Struct Multidisc Optim (2017) 56:413–434
DOI 10.1007/s00158-017-1672-x

RESEARCH PAPER

GPU parallel strategy for parameterized LSM-based topology optimization using isogeometric analysis

Zhaohui Xia[1] · **Yingjun Wang**[2] · **Qifu Wang**[3] · **Chao Mei**[3]

Received: 13 July 2016 / Revised: 6 February 2017 / Accepted: 14 February 2017 / Published online: 14 March 2017

Abstract This paper proposes a new level set-based topology optimization (TO) method using a parallel strategy of Graphics Processing Units (GPUs) and the isogeometric analysis (IGA). The strategy consists of parallel implementations for initial design domain, IGA, sensitivity analysis and design variable update, and the key issues in the parallel implementation, e.g., the parallel assembly race condition, are discussed in detail. The computational complexity and parallelization of the different steps in the TO are also analyzed in this paper. To better demonstrate the advantages of the proposed strategy, we compare efficiency of serial CPU, multi-thread parallel CPU and GPU by benchmark examples, and the speedups achieve two orders of magnitude.

Keywords Isogeometric analysis · Topology optimization · Level set method · CUDA · GPU parallel computing

✉ Yingjun Wang
wangyj84@scut.edu.cn

[1] Center for Modeling, Simulation and Imaging in Medicine (CeMSIM), Rensselaer Polytechnic Institute, Troy, NY 12180, USA

[2] School of Mechanical and Automotive Engineering, South China University of Technology, Guangzhou, Guangdong 510641, China

[3] National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China

1 Introduction

Topology optimization (TO), a computational technique to optimally distribute material within a prescribed design domain, is becoming one of the most efficient tools guiding designers and engineers in the early design stages. During the past three decades, TO has been used principally for traditional mechanics but its applications have been spreading to a wide range of other engineering domains (Bendsøe and Sigmund 2003; van Dijk et al. 2013). Several methods have been used in TO, such as SIMP (Rozvany 2001), ESO (Huang and Xie 2010), level set method (LSM) (Osher and Sethian 1988). Unlike other methods using explicit material representation, it is more convenient for the LSM-based TO method to model geometric constraints by adopting an implicit description of boundaries (Wang et al. 2003), although it has some drawbacks, e.g., depending on starting guess (Sigmund and Maute 2013). Some scholars have contributed on LSM-based TO and obtained some achievements (Allaire et al. 2013; Guo et al. 2014; Herrero et al. 2013; Luo et al. 2007; Otomori et al. 2012; Wang et al. 2003, 2016; Wang and Benson 2016; Wei et al. 2010; Xia et al. 2014a, b, c; Yamada et al. 2010). However, in most conventional level set-based methods using the finite difference method, the time step size has to be restrained to satisfy the Courant-Friedrichs-Lewy (CFL) condition so as to ensure the numerical stability of the time-marching process (Allaire et al. 2004; Luo et al. 2007, 2008; Wang et al. 2003). Furthermore, it usually requires the reinitialization of the level set function (LSF) when it becomes too flat or steep (Yamada et al. 2010), which decreases the efficiency of the TO. The parameterized LSMs are one family of those methods that can overcome these numerical shortcomings, which actually convert the original Hamilton-Jacobi PDE into a relatively simpler set of ordinary differential equations (ODEs) or an equivalent system of algebraic

equations, so that the Hamilton-Jacobi PDE does not have to be solved directly (Luo et al. 2008; Wang and Wang 2006a, b).

In practice, most of TO methods use the conventional finite element method (FEM) (Hughes 2012) for the structural analysis and sensitivity calculation. However, there are some drawbacks in the FEM, e.g., approximate model and low continuity (Wang and Benson 2015a, b). FEM is based on the interpolation points positioned inside the design domain, which are not suitable for the isogeometric interpolations since for the isogeometric method the control points are not necessarily located in the design domain (Wang and Benson 2015a, b). In the past few years, isogeometric analysis (IGA) (Cottrell et al. 2009; Hughes et al. 2005; Wang et al. 2015a, b), which uses the same basis functions for the geometric and computational models, has become one of the most efficient methods to replace conventional computational methods (e.g., the FEM and the boundary element method) in a variety of domains (Bazilevs et al. 2006; Hsu et al. 2011; Li and Qian 2011). IGA has been successfully applied in TO problems due to its high accuracy and continuity. Seo et al. (2010) first proposed an isogeometric TO by using trimmed spline surfaces. Dede et al. (2012) proposed a TO method with IGA in a phase field approach. Tavakkoli et al. (2013) presented a control point-based approach for structural TO. Wang and Benson (2015a, b) presented an accurate and efficient isogeometric TO method that integrated the non-uniform rational B-splines based IGA and the parameterized LSM for minimal compliance problems.

However, TO is a iterative computing process, which is computationally expensive and resource-consuming especially for a large-scale problem (Duarte et al. 2015). For solving more complicated and large-scale problems, parallel computing was used to the TO (Aage and Lazarov 2013). In recent years, especially after CUDA was released by NVIDIA (2007), graphical processing units (GPUs) have been successfully used in high-performance computations for complex scientific problems (Challis et al. 2014; Kuźnik et al. 2012; Xia et al. 2014a, b, c; Zegard and Paulino 2013). The GPU architectures are composed of hundreds, even thousands of cores specially designed for parallel computing. CUDA is used in this paper because of its better support by an NVIDIA GPU (Mukherjee et al. 2012), which is favorable in engineering applications (Georgescu et al. 2013; Ploskas and Samaras 2014; Wang et al. 2015a, b; Wei et al. 2015; Xia et al. 2015, 2016).

Recently, different strategies of GPU parallel TO have been proposed for large-scale problems. Wadbro and Berggren (2009) proposed an efficient method for large-scale TO problems with GPUs in heat conduction with over 4 million design variables. Schmidt and Schulz (2011) proposed a method of TO for structured meshes with a computationally intensive task on GPUs, which is faster than a 48-core shared memory CPU system. Suresh (2013) introduced an efficient algorithm and implementation for large-scale 3D TO problems, which can solve a 700 thousand DOFs problem in 125 s in a GPU. In addition, Herrero et al. (2013) proposed an implementation of the LSM-based TO in massively parallel computer architectures using GPU, which showed a predominance in optimizing large scale problems. Challis et al. (2014) presented a high resolution for LSM-based TO with a GPU implementation. A speedup reached approximately 13 times for more than 4 million design variables. Wu et al. (2016) proposed a scalable system equipped with a high-performance GPU solver for generating 3D objects using topology optimization, which can efficiently handle models comprising several millions of elements. However, their parallel strategies are not illustrated in detail.

Concurrently, there have been large efforts on IGA with GPU parallel computing. Kuźnik et al. (2012) developed a multi-thread multi-frontal parallel direct solver for two dimensional isogeometric FEM with 1 million DOFs under 448-core GPU. Woźniak et al. (2014) presented computational cost estimates for parallel shared memory isogeometric multi-frontal solver. Karatarakis et al. (2014) proposed the formulation of the stiffness matrix exhibiting several computational merits with GPU parallel computing, which can accelerate the computing by orders of magnitude. Woźniak (2015) analyzed the integration algorithm with higher order B-spline basis functions and provided sets of tasks that can be automatically scheduled and executed concurrently on GPU, and they can solve the 2D problems with one million DOFs.

