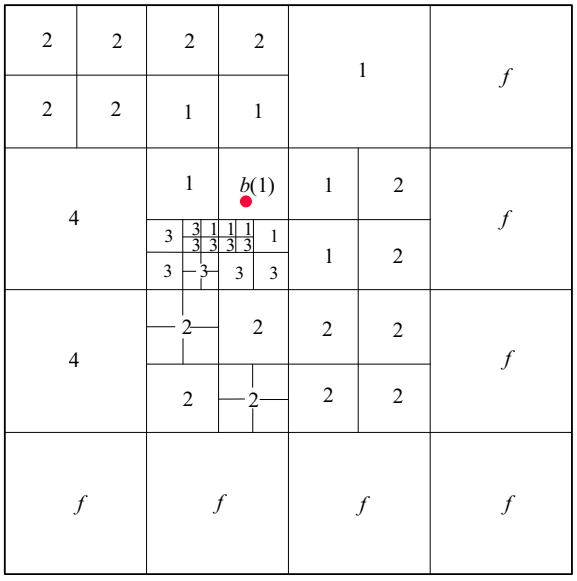


Fig. 2. Four lists for box b.

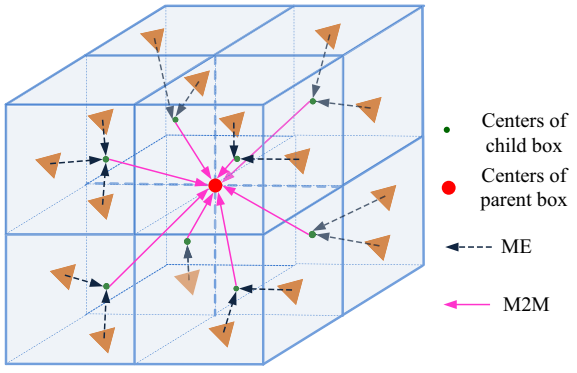


Fig. 3. ME and M2M of childless boxes.

where p is the number of expansion terms, $(\overrightarrow{\mathbf{Q}_c\mathbf{P}})$ is a vector from $\mathbf{Q}_c$ to $\mathbf{P}$, $\overline{M^k_{n,m}}(\mathbf{Q}_c)$ and $\overline{M^k_{j,n,m}}(\mathbf{Q}_c)$ are the complex conjugates of $M^k_{n,m}(\mathbf{Q}_c)$ and $M^k_{j,n,m}(\mathbf{Q}_c)$ (k = 1 or 2), and the formulations of ME moments and their coefficients $F^S_{ij,n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{P}})$ and $G^S_{i,n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{P}})$ are given in Appendix A.

The ME center may be moved from Q_c to $Q_{c'}$ by M2M translations

$$M^k_{n,m}(\mathbf{Q}_{c'}) = \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'} R_{n',m'}(\overrightarrow{\mathbf{Q}_{c'}\mathbf{Q}_c})\left[M^k_{n-n',m-m'}(\mathbf{Q}_c) + (\overrightarrow{\mathbf{Q}_{c'}\mathbf{Q}_c})_j M^k_{j,n-n',m-m'}(\mathbf{Q}_c)\right], \tag{8}$$

$$M^k_{j,n,m}(\mathbf{Q}_{c'}) = \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'} R_{n',m'}(\overrightarrow{\mathbf{Q}_{c'}\mathbf{Q}_c}) M^k_{j,n-n',m-m'}(\mathbf{Q}_c). \tag{9}$$

LE is similar to ME, which expands the integrals of $U_{ij}(\mathbf{P},\mathbf{Q})$ and $T_{ij}(\mathbf{P},\mathbf{Q})$ in Eq. (1) as

$$\int_{\Gamma_e} U_{ij}(\mathbf{P},\mathbf{Q}) t_j(\mathbf{Q}) d\Gamma(\mathbf{Q}) \cong \frac{1}{8\pi\mu}\sum_{n=0}^{p}\sum_{m=-n}^{n}\left[F^R_{ij,n,m}(\overrightarrow{\mathbf{P}_c\mathbf{P}}) L^1_{j,n,m}(\mathbf{P}_c) + G^R_{i,n,m}(\overrightarrow{\mathbf{P}_c\mathbf{P}}) L^1_{n,m}(\mathbf{P}_c)\right], \tag{10}$$

$$\int_{\Gamma_e} T_{ij}(\mathbf{P},\mathbf{Q}) u_j(\mathbf{Q}) d\Gamma(\mathbf{Q}) \cong \frac{1}{8\pi\mu}\sum_{n=0}^{p}\sum_{m=-n}^{n}\left[F^R_{ij,n,m}(\overrightarrow{\mathbf{P}_c\mathbf{P}}) L^2_{j,n,m}(\mathbf{P}_c) + G^R_{i,n,m}(\overrightarrow{\mathbf{P}_c\mathbf{P}}) L^2_{n,m}(\mathbf{P}_c)\right], \tag{11}$$

where $G^R_{i,n,m}$ and $F^R_{ij,n,m}$ are obtained from Eqs. (A.1) and (A.2) with $S_{n,m}$ replaced with $R_{n,m}$, and the LE moments are obtained by the M2L translations

$$L^k_{n,m}(\mathbf{P}_c) \cong (-1)^n \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'} \overline{S_{n+n',m+m'}}(\overrightarrow{\mathbf{Q}_c\mathbf{P}_c})\left[M^k_{n',m'}(\mathbf{Q}_c) - (\overrightarrow{\mathbf{Q}_c\mathbf{P}_c})_j M^k_{j,n',m'}(\mathbf{Q}_c)\right], \tag{12}$$

$$L^k_{j,n,m}(\mathbf{P}_c) \cong (-1)^n \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'} \overline{S_{n+n',m+m'}}(\overrightarrow{\mathbf{Q}_c\mathbf{P}_c}) M^k_{j,n',m'}(\mathbf{Q}_c). \tag{13}$$

When the LE point is moved from P_c to $P_{c'}$, we need to use the following L2L translations

$$L^k_{n,m}(\mathbf{P}_{c'}) \cong \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'} R_{n'-n,m'-m}(\overrightarrow{\mathbf{P}_c\mathbf{P}_{c'}})\left[L^k_{n',m'}(\mathbf{P}_c) - (\overrightarrow{\mathbf{P}_c\mathbf{P}_{c'}})_j L^k_{j,n',m'}(\mathbf{P}_c)\right], \tag{14}$$

$$L^k_{j,n,m}(\mathbf{P}_{c'}) \cong \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'} R_{n'-n,m'-m}(\overrightarrow{\mathbf{P}_c\mathbf{P}_{c'}}) L^k_{j,n',m'}(\mathbf{P}_c), \tag{15}$$

2.2. Procedure of FMBEM

The whole procedure of the FMBEM can be divided into four parts: (1) Tree structure construction, (2) Upward, (3) Downward and (4) Solution, where (2) and (3) are the core computation parts.

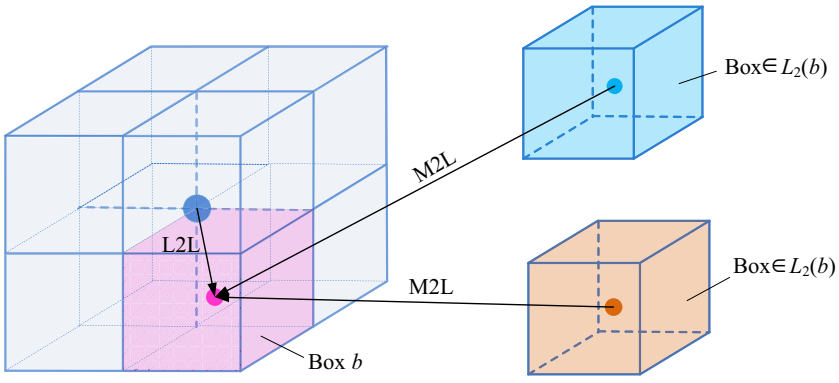


Fig. 4. M2L and L2L of box b.

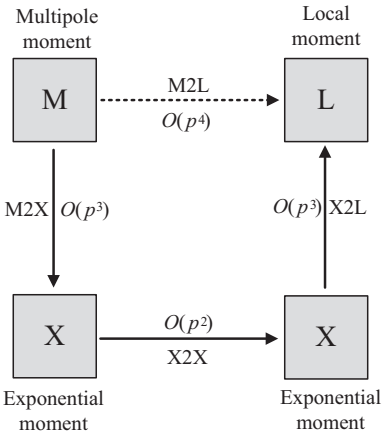


Fig. 5. Processes of exponential expansion.

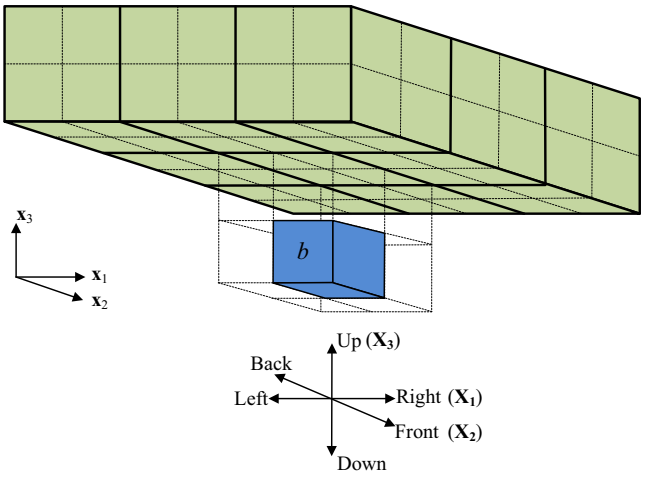


Fig. 6. Boxes of exponential expansions associated with box *b*.

