Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference
IDETC/CIE 2011
August 29-31, 2011, Washington, DC, USA

DETC2011-47981

BOUNDARY ELEMENT PARALLEL COMPUTATION FOR 3D ELASTOSTATICS USING CUDA

Yingjun Wang, Qifu Wang*, Gang Wang, Yunbao Huang, Yixiong Wei
National CAD Support Software Engineering Research Center
Huazhong University of Science and Technology
Wuhan, China 430074

ABSTRACT

Finite Element Method (FEM) is pervasively used in most of 3D elastostatic numerical simulations, in which Computer Aided Design (CAD) models need to be converted into mesh models first and then enriched with semantic data (e.g. material parameters, boundary conditions). The interaction between CAD models and FEM models stated above is very intensive. Boundary Element Method (BEM) has been used gradually instead of FEM in recent years because of its advantage in meshing. BEM can reduce the dimensionality of the problem by one so that the complexity in mesh generation can be decreased greatly. In this paper, we present a Boundary Element parallel computation method for 3D elastostatics. The parallel computation runs on Graphics Processing Unit (GPU) using Computing Unified Device Architecture (CUDA). Three major components are included in such method: (1) BEM theory in 3D elastostatics and the boundary element coefficient integral methods, (2) the parallel BEM algorithm using CUDA, and (3) comparison the parallel BEM using CUDA with conventional BEM and FEM respectively by examples. The dimension reduction characteristics of BEM can dispose the 3D elastostatic problem by 2D meshes, therefore we develop a new faceting function to make the ACIS facet meshes suitable for Boundary Element Analysis (BEA). The examples show that the GPU parallel algorithm in this paper can accelerate BEM computation about 40 times.

Keywords: parallel computation, BEM, 3D elastostatics, CUDA, ACIS mesh generation

*Corresponding Author, E-mail: wangqf@hust.edu.cn

1 INTRODUCTION

Typical product design involves the following steps: conceptual design, CAD modeling, meshing and model preparation for specific behavior study, finite element analysis, result analysis and optimization [1]. Designers need to spend much time in the pre-processings such as model transformation, model simplification and mesh generation when FEM is used [2-3]. Recently, BEM is introduced to deal with structural performance analysis, because it adopts Boundary Integral Equation (BIE) formulations [4] that reduce the dimensionality of the problem by one, which making the mesh generation easier. Besides the meshing advantage, BEM has higher precision than FEM. Because BEM adopts analytical solutions of differential operators and boundary integral in the interior of the solution domain, and just adopts numerical computation on the boundary, which is different from FEM that uses numerical computation in the whole solution domain.

However, BEM produces dense and nonsymmetric matrices and costs more time during computation, which decreasing the efficiency of product design. The matrices need $O(N^2)$ operations to compute the coefficients and $O(N^3)$ operations to solve the system by direct solvers or $O(N^2)$ operations by iterative solvers [5]. (N is the number of equations of the linear system or Degrees Of Freedom (DOFs)). In order to accelerate BEM computation, parallel computation methods have been utilized by several researchers. The first BEM parallel computation architecture was described by Symm [6], and this work was continued by Davies [7]. Since then, several research works have been published [8-10]. But there are still several problems such as load equilibrium, data transmission and network stability need to be handled urgently since these methods are based on Personal Computer (PC) cluster.

General parallel computations based on GPU made a great breakthrough in the last few years, especially since NVIDIA released the parallel computing architecture called CUDA in 2006. Following the breakthrough, GPU parallel computation has been used widely in scientific engineering computation. Kumar et al [11] presented a parallel implementation of expectation maximization for Gaussian mixture models on GPUs using CUDA, and the results show that speedup can reach 164 times. Takahashi and Hamada [12] implemented a GPU-accelerated 3D Helmholtz BEM program for GeForce 8-series GPUs. The program performed 6–23 times faster than a normal BEM program, but the computation of singular integral coefficients was still accomplished by CPU.

In this paper, we completed a GPU parallel BEM algorithm in 3D elastostatics using CUDA. In order to simplify the pre-processing, the ACIS meshes are applied directly here to avoid writing a mesh algorithm and integrating it with CAD software. The theory and the algorithm of BEM in 3D elastostatics are presented in Sec. 2. Section 3 describes the CUDA based parallel computation in processes of coefficient integrating and equation solving. Section 4 compares the CUDA based parallel BEM with conventional BEM and FEM respectively by two examples. Finally, a brief conclusion and future work are presented.

2 BEM IN 3D ELASTOSTATICS

2.1 Theory

Assuming the body forces are zero, the BIE of isotropic elastic solids with boundary S can be written as:

$$C_{ij}(e)u_j(e)+\int_s t_{ij}^*(e,f)u_j(f)ds = \int_s u_{ij}^*(e,f)t_j(f)ds \quad i,j=1,2,3 \tag{1}$$

where a summation is to be carried out over the range of two identical subscripts, $u_j(f)$ and $t_j(f)$ are the components in j direction of displacements and tractions at a point f, $C_{ij}(e)$ are free-term coefficients, $u_{ij}^*(e,f)$ and $t_{ij}^*(e,f)$ are the fundamental solution of Kelvin's problem [13].