In this paper, the parallelization of the TO problem with parameterized LSM and IGA is studied, and the computational procedure of each step of the optimization approach is discussed. The outline of the remainder of this paper is structured as follows: Sect. 2 briefly recapitulates NURBS-based IGA and level-set based TO and GPU parallel architecture. In Sect. 3, the parallel strategy for initial design domain with LSM is analyzed and presented. The parallel strategy using CUDA for IGA is discussed in Sect. 4. The implementing of the parallel algorithm for sensitivity analysis and update scheme of design variables is illustrated in Sect. 5. Several examples demonstrate the advantages of the proposed GPU parallel approach in Sect. 6. Finally, a brief summary is given in Sect. 7.

2 Basic theory

2.1 NURBS basic theory

In IGA, non-uniform rational B-splines (NURBS) (Piegl and Tiller 2012), constructed from B-splines, are commonly used

for the numerical discretization. A knot vector $\boldsymbol{\Pi}$, which consists of n spline basis functions, is a sequence of non-decreasing real numbers representing parametric coordinates of a curve:

$$\boldsymbol{\Pi} = \left\{\eta_1, \eta_2, \cdots, \eta_{n+p+1}\right\}, \tag{1}$$

where p is the order of the B-spline. The interval $[\eta_1, \eta_{n+p+1}]$ is called a patch, andthe knot interval $[\eta_i, \eta_{i+1})$ is called a span. Given a knot vector, the B-spline basis functions are recursively defined following the Cox-de Boor formula (De Boor 1972):for zero order ($p=0$),

$$B_{i,p}(\eta) = \begin{cases} 1 & if \ \ \eta_i \le \eta < \eta_{i+1}, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

and for non-zero order ($p>0$)

$$B_{i,p}(\eta) = \frac{\eta - \eta_i}{\eta_{i+p} - \eta_i} B_{i,p-1}(\eta) + \frac{\eta_{i+p+1} - \eta}{\eta_{i+p+1} - \eta_{i+1}} B_{i+1,p-1}(\eta). \tag{3}$$

Based on the tensor product formalism, two dimensional B-spline basis functions are constructed as

$$B_{i,p}^{j,q}(\eta, \zeta) = B_{i,p}(\eta) B_{j,q}(\zeta), \tag{4}$$

where $B_{i,p}(\eta)$ and $B_{j,q}(\zeta)$ are univariate B-spline basis functions of order p and q, corresponding to knot vectors $\boldsymbol{\Pi} = \{\eta_1, \eta_2, \ldots, \eta_{n+p+1}\}$ and $\boldsymbol{H} = \{\zeta_1, \zeta_2, \ldots, \zeta_{m+q+1}\}$. NURBS basis functions are obtained from B-splines by assigning a positive weight w_i to each basis function

$$N_{i,p}(\eta) = \frac{B_{i,p}(\eta) w_i}{\sum_{j=1}^{n} B_{j,p}(\eta) w_j}. \tag{5}$$

By the tensor product formulation, two dimensional NURBS basis functions are constructed as

$$N_{i,p}^{j,q}(\eta, \zeta) = \frac{B_{i,p}(\eta) B_{j,q}(\zeta) w_{i,j}}{\sum_{k=1}^{n} \sum_{t=1}^{m} B_{k,p}(\eta) B_{t,q}(\zeta) w_{k,t}}, \tag{6}$$

where $w_{i,j}$ is the weight value corresponding to the tensor product $B_{i,p}(\eta) B_{j,q}(\zeta)$.

2.2 LSM-based topology optimization

In the LSM, the structural boundary $\partial\Omega$ is implicitly embedded as the zero level set of a one-dimensional-higher level set function (LSF) $\Phi(\boldsymbol{x}, t)$ which is Lipschitz continuous (Osher and Fedkiw 2001), where t is a pseudo time. The LSF $\Phi(\boldsymbol{x}, t)$ is defined over a reference domain $D \subset R^d$ ($d=2$ or 3), and a two-dimensional model expressed with level set is shown in Fig. 1. The mathematical representation of the structure with level set can be defined as (Allaire et al. 2004; Luo et al. 2008)

$$\begin{cases} \Phi(\boldsymbol{x}, t) > 0 \ \Leftrightarrow \ \boldsymbol{x} \in \Omega \backslash \partial\Omega & \text{(material)}, \\ \Phi(\boldsymbol{x}, t) = 0 \ \Leftrightarrow \ \boldsymbol{x} \in \Gamma & \text{(interface)}, \\ \Phi(\boldsymbol{x}, t) < 0 \ \Leftrightarrow \ \boldsymbol{x} \in D \backslash \Omega & \text{(void)}. \end{cases} \tag{7}$$

Differentiating the LSF $\Phi(\boldsymbol{x}, t)$ with respect to the pseudo-time t, the Hamilton-Jacobi equation is obtained as (Luo et al. 2009):

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} - v_n |\nabla \ \Phi(\boldsymbol{x}, t)| = 0, \quad \Phi(\boldsymbol{x}, 0) = \Phi_0(\boldsymbol{x}), \tag{8}$$

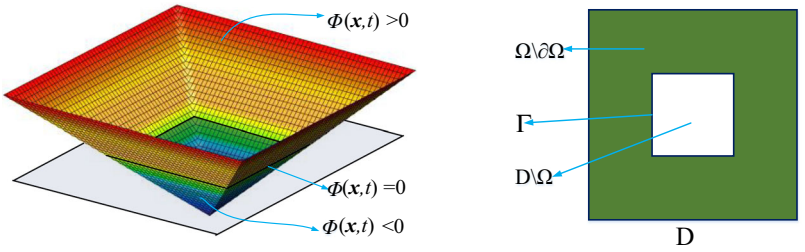
where the normal velocity $v_n = (\partial \boldsymbol{x}/\partial t) \cdot (\nabla \Phi / |\nabla \Phi|)$, and $\Phi_0(\boldsymbol{x})$ is the initial LSF. The Hamilton-Jacobi PDE is solved to move the boundary along the normal direction.

2.3 NURBS parameterized LSM-based TO

In the conventional level-set schemes, a level-set model mathematically described as the Hamilton-Jacobi PDE (Allaire et al. 2014). However, solving the Hamilton-Jacobi is difficult and time-consuming. To solve this problem, parameterized LSMs are used to convert the Hamilton-Jacobi problems into ordinary differential equations (Luo et al. 2007). There are several different interpolation functions that can be adopted for this parameterization, e.g., the linear B-spline basis functions used by Chen et al. (Chen et al. 2007, 2008), globally supported radial basis functions (RBFs) used by Wang et al. (Wang and Wang 2006a, b) and compactly supported radial basis functions (CS-RBFs) proposed by Luo et al. (Luo et al. 2007, 2008). These methods are all based on the interpolation points on the design domain, which is not suitable for the isogeometric interpolations since the control points are not necessarily in the design domain (Wang and Benson 2015a, b). In the NURBS-based parameterized LSM, the original PDE can be transformed into a set of ODEs that are simpler to solve numerically (Wang and Benson 2015a, b). The core idea of this method is to use the NURBS basis functions (not interpolation) to represent the LSF in a parameterized mode as

$$\Phi(\boldsymbol{x}, t) = \boldsymbol{N}(\boldsymbol{x})^T \boldsymbol{\varphi}(t) = \sum_i N_i(\boldsymbol{x}) \varphi_i(t), \tag{9}$$

where $\varphi_i(t)$ is the ith expansion coefficient associating with the ith grid point. $N_i(\boldsymbol{x})$ is the corresponding basis function of NURBS. After this parameterization, the LSF associated with both space and time is divided into the spatial terms $N_i(\boldsymbol{x})$ and the time dependent terms $\varphi_i(t)$, and only the latter are updated during the



(a) Level-set function and zero level-set. (b) Design domain and boundary.