2.2.1. Dual-information tree

The prerequisite of the FMM implementation is using the tree structure to store data. There are two versions of the tree structures in the FMM: the full tree structure [4,7] and the adaptive tree structure [6,34], and the later one is widely used nowadays. In the FMBEM, the final boundary integral equations are assembled from the unknowns of nodes but the boundary integrals are element-based, so the tree structure of the FMM cannot be used directly while high-order elements are used. In order to deal with this, a dual-information tree structure that contains both element and node information is constructed to satisfy the computational requirement of the FMBEM.

The tree structure is constructed by dividing the bounding box domain of problem into smaller box sub-domains recursively until the number of nodes in each box sub-domain is less than a prescribed number. When each sub-domain box is generated, whether elements are classified in the domain or not is determined by whether the center of the element in or out of the box (see Fig. 1). At the level 0, a cubic bounding box containing the whole domain is defined. Boxes on level $l+1$ are obtained by subdividing each box on level l into 8 equal child boxes, and then the empty child boxes are trimmed off. To implement the adaptive scheme, four lists associated with box *b* are defined as follows:

(1) *List* 1 (only for childless box *b*), denoted by $L_1(b)$, is the set consisting of *b* and all childless boxes adjacent to *b*.
(2) *List* 2, denoted by $L_2(b)$, is the set consisting of all children of the adjacent boxes of *b*'s parent that are well separated from *b*.
(3) *List* 3 (only for childless box *b*), denoted by $L_3(b)$, is the set consisting of all descendants of *b*'s adjacent boxes that are not adjacent to *b*.
(4) *List* 4, denoted by $L_4(b)$, is the counterpart of *List* 3, if $b. \in L_3(c)$ then $c \in L_4(b)$. Note that all boxes in $L_4(b)$ are childless.

Fig. 2 shows the four lists for a box *b* in 2D where *f* represents the far field boxes of *b*, and the lists of 3D problems can be obtained in the same way by replacing squares with cubes.

2.2.2. Upward

Upward pass is divided into two steps: (1) compute the multipole moment of each childless box *b* at its center by accumulating all the element integrals in *b*; and (2) compute the multipole moments of other boxes by M2M translation from the penultimate level to level 2. All the multipole moments will be obtained except the boxes belonging to level 0 and level 1. An example of ME and M2M is shown in Fig. 3.

2.2.3. Downward

Downward pass is divided into following steps:

(1) From level 2 to the lowest level, loop over each box *b* of level *l*, use M2L to translate the multipole moments of $L_2(b)$ to the local moment of *b* and add them together. Then, from level 3 to the lowest level, loop over each box *b* of level *l*, use L2L translation to translate local moment of *b*'s parent to *b* and add to the original local moment of *b*. Finally, use LE to compute the *U*/*T* integrals at each node of each childless box *b*. An example of M2L and L2L is shown in Fig. 4.
(2) Loop over each childless box *b* (level independent), use direct computation as conventional BEM to compute the *U*/*T* integrals from elements of each box in $L_1(b)$ to each node in *b*, and accumulate the integrals.
(3) Loop over each childless box *b* (level independent), use ME (the number of nodes in *b* is larger than the square of expansion term p^2) or direct computation as conventional BEM (the number of nodes in *b* is not larger than p^2) to compute the *U*/*T* integrals from elements of each box in $L_3(b)$ to each node in *b*, and accumulate the integrals.
(4) From level 3 to the lowest level, loop over each box *b* of level *l*, use direct computation as conventional BEM to compute the *U*/*T* integrals from elements of each box in $L_4(b)$ to each node in *b*, and accumulate the integrals.

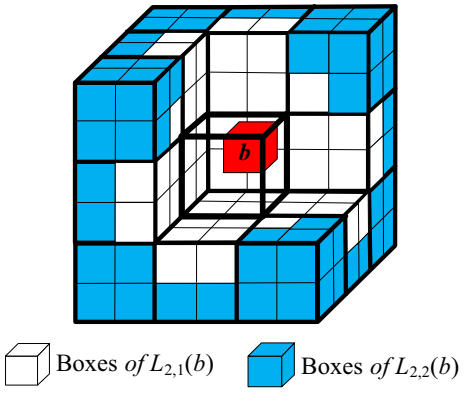


Fig. 7. Classification of $L_2(b)$ based on Eq. (16).

2.2.4. Solution

The coefficient matrix of the BEM equation system Eq. (5) is dense and nonsymmetric. Iterative solvers are more efficient than direct solvers in large-scale equations. Generalized minimum residual (GMRES) method is one of most efficient iterative methods and widely used in BEM equation solutions [35], which is also used in this paper. In the GMRES solution, the most time-consuming part is matrix–vector multiplication ($\mathbf{A}\lambda$) whose complexity is $O(N^2)$. The core idea of the FMBEM solution is to use the FMM to accelerate the matrix–vector multiplication in GMRES, without forming the entire matrix **A** explicitly. In order to accelerate the convergence of the iterative solutions, the leaf based preconditioner [36] is used in this work.

3. Level-skip M2L and RBMM in FMBEM

3.1. Level-skip M2L

In the FMBEM, M2L is one of the most time-consuming parts and will be the most expensive when the expansion term p is large enough. Therefore, improving M2L can obviously improve the efficiency of the FMBEM, which attracts researchers to work on it.

Greengard and Rokhlin [7] used the exponential expansion to accelerate M2L. The key concept of the exponential expansion method is using less complex expansions to replace M2L as shown in Fig. 5, but it will increase memory cost to store the exponential moments and does not result in acceleration when the number of expansion terms is small. Besides, the exponential expansions are restricted to the boxes in $+X_3$ direction of a box b as Fig. 6, and extra rotation operations need to be applied to the boxes in other directions. More details of exponential expansion can be found in [33].

Unlike the exponential expansion method, the level-skip methods accelerate M2L translation by reducing the number of M2L translations. Later, Bapat and Liu [9] presented an efficient parent-to-child level-skip M2L translation, but it sacrifices accuracy for efficiency. Gumerov and Duraiswami [8] presented an child-to-parent level-skip M2L translation to accelerate all expansion terms almost without memory increase, but they did not describe the detailed implementation. In this section, we will detailedly present an available method of the child-to-parent level-skip M2L translation.

For an arbitrary box b, all boxes in $L_2(b)$ may be subdivided into two sublists $L_{2,1}(b)$ and $L_{2,2}(b)$, where $L_{2,1}(b)$ represents the boxes in $L_2(b)$ which are near the center of b's parent box, and $L_{2,2}(b)$ represents the boxes in $L_2(b)$ which are far from the center of b's parent box (see Fig. 7). Assuming that the edge length of box b is l and the distance from the center of a box in $L_2(b)$ to the center of b's parent box is r, and then one of the classified rules for arbitrary box $c \in L_2(b)$ may be concluded as

$$\begin{cases} \frac{\sqrt{3}}{2}l \leqslant r \leqslant \frac{\sqrt{35}}{2}l, & c \in L_{2,1}(b) \\ \frac{\sqrt{43}}{2}l \leqslant r \leqslant \frac{5\sqrt{3}}{2}l, & c \in L_{2,2}(b) \end{cases}. \tag{16}$$

The boxes in $L_{2,2}(b)$ can do M2L translation to b's parent box instead of box b itself, while still satisfying the specified error bound. This M2L skips from child level to parent level, so it can be called child-to-parent level-skip M2L translation in this paper. By using this technique for the M2L translation, the max number of operations per box is reduced from 189 to 119.

For a box b, in tree construction, the only change is splitting the $L_2(b)$ into $L_{2,1}(b)$ and $L_{2,2}(b)$ according to the box position; and in downward, the M2L translation needs to be computed by two parts: one is for the boxes in $L_{2,1}(b)$ and the other is for those in $L_{2,2}(b)$. It is noted that the local moments of boxes at level 1 is no longer equal to 0 (original M2L does not perform at level 1, so the local moments is 0), and therefore the L2L translation in downward should start from level 2 instead of level 3. The computational cost of the L2L in level 2 is so small that it can be neglected.

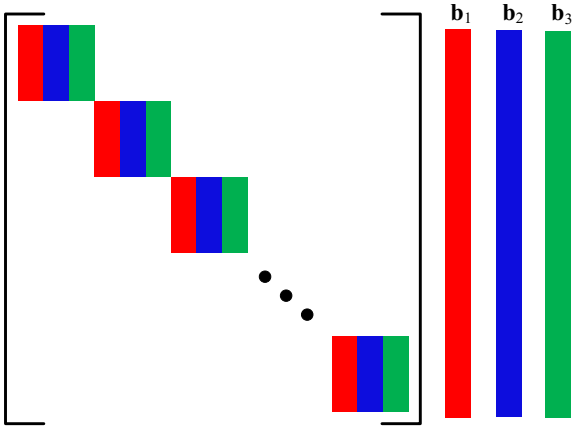


Fig. 8. Distribution of diagonal block coefficients of matrix H.

