GPU parallel computation of topology optimization based on EFG method

***†S. G. Gong, Q. L. Liu, G. L. Xie, H. L.Lu, and J. P. Zhang**

School of Mechanical Engineering, Xiangtan University, Xiangtan 411105, China

*Presenting author: gongsg@xtu.edu.cn

†Corresponding author: gongsg@xtu.edu.cn

Abstract

Aiming at the problem of low efficiency and time-consuming in topology optimization based on element-free Galerkin (EFG) method, a new implementation of topology optimization based on EFG method and Graphic Processing Unit (GPU) parallel computing is presented by using node-by-node method and interacting nodes pairs to carry out the sensitivity analysis, and the corresponding computational formulas are derived. The GPU parallel execution model is designed to be used in the sensitivity analysis of objective function and implementing of optimization criterion (OC) method, and the flowchart is also given. The two examples of topology optimization are achieved, and the results obtained show that the proposed method is verified to be efficient and feasible. On the premise that the calculating accuracy is met, the 24 times speedup is obtained.

Keywords: Topology Optimization; GPU; Parallel Computing; EFG method; Sensitivity Analysis

1. Introduction

Structural topology optimization has played an important role in lightweight design of industrial product parts [Zargham et al. (2016); Chen et al. (2016)]. Its numerical methods are mainly based on element based method, such as the finite element. The numerical instabilities, such as checkerboards, mesh-dependencies and local minima, can occur in application of topology optimization because of using element based method [Sigmund O (1998)]. To get clear topological outline, the different topology optimization methods or filtering technologies have been put forward, such as level set method, evolutionary structural optimization method(ESO) , and so on [Zargham et al. (2016)].

The Element-Free Galerkin (EFG) method requires only nodal data, and the element connectivity between nodes is eliminated [Lu et al. (1994)]. As the EFG method has high rate of convergence, high computing accuracy and good computing stability, the scholars have tried to us it in topology optimization. Gong et al. [2012] has presented the topology optimization method based on EFG method by selecting the nodal density as the design variables. His research results show that the numerical instabilities in topology optimization have significantly improved. The same conclusion has also achieved by authors [Yang et al. (2016)]. Despite all this, the EFG method also has some shortcomings that are the EFG method has poor computational efficiency and time-consuming [Belytschko et al. (1996)]. For this reason, many scholars try to make use of parallel algorithm to improve computing efficiency of EFG method. Singh [2004] has researched parallel assembling of stiffness matrix and parallel solving of linear equations, and achieved the parallel computing of EFG method. Zeng et al. [2008] discussed the parallel computing the shape functions and their derivatives of Moving Least Squares method, and his research results show the EFG method

has large parallel potential and high parallel efficiency.

In particular, as the NVIDIA Company released Compute Unified Device Architecture (CUDA) and the Open Compute Language (OpenCL) in recent years, parallel acceleration of Graphic Processing Unit (GPU) has received considerable attention. Karatarakis et al. [2013] presented the GPU parallel algorithm to assemble stiffness matrix of EFG method. And the GPU parallel algorithm of solving linear equations by using conjugate gradient method has been presented by authors [Bolz et al. (2003)]. Gong et al. [2015] presented a GPU acceleration parallel algorithm of EFG method by using node pair-wise approach to assemble the stiffness matrix, the maximum speedup ration is up to 17 times. Stephan et al. [2011] investigates the GPU parallel computing of topology optimization with the solid isotropic material with penalization approach based on FEM, and they found that the computing efficiency of GPU is faster than a 48 core shared memory CPU system. In addition, Challis et al. [2014] has carried out the parallel computing of topology optimization with level set method based on FEM. They have also found that the GPU is utilized more effectively at the higher scale problem. To solve large FE model in topology optimization, Martínez-Frutos et al. [2016] has presented a multi-GPU system in terms of memory consumption and processing time. Cai et al. [2016] proposed the parallel computing method of Bi-directional Evolutionary Structural Optimization (BESO) based on MatLab and GPU. In a word, the GPU parallel computing has become a very popular accelerating computational method.

Hence, in this paper, our purpose is to present an entire topology optimization method based on the GPU parallel computing and EFG method. The parallel algorithm of sensitivity analysis in topology optimization and optimization criterion (OC) method will be explored in detail, and the flow chart of GPU parallel computing is given then. The two numerical examples are achieved based on the proposed method, and the influence of nodes to speedup will be discussed.

2. Topology optimization based on EFG method

In this work, we utilize the penalty function method to impose the essential boundary condition because the shape functions in the EFG method do not satisfy the Kronecker delta property. The minimum structural compliance c is chosen as the objection function, and the volume ratio f as the constraint, the relative density of nodes ρ_I as the design variables, the topology optimization model by using Solid Isotropic Material with penalization (SIMP) interpolation can is now given as follows.

$$\begin{cases} \text{find} & \rho(x) \\ \min & \mathrm{c}(\boldsymbol{\rho}) = \left(\boldsymbol{F} + \boldsymbol{F}^a\right)\boldsymbol{U} \\ \text{s.t} & V = fV_0 \\ & \left(\boldsymbol{K} + \boldsymbol{K}^a\right)\boldsymbol{U} = \boldsymbol{F} + \boldsymbol{F}^a \\ & 0 < \rho_{\min} \le \rho_I \le 1 \end{cases} \tag{1}$$

where $\boldsymbol{K}$ is the global stiffness matrix, $\boldsymbol{K}^a$ is the global penalty stiffness matrix, $\boldsymbol{F}$ is the global force vector. $\boldsymbol{F}^a$ denotes the global penalty force vector, and V represents the material volume of design domain, These parameters can be defined as follows, respectively.