Fig. 1 2D design domain and level-set model. **a** Level-set function and zero level-set. **b** Design domain and boundary

optimization procedure. Substituting (9) to (8), the Hamilton-Jacobi PDE is rewritten as

$$\boldsymbol{N}(\boldsymbol{x})^T \frac{\partial \boldsymbol{\varphi}(t)}{\partial t} - v_n \left| (\nabla \boldsymbol{N}(\boldsymbol{x}))^T \boldsymbol{\varphi}(t) \right| = 0, \tag{10}$$

where the v_n related to the time derivative of the expansion coefficients is

$$v_n = \frac{\boldsymbol{N}(\boldsymbol{x})^T}{\left| (\nabla \boldsymbol{N}(\boldsymbol{x}))^T \boldsymbol{\varphi}(t) \right|} \frac{\partial \boldsymbol{\varphi}(t)}{\partial t}. \tag{11}$$

The parametric coordinates η and ζ are used, thus the control points can be adopted to interpolate the LSF of the design domain, and (9) may be written as

$$\Phi(\boldsymbol{x}, t) = \Phi(\boldsymbol{x}(\eta, \zeta), t) = \sum_i N_i(\eta, \zeta) \varphi_i(t). \tag{12}$$

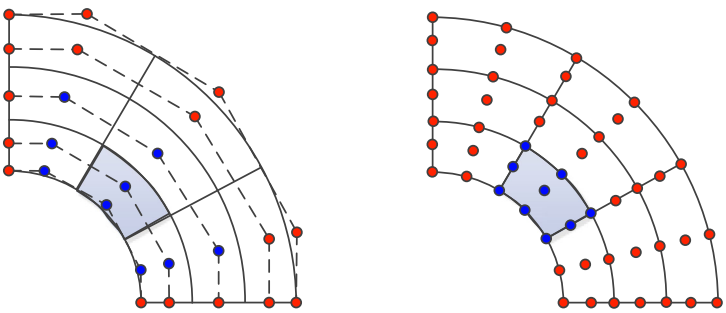
where $N_i(\eta, \zeta)$ is the basis function of the ith control point influencing on (η, ζ). Compared with conventional FEM, the NURBS basis functions of IGA are referred to control points instead of nodes, which are not interpolatory like the Lagrangian FEM basis functions (Luo et al. 2007, 2008). The spatial discretization of the different schemes with quadratic elements is shown Fig. 2, which shows that the number of DOFs is much smaller in IGA, and the control points influencing an element are not necessarily in the element domain and may be out of the problem domain.

The procedure of the NURBS-based parameterized TO for minimum compliance problem can be described as Fig. 3. The boxes inside the dash lines represent the main steps in the loop, the optimal topology is obtained when the change of objective functions between two iterations is lower than a specified tolerance.

2.4 GPU parallel architecture

GPUs were originally used for graphic processing but are increasingly applied to scientific computations due to their outstanding computational power, especially since NVIDIA released CUDA in 2007 (Kirk 2007). 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 are always run together on a processor (core) as shown in Fig. 4 (SFU represents Super Function Unit). The warp is the unit of thread scheduling in streaming multiprocessors (SMs), which includes a number of hardware streaming processors (SPs). Figure 3 shows the division of blocks into warps in an implementation. A SM is designed to execute all threads in a warp following the single instruction, multiple data (SIMD) model. Note that these threads will apply the same instruction to different portions of the data. As a result, all threads in a warp will always have the same execution time.

Several memories can be used by programmers to achieve high efficiency in GPU (Kirk and Wen-mei 2012). Global



(a) Quadratic NURBS elements of IGA. (b) Quadratic Lagrange elements of FEM.

Fig. 2 Spatial discretization of IGA and FEM for a quarter annulus model. **a** Quadratic NURBS elements of IGA. **b** Quadratic Lagrange elements of FEM

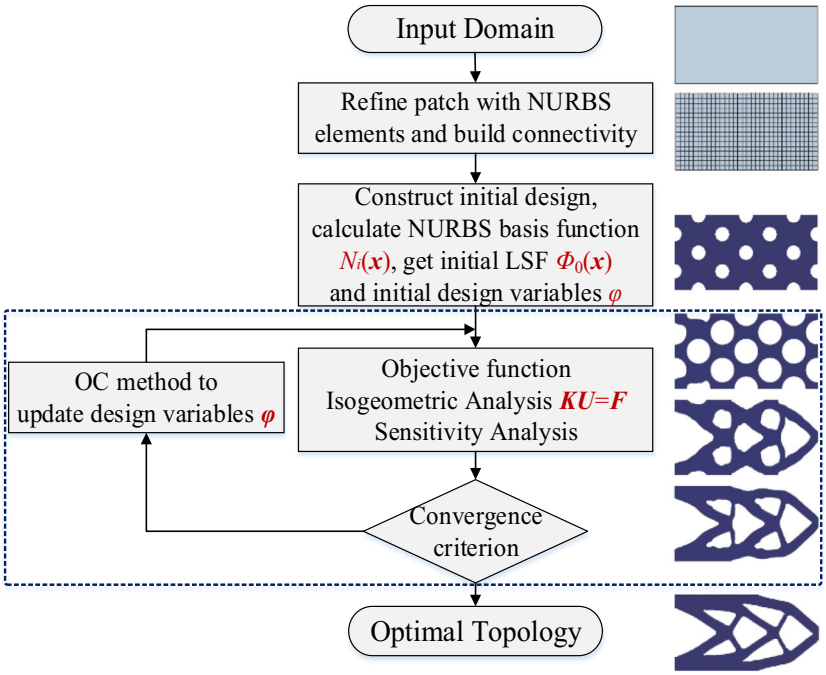


Fig. 3 TO procedure for the compliance minimization problem

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).