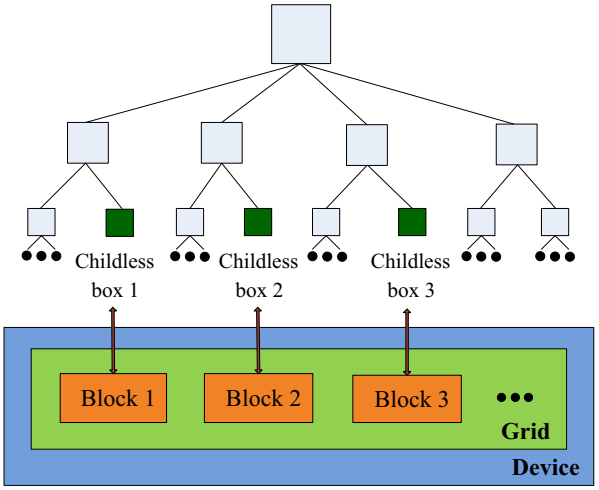


Fig. 9. Correspondence of childless boxes and blocks.

3.2. RBMM in FMBEM

In 3D elasticity, the diagonal block coefficients of matrix **H** in Eq. (4) contain strongly singular integrals and free-term coefficients which are commonly evaluated by the rigid body motion method (RBMM) in the conventional BEM as follows

$$\mathbf{H}_{ij}^{PQ} = (\delta_{PQ} - 1)\sum_{Q=1}^{N}\mathbf{H}_{ij}^{PQ}, \tag{17}$$

where N is the number of nodes, δ_{PQ} is the Kronecker delta, the subscripts i and j range from 1 to 3, and the superscripts P and Q refer to the nodal number.

However, the RBMM cannot be used in the FMBEM because most of the coefficients of matrix **H** are not generated explicitly. Although the analytical integration [18] or the CPV integration [19] can be used to evaluate the strongly singular integrals, the free-term coefficients have to be evaluated by other methods, and the linear-dependence relationship between the diagonal and non-diagonal coefficients is cut off, which can influence the matrix condition because of the errors between analytical and numerical computations.

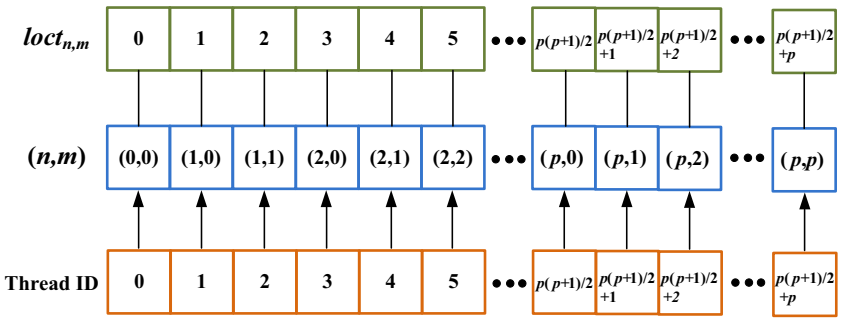


Fig. 10. Thread distribution in a block.

In this section, we will present a RBMM that can evaluate the diagonal block coefficients of matrix **H** by the FMBEM. The core idea of our method is using particular boundary conditions to construct equations and solve them to get the diagonal block coefficients of matrix **H**. For 3D problems, set free term coefficients and traction of each node to 0, and assign the displacement coordinate components of each node to be $\{-1,0,0\}$, $\{0,-1,0\}$ and $\{0,0,-1\}$ respectively, and then the following three separate equation systems can be obtained

$$\mathbf{H}'\mathbf{u}_1 = \mathbf{0}, \tag{18}$$

$$\mathbf{H}'\mathbf{u}_2 = \mathbf{0}, \tag{19}$$

$$\mathbf{H}'\mathbf{u}_3 = \mathbf{0}, \tag{20}$$

where $\mathbf{H}'$ is matrix **H** of Eq. (4) with 0 free term coefficients, and $\mathbf{u}_1 = \{-1,0,0,\ldots,-1,0,0\}^T$, $\mathbf{u}_2 = \{0,-1,0,\ldots,0,-1,0\}^T$, $\mathbf{u}_3 = \{0,0,-1,\ldots,0,0,-1\}^T$.

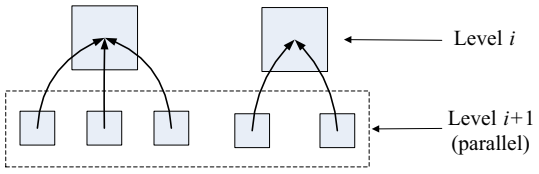
The FMBEM is used to compute $\mathbf{H}'\mathbf{u}_i$. When the diagonal block coefficients of matrix $\mathbf{H}'$ (the coefficients of each node to itself in) is temporarily set to 0 in the near field direct integration (NFDI) of downward, the equations become

$$\mathbf{H}'\mathbf{u}_1 = \mathbf{b}_1, \tag{21}$$

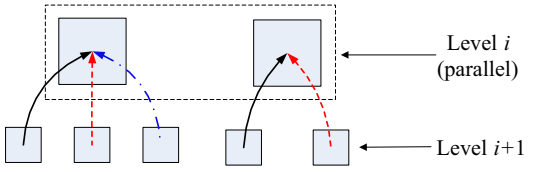
$$\mathbf{H}'\mathbf{u}_2 = \mathbf{b}_2, \tag{22}$$

$$\mathbf{H}'\mathbf{u}_3 = \mathbf{b}_3, \tag{23}$$

where $\mathbf{b}_1$, $\mathbf{b}_2$ and $\mathbf{b}_3$ are the vectors consisting of diagonal block coefficients of matrix **H** as shown in Fig. 8.



(a) Parallelization of child level boxes



(b) Parallelization of parent level boxes

Fig. 11. Two parallel strategies.

The idea of our method is the same as RBMM, and successfully used in FMBEM without computing all coefficients of matrix **H**. All the diagonal block coefficients of matrix **H** only need to be evaluated and stored once, so it will not influence the whole efficiency of the FMBEM.

4. GPU parallel computing of FMBEM

GPUs were originally used for graphic processing but are increasingly being applied to scientific computations due to their outstanding computational power, especially since NVIDIA released the Compute Unified Device Architecture (CUDA) in 2007 [37].

Under CUDA, a program is composed of both host part (CPU) and device part (GPU), where host part is in charge of serial parts, and device part is in charge of parallel parts. The functions defined in device part are called kernel functions. Threads are the units of execution on the GPU. A certain number of threads bundled together form a thread block. Within a block, threads are executed in groups called warps (32 threads) which always run together on a processor (core). There are several memories that can be used by programmers to achieve high efficiency. Global memory, shared by the whole GPU which has a noticeable latency, usually stores input and output data; registers are allocated to individual threads, and each thread can only access its own registers (access speed is very high); shared memory can be accessed by all threads in a block almost as fast as registers; constant memory is cached, where host can write and read but device can only read; local memory which is allocated to each thread usually stores data when register is used out; and texture memory has some special functions (e.g. texture rendering).

NVIDIA GTX 580 GPU is used to run our parallel FMBEM program, which contains 16 streaming multiprocessors and 512 CUDA cores. Global memory occupies most of the available memory (1.5 GB). The 32 processors in a multiprocessor share 64 KB of RAM with a configurable partitioning of shared memory and L1 cache, and all multiprocessors share 64 KB constant memory.

In this section, A CUDA based GPU parallel algorithm of the adaptive FMBEM with level-skip M2L is presented according to the intrinsic parallelism of boundary elements and levels of the adaptive tree, which accelerates ME, M2M translation, M2L translation, L2L translation and NFDI in the FMBEM.

4.1. ME parallel strategy

The MEs of childless boxes are independent so they can be computed in parallel in different levels of the adaptive FMBEM. From Eqs. (A.1)–(A.4), it can be obtained that the subscript (n,m) computations of the moments $M^k_{n,m}$ and $M^k_{j,n,m}$ are independent. A parallel strategy is presented as follows: one block completes the computations of one childless box (one-block-one-box) as Fig. 9, and each

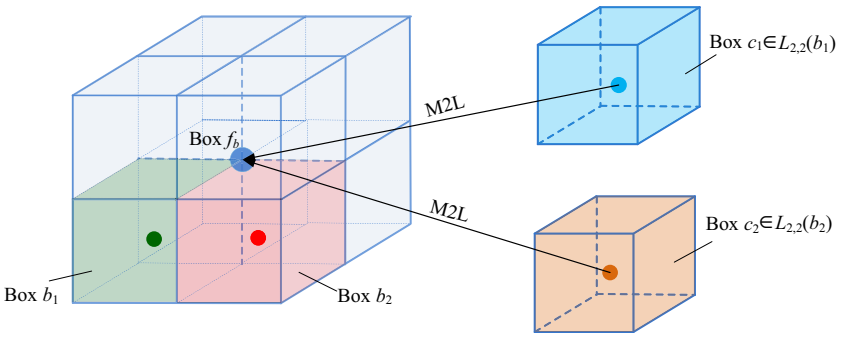


Fig. 12. Data conflict of skip-level M2L translations.

thread in the block completes the moment component computation corresponding to subscript (n, m). Assume the expansion term is p, the subscript (n, m) satisfies

$$\begin{cases} 0 \leqslant n \leqslant p \\ -n \leqslant m \leqslant n \end{cases} (n, m \in \text{integers}), \tag{24}$$

where only the moment components whose subscript $m \geqslant 0$ need to be computed and the others can be obtained from the complex conjugate relationship as Eqs. (A.7) and (A.8).

One dimension arrays are used in graphic memory for ME coefficients, and therefore $M_{n,m}^k$ are stored in an array, and $M_{j,n,m}^k$ are stored in three arrays depending on the value of j. The location of (n, m) in an array is

$$loct_{n,m} = n(n+1)/2 + m. \tag{25}$$

One thread is assigned to compute one moment component corresponding to subscript (n, m). Then the number of threads in one block is $(p + 1)(p + 2)/2$, and the thread to (n, m) mapping relationship is shown in Fig. 10.