$$\boldsymbol{K} = \int_\Omega \rho_I^{(p)} \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D}\boldsymbol{B}\mathrm{d}\Omega, \qquad \boldsymbol{K}^\alpha = \alpha \int_{\Gamma_u} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \cdot \mathrm{d}\Gamma,$$

$$\boldsymbol{F} = \int_{\Gamma_t} \boldsymbol{\Phi}^{\mathrm{T}} \bar{\boldsymbol{t}} \mathrm{d}\Gamma + \int_\Omega \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{b}\mathrm{d}\Omega, \qquad \boldsymbol{F}^\alpha = \alpha \int_{\Gamma_u} \boldsymbol{\Phi}^{\mathrm{T}} \bar{\boldsymbol{u}} \mathrm{d}\Gamma, \qquad V = \int_\Omega \rho \mathrm{d}\Omega$$

in which $\boldsymbol{B}$ is the geometric matrix, $\boldsymbol{D}$ is the elasticity matrix. α is the penalty factor, in general, $\alpha = \left(10^3 \sim 10^7\right) E$, and E is the elastic modulus. $\boldsymbol{\Phi}$ denotes the shape function; $\bar{\boldsymbol{t}}$ are the traction forces applied on the natural boundary, $\boldsymbol{b}$ is the body force vector, $\bar{\boldsymbol{u}}$ is the known displacements on the essential boundary, $\boldsymbol{U}$ is the approximation nodal displacement, and V_0 is the initial volume of the design domain. In the meantime, in order to avoid the singularity in the optimal process, the lower limiting value of relative density $\rho_{\min}$ is set to 0.001. Meanwhile, the relative density of any node in the design domain is given as follow

$$\rho = \sum_{I=1}^{N_c} \phi_I \rho_I$$

On the other hand, we select the Optimization Criterion (OC) method [Sigmund (2001)] to update the design variables. The iteration scheme is given by

$$\rho_I^{(k+1)} = \begin{cases} \max\left(\rho_{\min}, \rho_I^{(k)} - m\right) & \text{if } \max\left(\rho_{\min}, \rho_I^{(k)} - m\right) \geq \left(\mathrm{B}_I^{(k)}\right)^\eta \rho_I^{(k)} \\ \left(\mathrm{B}_I^{(k)}\right)^\eta \rho_I^{(k)} & \text{if } \max\left(\rho_{\min}, \rho_I^{(k)} - m\right) < \left(\mathrm{B}_I^{(k)}\right)^\eta \rho_I^{(k)} < \min\left(1, \rho_I^{(k)} + m\right) \\ \min\left(1, \rho_I^{(k)} + m\right) & \text{if } \left(\mathrm{B}_I^{(k)}\right)^\eta \rho_I^{(k)} \geq \min\left(1, \rho_I^{(k)} + m\right) \end{cases} \tag{2}$$

where superscript k is the number of iteration. In order to guarantee the stable of iteration, the damping coefficient η and a positive move-limit m are introduced, respectively, and B_I can be found from the optimality condition as

$$B_I = \left(-\frac{\partial c}{\partial \rho_I}\right) \Big/ \left(\lambda \frac{\partial V}{\partial \rho_I}\right)$$

where λ is a Lagrange multiplier that can be obtained by a bi-section algorithm.

The sensitivity analysis of the objective function and volume can be expressed as follows, respectively.

$$\frac{\partial V}{\partial \rho_I} = \int_\Omega \phi_I \mathrm{d}\Omega \tag{3}$$

$$\frac{\partial c}{\partial \rho_I} = -\boldsymbol{U}^{\mathrm{T}} \frac{\partial \boldsymbol{K}}{\partial \rho_I} \boldsymbol{U} \tag{4}$$

where $\frac{\partial \boldsymbol{K}}{\partial \rho_I}$ is given by

$$\frac{\partial \boldsymbol{K}}{\partial \rho_I} = \int_\Omega p \rho^{p-1} \phi_I \boldsymbol{B}^{\mathrm{T}} \boldsymbol{D} \boldsymbol{B} \mathrm{d}\Omega \tag{5}$$

where superscript p is used as a penalty factor enforcing a 0/1 distribution as intermediate values are greatly reduced.

More details on topology optimization based on EFG method in general can reference to the literature [Gong et al. (2012); Gong et al. (2009)].

3. Implementation of GPU parallel algorithm

3.1 Date storage and access

NVIDIA's GPU employing the isolated storage system of multi-level memory is an independent computing system, but there exist giant difference in feature of different memory, and their location, access permission and life cycle are different. So the rational allocation and use of GPU memory in the CUDA architecture is important for improving the performance of GPU program, especially topology optimization based on EFG method requires processing huge amount of data, and using frequencies and ways between different data are also different. Aiming at these problems, we take into account following ways to carry out optimization of data storage and access.

1) Subject to the limit of transmission efficiency of the PCI-E port, the data transmission bandwidth between CPU and GPU is much less than that of video memory. In order to avoid time-consuming in the data transmission, according to the needs of the program design, the prepared basic data will be transferred once from CPU to GPU instead of being transferred repeatedly, and only the residual error will be returned to CPU for looping control in the each iteration process. Finally, the optimization results are also transmitted to the host memory after the optimization.

2) The memory capacity of stiffness matrix in the EFG method is much bigger than the transitional FEM. Meanwhile, the data including the model data, updated data in the iterative

can be only stored in the GPU global memory. But the global memory has high access delay, and it is easy to become the access bottleneck. In order to achieve the highest access rate, the address of the first element in each row of data array is aligned by programming. This approach will be beneficial to satisfy the combined access requirements and ensure the maximum effective bandwidth.

3) Each adding a processor in the GPU has a cache to speed up reading operation from constant memory, and compared with the global memory, the constant memory has smaller delay and faster speed for access operation. Hence, for the sake of effectively reducing the access to global memory and improving process performance, the constants accessed frequently, for example, penalty factor, numerical damping coefficient, material properties, etc, will be stored in the constant memory.

4) To take full advantage of the register in the GPU chip, which it has the characteristic of high bandwidth, low delay and small capacity, the number of intermediate variables should be minimized. If the intermediate variables are overmuch, they should be stored in shared memory instead of register, and the registers should be allocated rationally for each thread according to the difference task.