When the boundary is discretized into N const Triangle Elements (TEs), the number of nodes is N, Eqn. (1) can be reduced to the following:

$$C_{ij}(e)u_j(e)+\sum_{f=1}^{N}\int_{S_f} t_{ij}^*(e,f)u_j(f)ds = \sum_{f=1}^{N}\int_{S_f} u_{ij}^*(e,f)t_j(f)ds \quad i,j=1,2,3;e,f=1,2,\cdots N \tag{2}$$

where $C_{ij}(e)=\frac{1}{2}\delta_{ij}$ (δ_{ij} is Kronecker delta), $h_{ij}(e,f)$ and $g_{ij}(e,f)$ are integral coefficients of u_j and t_j:

$$h_{ij}(e,f)=\int_{S_f} t_{ij}^*(e,f)ds$$
$$g_{ij}(e,f)=\int_{S_f} u_{ij}^*(e,f)ds \tag{3}$$

Equation (2) has the matrix representation:

$$\left[\mathrm{H_{ef}}\right]_{\mathrm{N\times N}}\left\{\mathrm{u_f}\right\}_{\mathrm{1\times N}}=\left[\mathrm{G_{ef}}\right]_{\mathrm{N\times N}}\left\{\mathrm{t_f}\right\}_{\mathrm{1\times N}} \tag{4}$$

where

$$\mathrm{H}_{ef}=\left[H_{ef}^{ij}\right]_{3\times3},\ \mathrm{G}_{ef}=\left[G_{ef}^{ij}\right]_{3\times3}$$
$$\mathrm{u}_f=\left[u_f^1 \quad u_f^2 \quad u_f^3\right]^{\mathrm{T}},\ \mathrm{t}_f=\left[t_f^1 \quad t_f^2 \quad t_f^3\right]^{\mathrm{T}}$$
$$H_{ef}^{ij}=\frac{1}{2}\delta_{ij}\delta_{ef}+h_{ij}(e,f),\quad G_{ef}^{ij}=g_{ij}(e,f)$$

u_f and t_f are vectors of displacement and traction at point f, each vector has 3 components, H_{ef} and G_{ef} are 3×3 submatrices.

The integral computations of off-diagonal submatrices' elements of matrix H and matrix G in Eqn. (4) are nonsingular. The computations can be obtained by Eqn. (3). By introducing a dimensionless oblique coordinate system (ξ_1,ξ_2) as Fig. 1, the integrals about *ds* can change to the integrals about $d\xi_1 d\xi_2$ as Eqn. (5), then we can use Hammer numerical quadrature method [14] to calculate them.

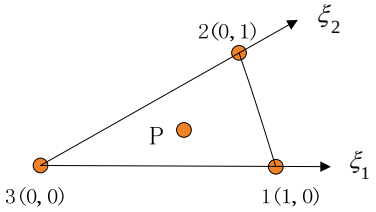

Figure 1. Oblique coordinate system

$$H_{ef}^{ij}=h_{ij}(e,f)=\int_{S_f} t_{ij}^*(e,f)ds = \int_0^1\int_0^{1-\xi_2} t_{ij}^*(e,f)J(\xi_1,\xi_2)d\xi_1 d\xi_2$$
$$G_{ef}^{ij}=g_{ij}(e,f)=\int_{S_f} u_{ij}^*(e,f)ds = \int_0^1\int_0^{1-\xi_2} u_{ij}^*(e,f)J(\xi_1,\xi_2)d\xi_1 d\xi_2 \tag{5}$$

where *J* is the Jacobian of the coordinate transformation.

The integrals of diagonal submatrices' elements of matrix G in Eqn. (4) are $O(1/r)$ singular. By dividing triangle s_e into 3 sub-triangles $s_{e,1}$, $s_{e,2}$, $s_{e,3}$ as Fig. 2, a new triangle polar coordinate transformation [15] as Fig. 3 can be used to remove $O(1/r)$ singularity of each subtriangle. The transformation formula is present as Eqn. (6). Then Hammer numerical quadrature formula being used to compute the integrals of sub-triangles ($G^{ij}_{ee,1}, G^{ij}_{ee,2}, G^{ij}_{ee,3}$), $G^{ij}_{ee} = \sum_{k=1}^{3} G^{ij}_{ee,k}$, but this method is ineffective for $O(1/r^2)$ singularity.

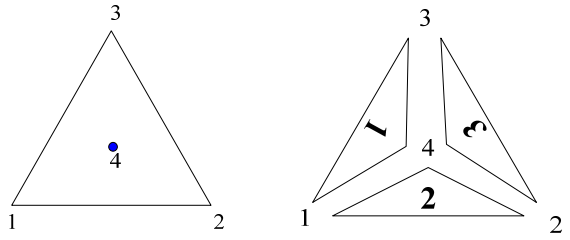

Figure 2. Triangle subdivision

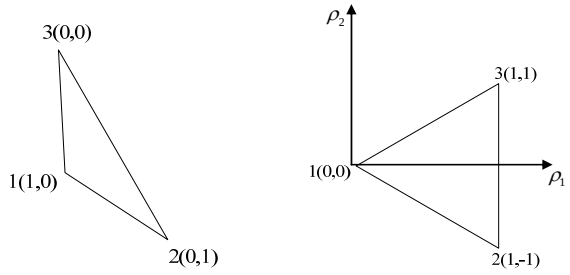

Figure 3. Triangle mapped onto an isoceles triangle

$$\begin{cases} \xi_1 = 1-\rho_1^t \\ \xi_2 = \dfrac{1}{2}\rho_1^{t-1}(\rho_1-\rho_2) \end{cases} \tag{6}$$

The element integrals of diagonal submatrices of matrix H in Eqn. (4) have $O(1/r^2)$ singularity. The integrals can be computed by rigid body movement method [16] as:

$$H^{ij}_{ee} = (\delta_{ef} - 1)\sum_{f=1}^{N} H^{ij}_{ef} \tag{7}$$

When all the elements of matrices are obtained, the BEM equations are represented as:

$$Ax = b \tag{8}$$

where A is the coefficient matrix which is dense and nonsymmetric, x is the unknown vector, b is known right-hand-side vector. Solving the Eqn. (8), the boundary results ($\{ru^1_e, ru^2_e, ru^3_e\}^N_{e=1}$, $\{rt^1_e, rt^2_e, rt^3_e\}^N_{e=1}$) are obtained.