In this parallel mode, all the multipole moments are stored in shared memory that can avoid the access latency of global memory, and the loads between each thread are balanced. When the number of leaves is large enough, the computing capability of GPU can be fully utilized. The values of multipole moments are copied from shared memory to global memory when all the moment components are obtained. The parallel GPU ME algorithm is demonstrated as follows

Algorithm 1: GPU parallel ME

```
// Number of blocks numblock is equal to the number of
   childless boxes
// Number of threads in a block is p(p + 1)/2 + p + 1
// Variable bid is block ID and tid is local thread ID in bid
For bid=0,1,...,numblocks in parallel do
  If tid = 0 then
    Get the number of a childless box b and the coordinates of
   its center
  End if
  Get ME moment components through Eqs. (A.1)–(A.4),
   where tid computes the component of the tid-th location.
End for
```

Remark: In Algorithm 1, the "**For** ...*numblocks* **in parallel do**" structure is not a real loop structure, which is just used to describe the parallel process. This representation method will be also used in the following algorithms.

4.2. Moment translation parallel strategy

4.2.1. M2M parallel strategy

The M2M translations are from the boxes of the lowest level to the boxes at level 2 recursively. It means that the M2M translations of different levels cannot be parallelized; however, all M2M translations at the same level can be parallelized. There are two parallel strategies as shown in Fig. 11.

(1) Parallelization of child level boxes (Fig. 11(a)). The advantage of this strategy is that the number of the child level boxes is large (the parallel scale is large), but it has a disadvantage that the M2M translations from different boxes of the child level need to be accumulated to their parent boxes, which causes the accumulative process unable to be parallelized because of data writing conflict.
(2) Parallelization of parent level boxes (Fig. 11(b)). Although the parallel scale of this strategy is smaller than the parallelization of child level boxes, it can avoid data writing conflict in the process of M2M translations. Therefore, this strategy is used in the M2M translation in this work.

According to Eqs. (8) and (9), it is easy to find that each component represented by subscript (n, m) can be computed in parallel, so the parallelization strategy of subscript (n, m) in the ME can also be applied to the M2M translation. The GPU parallel algorithm of the M2M translations is described as follows

Algorithm 2: GPU parallel M2M translation

```
// Number of tree levels is nlevel, Number of blocks required at
   level i is numblocks_i
// Number of threads in a block is p(p + 1)/2 + p + 1
// Variable bid is block ID which represents box b_bid and tid is
   local thread ID in bid
  For level l = nlevel-1, nlevel-2,...,2 do
    For bid = 0,1,...,numblocks_l-1in parallel do
      For each box b ∈ child boxes of box b_bid do
        Each thread tid parallel translates the ME moment to
   the subscript (n, m) component of ME moment of box b_bid
   by Eqs. (8) and (9), then adds it to the counterpart of the ME
   moment component of box b_bid
      end for c
    end for bid
  end for l
```

4.2.2. M2L parallel strategy

In general, the M2L translations are evaluated level by level. In fact, the M2L translations of different levels are independent,

which means they can be evaluated in parallel. Therefore, in the GPU parallel computing, we can use one block to charge the M2L translations of one box, and use one thread to evaluate the LE component of each subscript pair (n, m).

When the level-skip M2L translation approach is used, *List* 2 of a box b, denoted by $L_2(b)$, is divided into the near sublist $L_{2,1}(b)$ and far sublist $L_{2,2}(b)$ as Fig. 7. In the parallel M2L translation process, data writing conflict will occur. As shown in Fig. 12, box b_1 and box b_2 share the same parent box f_b, and box c_1 and box c_2 belong to $L_{2,2}(b_1)$ and $L_{2,2}(b_2)$ respectively, when the level-skip M2L translations of boxes in $L_{2,2}(b_1)$ and $L_{2,2}(b_2)$ are computed in parallel, the translations from box b_1 and box b_2 may happen at the same time that results in data writing conflict in the moment summation process of box f_b.

In order to solve this problem, we change the tree structure and use $L'_{2,2}(f_b)$ to replace $L_{2,2}(f_b)$ for each box f_b from the lowest level (level *nlevel*) to level 1 as

$$L'_{2,2}(f_b) = L_{2,2}(b_1) \cup L_{2,2}(b_2), \ldots, \cup L_{2,2}(b_i). \tag{26}$$

It can be seen that $L'_{2,2}(f_b)$ is a set of all $L_{2,2}(b_i)$ of box f_b's child box b_i and the data conflict can be avoided. Meanwhile, the M2L translations at level 1 need to be computed, but the computational cost of the M2L translations at level 1 is very small. The GPU parallel algorithm of the M2L translations is demonstrated as follows

Algorithm 3: GPU parallel M2L translation

// Number of tree levels is *nlevel*, C_i is the *i*th level of the tree structure
// The total number of boxes from C_1 to C_{nlevel} is $N1$, and the corresponding number of blocks is *numblocks*1 = $N1$
// The total number of boxes from C_2 to C_{nlevel} is $N2$, and the corresponding number of blocks is *numblocks*2 = $N2$
// Number of threads in a block is $p(p + 1)/2 + p + 1$
// Variable *bid* be block ID which represents box b_{bid} and *tid* is local thread ID in *bid*
// Steps (1) and (2) need to call the kernel function respectively
(1) **For** *bid* = 0, 1,…,*numblocks*2-1**in parallel do**
  For each box $c \in L_{2,1}(b_{bid})$ **do**
    Each thread *tid* parallel translates ME moment to the subscript (n,m) component of LE moment of box b_{bid} by Eqs. (12) and (13), then adds it to the counterpart of the LE moment component of box b_{bid} together
  end for *c*
end for *bid*
(2) **For** *bid* = 0, 1,…,*numblocks*1-1**in parallel do**
  For each box $c \in L_{2,2}(b_{bid})$ **do**
    Each thread *tid* parallel translates ME moment to the subscript (n,m) component of LE moment of box b_{bid} by Eqs. (12) and (13), then adds it to the counterpart of the LE moment component of box b_{bid} together
  end for *c*
end for *bid*

4.2.3. L2L parallel strategy

In the L2L translation, similar to the M2M translation, only the translations at the same level can be parallelized, but it needs to be evaluated from level 2 to the lowest level which is opposite to the M2M translation. It means that the data writing conflict will not occur in the L2L translation, so we utilize the strategy of parallelization of child level boxes (Fig. 11(a)) to the L2L translation. The GPU parallel algorithm of the L2L translations is demonstrated as follows

Algorithm 4: GPU parallel L2L translation

// Number of tree levels is *nlevel*, Number of blocks required at level *i* is *numblocks_i*
// Number of threads in a block is $p(p + 1)/2 + p + 1$
// Variable *bid* is block ID which represents box b_{bid} and *tid* is local thread ID in *bid*
For level *l* = 2, 3,…,*nlevel* **do**
  For *bid*=0, 1,…,*numblocks_l*-1 **in parallel do**
  Each thread *tid* parallel translates the LE moment of box b_{bid}'s parent box to the subscript (n,m) component of the LE moment of box b_{bid} by Eqs. (14) and (15), then adds it to the counterpart of the LE moment component of box b_{bid}
  end for *bid*
end for *l*

4.3. NFDI parallel strategy

In the FMBEM, each node needs to use direct integration to complete the element integrals which are near itself, and this direct integration is called near field direct integration (NFDI). The NFDI may be used in the computations of *List* 1, *List* 3 and *List* 4 of each childless box.

Element integrals are box independent, and thus they can be computed in one-block-one-box parallel mode. We compute integrals of one element in a field box to all nodes in a source box (Fig. 13(b)) in parallel, and then loop the elements in the field box to complete the box computation. In this way, the data writing conflict of the summation of element integrals to one node as shown in Fig. 13(a) is avoided.

According to our parallel strategy, the number of threads in a block is set to the maximum number of nodes allowed in a leaf, and the shared memory is used to store the integral results. The parallel CUDA NFDI algorithm is described as follows

Algorithm 5: GPU parallel NFDI

// Number of blocks *numblock* is equal to the number of childless boxes
// Number of threads in a block *x* is the max number of nodes allowed in a box
// Use *Nn_i* and *Ne_i* to represent the number of nodes and the number of elements in box b_i respectively
// Variable *bid* is block ID which represents box b_{bid} and *tid* is local thread ID in *bid*
For *bid* = 0, 1,…,*numblocks*-1**in parallel do**
  For each box $c \in L_1(b_{bid}) \cup L_3(b_{bid}) \cup L_4(b_{bid})$ **do**
    If *tid*<*Nn_bid* **then**
      For element $e \in$ box *c* **do**
        Each thread *tid* parallel does the integral computation of element *e* to all nodes in box b_{bid}, and then accumulates to the original integral of the nodes
      end for *e*
    end if
  end for *c*
end for *bid*

4.4. Solution parallel strategy

The GMRES method which we used in the FMBEM is an iterative method for the numerical solution of a system of linear equations as **Ax** = **b**. Each iterative process is dependent and cannot be parall-

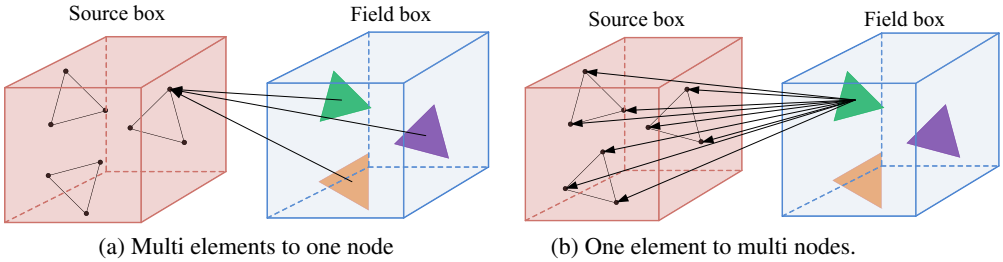


Fig. 13. Modes of element-to-node integration in NFDI.