3.2 Parallel algorithm of sensitivity analysis

In the conventional computing of topology optimization, the sensitivity analysis of objective function can be achieved by looping for integral point. That is to say, first we calculate the sensitivity component of all nodes in the influent domain of integral point, and then accumulate them by node number to get the sensitivity array. Due to the node may also appear in the influent fields of multiple Gauss integral points simultaneously. It means that the address of each node in the sensitivity array will be accessed repeatedly, as shown in Fig. 1. This way is not suitable for parallel computing because it is easy to lead to the conflicts of data storage. Therefore, we present a parallel algorithm to carry out the sensitivity analysis by using the interacting nodes pairs that is two nodes in which influence domain between them has overlap region [Gong et al. (2015)], as shown in Fig. 2. This parallel scheme adopts node-by-node method to circularly develop the sensitivity computing of objective function, and the address of node in sensitivity array will be accessed only once. This approach can eliminate the conflicts of data storage and be suitable for parallel computing.

Moreover, there exist a lot of the intermediate variables in the traditional calculation. These variables will appear in an array form to participate in operation. It will seriously degrade the global performance of GPU programs because these variables can only store in global memory. Aiming at this problem, by combining the parallel algorithm as shown in Fig. 2, we propose a following processing method for the sensitivity computing of objection function.

For two-dimension problem, the global stiffness matrix $\boldsymbol{K}$ is a $n \times n$ matrix, and each element $\boldsymbol{K}_{LM}$ in matrix is a 2×2 sub-blocks. Therefore, $\dfrac{\partial \boldsymbol{K}}{\partial \rho_I}$ is also comprised by a series of

sub-blocks $\frac{\partial \boldsymbol{K}_{LM}}{\partial \rho_I}$. According to Eq. (5), $\frac{\partial \boldsymbol{K}_{LM}}{\partial \rho_I}$ can be expressed as

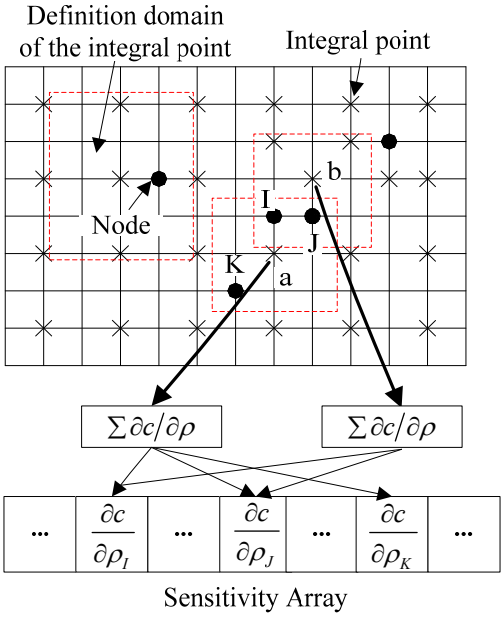

Fig. 1 Assembly the sensitivity matrix by looping for integral point

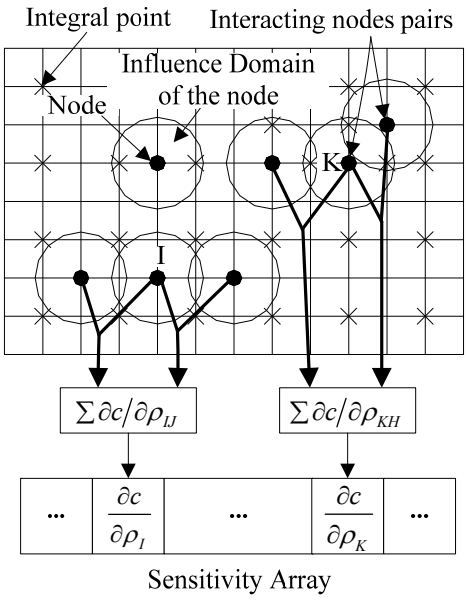

Fig. 2 Assembly the sensitivity matrix by using node-by-node method

$$\frac{\partial \boldsymbol{K}_{LM}}{\partial \rho_I} = \int_\Omega p\rho^{p-1}\phi_I \boldsymbol{B}_L^{\mathrm{T}} \boldsymbol{D}\boldsymbol{B}_M \mathrm{d}\Omega \tag{6}$$

In the EFG method, the global stiffness matrix is also banded sparse matrix, and it means that the value of the most of 2×2 blocks in $\frac{\partial \boldsymbol{K}}{\partial \rho_I}$ is zero. Only the block element value corresponding to the interacting nodes pairs of node I is not zero. So we can obtain the following equation according to Eq. (4).

$$\frac{\partial c}{\partial \rho_I} = -\sum_{i=1}^{NP_I}\left\{\left(\boldsymbol{u}_L^{\mathrm{T}} \frac{\partial \boldsymbol{K}_{LM}}{\partial \rho_I}\boldsymbol{u}_M\right)_i\right\} \tag{7}$$

where $\boldsymbol{u}_L = \left[u_{Lx}, u_{Ly}\right]^{\mathrm{T}}$ is the displacement vector of node L, and NP_I is the number of interacting nodes pairs of node I.