2.2 Computation Steps

$S = \{s_e\}^N_{e=1}$ is a set of TEs of solid faces, $V = \{v_\alpha\}^p_{\alpha=1}$ is a set of TEs' vertices, the BEM computation can be implemented as follows:

Input $V = \{v_\alpha\}^p_{\alpha=1}$ and their coordinates $\{x_\alpha, y_\alpha, z_\alpha\}^p_{\alpha=1}$, TEs' vertex number $\{nv_e, nv_{e+1}, nv_{e+2}\}^N_{e=1}$, and boundary conditions $\{u^1_e, u^2_e, u^3_e\}^N_{e=1}$, $\{t^1_e, t^2_e, t^3_e\}^N_{e=1}$.

Output boundary results $\{ru^1_e, ru^2_e, ru^3_e\}^N_{e=1}, \{rt^1_e, rt^2_e, rt^3_e\}^N_{e=1}$.

Step 1 compute the coordinates $\{x^{'}_e, y^{'}_e, z^{'}_e\}^N_{e=1}$ of all TEs' nodes $\{nod_e\}^N_{e=1}$.

Step 2 compute Jacobians of the coordinate transformation $\{J_e\}^N_{e=1}$.

Step 3 compute the direction cosines $\{n_{ex}, n_{ey}, n_{ez}\}^N_{e=1}$ of all TEs' outward normal

Step 4 **for** $e = 1$ **to** N **do**
  for $f = 1$ **to** N **do**
    if $nod_e \notin s_f$ **then** compute H^{ij}_{ef} and G^{ij}_{ef} $(e \neq f)$
    else divide TE s_f into $s_{f,1}, s_{f,2}, s_{f,3}$,compute $G^{ij}_{ff,1}, G^{ij}_{ff,2}, G^{ij}_{ff,3}$, $G^{ij}_{ff} = \sum_{k=1}^{3} G^{ij}_{ff,k}$.
    end
    compute $H^{ij}_{ee} = (\delta_{ef} - 1)\sum_{f=1}^{N} H^{ij}_{ef}$
  end

Step 5 arrange elements to form equations Ax=b, solve the equations.

From the steps above, it is easy to find that the computations of H^{ij}_{ef} and G^{ij}_{ef} $(e \neq f)$ just need the data of TE e and TE f; G^{ij}_{ee} just need the data of TE e; H^{ij}_{ee} need the values of H^{ij}_{ef} $(e \neq f)$. Strong independence exists in the coefficient integrals, which is suitable for parallel computation.

The BEM coefficient matrix is dense and nonsymmetric. Iterative solvers are more efficient than direct solvers in large scale equations. Krylov subspace methods are effective iterative methods for systems of linear equations, the best known Krylov subspace methods are generalized minimum residual (GMRES), biconjugate gradient stabilized (BiCGSTAB), Conjugate gradient, etc. Comparing with other Krylov subspace methods, GMRES is more efficient to BEM solution [17]. GMRES method is chosen to solve the BEM equations in this study. The majority of the solution time is spent on the linear algebra computation when using GMRES method. The linear algebra computation is very suitable for parallel computation.

3 PARALLEL COMPUTATION USING CUDA

3.1 CUDA Computation Flow

CUDA is a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA GPUs that is accessible to software developers through variants of industry standard programming languages [18]. Under the CUDA, a program is completed by both Host port (CPU) and Device port (GPU), where Host port is in charge of serial parts, and Device port is in charge of parallel parts. The functions defined in Device port is called kernel functions. Threads are the executable units of GPU. The threads that reside on the same processor core structure a thread block. There is a limit to the number of threads per thread block, but a kernel can be executed by multiple equally-shaped thread blocks. Therefore the total number of threads is equal to the number of threads per block multiplying the number of blocks. CUDA is especially suitable for fine granular data parallel computation because of large number of threads.

The flow of CUDA parallel BEM program in the paper is shown in Fig. 4. Host inputs the vertices' coordinates, vertices' number, etc, and transmits the data to graphics memory. Device parallel computes the coefficient integrals and forms the equations according to the boundary conditions. Finally, Host cooperates with Device to solve equations, and the results are obtained.

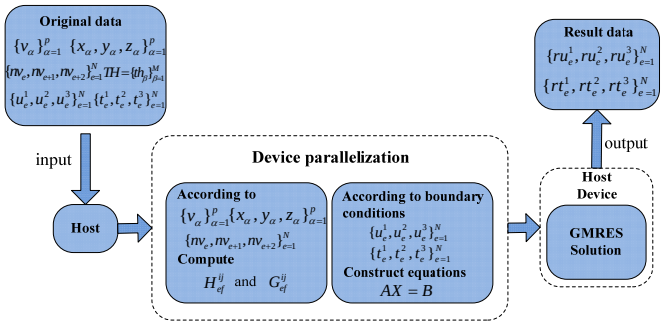

Figure 4. Flow of CUDA parallel BEM

3.2 Parallel Computations of Coefficients

In order to keep balance in threads' loads, there are 3 kernel functions used to complete all the coefficient computations:

1) Kernel1 computes H_{ef}^{ij} and G_{ef}^{ij} $(e \neq f)$;

2) Kernel2 computes G_{ee}^{ij}, and every TE (s_e) is divided into 3 sub-TEs ($s_{e,1}$,$s_{e,2}$,$s_{e,3}$) and needs 3 threads to work;

3) Kernel3 computes H_{ee}^{ij}.