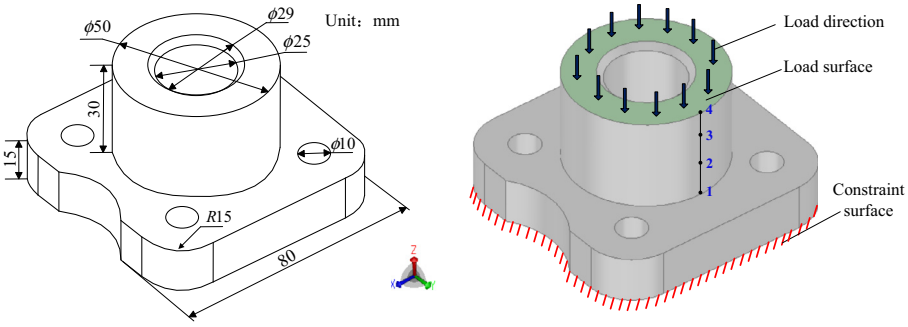


Fig. 14. The size and boundary conditions of bracket model.

Table 1
Relative errors of right hand side vector **b** using different M2L translations.

p	M2L	M2L_E		M2L_S
		$g = 8$	$g = 17$	
6	1.283×10^{-4}	6.466×10^{-4}	1.284×10^{-4}	2.150×10^{-4}
8	1.552×10^{-5}	6.336×10^{-4}	1.582×10^{-5}	2.762×10^{-5}
10	2.275×10^{-6}	6.335×10^{-4}	2.208×10^{-6}	5.240×10^{-6}
12	4.195×10^{-7}	6.334×10^{-4}	4.362×10^{-7}	1.203×10^{-6}

Table 2
Time costs in M2L translations of different M2L translation methods (unit: s).

p	M2L	M2L_E		M2L_S
		$g = 8$	$g = 17$	
6	0.846	1.001	4.913	0.625
8	2.107	1.335	6.630	1.615
10	4.837	1.767	8.819	3.564
12	8.822	2.304	11.205	6.317

Table 3
Memory costs of the FMBEM using different M2L methods (unit: MB).

p	M2L	M2L_E		M2L_S
		$g = 8$	$g = 17$	
6	48	53	62	48
8	49	54	63	49
10	50	55	65	50
12	51	57	66	51

elized, but more than 90% computational cost is spent in linear algebra operations which can be computed in parallel.

The linear algebra operations in the GMRES can be divided into five types [38]: (1) matrix vector multiplication; (2) vector dot product; (3) vector Euclidean norm; (4) vector real multiplication and (5) vector multiply–add operation. The most complicated operation is matrix vector multiplication which is completed by the FMBEM. The rest of the four types can be easily completed by assigning suitable blocks and threads according to the computational scale, which will not be discussed in detail here.

5. Numerical examples

The FMBEM programs are executed on a desktop computer: CPU is Intel Core i7-2600k 3.4 GHz (1 core is used), GPU is NVIDIA GTX 580 GPU, OS is Windows 7 Professional, 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/C++ language).

5.1. Bracket model

The sizes and boundary conditions of the bracket model are shown in Fig. 14. Elastic modulus is 200 GPa, Possion's ratio is 0.3, and the load is 20 MPa ($-Z$ direction). The surfaces of the model are meshed with 6646, 14,172 and 55,560 linear triangular elements, respectively, and the degrees of freedom (DOFs) are 9945, 21,234 and 83,316. The number of expansion term p is 8, and the maximum number of nodes allowed in a childless box is set to 20. The convergence tolerance of GMRES solution is set to 10^{-3}.

5.1.1. Comparison of different M2L translations

In this paper, the level-skip M2L method is used to accelerate the M2L translations. Therefore, we compare the level-skip M2L with the exponential-expansion based M2L and the conventional M2L, which are distinguished by M2L_S, M2L_E and M2L in the following of this section. The bracket model with 9945 DOEs is used here, and the number of expansion term p is chosen 6, 8, 10, 12 for the multipole expansion and location expansion, respectively. The number of expansion term g is chosen 8 and 17 for the exponential expansion, which contains 3 and 6 digital accuracy, respectively [34].

Table 4
Relative errors of displacements at the sample points.

DOFs	Point 1		Point 2		Point 3		Point 4	
	F/B (%)	F/A (%)	F/B (%)	F/A (%)	F/B (%)	F/A (%)	F/B (%)	F/A (%)
2874	2.1	6.4	0.5	3.3	0.5	5.0	0.5	2.0
5493	1.9	1.8	0.4	0.4	0.2	1.1	0.2	1.0
9945	0.9	1.0	0.3	0.6	0.2	1.5	0.2	1.3
21,234	1.1	1.6	0.3	1.2	0.2	1.3	0.2	1.2
83,316	–	0.9	–	0.3	–	1.7	–	0.6

Table 5
Relative errors of **b** and **u** between FBEM_G and FBEM_C.

DOFs	9945	21,234	83,316
Error ε_b	7.130×10^{-4}	2.952×10^{-3}	8.213×10^{-4}
Error ε_u	2.026×10^{-3}	5.584×10^{-3}	2.255×10^{-3}

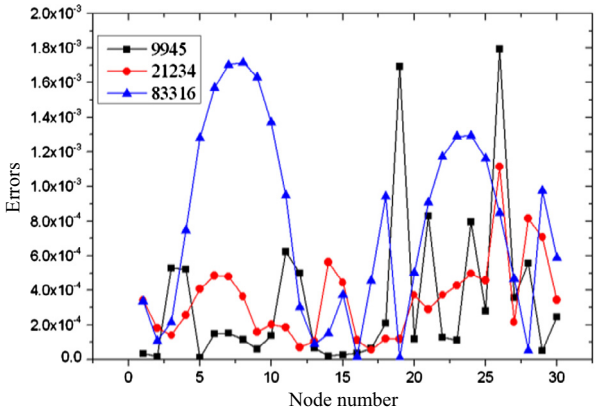


Fig. 15. The FBEM_G/FBEM_C errors in load direction of selected nodes.

In order to avoid the error introduced by solution, we use the right hand side vector **b** of equation system **Ax** = **b** to compare the accuracy and efficiency of different M2L translation methods. The vector **b** obtained from conventional BEM is used as reference, so the relative error of vector **b** is defined as

$$\varepsilon = \frac{\|\mathbf{b}_{\mathrm{BEM}} - \mathbf{b}_{\mathrm{FMBEM}}\|_2}{\|\mathbf{b}_{\mathrm{BEM}}\|_2}, \tag{27}$$

where $\mathbf{b}_{\mathrm{BEM}}$ and $\mathbf{b}_{\mathrm{FMBEM}}$ are the right hand side vectors of the conventional BEM and the FMBEM, respectively.

Table 1 lists the relative errors of vector **b** using different M2L translations. The accuracies of both M2L and M2L_S increase with the number of expansion term p. The accuracy of M2L_S is a little lower than that of M2L, but still keeps in the same order of magnitude. The accuracy of M2L_E is not only associated with expansion term p but also with the expansion term g, and high accuracy cannot be obtained if either of them is small.

Table 2 lists the times costs in the M2L translations using different M2L methods. The rising trends of time costs of M2L and M2L_S are consistent when the number of expansion term p increases, and acceleration of M2L_S is irrelevant to the number of expansion term. The M2L_E accelerates the M2L translation only when p equals to 6, 8 and 10 and g equals to 8, but its rising speed of time cost is slower than that of other methods when the number of expansion term p increases. This means that M2L_E makes full use of its advantages only in the situation that p is very large.

The memory costs of the FMBEM using different M2L methods are shown in Table 3, where it can be found that the M2L_E method needs to use more memory than other methods. The memory costs between M2L and M2L_S are almost the same.

Comprehensively comparing the accuracy, efficiency and memory cost, the M2L_S performs the best, that is why it is used in this paper. In the following sections, all the M2L translations uses the M2L_S methods if no explicit explanation is made.

5.1.2. Accuracy analysis of CPU-FMBEM

Before discussing the GPU-FMBEM, the accuracy of the CPU-FMBEM is analyzed, which is the basis to demonstrate the accuracy

Table 8
GPU/CPU speedups of different parts correspond to Table 7.

DOFs	GPU/CPU speedups				
	ME	M2M	M2L	L2L	NFDI
9945	7.8	2.0	30.7	2.3	6.7
21,234	9.4	5.3	32.1	5.8	7.3
83,316	9.5	7.8	35.2	7.2	7.5

Table 6
Iterative number, time and memory of different solving scales.

DOFs	CPU			GPU		
	Iterative number	Total time (s)	Total memory (MB)	Iterative number	Total time (s)	Total memory (MB)
9945	14	48.047	49	14	5.64	41
21,234	19	139.718	82	19	14.25	57
83,316	26	579.375	303	26	55.454	133

Table 7
Time spent in different parts of one upward–downward process (unit: s).