Substituting Eq. (6) into the Eq. (7), we have

$$\frac{\partial c}{\partial \rho_I} = -\sum_{i=1}^{NP_I}\left\{\left(\boldsymbol{u}_L^{\mathrm{T}}\left(\int_\Omega p\rho^{p-1}\phi_I \boldsymbol{B}_L^{\mathrm{T}} \boldsymbol{D}\boldsymbol{B}_M \mathrm{d}\Omega\right)\boldsymbol{u}_M\right)_i\right\} \tag{8}$$

The Eq. (8) can also be written as

$$\frac{\partial c}{\partial \rho_I}=\sum_{i=1}^{NP_I}\left\{\left(\begin{array}{l}\phi_I p\rho_I^{(p-1)}\dfrac{E}{1-\nu^2}HJ(u_{Lx}u_{Mx}\phi_{L,x}\phi_{M,x}+u_{Lx}u_{Mx}\phi_{L,y}\phi_{M,y}\dfrac{1-\nu}{2}+\\ \nu u_{Ly}u_{Mx}\phi_{L,y}\phi_{M,x}+u_{Ly}u_{Mx}\phi_{L,x}\phi_{M,y}\dfrac{1-\nu}{2}+\nu u_{Lx}u_{My}\phi_{L,x}\phi_{M,y}+\\ u_{Lx}u_{My}\phi_{L,y}\phi_{M,x}\dfrac{1-\nu}{2}+u_{Ly}u_{My}\phi_{L,y}\phi_{M,y}+u_{Ly}u_{My}\phi_{L,x}\phi_{M,x}\dfrac{1-\nu}{2})\end{array}\right)_i\right\} \tag{9}$$

where E is Young's modulus and ν is Poisson's ratio; H , J are the weight coefficient and the Jacobi coefficient of integral point, respectively. $\phi_{L,x}$, $\phi_{L,y}$ denote the derivative for shape function ϕ_L with respect to x and y, respectively.

Although this approach may increase the complexity of programming, it can ensure that most of the intermediate variables could be stored in the registers of GPU. And it could be beneficial to improve the performance of program.

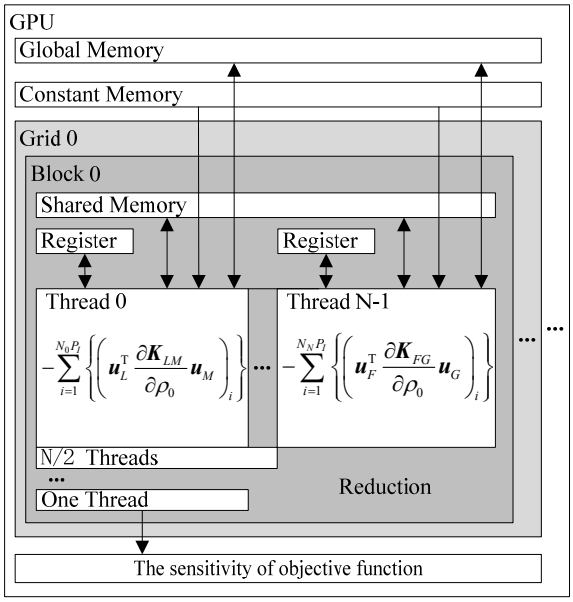

Fig.3 Parallel execution model based on CUDA architecture for the sensitivity of objective function

Synthesizing the above ideas, we present the parallel execution model based on CUDA architecture to achieve parallel computing for the sensitivity analysis of objective function, as shown in Fig. 3. That is, each thread reads data from the global memory and constant memory, and the registers and shared memory is used to store the intermediate variables. And then computing the sensitivity value of objective function of each node by using parallel Reduction Algorithm [Gong et al. (2015)], and its final results will store in global memory. Meanwhile, there are two parallel hierarchies in this algorithm. One or outer loop layer is the computing of nodes, that is, each thread blocks will calculate the objective function sensitivity of one node. The other or inner loop layer is the integral points in the influence domain of node, and each thread in the thread blocks will deal with the computing of an integral point. Finally, we

can obtain the sensitivity component of this node objective function.

3.3 Parallel algorithm of OC method

It shares less time in the whole topology optimization that the OC method is used to update the design variables, but to obtain the final optimal result need repeatedly iterate in the structural topology optimization. Only if the OC method is implemented in the CPU serial computing，a large amounts of data need to be transferred repeatedly between CPU and GPU, and it will seriously reduce the effectiveness of other parallel algorithm. On the other hand, the computing between each step in the iteration process is a serial operation, but some operation including the replacement of design variable, the calculation of the relative density, etc, may be a parallel operation in data level. So these operations can also make use of GPU to carry out parallel computation. Therefore, according to heterogeneous of CPU and GPU as well as the CUBLAS library, we present a parallel algorithm of implementing OC method on GPU, and it is controlled by using a sub-function in this paper. Detailed algorithm flow is expressed as follows.

1) Initialize the CUBLAS library.
2) Define the upper limit $l1$ and lower limit $l2$ of finite interval in host computer, which this interval is used to calculate the Lagrange multiplier by using bisection algorithm, and assigning their initial value at the same time.
3) Copy $\rho^{(k+1)}$ to $\rho^{(k)}$ by calling library function *cublasDcopy*() in CUBLAS library.
4) while($l2$-$l1$>1.0e-4)
   {
   4.1) Calculate the Lagrange multiplier $\lambda_{(i+1)}$ by using bisection algorithm.
   4.2) After updating the Lagrange multiplier, Calculate the design variables $\rho_{(i+1)}^{(k+1)}$ by calling sub-function: *rou_Kernel*<<<*noumnod*, 1>>>() where *numnod* is the total number of nodes in the design domain. This sub-function is written according to Eq.(2), and there is a one-to-one mapping between thread block and node, that is, each thread block will calculate the density $\left(\rho_{(i+1)}^{(k+1)}\right)_I$ of one node.
   4.3) While Lagrange multiplier is $\lambda_{(i+1)}$, calculate the relative density of integral point, that is, $\rho_{in_{(i+1)}} = \sum_{I=1}^{N_c}\left[\phi_I\left(\rho_{(i+1)}^{(k+1)}\right)_I\right]$, by calling sub-function: *rou_integral_Kernel*<<<*numq*2, 32>>>() that is written according to calculation formula of $\rho_{in_{(i+1)}}$, where *numq*2 is the total number of integral points in design domain, and N_c is the number of nodes in influence field of integral point. There

are two parallel hierarchies in this sub-function. One is integral point layer, that is, each thread block computes the relative density of an integral point. The other is node in influence field of this integral point, that is, there is one-to-one mapping between each thread in thread block and node in influence field of this integral point. When the relative density component of each node is gained by using calculation formula $\left[\phi_I\left(\rho_{(i+1)}^{(k+1)}\right)_I\right]$, the relative density of integral point can be obtained by utilizing Reduction Algorithm to accumulate each component.