The mapping relationship between coefficient integrals and threads is illustrated in Fig. 5 (neglect matrix partition). If the BEM matrix scale is too large, it has to be parted. The mapping relationship needs to be revised that makes the threads match to each block matrix.

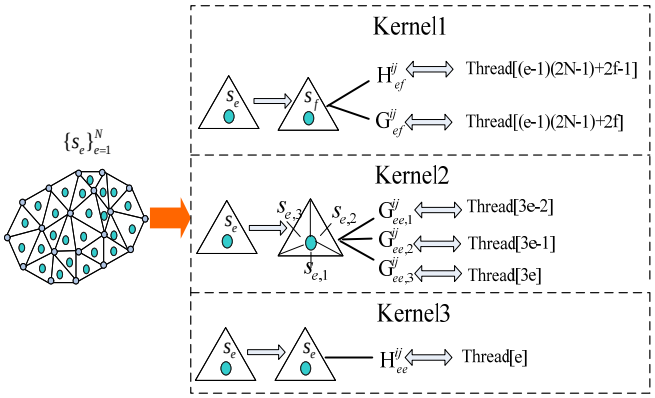

Figure 5. Thread mappings of coefficients

The parallel computation steps of coefficient integrals are as follows:

M is the total thread number, and M' is the necessary thread number when the computations of matrices H and G can be finished parallel.

Input TEs $S=\{s_e\}_{e=1}^{N}$, TEs' vertices $V=\{v_\alpha\}_{\alpha=1}^{p}$ and their coordinates $\{x_\alpha, y_\alpha, z_\alpha\}_{\alpha=1}^{p}$,TEs' vertex number $\{nv_e, nv_{e+1}, nv_{e+2}\}_{e=1}^{N}$, thread set $TH=\{th_\beta\}_{\beta=1}^{M}$, boundary conditions $\{u_e^1, u_e^2, u_e^3\}_{e=1}^{N}$ and $\{t_e^1, t_e^2, t_e^3\}_{e=1}^{N}$.

Output boundary results $\{ru_e^1, ru_e^2, ru_e^3\}_{e=1}^{N}, \{rt_e^1, rt_e^2, rt_e^3\}_{e=1}^{N}$.

Step 1 compute the coordinates $\{x_e^{'}, y_e^{'}, z_e^{'}\}_{e=1}^{N}$ of all TEs' nodes $\{nod_e\}_{e=1}^{N}$.

Step 2 compute Jacobians of the coordinate transformation $\{J_e\}_{e=1}^{N}$.

Step 3 compute the direction cosines $\{n_{ex}, n_{ey}, n_{ez}\}_{e=1}^{N}$ of all TEs' outward normal .

Step 4 **if** M≥M' **then**

① $th_1 \sim th_{2N(N-1)}$ compute $\{H_{ef}^{ij}\}_{e=1, f=1}^{e=N, f=N}$ and $\{G_{ef}^{ij}\}_{e=1, f=1}^{e=N, f=N}$ $(e \neq f)$,

② $th_1 \sim th_{3N}$ divide $\{s_e\}_{e=1}^{N}$ into $\{s_{e,1}, s_{e,2}, s_{e,3}\}_{e=1}^{N}$, compute $G_{ee,1}^{ij}, G_{ee,2}^{ij}, G_{ee,3}^{ij}$, $G_{ee}^{ij} = \sum_{k=1}^{3} G_{ee,k}^{ij}$.

③ $th_1 \sim th_N$ compute $\{H_{ee}^{ij}\}_{e=1}^{N} = \{(\delta_{ef} - 1)\sum_{f=1}^{N} H_{ef}^{ij}\}_{e=1}^{N}$

else divide matrices H and G into block matrices, get the matrix sets $QH=\{qh_k\}_{k=1}^{L}$ and $QG=\{qg_k\}_{k=1}^{L}$, the scale is N1×N1 whose elements can be parallel finished by M threads.

for k=1 **to** L do

if $qh_k \cap \{H_{ee}^{ij}\}_{e=1}^{N} = \varnothing$ **then** $th_1 \sim th_{N1 \cdot N1}$

compute $H_{ef}^{ij} \in qh_k$

else $th_1 \sim th_{N1(N1-1)}$ compute $H_{ef}^{ij} \in qh_k$ $(e \neq f)$

if $qg_k \cap \{G_{ee}^{ij}\}_{e=1}^{N} = \varnothing$ **then**

$th_1 \sim th_{N1 \cdot N1}$ compute $G_{ef}^{ij} \in qg_k$

else① $th_1 \sim th_{N1(N1-1)}$ compute $G_{ef}^{ij} \in qg_k$ ($n \neq m$),

② $th_1 \sim th_{3N1}$ compute $\{G_{ee,1}^{ij}, G_{ee,2}^{ij}, G_{ee,3}^{ij}\} \in qg_k$,

get $\{G_{ee}^{ij}\} \in qg_k$

end

parallel compute $\{H_{ee}^{ij}\}_{e=1}^{N} = \{(\delta_{ef} - 1)\sum_{f=1}^{N} H_{ef}^{ij}\}_{e=1}^{N}$