DOFs	CPU					GPU				
	ME	M2M	M2L	L2L	NFDI	ME	M2M	M2L	L2L	NFDI
9945	0.235	0.006	0.954	0.007	1.437	0.030	0.003	0.031	0.003	0.219
21,234	0.516	0.021	2.89	0.023	3.088	0.055	0.004	0.094	0.004	0.422
83,316	1.969	0.086	9.89	0.093	8.746	0.208	0.011	0.281	0.013	1.172

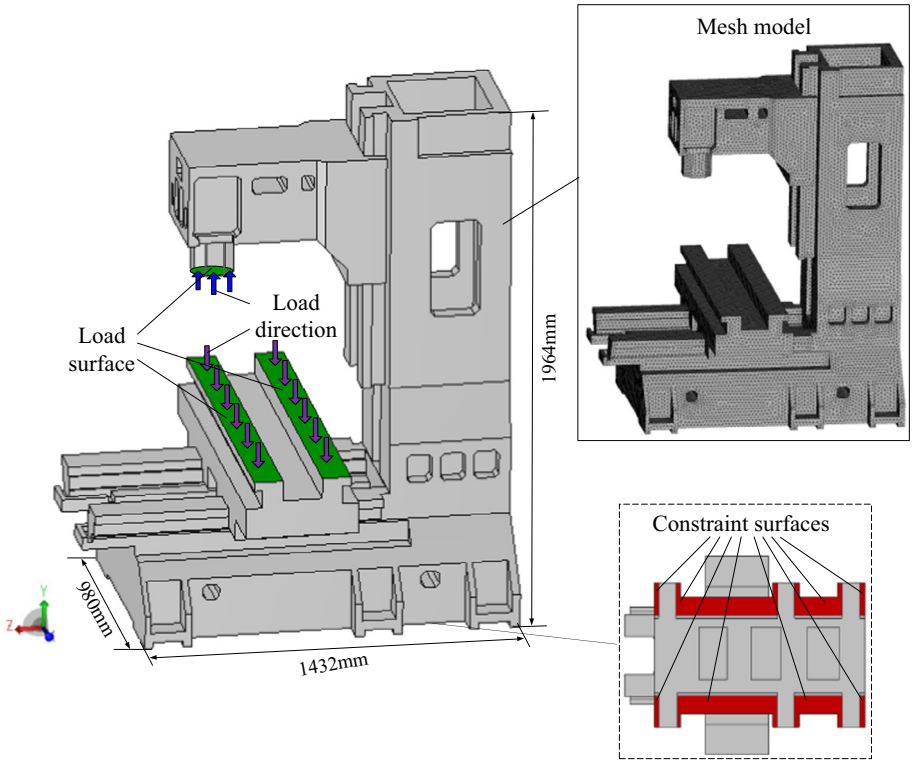


Fig. 16. Machine tool model.

Table 9
Computational parameters of machine tool with different methods.

Numerical method	DOFs	Iterative number	Time (s)	Memory (MB)
FMBEM_C	251,988	93	6455.6	973
FMBEM_G	251,988	94	577.1	568
FEM_1	534,214	–	1012.2	1458||7802
FEM_2	3,677,743	–	7605.5	5131||61478

of the GPU-FMBEM. We pick up 4 points along the load direction (see Fig. 14) as the sample points to compare the displacement results between the FMBEM, conventional BEM using 10^{-5} convergence tolerance and the finite element software ANSYS [39] using 16,041 quadratic tetrahedral elements with 74,688 DOFs. Another two meshes with 1932 and 3678 elements (DOFs are 2874 and 5493) are also discussed to compare the accuracy.

Table 4 lists the relative errors of displacements at the sample points, where F/B and F/A represent the relative errors of FMBEM/BEM and FMBEM/ANSYS, respectively. The symbol "——" represents that the problem scale is too large to be solved by the desktop using the conventional BEM. It can be found that the relative errors between the FMBEM and the BEM are less than 0.6% except point 1, and the max error at point 1 is only 2.1%. The relative errors between the FMBEM and ANSYS are less than 2% except the case that the FMBEM with coarse elements. These demonstrate the accuracy of the FMBEM. The reason for that the displacement of point 1 is less accurate is that the displacement value is small which is more sensitive to the computational errors.

5.1.3. Accuracy and efficiency analysis of GPU-FMBEM

We use the right hand side vector **b** of equation system **Ax** = **b** and displacement result vector **u** to analyze the accuracy of the FMBEM. The relative errors of vector **b** and **u** are defined as

$$\varepsilon_b = \frac{\|\mathbf{b}_{\text{FMBEM\_C}} - \mathbf{b}_{\text{FMBEM\_G}}\|_2}{\|\mathbf{b}_{\text{FMBEM\_C}}\|_2}, \tag{28}$$

$$\varepsilon_u = \frac{\|\mathbf{u}_{\text{FMBEM\_C}} - \mathbf{u}_{\text{FMBEM\_G}}\|_2}{\|\mathbf{u}_{\text{FMBEM\_C}}\|_2}, \tag{29}$$

where subscript FMBEM_C represents the results of CPU (double precision) and FMBEM_G represents the results of GPU (single precision). We can respectively obtain the errors between double precision and single precision in a FMM process and iterative solution by the comparisons of vector **b** and vector **u**.

In Table 5, it can be found that errors exist between GPU and CPU, but the errors are small. In this example, the magnitude of the max error is 10^{-3} which is acceptable by engineering computation. Comparing error ε_b and error ε_u, we can find out that ε_u is larger than ε_b but still in the same order of magnitude, which means that the errors accumulate in the iterative processes but the speed of error increase is slow. Fig. 15 shows the errors in load direction of randomly selected nodes, and the scale of the max error is 10^{-3}, which is consistent with that of Table 5.

Table 6 shows the number of iterations, time and memory of different problem sizes. Although in some cases, using single precision increases iterative number of solution [35], this phenomenon does not occur in the example. From the total time in Table 6, it can be seen that GPU computing has obviously accelerated the FMBEM, and the speedups of 9945, 21,234 and 83,316 DOFs are 8.5, 9.8 and 10.4 respectively. The speedups increase with the solving scales because large-scale computation can fully utilize the computational capability of GPU, but the speedups will not increase when the solving scale reach a certain degree. The reason why memory of GPU is less than that of CPU is that GPU uses single precision data which occupy less memory than double precision data used by CPU.

Table 7 shows the time spent in different parts of one upward–downward process, and Table 8 gives the corresponding GPU/CPU

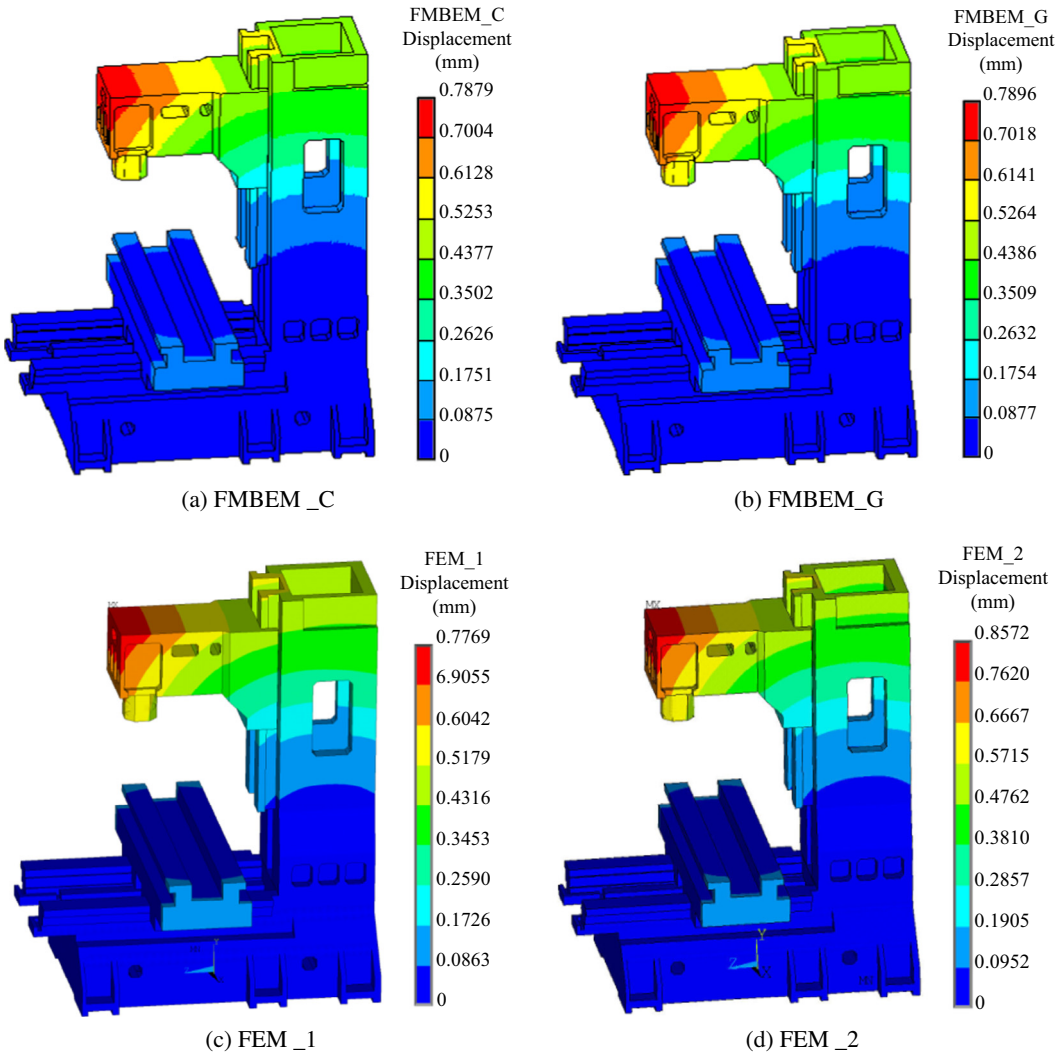


Fig. 17. Displacement of machine tool model.

speedups. GPU greatly accelerates the FMBEM with different solving scales, especially for the M2L whose speedup is more than 30, which is comparable to the 4–8 times speedup between the GPU and multi-core parallelized CPU in [29] and the 4–32 times GPU/CPU speedup of the FMM downward pass in [8]. The acceleration effects of different parts is summarized as: M2L > ME > NFDI > L2L ≈ M2M. ME, M2L and NFDI spend most of time (more than 95% of the total time in the FMBEM process), especially the M2L and NFDI. In the GPU computing, the NFDI is the most time-consuming part because its speedup is much smaller than that of M2L, so this part will be further optimized in the future.