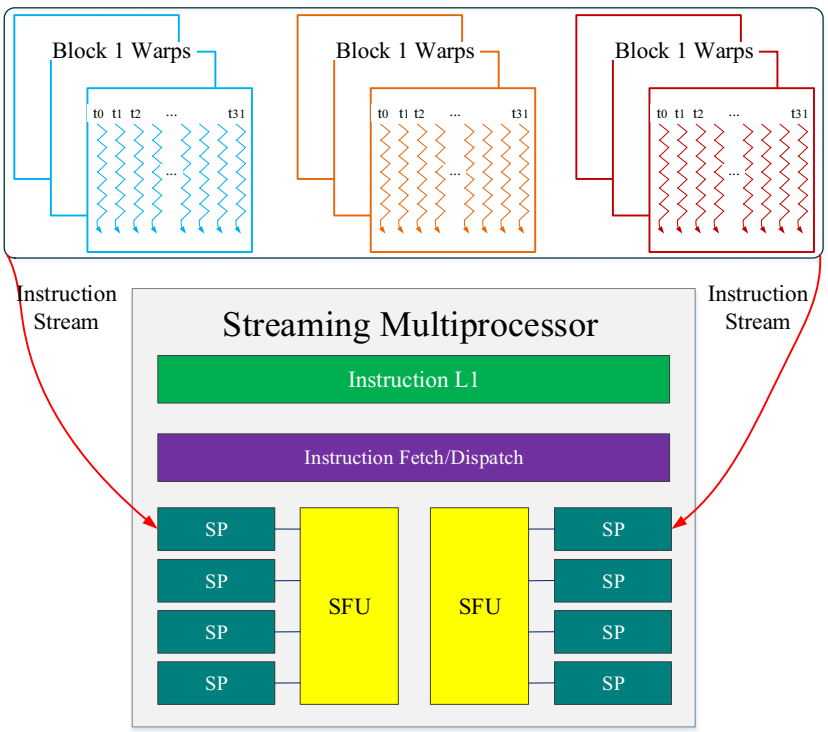


Fig. 4 Blocks are partitioned into warps for thread scheduling

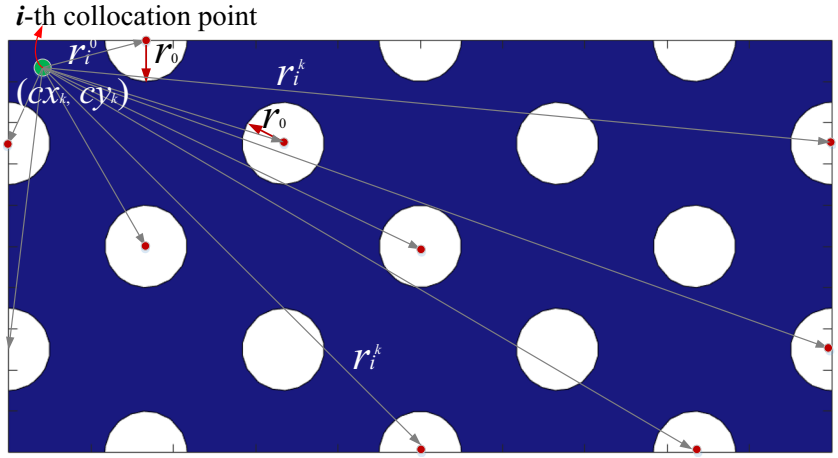


Fig. 5 Initial LSF value of a collocation point

3 Parallel strategy for initial design domain

3.1 Initial design domain

Through Sect. 2.3, the initial LSF as $\Phi(\boldsymbol{x}, 0)$ is known when the geometry of the initial design domain is known, but the initial expansion coefficients of control points as $\varphi_i(0)$ are unknown. If the number of control points is n, n collocation points need to be sampled in the initial design domain to set up the equations as (9). In this work, the Greville abscissae is used to sample the collocation points (Johnson 2005). For a collocation point on a NURBS surface, there are two Greville abscissae representing the coordinates in η and ζ coordinates, respectively. Applying (9) at each collocation point yields equations that may be assembled into a linear equation system

$$\boldsymbol{\Phi} = \boldsymbol{A}\boldsymbol{\varphi}, \tag{13}$$

where $\boldsymbol{\Phi}$ is a vector consisting of initial LSF values at all collocation points, $\boldsymbol{A}$ is a matrix consisting of the NURBS basis functions values corresponding to the collocation points, and $\boldsymbol{\varphi}$ is a vector of expansion coefficients at control points. When $\boldsymbol{\varphi}$ is updated, the new LSF values are evaluated, while the matrix $\boldsymbol{A}$ will not change. To construct the initial design, holes are introduced into the design domain that are used to describe the shape of the optimization results. Different number of initial holes would not influence the final topology significantly (Luo et al. 2009). The initial LSF values can be obtained from

$$\begin{aligned} r_i^k &= \sqrt{(x_i - cx_k)^2 + (y_i - cy_k)^2} - r_0, \quad k = 1, 2, \cdots, m, \\ \Phi(\boldsymbol{x}, t)_i &= \min\left(r_i^1, r_i^2, \cdots, r_i^k\right), \quad i = 1, 2, \cdots, n, \end{aligned} \tag{14}$$

where x_i and y_i are the physical coordinates of the ith collocation point, r_0 is the radius of the initial holes, r_i^k is the distance between the ith collocation point and the kth initial hole. The initial LSF value of a collocation point is the minimum value of r_i^k. cx_k and cy_k are the physical coordinates of the kth initial hole as shown in Fig. 5.

In the NURBS-based LSM, the vertices of the NURBS spans are used (i.e., elements in IGA) as interpolation points,

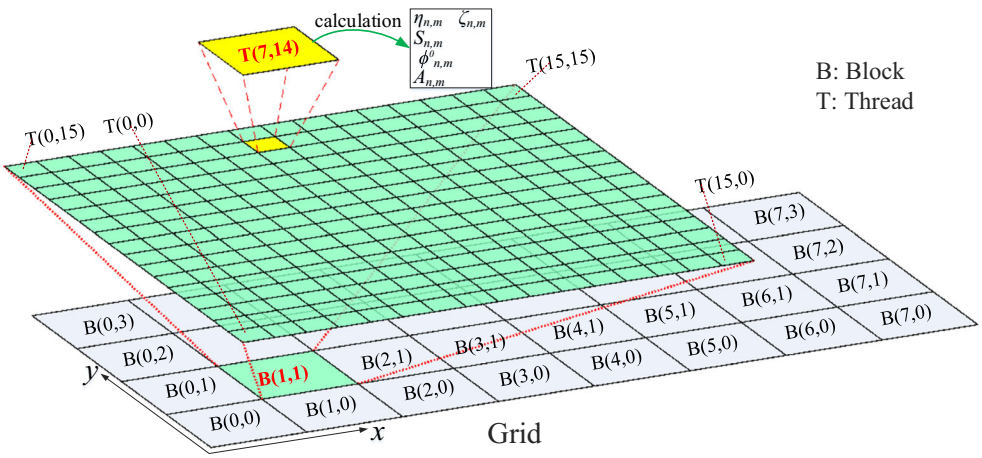


Fig. 6 CUDA parallel strategy for initial design domain

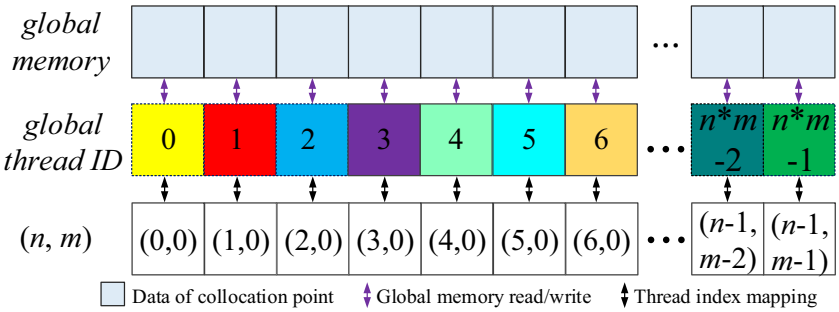


Fig. 7 Stretch of thread distribution

which is used to track the zero-level set boundaries and perform the visualization. The physical coordinates of the points are available by the point inversion (projection) algorithm (Piegl and Tiller 2012), so meshing is not needed. The LSF values of the interpolation points may be obtained by (13) as matrix $\boldsymbol{B}$.