4.4) While Lagrange multiplier is $\lambda_{(i+1)}$, calculate the total volume V by calling sub-function *Vtol_Kernel*<<<1, 1024>>>() that is written by using expression $V_{(i+1)}=\int_\Omega \rho_{in}\mathrm{d}\Omega$. In this sub- function, each thread in the thread blocks only computes the volume component of a integral point, and the total volume $V_{(i+1)}$ of design domain can be obtained by accumulating volume component.

4.5) Judge the size of design domain volume $V_{(i+1)}$. if less than, then $l2=\lambda_{(i+1)}$, else $l1=\lambda_{(i+1)}$.

}

5) Clear the environment of CUBLAS library

6) Return the design variable value of node $\rho_{(i+1)}^{(k+1)}$.

3.4 Flowchart of parallel algorithm based on GPU

Combining the above ideas with the traditional topology optimization algorithm based on EFG method, in this paper, we propose the heterogeneous parallel program structure of CPU and GPU on CUDA framework, and the flowchart of parallel algorithm is shown in Fig.4. The CPU is responsible for the main control operation, calculation with shorter time-consuming or inconvenient parallel operation, and the final result output, etc. The GPU is responsible for iterative calculation, calculation of the shape function and its derivatives, etc, which they are the most time-consuming in structural topology optimization.

Meanwhile, the pretreatment in Fig.4 mainly include the following contents and procedure.

1) Set up integral points.

2) Calculate influence field radius of node and definition field radius of integral point.

3) Establish the list of integral points in influence field of nodes, and nodes in definition field of integral points.

4) Determine interacting nodes pairs.

5) Calculate global force vector and global penalty force vector.

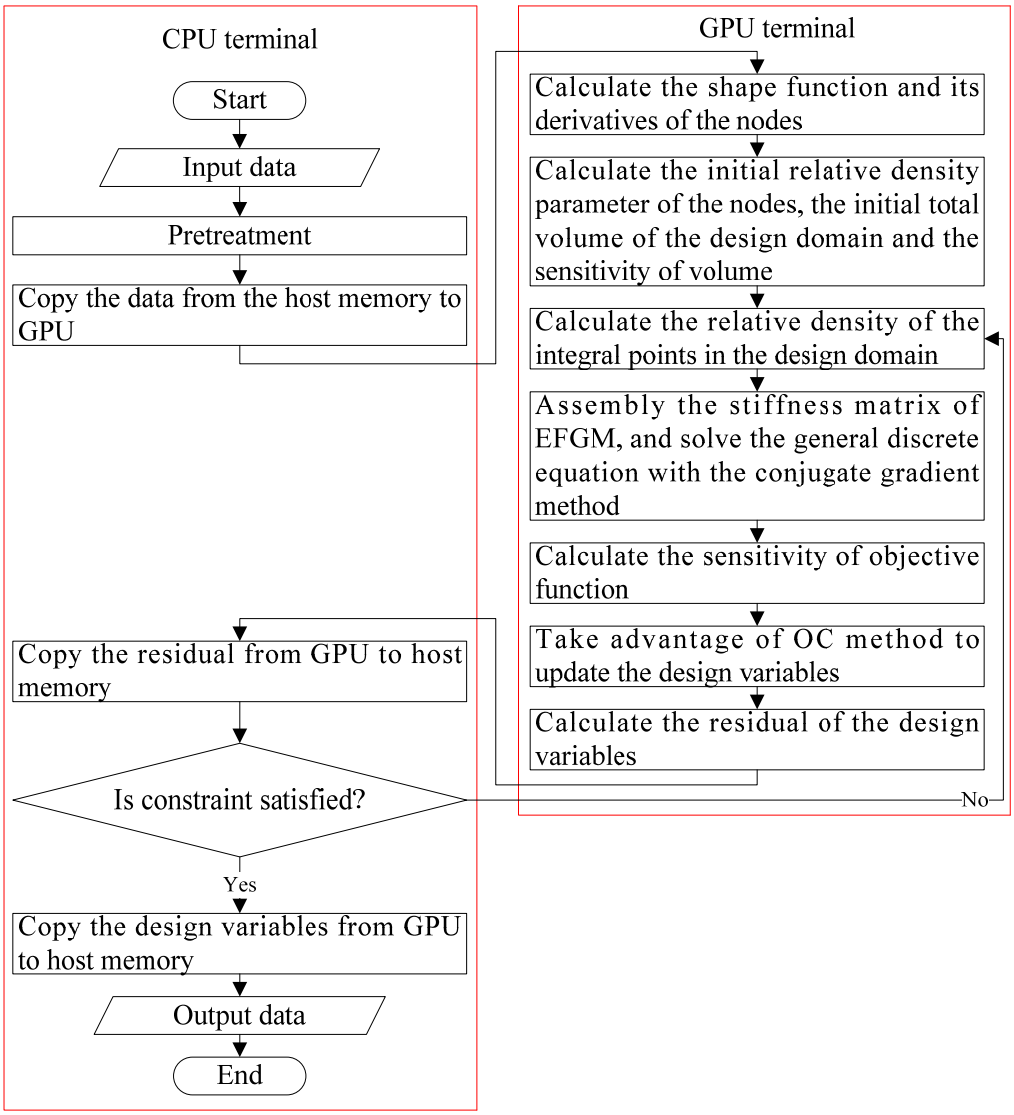

Fig. 4 Flowchart of GPU parallel algorithm for structural topology optimization

4 Numerical examples

The numerical examples are used to verify the feasibility of presented method in this paper, and are run on the following hardware and software, which CPU is Intel Core i5-3330 that has four physical cores at 3.0GHz, and GPU is the GeForce GTX 660 with 2GB GDDR5 memory, the version of CUDA is 3.0, and the operating system is 64 bit WIN 7 system. On the other hand, the same parameters are used in the following examples, that is, Penalty factor is $p = 3.0$, volume constraint is 30%, numerical damping coefficient is $\eta = 0.5$ and move-limit is $m = 0.2$. The termination condition of optimal iterative is $\left|\rho_I^{(k+1)} - \rho_I^{(k)}\right| \le 0.1$. The influence domain radius of node $\boldsymbol{x}_I$ is 2.5 times the size of the minimum distance between node $\boldsymbol{x}_I$ and other nodes, and the 2×2 Gauss points is assigned in each integration cell.