3.3 Parallel Computations of GMRES

The GMRES method is an iterative method for the numerical solution of a system of linear equations as $Ax = b$. The nth Krylov subspace for the equations $Ax = b$ is $K_n = span\{b, Ab, \cdots, A^{n-1}b\}$. GMRES approximates the exact solution by the vector $x_n \in K_n$ that minimizes the norm of the residual $b - Ax_n$. The vectors $b, Ab, \cdots, A^{n-1}b$ are almost linearly dependent, instead of these vectors, the Arnoldi iteration is used to find orthonormal vectors $r_1, r_2, \cdots, r_n$ which form a basis for K_n. To accelerate the iterative solution, we use the diagonal coefficients of matrix A to form a precondition matrix M which can reduce the number of iterations for a given tolerance. When the number of iterations increases, the requiring storage of the orthonormal vectors increases. In order to solve this problem, we restart the GMRES algorithm every *m* steps. The GMRES algorithm flow is described below:

Step 1 Start:

choose x_0 and m,

compute $r_0 = b - Ax_0, \beta = \| r_0 \|, v_1 = r_0 / \|r_0\|$.

Step 2 Arnoldi iteration:

for $i = 1 : m$

$\hat{v}_{i+1} = M^{-1} A v_i$

for $k = 1 : i$

$h_{ki} = (\hat{v}_{i+1}, v_k)$, $\hat{v}_{i+1} = \hat{v}_{i+1} - h_{ki} v_k$

end

$h_{i+1,i} = \| \hat{v}_{i+1} \|$, $v_{i+1} = \hat{v}_{i+1} / h_{i+1,i}$

end

Step 3 Form the approximate solution:

define $W_m = [v_1, \cdots, v_m]$, $\tilde{H}_m = \{h_{l,j}\}_{1 \le l \le j+1, 1 \le j \le m}$

compute $x_m = x_0 + W_m y_m$,

where y_m minimizes $\| \beta e_1 - \tilde{H}_m y_m \|$.

Step 4 Restart:

compute $r_m = b - Ax_m$; **if** satisfied **then** stop

else compute $x_0 = x_m$, $v_1 = r_m / \|r_m\|$, go to **Step 2**.

Assuming the number of iterations for an N×N matrix A is s, the complexity and the times of linear algebra computations are shown in Tab. 1. When the iteration number is a small number, the Arnoldi iteration takes the majority time of solution. From Tab.1, we can get that most of the time of the Arnoldi iteration is spended on matrix-vector multiplication (gemv in Tab. 1) which has high parallelism.

Table 1. Complexity and times of linear algebra computations

computation	expression	complexity	times
gemv	y=Ax	O(n^2)	s+1
dot	a=(x,y)	O(n)	s(s+1)/2
nrm2	a=‖x‖	O(n)	s+1
scal	x=ax	O(n)	s+1
axpy	y=ax+y	O(n)	s(s+1)/2+2

In this study, the solution of BEM system equations by GMRES method is completed by both Host port and Device port under the CUDA. The coefficient matrix *A* and the vector *b* are generated by Device port, stored in the global memory on GPU. Host port controls the flow of the solution. Device port parallel completes matrix-vector multiplication and other linear algebra computations.

The matrices generated in the computations can be stored in row format or column format. There is no difference between the two formats in forming the matrices H and G, because all the elements in H and G are independent. However, the formats have influence on the linear algebra computations in the solution. The column format is adopted in this study. The column data of matrices is put into neighbour addresses to satisfy the coalesced access principle which requires the neighbour threads access neighbour addresses. CUDA can read the data quickly during the process of linear algebra computations when the column format is used.

4 EXAMPLE AND ANALYSIS

4.1 Preparation Work

The 3D elastostatics BEM program codes are executed on a desktop computer:

CPU: Intel Core2 Quad CPU Q9550 2.83GHz;
OS: Windows XP Professional;
GPU: NVIDIA GeForce GTX 285 1GB;
RAM: DDR3 SDRAM 2GB;
COMPILERS: Microsoft Visual Studio 2005，NVIDIA CUDA 3.0.

The characteristic advantage of the BEM is the reduction in dimensionality of the problem by one, which means that 2D meshes can be used for the 3D elastostatics BEA. In this study,

we use the ACIS facet meshes for BEA to avoid writing a particular mesh algorithm.

The 3D ACIS Modeler is a 3D modelling engine owned by Spatial Corporation. It is the geometry kernel of AutoCAD, Autodesk Inventer, and many other well known CAD applications [19]. The Faceter Component is one of the ACIS components, which generates and controls approximate polygonal representations. The Faceting function of the Faceter Component generates approximate polygonal representations for the faces of entities while maintaining edge consistency between adjacent faces. Face faceting is performed by subdividing the face in parameter space with a grid whose increments are determined through refinements. The faceted representation of a face is also called a mesh [20].

Considering TE is used easily and widely, a triangle meshing algorithm for BEM based on ACIS meshing is proposed in this study. Because the origin ACIS face meshing function does not satisfy the BEA requirements, some improvements are completed as follows:

1) reordering vertex number of each TE as counterclock-wise;

2) extracting vertices' coordinates and reordering them as BEA need;

3) adding element control function which can control element size, normal tolerance, and grid mode.

The ACIS meshing algorithm is implemented by developing the 3D CAD software InterSolid whose kernel is ACIS, developed by National CAD Support Software Engineering Research Center of China. There are 4 parameters used to control the meshes as Fig. 6.