3.2 CUDA parallel strategy for initial design domain of LSM

3.2.1 Hierarchy of CUDA thread and memory

Through analyzing the initial design domain procedure, its computations are amenable to parallelization. A CUDA kernel is executed in a grid of thread blocks indexed by a 2D block identification (id) in the form (row, column). Herein, "One-thread-one-collocation point" mode of parallelization with CUDA is adopted, and 2D grids and blocks (Kirk and Wen-mei 2012) are chosen since the TO problem is 2D. Each thread can be properly used to deal with a collocation point. The number of threads n_t in a block is usually set as times of 32, which is the size of a warp (Kirk and Wen-mei 2012), and n_t can be adjusted to obtain the optimal performance for different GPU devices. Figure 6 shows such an arrangement for parallelizing the initial design domain with $nx*ny$ ($nx=128$, $ny=64$) collocation points, and n_t is set to be $16*16=256$ according to "latency tolerance" mechanism (Kirk and Wen-mei 2012). Then the number of blocks in a grid n_b can be chosen not less than $nx*ny/n_t$.

Each thread is assigned to charge the computation of parametric coordinate η and ζ, physical coordinates $\boldsymbol{S}$, NURBS basis functions and the initial LSF values Φ_0 of one collocation point (m, n). CUDA uses grid and blocks to manage these threads, and each thread has a unique global index, and the mapping of the CUDA threads to all the computational stencils is shown in Fig. 7. To access a piece of data stored in the GPU global memory from a single thread in a block, the global index of the thread needs to be computed as $T_{n,m}=n+m*ny$.

Figure 8 presents a simple layout of the CUDA thread and memory hierarchy in this parallel strategy. The data stored in

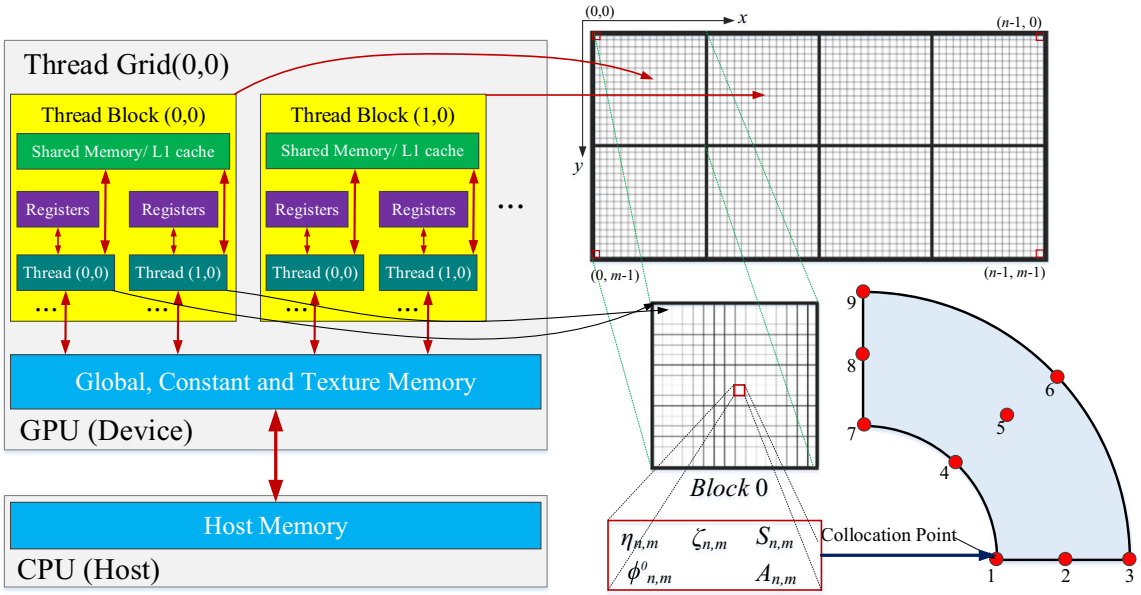


Fig. 8 The model of the CUDA memory

the host memory is firstly copied to the global memory of the device, and proper use of shared memory will greatly benefit the program performance especially when there are a lot of data reuse (Kirk and Wen-mei 2012). Constant memory is used to store important input parameters herein such as the order of the NURBS (p and q), the number of collocation points (nx and ny), the domain size and the minimum radius of the holes (r_0), etc., which can reduce the required memory bandwidth (Kirk and Wen-mei 2012). The data copied into the constant memory will not be changed during kernel execution, which make CUDA directs the hardware to aggressively cache the constant memory variables. Thus they do not need to be passed to the kernel as parameters.

3.2.2 Parallel strategy for initial design domain

A simplified parallel algorithm for initial design domain is presented in Table 1. The ← sign shows the variable assignment in the local memory and the double line arrow ⇒/⇐ represents the device memory read/write access. The variables with a star superscript are stored in the block shared memory. p and q are the degrees of NURBS. The *SurfacePoint*() function uses the global thread index ij to find the collocation point index, which enables the proper calculations in the GPU. The *Nurbs2DBasis*() function calculates the NURBS basis functions related to each collocation point. ***mx*** and ***ix*** arrays store the maximum value and index value corresponding to the objective function which will be used in sensitivity analysis. The initial level set values $\boldsymbol{\Phi}$ can be obtained from the minimum distance between the holes and collocation points. r_0 is the minimum radius of the initial holes.

Table 1 Parallel algorithm for initial design domain

Segment 1 – **Kernel** 1, Calculate Greville abscissae and physical coordinates of collocation points

m, n: number of colocations points	en: element number, ed: DOFs of en
cp: control point numbers, ***ed***: element ID	***u***, ***v***: knot vectors, ***w***: weights

1: $i \leftarrow blockIdx.x*blockDim.x + threadIdx.x$
2: $j \leftarrow blockIdx.y*blockDim.y + threadIdx.y$
3: $ij \leftarrow i*n + j$
4: *if* $i < m$ && $j < n$
5: $\eta_{ij} \leftarrow \frac{v_i+\ldots+v_{i+p-1}}{p}$; $\zeta_{ij} \leftarrow \frac{v_i+\ldots+v_{i+q-1}}{q}$;
6: *__syncthreads*()
7: $\boldsymbol{tem} \Leftarrow SurfacePoint(n, p, \boldsymbol{u}, m, q, \boldsymbol{v}, C, \boldsymbol{\eta}_{ij}, \boldsymbol{\zeta}_{ij})$
8: $\boldsymbol{S}_{ij} \Leftarrow \frac{tem_0}{tem_2}$; $\boldsymbol{S}_{ij} \Leftarrow \frac{tem_1}{tem_2}$;
9: $cp_{en} \Rightarrow ed$; $ed_{ij} \Leftarrow en$;
10: $\boldsymbol{R}^* \Leftarrow Nurbs2DBasis(\boldsymbol{\eta}_{ij}, \boldsymbol{\zeta}_{ij}, p, q, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}^T)$
11: $\boldsymbol{A}_{ij,ed} \Leftarrow \boldsymbol{R}^*$; $[mx_{ij}, I] \Leftarrow max(\boldsymbol{R}^*)$; $ix_{ij}=ed_I$;
12: *for* k=0 to N_{holes}-1 *do*
13: $\boldsymbol{tm}^*_k \Leftarrow \sqrt{S_{ij}\text{-}x_k,\ S_{ij}\text{-}y_k}$ - r_0
14: *end*
15: $\Phi_{ij} \Leftarrow max(\boldsymbol{tm}^*)$
16: *end*

To assemble (16) as shown in Fig. 9, the initial LSF value $\boldsymbol{\Phi}$ and matrix $\boldsymbol{A}$ in each collocation point should be evaluated. Then the vector of expansion coefficient φ can be solved, which are the design variables in the TO procedure. The scale of $\boldsymbol{A}$ is very large for large scale problems, thus it needs to be stored with sparse format to save memory. Apart from the reduced number of accesses to the global matrix, the calculation herein does not require lookups, thus the coordinate list (COO) format is appropriate herein.