5.2. Machine tool model

The major dimensional sizes, boundary conditions and mesh model of the machine model are shown in Fig. 16. Elastic modulus is 260 GPa, Possion's ratio is 0.3, the load value of the spindle surface is 15 MPa (Z direction), and the load of the worktable is 5 Mpa ($-Z$ direction). The surfaces of the machine tool are meshed with 168,028 linear triangular elements, and the degrees of freedom (DOFs) are 251,988. The number of expansion term p is 6, and the maximum number of nodes allowed in a childless box is 32.The convergence tolerance of GMRES solution is set to 10^{-3}. In order to verify the results of the FMBEM, the software ANSYS is used, which used linear and quadratic tetrahedral elements (the same edge size as those in the FMBEM) to discretize the machine tool model and the DOFs is 493,919 and 3,699,943, respectively.

Computational parameters of machine tool adopting different methods are shown in Table 9, where FMBEM_C represents the CPU serial solution, FMBEM_G represents the GPU parallel solution, and FEM_1 and FEM_2 represent ANSYS solution with linear and quadratic elements, respectively. The memory data of ANSYS are separated by symbol '||', where the front one is memory allocated for solver and the other one is the memory required for in-core mode solution (only uses RAM). ANSYS used out-of-core mode to solve the machine tool model because the in-core required memory has exceeded the available RAM of the computer.

From the computational time, it can be found that FMBEM_C is slower than FEM_1 but a little faster than FEM_2, and FMBEM_G is faster than all the others. The speedup of FMBEM_G/FMBEM_C reaches 11.2 times, and the speedups of FMBEM_G/FEM_1 and FMBEM_G/FEM_2 are 12.2 and 1.9 times. Note that the efficiency comparison is just given as a reference which cannot represent the algorithm comparison of the FMBEM and FEM, because the FMBEM codes are just written for research without professional optimization. From the memory cost, we can find that FMBEM_C and FMBEM_G consume much less memory than ANSYS, which

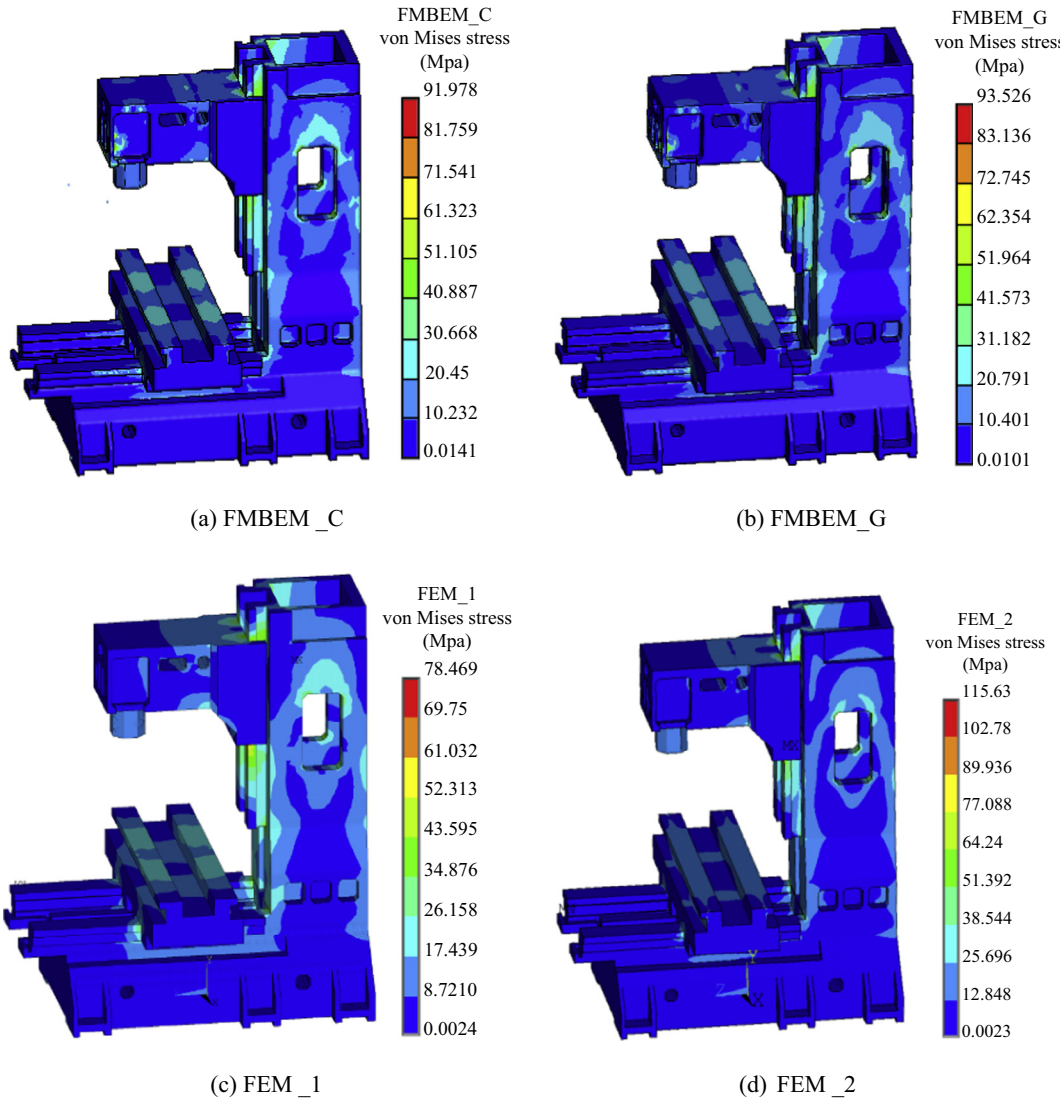


Fig. 18. von Mises stress of machine tool model.

means that the FMBEM can solve large problems using limit memory resource.

Figs. 17 and 18 show the displacement and von Mises stress of machine tool model respectively. The results among FMBEM_C, FMBEM_G, FEM_1 and FEM_2 are consistent. The displacement error of FMBEM/FEM_2 is less than 8% which is smaller than that of FEM_1/FEM2, and the stress error (less than 20%) of FMBEM/FEM_2 is also smaller than the error of FEM_1/FEM_2. The reason for this is that linear triangular elements used in the FMBEM have C^1 continuity in displacement but only have C^0 continuity in stress, while the quadratic tetrahedron elements used in ANSYS have C^2 and C^1 continuity in displacement and stress respectively. Therefore, the quadratic triangular elements is planned to be used in the FMBEM to improve the accuracy in the following research. One thing should be noted that these error are smaller than those between linear and quadratic elements in ANSYS which demonstrates the accuracy of the FMBEM.

6. Conclusions

In this paper, we presented GPU parallel strategies for different parts of the FMBEM with level-skip M2L in 3D elasticity. We first briefly introduced the theory and procedure of the adaptive FMBEM whose tree structure contains both node and element information. In order to accelerate the M2L translation, an available child-to-parent level-skip M2L translation was presented in detail. Meanwhile, a RBMM suitable for FMBEM was proposed to avoid the CPV integration and the computation of free term coefficients. After that, the GPU parallel algorithms of the adaptive FMBEM are presented according to the intrinsic parallelism of boundary elements and boxes of the adaptive tree, which accelerates ME, M2M translation, M2L translation, L2L translation and NFDI in the FMBEM. Two examples were used to verify our algorithms, and the results showed that the theories and algorithms in this paper have advantages of high computing efficiency, large solving scale and strong adaptability, which have a promising future in engineering.

In the future, we will further improve our FMBEM algorithms, especially in computational accuracy improvement and computational speed acceleration. In the computational accuracy improvement, quadratic elements will be used and the nearly singular integrals will be dealt with by some especial techniques like subdivision of the integration region [40] or semi-analytical algorithm [41]. In the speed acceleration part, we plan to use the reusable intrinsic sample point algorithm [42] to accelerate the NFDI in

the FMBEM. Besides that, we will expand our FMBEM to other engineering problems, such as thermodynamics, fluid mechanics and acoustics.

Acknowledgements

This research has been supported by two National Natural Science Foundations of China (51475180 and 51275182), and also sponsored in part by the National Key Technology R & D Program of China (Grants: 2012BAF08B01) and NSF grant CMMI-1068106.