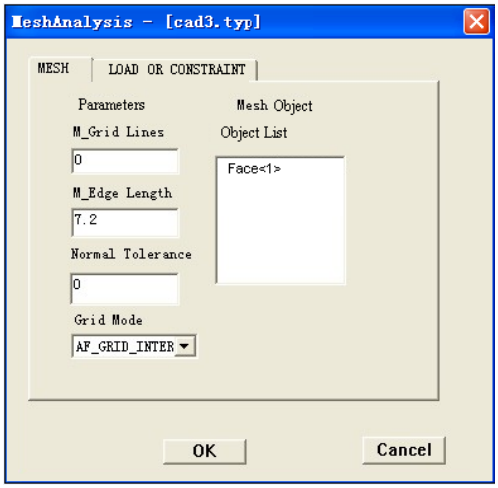

Figure 6. Dialog box of ACIS meshing

Where

M_Grid Line controls the total edge number, default is 500;

M_Edge Length controls the max element edge length;

Normal Tolerance is the angle between the surface normals at the two adjacent vertices of a facet element, by setting this normal deviation refinement, the user specifies how accurately the facets are representing the surface and the quality of rendering desired;

Grid Mode determines whether a grid is used and whether the points where the grid cuts the edges should be inserted into the edge discretization.

In order to simplify the pre-processing of BEA, the boundary constraint adding function and boundary load adding function are developed in InterSolid software, shown in Fig. 7.

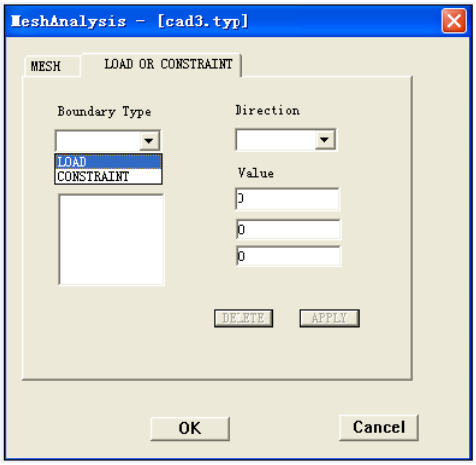

Figure 7. Dialog box of constraint and load

The data outputted from InterSolid can be used by BEA directly. CAD/CAE integration can be realized well during the developed BEM program codes under InterSolid. The flowchart of CAD/CAE design analysis is shown in Fig. 8.

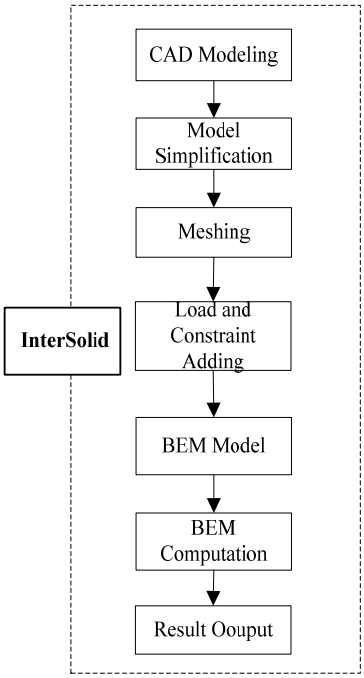

Figure 8. Flowchart of CAD/CAE design analysis

4.2 Results and Analysis

Using a link and a fork part as examples, the constraints and loads are shown in Fig. 9, Elastic modulus is 200000 Mpa, Possion ratio is 0.3, load value is 10Mpa.

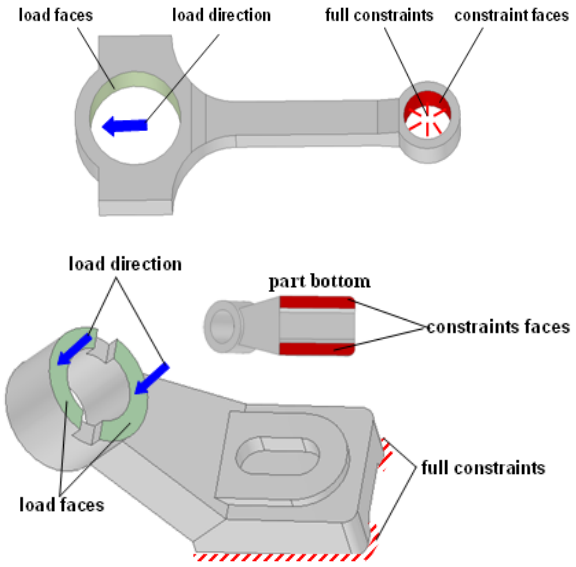

Figure 9. Constraints and loads of the part

Both BEM and FEM (using software ANSYS10.0) are discussed. The initial conditions are listed in Tab. 2. BEM uses the const TEs, FEM uses the tetrahedral quadratic elements. The face element number of the BEM is much less than the volume element number of the FEM when the same element size is used. This is because the BEM has dimension reduction characteristics that can decrease the unknowns' scale.

Table 2. Initial conditions of examples

Examples	BEM Element number	BEM DOFs	FEM Element number	FEM DOFs
Link	2706	8118	9217	46980
Fork part	3390	10170	13628	64962

Table 3 lists each part of time consumption when we use 4 point Hammer integrals and GMRES solver. Table 4 lists the speedups corresponding to Tab. 3. Table 5 lists each part of time when 7 point Hammer integrals and GMRES solver are used. Table 6 lists the speedups corresponding to Tab. 5.