4 Parallel strategy of isogeometric analysis for NURBS based TO

4.1 Isogeometric analysis

In IGA-based TO, IGA is used for the sensitivity analysis, discretizing the design domain Ω into elements, and the same NURBS basis functions are used for geometric representation and sensitivity analysis. Therefore, no spatial discretization error is introduced in the TO. For 2D problems, a variable value x (e.g., coordinate, displacement and force.) corresponding to a point with the geometric parametric coordinate (η, ξ) is evaluated from the control point values as

$$x(\eta, \zeta) = \sum_i N_i(\eta, \zeta) x_i, \tag{15}$$

where N_i is the basis function of the ith control point that influence on (η, ζ), and x_i is the corresponding value of the control point. The spatial discretization with a patch containing 3×3 quatratic NURBS elements is shown Fig. 10 (Wang and Benson 2015a, b). Additionally, the continuity between the quadratic NURBS elements is C^1.

The discrete equilibrium equation may be written as (Hughes 2012)

$$\boldsymbol{KU} = \boldsymbol{F}, \tag{16}$$

in which $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{U}$ is the displacement vector and $\boldsymbol{F}$ is the external force vector associated with the control points. The stiffness matrix $\boldsymbol{K}$ consists of the element stiffness matrix $\boldsymbol{K}_e$ calculated by IGA.

4.2 Implementing isogeometric analysis with CUDA

The assembly of the stiffness matrix is a computationally demanding task. To build the global stiffness matrix, the local stiffness matrix $\boldsymbol{K}_e$ should be built by adding the contributions

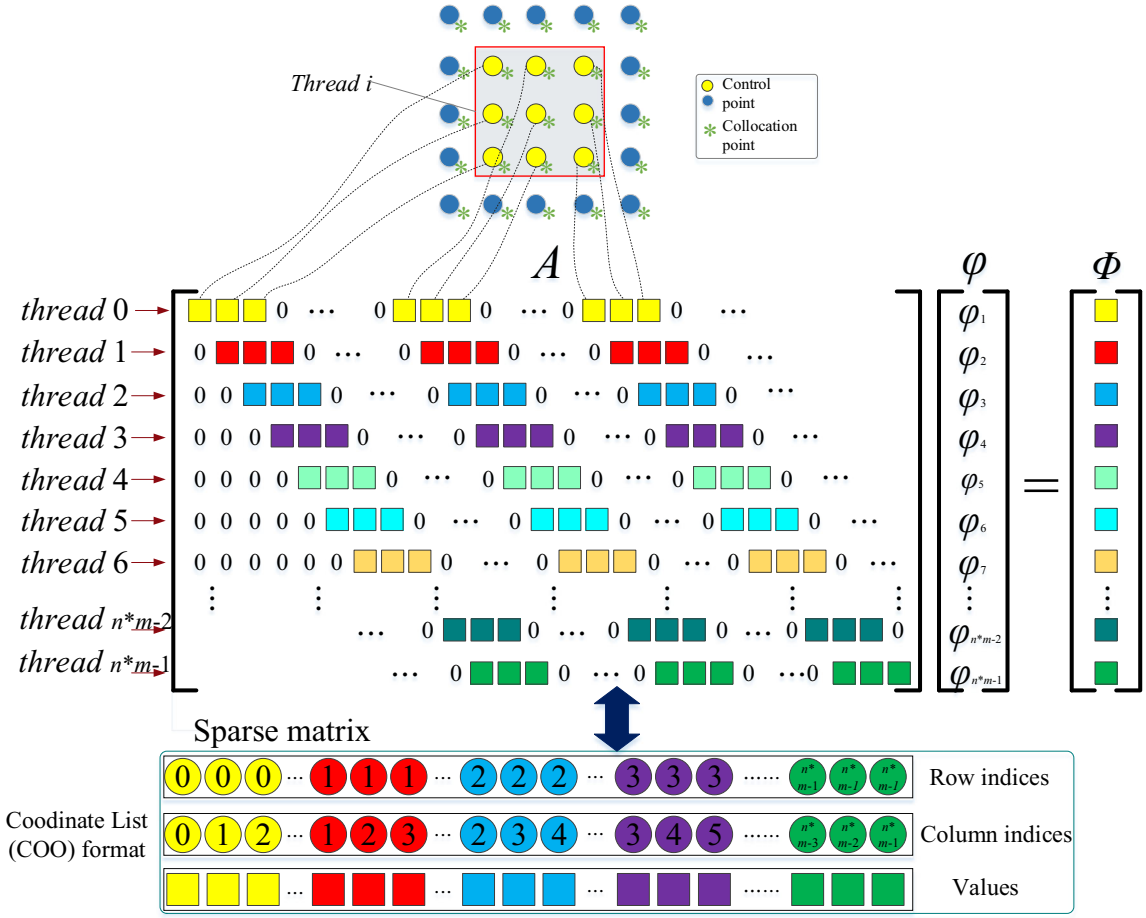


Fig. 9 Parallel strategy for assembling the equation

of all Gauss points of each element separately, then they will be subsequently appended to the global stiffness matrix as (17) in the appropriate positions:

$$\boldsymbol{K} = \sum_{e} \boldsymbol{K}_e. \tag{17}$$

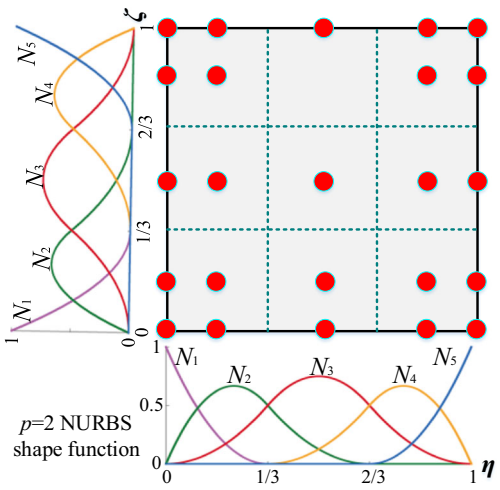


Fig. 10 Spatial discretization of IGA with quadratic NURBS elements

4.2.1 CUDA thread race conditions

In the IGA procedure, "one-thread-one-element" mode of parallelization is adopted as shown in Fig. 11. A typical problem is the racing of CUDA threads, where no less than two threads (e.g., *i* with red color and *j* with blue color) attempt to access the same memory location concurrently as shown in Fig. 12. Thus there might be conflicting updates to the same coefficients of the stiffness matrix.

The aforementioned race conditions can be avoided with proper synchronization with atomic operations in massively GPU parallel computing, but this is detrimental to performance since all updates are serialized (Karatarakis et al. 2014). To address this problem, an approach combining the advantages of MATLAB and the acceleration with GPUs is proposed, which is inspired by the approach described by Davis (Davis 2007). With this method, the most compute-intensive part will be executed on GPUs using CUDA, and the process of assembling the global stiffness matrix $\boldsymbol{K}$ can be illustrated as Fig. 13. Different from Fig. 11, the summation from local stiffness matrices to the global stiffness matrix is executed by sparse function in MATLAB.