Appendix A

$$M^1_{n,m}(\mathbf{Q}_c) = \int_{\Gamma_e} (\overrightarrow{\mathbf{Q}_c\mathbf{Q}})_j R_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{Q}}) t_j(\mathbf{Q}) d\Gamma(\mathbf{Q}), \tag{A.1}$$

$$M^1_{j,n,m}(\mathbf{Q}_c) = \int_{\Gamma_e} R_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{Q}}) t_j(\mathbf{Q}) d\Gamma(\mathbf{Q}), \tag{A.2}$$

$$M^2_{n,m}(\mathbf{Q}_c) = E_{jpkl} \int_{\Gamma_e} \frac{\partial}{\partial x_p} [(\overrightarrow{\mathbf{Q}_c\mathbf{Q}})_j R_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{Q}})] n_k(\mathbf{Q}) u_l(\mathbf{Q}) d\Gamma(\mathbf{Q}), \tag{A.3}$$

$$M^2_{j,n,m}(\mathbf{Q}_c) = E_{jpkl} \int_{\Gamma_e} \frac{\partial}{\partial x_p} [R_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{Q}})] n_k(\mathbf{Q}) u_l(\mathbf{Q}) d\Gamma(\mathbf{Q}), \tag{A.4}$$

$$F^S_{ij,n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{P}}) = \frac{\lambda+3\mu}{\lambda+2\mu} \delta_{ij} S_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{P}}) - \frac{\lambda+\mu}{\lambda+2\mu} (\overrightarrow{\mathbf{Q}_c\mathbf{P}})_j \frac{\partial}{\partial x_i} S_{n,m}(\overrightarrow{\mathbf{Q}_c\mathbf{P}}), \tag{A.5}$$

$$G^S_{i,n,m}(\overrightarrow{\mathbf{Q}_c P}) = \frac{\lambda+\mu}{\lambda+2\mu} \frac{\partial}{\partial x_i} S_{n,m}(\overrightarrow{\mathbf{Q}_c P}), \tag{A.6}$$

where λ and μ are Lame coefficients, E_{jpkl} is elastic modulus tensor and equals to $\lambda\delta_{ik}\delta_{lp} + \mu(\delta_{il}\delta_{jp} + \delta_{ip}\delta_{kl})$, $S_{n,m}$ and $R_{n,m}$ are solid harmonic functions [33].

$$M^k_{n,-m}(\mathbf{Q}_c) = (-1)^m \overline{M^k_{n,m}(\mathbf{Q}_c)} \ (m \geqslant 0), \tag{A.7}$$

$$M^k_{j,n,-m}(\mathbf{Q}_c) = (-1)^m \overline{M^k_{j,n,m}(\mathbf{Q}_c)} \ (m \geqslant 0), \tag{A.8}$$

where $k = 1, 2$.

References

[1] Tadeu AJB, Godinho L, Santos P. Performance of the BEM solution in 3D acoustic wave scattering. Adv Eng Softw 2001;32:629–39.
[2] Chahine GL, Kalumuck KM. BEM software for free surface flow simulation including fluid–structure interaction effects. Int J Comput Appl Technol 1998;11:177–98.
[3] Wilde AJ, Aliabadi MH. Direct evaluation of boundary stresses in the 3D BEM of elastostatics. Commun Numer Methods Eng 1998;14:505–17.
[4] Greengard L, Rokhlin V. A fast algorithm for particle simulations. J Comput Phys 1987;73:325–48.
[5] Barnes J, Hut P. A hierarchical O(NlogN) force-calculation algorithm. Nature 1986;324:4.
[6] Carrier J, Greengard L, Rokhlin V. A fast adaptive multipole algorithm for particle simulations. SIAM J Scient Stat Comput 1988;9:669–86.
[7] Greengard L, Rokhlin V. A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numer 1997;6:229–69.
[8] Gumerov NA, Duraiswami R. Fast multipole methods on graphics processors. J Comput Phys 2008;227:8290–313.
[9] Bapat MS, Liu YJ. A new adaptive algorithm for the fast multipole boundary element method. Comput Model Eng Sci (CMES) 2010;58:161–84.
[10] Shen L, Liu YJ. An adaptive fast multipole boundary element method for three-dimensional potential problems. Comput Mech 2007;39:681–91.
[11] Liu YJ, Nishimura N. The fast multipole boundary element method for potential problems: a tutorial. Eng Anal Bound Elem 2006;30:371–81.
[12] Chaillat S, Bonnet M, Semblat JF. A multi-level fast multipole BEM for 3-D elastodynamics in the frequency domain. Comput Methods Appl Mech Eng 2008;197:4233–49.
[13] Wei YX, Wang QF, Wang YJ, Huang YB. Optimizations for elastodynamic simulation analysis with FMM-DRBEM and CUDA. Comput Model Eng Sci 2012;86:241.
[14] Yoshida K, Nishimura N, Kobayashi S. Application of new fast multipole boundary integral equation method to crack problems in 3D. Eng Anal Bound Elem 2001;25:239–47.
[15] Lu BZ, Cheng XL, Huang JF, McCammon JA. An adaptive fast multipole boundary element method for Poisson-Boltzmann electrostatics. J Chem Theory Comput 2009;5:1692–9.
[16] Wang YJ. Research on GPU parallel fast multipole boundary element method in structural analysis. Wuhan, China: Huazhong University of Science and Technology; 2013.
[17] Lachat JC, Watson JO. Effective numerical treatment of boundary integral equations: a formulation for three-dimensional elastostatics. Int J Numer Meth Eng 1976;10:991–1005.
[18] Li HB, Han GM, Mang HA. A new method for evaluating singular integrals in stress analysis of solids by the direct boundary element method. Int J Numer Meth Eng 1985;21:2071–98.
[19] Guiggiani M, Gigante A. A general algorithm for multidimensional Cauchy principal value integrals in the boundary element method. J Appl Mech 1990;57:906–15.
[20] Ning D-z, Teng B, Zhao H-t, Hao C-l. A comparison of two methods for calculating solid angle coefficients in a BIEM numerical wave tank. Eng Anal Bound Elem 2010;34:92–6.
[21] Lei T, Yao ZH, Wang HT, Wang PB. A parallel fast multipole BEM and its applications to large-scale analysis of 3-D fiber-reinforced composites. Acta Mech Sin 2006;22:225–32.
[22] Wang HT, Lei T, Li J, Huang JF, Yao ZH. A parallel fast multipole accelerated integral equation scheme for 3D Stokes equations. Int J Numer Meth Eng 2007;70:812–39.
[23] Takahashi Y, Iwashita T, Nakashima H, Wakao S, Fujiwara K, Ishihara Y. Performance evaluation of a parallel fast multipole accelerated boundary integral equation method in electrostatic field analysis. IEEE Trans Magn 2011;47:1174–7.
[24] Darve E, Cecka C, Takahashi T. The fast multipole method on parallel clusters, multicore processors, and graphics processing units. CR Mec 2011;339:185–93.
[25] Lashuk I, Chandramowlishwaran A, Langston H, Nguyen T-A, Sampath R, Shringarpure A, et al. A massively parallel adaptive fast multipole method on heterogeneous architectures. Commun ACM 2012;55:101–9.
[26] Yokota R, Barba LA, Narumi T, Yasuoka K. Petascale turbulence simulation using a highly parallel fast multipole method on GPUs. Comput Phys Commun 2013;184:445–55.
[27] Nguyen QM, Dang V, Kilic O, El-Araby E. Parallelizing fast multipole method for large-scale electromagnetic problems using GPU clusters. IEEE Antennas Wireless Propag Lett 2013;12:868–71.
[28] Hamada S. GPU-accelerated indirect boundary element method for voxel model analyses with fast multipole method. Comput Phys Commun 2011;182:1162–8.
[29] Takahashi T, Cecka C, Fong W, Darve E. Optimizing the multipole-to-local operator in the fast multipole method for graphical processing units. Int J Numer Meth Eng 2012;89:105–33.
[30] Yokota R, Bardhan JP, Knepley MG, Barba LA, Hamada T. Biomolecular electrostatics using a fast multipole BEM on up to 512 GPUs and a billion unknowns. Comput Phys Commun 2011;182:1272–83.
[31] Gao XW, Davies TG. Boundary element programming in mechanics. Cambridge University Press; 2002.
[32] Liu YJ. Fast multipole boundary element method: theory and applications in engineering. Cambridge University Press; 2009.
[33] Yoshida K. Applications of fast multipole method to boundary integral equation method. Japan: Dept of Global Environment Eng Kyoto Univ; 2001.
[34] Cheng H, Greengard L, Rokhlin V. A fast adaptive multipole algorithm in three dimensions. J Comput Phys 1999;155:468–98.
[35] Xiao H, Chen ZJ. Numerical experiments of preconditioned Krylov subspace methods solving the dense non-symmetric systems arising from BEM. Eng Anal Bound Elem 2007;31:1013–23.
[36] Wang HT, Yao ZH, Wang PB. On the preconditioners for fast multipole boundary element methods for 2D multi-domain elastostatics. Eng Anal Bound Elem 2005;29:673–88.
[37] NVIDIA C. NVIDIA CUDA programming guide. Version 4.0; 2011.
[38] Wang YJ, Wang QF, Wang G, Huang YB, Wei YX. Boundary element parallel computation for 3D elastostatics using CUDA. In: Proceedings of the ASME IDETC/CIE Washington, DC: ASME; 2011.
[39] ANSYS I. ANSYS 13.0 online help; 2010.
[40] Beer G, Smith I, Duenser C. The boundary element method with programming: for engineers and scientists. Springer Verlag; 2008.
[41] Niu Z, Wendland WL, Wang X, Zhou H. A semi-analytical algorithm for the evaluation of the nearly singular integrals in three-dimensional boundary element methods. Comput Methods Appl Mech Eng 2005;194:1057–74.
[42] Kane JH, Gupta A, Saigal S. Reusable intrinsic sample point (RISP) algorithm for the efficient numerical integration of three dimensional curved boundary elements. Int J Numer Meth Eng 1989;28:1661–76.