Table 3. Computation time using 4 point Hammer integrals

Examples	CPU Coefficient time t1	CPU Solution time t2	GPU Coefficient time t3	GPU Solution time t4
Link	7.281s	43.992s	0.176s	0.984s
Fork part	9.046s	71.719s	0.234s	1.813s

Table 4. Speedups using 4 point Hammer integrals

Examples	Speedup1 t1/t3	Speedup2 t2/t4	Speedup3 (t1+t2)/(t3+t4)
Link	41.37	44.71	44.20
Fork part	38.66	39.56	39.46

Table 5. Computation time using 7 point Hammer integrals

Examples	CPU Coefficient time t1	CPU Solution time t2	GPU Coefficient time t3	GPU Solution time t4
Link	11.466s	45.614s	0.213s	0.990s
Fork part	14.574s	20.968s	0.296s	0.469s

Table 6. Speedups using 7 point Hammer integrals

Examples	Speedup1 t1/t3	Speedup2 t2/t4	Speedup3 (t1+t2)/(t3+t4)
Link	53.83	46.07	47.45
Fork part	49.24	44.71	46.46

The results show that the GPU parallel computations are much faster than CPU computations in both coefficient integrals and GMRES solutions. The speedups can reach about 40 times.

Comparing Tab. 3 with Tab. 5, the influence of integral accuracy can be obtained. More time is needed in coefficient integrals when 7 point integrals is used, but higher accuracy is reached that can influence the coefficient value. When the coefficients change, the solution time is influenced. The solution time using 7 point integrals is a little longer than using 4 point integrals in the link; but the fork part solution time using 7 point integrals is much shorter than using 4 point integrals. Several other models are tested in order to explore the influence on integral accuracy. It is found that most models (especially complicated geometries) using 7 point integrals are faster than those using 4 point integrals in GMRES solutions as the fork part, other few models using 7 point integrals are equal to or a little slower than that using 4 point integrals. So the high accurate integral is recommended to BEM.

The Gaussian Elimination (GE) method, a direct solution method, is compared with GMRES. The GE method solver took 576.92s and 1076.27s to solve the link and the fork part. It is much slower than GMRES and is not suitable for large scale problems despite of its high accuracy.

Table 7 lists the GMRES solutions between the TEs generated by ACIS and the TEs generated by ANSYS where 7 point integrals are used. It can be seen that the solution using TEs generated by ANSYS is faster than that generated by ACIS (especially for complicated geometries). Because there is a good meshing algorithm in ANSYS to generate better quality elements that are more uniform and less sharp triangle elements.

Table 8 lists the computation time between the BEM using CUDA (7 point Hammer integrals) and the FEM (ANSYS 10.0). It shows that the BEM using CUDA is faster than the FEM both in coefficient computations and solutions. But when

the models are very complicated, the conventional BEM solution time will increase quickly, more quickly than FEM, the BEM using CUDA will be slower than FEM. In order to solve this problem, a fast BEM has to be adopted which is not discussed in this paper.

Table 7. Different quality TEs' influence on solutions

Examples	ACIS Element number	ANSYS Element number	ACIS Solution time	ANSYS Solution time
Link	2706	2786	0.990s	0.879s
Fork part	3390	3410	0.469s	0.248s

Table 8. Computation time of BEM using CUDA and FEM

Examples	FEM Coefficient time	FEM Solution time	BEM Coefficient time	BEM Solution time
Link	4.343s	7.570s	0.213s	0.990s
Fork part	5.687s	15.907s	0.296s	0.469s

Figure 10 shows the displacement results of the examples at X direction, including BEM using CUDA and FEM. The displacement distributions of BEM using CUDA and FEM are the same. The BEM using CUDA displacement values have some errors in the areas where the displacements change greatly. This is because the BEM uses const TEs whose accuracy is low, especially for large deformation areas, the FEM uses quadratic elements whose accuracy is high. A detail discussion can be found in literature [21]. When BEM uses linear elements or quadratic elements, this problem will be solved.

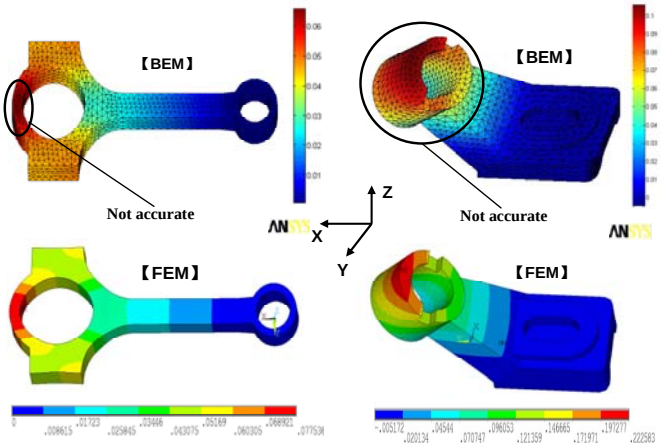

Figure 10. Results of examples

5 CONCLUSIONS AND FUTURE WORK

In this paper, we set up a 3D elastostatics BEA framework which uses ACIS meshing to generate TEs and GPU (under CUDA) parallel method to accelerate computation. The framework merges the CAE analysis with CAD modeling in InterSolid that decreases the pre-processing time substantially. The ACIS meshing generates TEs quite fast, because it directly acquires the TE information from the triangle facets which was used to represent geometric faces in ACIS. Although BEM can easily mesh models, its efficiency in solution is so poor that GPU parallel method is introduced for fast computation. The examples show that the BEM using CUDA in this paper is much faster than the conventional counterpart (about 40 times).

In the future, there are still several problems to be solved to improve and complete our work:

1) The first is to improve the element quality of ACIS meshing. The purpose of ACIS meshing is faceted representation, accordingly the meshes on surfaces are dense and the meshes on the planes are sparse, so that some large deformation meshes appear at the intersections between surfaces and planes. The quality of ACIS large deformation mesh elements is not suitable for BEA. In order to improve the quality of TEs, a good triangle mesh optimization algorithm based on ACIS is demanded indeed.