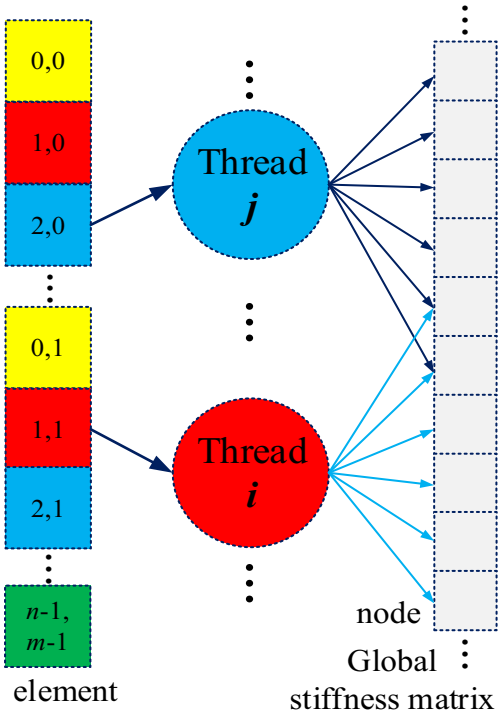


Fig. 11 Assembling stiffness matrix

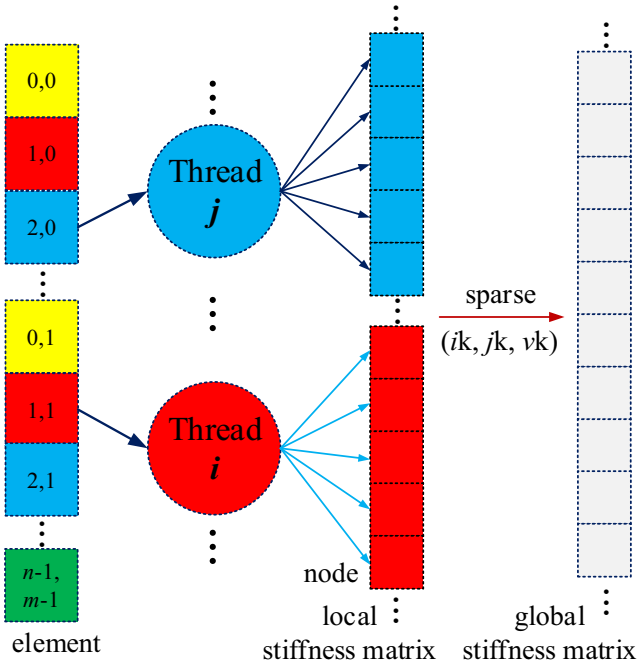


Fig. 13 Assembling stiffness matrix without conflicting data

The sparse function takes three vectors as input arguments: row and column indices (***iK***, ***jK***) of the non-zero matrix entries, which are collected in the third vector (***vK***) as shown in Fig. 14. Specifying the same row and column indices multiple times results in a summation of the corresponding entries.

4.2.2 Data structure and parallel strategy for IGA

The global stiffness matrix is stored in sparse format (Wong et al. 2015). Arrays of C/C++ structure are used to store the three vectors (***iK***, ***jK*** and ***vK***) of $\boldsymbol{K}_e$, which is efficiently performed on the sparse function avoiding the summation in CUDA to avoid thread conflict. To accelerate the computational procedures, the CUDA kernel should be integrated into MATLAB, and the written C code for controlling the CUDA should be adapted to MATLAB. Therefore, MEX-files (Andrews 2012), which can dynamically link subroutines produced from C/C++ code, are utilized. The hybrid programming model with MATLAB, C and CUDA is shown in Fig. 15.

A simplified parallel algorithm for IGA is presented in Table 2, and the symbols represents the same meanings as

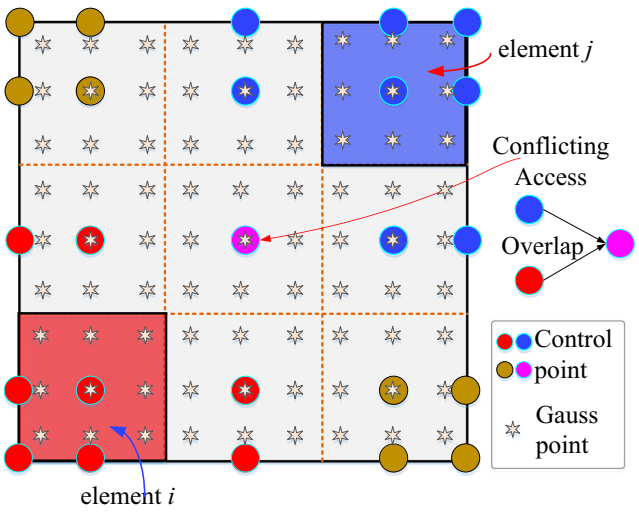


Fig. 12 Conflicting access for assembling K

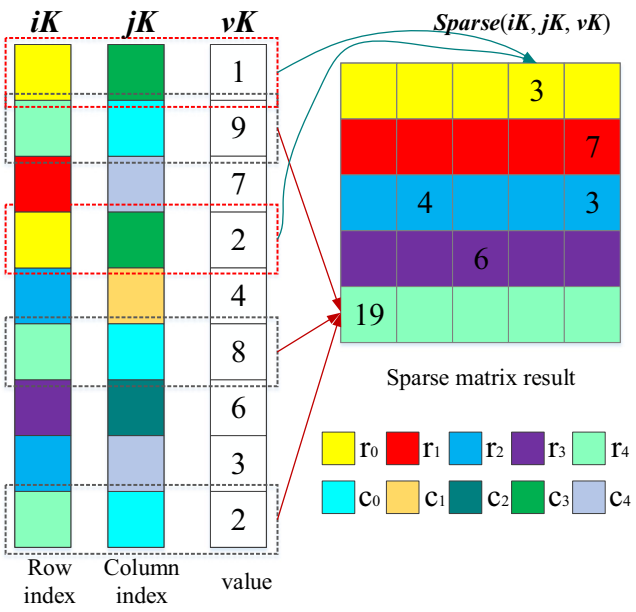
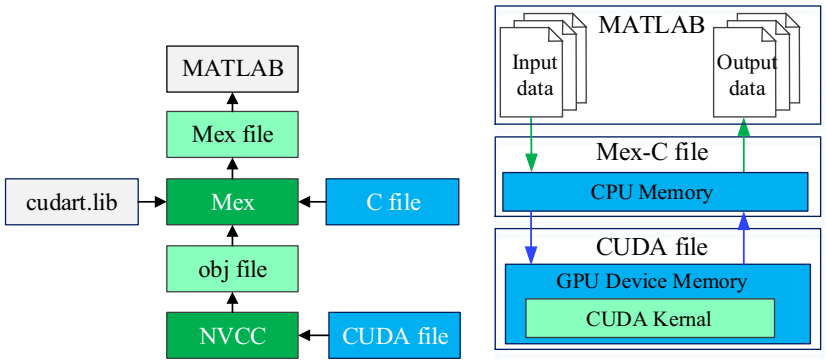


Fig. 14 An example of the summation with sparse function



(a) Process of building Mex-files. (b) Data process flow.