4.1 Example I- cantilever beam

The cantilever beam is a classical topology optimization problem, and its model is shown in Fig. 5(a). The cantilever assumed to be in a state of plane stress has characteristic height:

$H = 20\text{m}$ and width: $L = 40\text{m}$ and is considered to be of unit depth. It is fixed the left side and loaded with a point force $t = 1\text{N}$ at the lower of the right side. The design domain is discretized by 17×32 nodes uniformly distributed, as shown in Fig. 5(b), and is assigned by 496 integration cells. The elastic property of material is: Young's modulus $E = 1\text{Pa}$, Poisson' ratio $\nu = 0.3$.

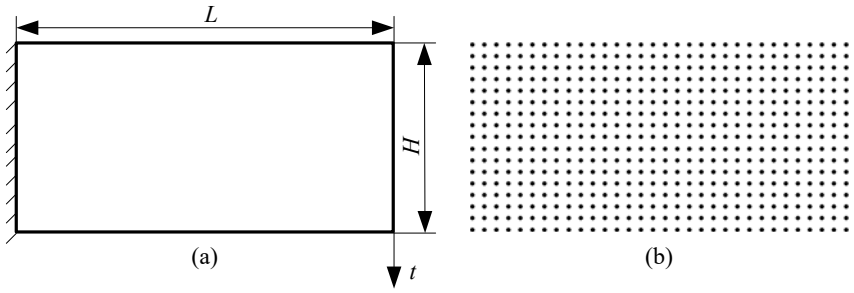

Fig. 5 Cantilever beam (a) geometry model, (b) discrete nodes

After 22 times of iteration, the program can reach the convergence precision. The optimization results obtained by using the GPU parallel algorithm and traditional algorithm are shown in Fig. 6, respectively. The figure 6 shows that both of results are good agreement. It means that the presented GPU parallel algorithm is feasibility and effective.

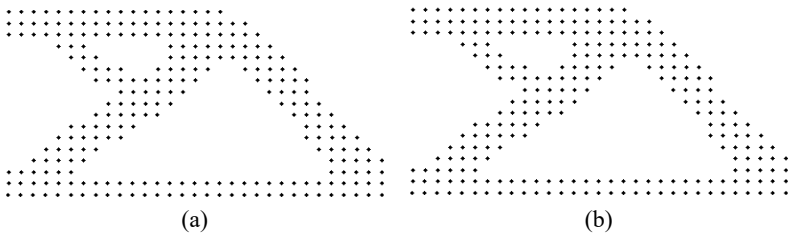

Fig. 6 Optimization results (a) traditional algorithm, (b) GPU parallel algorithm

When other parameters remain unchanged, the optimal results under different number of nodes can also be obtained by using GPU parallel algorithm, as shown in Fig.7.

In this paper, the speedup is defined as following

$$s_p = \frac{t_{\text{CPU}}}{t_{\text{GPU}}} \tag{10}$$

where t_{CPU}, t_{GPU} denotes for the run time of the traditional computation and parallel computation, respectively. The relation between speedup and nodes is shown in Fig.8.

In the case of different nodal spacing, the different topology results can be found in Fig.7, but their topological layout is similar. Meanwhile, it also implies that the GPU parallel algorithm presented is feasible to find the tiny structure in topological optimization.

The figure 8 shows that the speedup will increase with increasing of the number of nodes, and the maximum speedup is reached about 24 times. It is because GPU has a powerful floating-point computing power and is very suitable for computing of large scale problems. Meanwhile, it also means that the GPU parallel algorithm presented in this paper can effectively improve the computing efficiency of topology optimization.

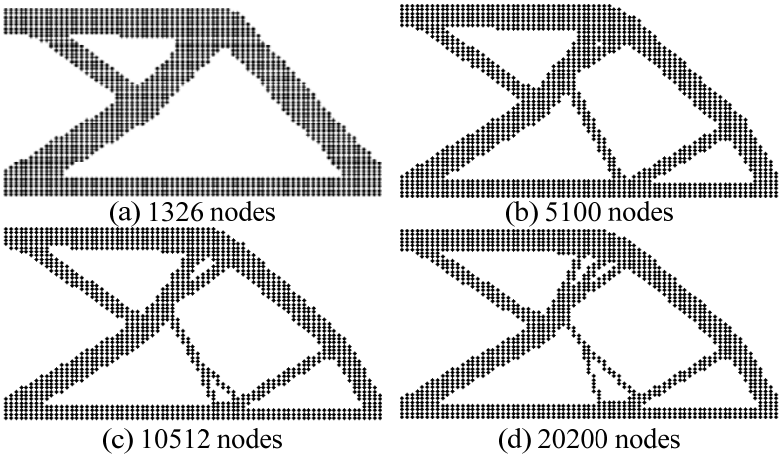

Fig. 7 Optimization results of cantilever beam under different number of nodes

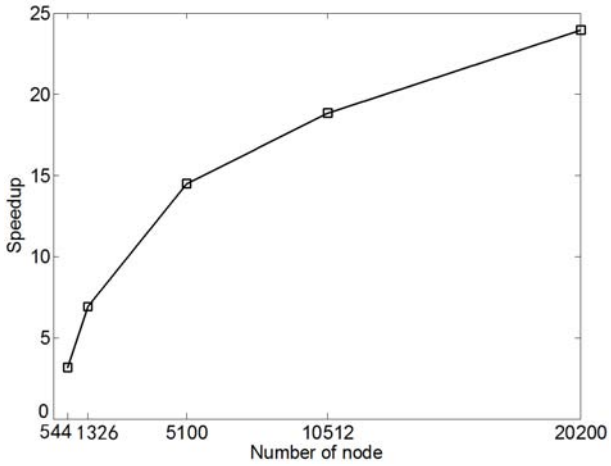

Fig.8 Relation between speedup and node

4.2 Example II-Deep beam with opening hole

A deep beam model with opening hole is showed in Fig. 9(a). It is subjected to a point load $P = 3000\text{kN}$, and Young's modulus is $E = 20820$ MPa; Poisson's ratio is $\nu = 0.15$. On the other hand, the initial width of this concrete beam is $b = 400\text{mm}$. The design domain is discretized by 545 nodes as shown in Fig. 9(b).