2) Another is to apply linear elements or quadratic elements to replace const elements because of the low accuracy of const elements.

3) The third is to apply GPU parallel computation to fast BEM such as Fast Multipole BEM, Adaptive Cross Approximation BEM, which accelerate the solutions of BEM to $O(n)$. Owing to the matrices produced by BEM are smaller than FEM matrices, if the solution computation is $O(n)$, the BEM does not only require less memory but also computes much faster than FEM. When the GPU parallel technology is applied to fast BEM, the speed of computation will be promoted several times so that large scale problems can be solved by PC.

4) The fourth is to add a model automatic simplification function that can simplify the small features of models. The simplified models are more easily meshed and produce fewer elements, which can accordingly promote the CAE efficiency.

ACKNOWLEDGMENTS

This research has been supported by the National Natural Science Foundation of China (50975107 and 60804050).

REFERENCES

[1] Lou, R., et al, 2009. "Towards CAD-less finite element analysis using group boundaries for enriched meshes manipulation". ASME International Design Engineering Technical Conference, San Diego, California, DETC2009-86575.

[2] Raphael, B. and Krishnamoorthy, C., 1993. "Automating finite element development using object oriented techniques". *Engineering Computations*, 10(3), pp. 267-278.

[3] Park, H.S. and Dang, X.P., 2010. "Structural optimization based on CAD-CAE integration and metamodeling techniques". *Computer-Aided Design*, 42(10), pp. 889-902.

[4] Brebbia, C., Telles, J. and Wrobel, L.,1984. *Boundary*

element techniques: theory and applications in engineering. Springer-Verlag, Berlin.
[5] Liu, Y., 2009. *Fast multipole boundary element method : theory and applications in engineering*. Cambridge University Press, New York,Chap.3, pp.47-48.
[6] Symm, G., 1984. "Boundary elements on a distributed array processor". *Engineering Analysis*, 1(3), pp. 162-165.
[7] Davies, A., 1988. "The boundary element method on the ICL DAP". *Parallel Computing*, 8(1-3), pp. 335-343.
[8] Davies, A., 1996. "Parallel implementations of the boundary element method". *Computers and Mathematics with Applications*, 31(6), pp. 33-40.
[9] Cunha, M., Telles J., and Coutinho, A.,2004. "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.
[10] Araujo, F.C., d'Azevedo, E.F, and Gray,L.J., 2010. "Boundary-element parallel-computing algorithm for the microstructural analysis of general composites". *Computers and Structures*, 88(11-12), pp. 773-784.
[11] Kumar, N., Satoor, S. and Buck, I.2009. "Fast Parallel Expectation Maximization for Gaussian Mixture Models on GPUs Using CUDA". *11th IEEE International Conference on High Performance Computing and Communications*, pp.103-109.
[12] Takahashi, T. and Hamada, T. 2009. "GPU-accelerated boundary element method for Helmholtz'equation in three dimensions". *International Journal for Numerical Methods in Engineering*, 80(10), pp. 1295-1321.
[13] Cruse, T., 1969. "Numerical solutions in three dimensional elastostatics". *International Journal of Solids and Structures*, 5(12), pp. 1259-1274.
[14] Dunavant, D., 1985. "High degree efficient symmetrical Gaussian quadrature rules for the triangle". *International Journal for Numerical Methods in Engineering*, 21(6), pp. 1129-1148.
[15] Hu, S., Chen, G. 1997. "A new triangle polar coordinate transformation". *Chinese Journal of Computational Mechanics*, 14(3), pp.1129– 1148. (in Chinese)
[16] Gao, X. and Davies,T., 2002. *Boundary element programming in mechanics*. Cambridge University Press, New York,Chap.4, pp.51-52.
[17] Xiao, H. and Chen, Z., 2007. "Numerical experiments of preconditioned Krylov subspace methods solving the dense non-symmetric systems arising from BEM". *Engineering Analysis with Boundary Elements*, 31(12), pp. 1013-1023.
[18] NVIDIA Corporation, 2010. NVIDIA CUDA programming guide, Version 3.0.
[19] Kaufmann, H., 2009. "Dynamic Differential Geometry in Education". *Journal for Geometry and Graphics*, 13(2), pp. 131-144.
[20] Spatial Corporation, 2004. ACIS R15 Online Help.
[21] Lei, T., Yao, Z., and Wang, H., 2007. "High performance parallel computations of 3-D fast multipole boundary element method". *Journal of Tsinghua University* (*Science and Technology*), 47(2), pp. 280-283. (in Chinese)

ANNEX A

HAMMER INTEGRAL TABLE

Point number	i	Ordinate (ξ_1^i)	Ordinate (ξ_2^i)	Ordinate (ξ_3^i)	Weights(W_i)
m=4	1	1/3	1/3	1/3	-9/16
	2	3/5	1/5	1/5	25/48
	3	1/5	3/5	1/5	25/48
	4	1/5	1/5	3/5	25/48
m=7	1	0.33333333	0.33333333	0.33333333	0.22500000
	2	0.79742699	0.10128651	0.10128651	0.12593918
	3	0.10128651	0.79742699	0.10128651	0.12593918
	4	0.10128651	0.10128651	0.79742699	0.12593918
	5	0.05971587	0.47014206	0.47014206	0.13239415
	6	0.47014206	0.05971587	0.47014206	0.13239415
	7	0.47014206	0.47014206	0.05971587	0.13239415