After iterating 20 times, the program reach the convergence precision. The final optimization results are shown in Fig. 10. The relation between speedup and number of nodes can be also obtained, as shown in Fig.11, and Fig.12 shows the optimal results under different number of nodes.

Figure 10 shows that the result obtained by using GPU parallel algorithm is good agreement with one gained by traditional algorithm. When the number of nodes is different, the basic topology layout is similar, but tiny structure in topological results can be found in Fig.12. And with the increasing of number of nodes, the speedup will also increase, as shown in Fig.11. The maximum speedup will reach 22.3 times.

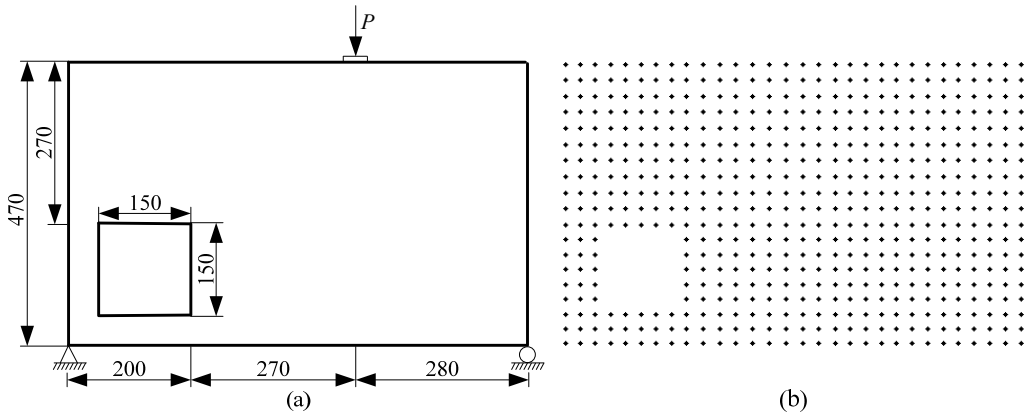

Fig. 9 Deep beam with opening hole (a) geometry model, (b) distribution of nodes

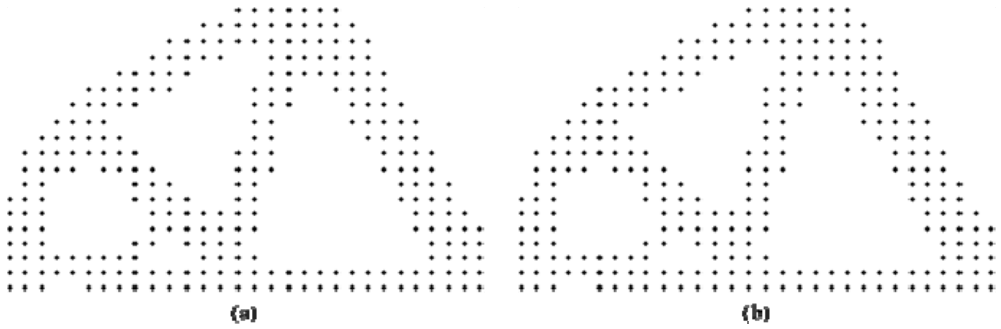

Fig. 10 Optimization result (a) traditional algorithm, (b) GPU parallel algorithm

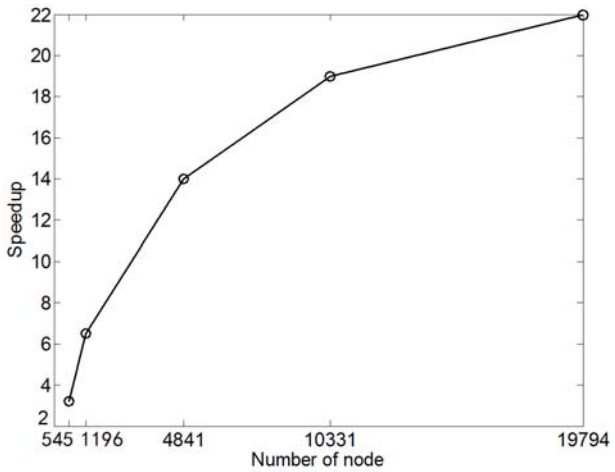

Fig.11 Speedup for beep beam model

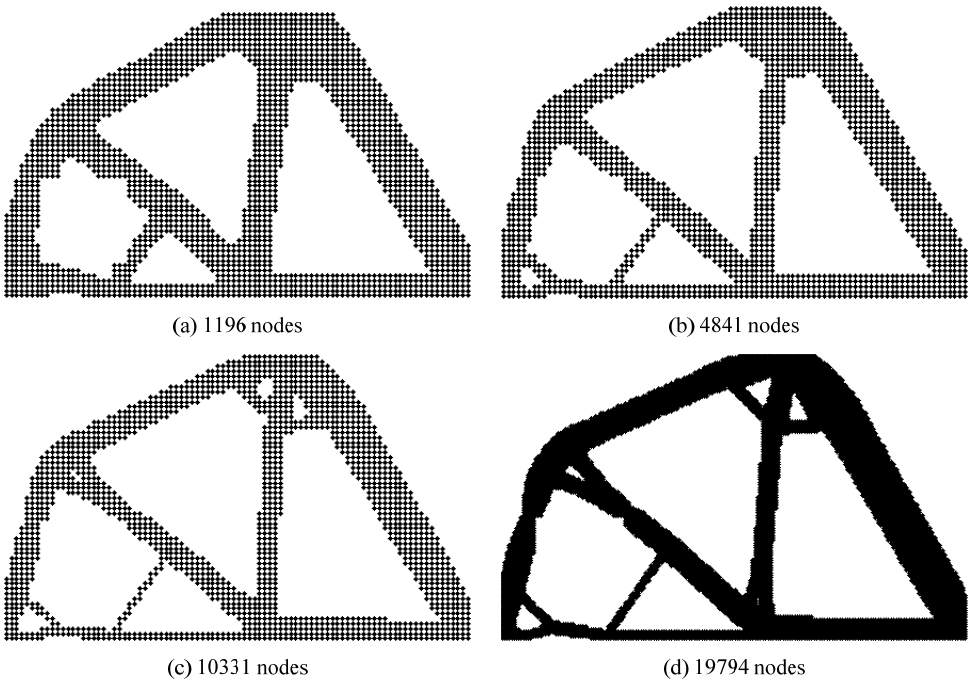

Fig. 12 Optimization results of beep beam under different number of nodes

Comparing Fig.12 and Fig.8, we can find that both of them are basically the same, and the size of speedup is related to the number of node.

5 Conclusions

In this paper, we present a new implementation of topology optimization based on GPU parallel acceleration by using the EFG method to carry out the numerical analysis. Its purpose is to reduce the computational cost of topology optimization. The parallel algorithm of the sensitivity analysis and OC method is discussed by using node-by-node method and interacting nodes pairs, and the corresponding flowchart and thread handling is given, respectively. Meanwhile, to improve the whole performance of GPU program and reduce the number of intermediate variables, the computational equation of sensitivity analysis for objection function is derived, so that the intermediate variables can be stored in the register of GPU. The presented method is verified by two numerical examples, and topological layouts obtained by using GPU parallel acceleration algorithm are good agreement with that using traditional algorithm. By studying on the speedup, it can be seen that the speedup will increase with the increase of number of node, and the maximum speedup may reach about 24 times in our given example. It means that the presented method in this paper is feasible and valid, and it can observably save the computation time of topology optimization.

Acknowledgment

This research is funded by National Natural Science Foundation of China [51375417, 51405415, 51475403]. The financial support to the first author is gratefully acknowledged.

References

[1] Zargham, S., Ward, T. A., Ramli, R. and Badruddin, I. A. (2016) Topology optimization: a review for structural designs under vibration problems, *Structural & Multidisciplinary Optimization*, **53**(6), 1157-1177.

[2] Chen, X. (2016) Topology optimization of microfluidics - A review, *Microchemical Journal*, **127**, 52-61.

[3] Sigmund, O. (1998) Numerical instabilities in topology optimization A survey on procedures dealing with checkboards, mesh-dependencies and local minima, *Structural Optimization*, **16**, 68-75.

[4] Lu, Y. Y., Belytschko, T. and Gu, L. (1994) A new implementation of the element free Galerkin method, *Computer Methods in Applied Mechanics & Engineering*, **113**(3-4), 397-414.

[5] Gong S. G., Chen M., Zhang J. P. and He R. (2012) Study on modal topology optimization method of continuum structure based on EFG method, *International Journal of Computational Methods*, **9**(1):1240005(12pags).

[6] Yang, X., Zheng, J. and Long, S. (2016) Topology optimization of continuum structures with displacement constraints based on meshless method, *International Journal of Mechanics & Materials in Design*, 1-10.

[7] Belytschko, T., Krongauz, Y., Organ, D., Fleming M. and Krysl P. (1996) Meshless methods: an overview and recent developments, *Computer methods in applied mechanics and engineering*, **139**(1), 3-47.

[8] Singh, I. V. (2004) Parallel implementation of the EFG method for heat transfer and fluid flow problems, *Computational Mechanics*, **34**(6), 453-463.

[9] Zeng, Y. S., Zeng, Q. H. and Lu, D. T. (2008) Parallel computing of element-free Galerkin method for elasto-dynamics, *Chinese Journal of Computational Mechanic*s, **25**(3), 385-391.

[10] Karatarakis, A., Metsis, P. and Papadrakakis, M. (2013) GPU-Acceleration of stiffness matrix calculation and efficient initialization of EFG meshless methods, *Computer Methods in Applied Mechanics and Engineering*, **258**, 63-80.

[11] Bolz, J., Farmer, I., Grinspun, E. and Der P. (2003) Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Transactions on Graphics*, **22**(3), 917-924.

[12] Gong, S. G., Liu, Q. L., Lu, H. S., Zhou Z. Y. and Zhang J. (2015) Parallel computing and application of Element-free Galerkin method for GPU acceleration, *Chinese Journal of Computational mechanics*, **32**(6), 745-751.

[13] Stephan, S. and Volker, S. (2011) A 2589 line topology optimization code written for the graphics card, *Computing and Visualization in Science*, **14**, 249-256.

[14] Challis, V. J., Roberts, A. P. and Grotowski, J. F. (2014) High resolution topology optimization using graphics processing units(GPUs), *Structural and Multidiscipline Optimization*, **49**(2), 315-325.

[15] Martínez-Frutos, J. and Herrero-Pérez, D. (2016) Large-scale robust topology optimization using multi-GPU systems, *Computer Methods in Applied Mechanics & Engineering*, **311**, 393-414.

[16] CAI, Y. and Li S. (2016) Graphics processor unit parallel computing in Matlab and its application in topology optimization, *Journal of Computer Applications*, **36**(3), 628-632.

[17] Sigmund, O. (2001) A 99 line topology optimization code written in Matlab, *Structural and Multidiscipline Optimization*, **21**(2), 120-127.

[18] Gong, S. G., Liu, X., Xie, G. L. and Zhang J. P. (2009) Topology optimization under multi-load cases based on element-free Galerkin method, *Journal of Mechanical Engineering*, **45**(12), 137-142.