Fig. 15 Hybrid programming model of MATLAB, C and CUDA. **a** Process of building Mex-files. **b** Data process flow

Table 1. The *BoundElement*() function calculates the densities ***dn*** and solid area ***a*** of each element *ij*. $\boldsymbol{\Phi}_e$ represents the LSF values of the interpolation points. The *FilterSensitivity*() function is a filtering scheme, which is used to smooth the free design boundary within a narrowband region to practically improve the smoothness of the strain energy densities by filtering the element pseudo-densities using a window function (Luo et al. 2008). The *CalcEleDOF*() function computes the DOFs of the specific element. The *parent2ParametricSpace*() function is used to compute coordinates in parametric space. The *jacobianMapping*() function is used to transform Jacobi matrix. *tol* is the minimum tolerance of each element's density. ***D*** is the plane-stress elasticity

Table 2 Parallel algorithm for isogeometric analysis

Segment 1 – Kernel 1, Isogeometric analysis

idx: indices of elements
P: coordinates of control points
cp: control point numbers
Q, ***W***: coordinates and weights of Gauss points
en: element number, *ed*: DOFs of *en*
elU, ***elV***: range of elements
u, ***v***: knot vectors, ***w***: weights
N_{gs}: number of Gauss points

1: $i \leftarrow blockIdx.x*blockDim.x+threadIdx.x$
2: $j \leftarrow blockIdx.y*blockDim.y+threadIdx.y$
3: $ij \leftarrow i*n+j$
4: *if* $i<m$ && $j<n$
5: $[\boldsymbol{a}_{ij}, \boldsymbol{dn}_{j,i}] \Leftarrow BouldElement(\boldsymbol{\Phi}_e)$
6: $A_{\boldsymbol{0}} \Leftarrow A_{\boldsymbol{0}}+ a_{ij}$
7: $DN_{j,i} \Leftarrow FilterSensitivity(DN_{j,i})$
8: $\boldsymbol{idx}_{ij,0} \Rightarrow iu$; $\boldsymbol{idx}_{ij,1} \Rightarrow iv$;
9: $elU_{iu} \Rightarrow e\eta$; $elV_{iv} \Rightarrow e\zeta$;
10: $\boldsymbol{cp}_{ij} \Rightarrow ed_0$; $\boldsymbol{P}_{ed0} \Rightarrow \boldsymbol{p}$;
11: $ed \Leftarrow CalcEleDOF(ed_0)$; $nn \Leftarrow len(ed_0)$;
12: *if* $DN_{j,i} < tol$
13: $DN_{j,i} \Leftarrow tol$
14: *end*
15: *for* k=0 to $\boldsymbol{N}_{gs}$-1 *do*
16: $\boldsymbol{Q}_k \Rightarrow \boldsymbol{pt}$; $\boldsymbol{w}_k \Rightarrow \boldsymbol{wt}$;
17: $\eta \Leftarrow parent2ParametricSpace(\boldsymbol{e\eta}, \boldsymbol{pt}_0)$; $\zeta \Leftarrow parent2ParametricSpace(\boldsymbol{e\zeta}, \boldsymbol{pt}_1)$;
18: $J_2 \Leftarrow jacobianMapping\ (\boldsymbol{e\eta}, \boldsymbol{e\zeta})$
19: $[d\boldsymbol{\eta}, d\boldsymbol{\zeta}] \Leftarrow Nurbs2Dders([\eta, \zeta], p, q, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}^{\mathrm{T}})$
20: $J \Leftarrow \boldsymbol{p}^{\mathrm{T}}*[d\boldsymbol{\eta}^{\mathrm{T}}\ d\boldsymbol{\zeta}^{\mathrm{T}}]$; $J_1 \Leftarrow det(J)$;
21: $d\boldsymbol{R}^{*} \Leftarrow [d\boldsymbol{\eta}^{\mathrm{T}}\ d\boldsymbol{\zeta}^{\mathrm{T}}]*J^{-1}$
22: $\boldsymbol{B} \Leftarrow$ convert($d\boldsymbol{R}^{*}$)
23: $\boldsymbol{K}^{\boldsymbol{e}}{}_{ed,ed} \Leftarrow \boldsymbol{K}^{\boldsymbol{e}}{}_{ed,ed} + \boldsymbol{DN}_{j,i}*\boldsymbol{B}^{\mathrm{T}}*\boldsymbol{D}*\boldsymbol{B}*J_1*J_2*\boldsymbol{wt}$
24: *end*
25: *end*

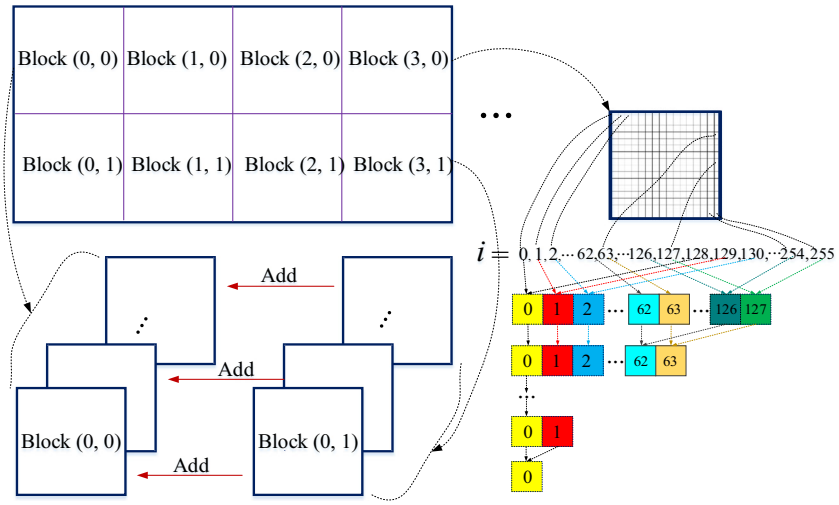


Fig. 16 Reduction summation for solid area A0

matrix. The contributions of all Gauss points of the element are accumulated to $\boldsymbol{K}_e$, which will be assembled to $\boldsymbol{K}$ through MATLAB built-in *sparse* function.

Note that the solid area $\boldsymbol{a}_{ij}$ of each element should be accumulated as shown in Fig. 16, which will lead to access conflict in GPU memory. To tackle this problem, a parallel reduction algorithm is adopted here. Threads 0–127 (256 threads per block)

Table 3 Parallel algorithm for sensitivity analysis

Segment 1 – Kernel 1, Sensitivity analysis

cp: control point numbers, ***ed***: element ID
U, ***V***: parametric coordinates
P: coordinates of control points
u, ***v***: knot vectors, ***w***: weights
sn: strain energy of nodes
sd: strain energy density
dJ: derivation of the objective function
*d**C***: derivation of the constraint function

1: $i \leftarrow blockIdx.x*blockDim.x+threadIdx.x$
2: $j \leftarrow blockIdx.y*blockDim.y+threadIdx.y$
3: $ij \leftarrow i*n+j$
4: *if* $i<m$ && $j<n$
5: $\boldsymbol{ed}_{ij} \Rightarrow en$
6: $\boldsymbol{ex} \Leftarrow en\%(m\text{-}p)$; $\boldsymbol{ey} \Leftarrow floor(\frac{en}{m-p})$;
7: $\boldsymbol{U}_{ij} \Rightarrow \eta$; $\boldsymbol{V}_{ij} \Rightarrow \zeta$;
8: $\boldsymbol{cp}_{ij} \Rightarrow ed_0$; $\boldsymbol{P}_{ed0} \Rightarrow \boldsymbol{p}$;
9: $ed \Leftarrow CalcEleDOF(ed_0)$; $nn \Leftarrow len(ed_0)$;
10: $[d\boldsymbol{\eta}, d\zeta] \Leftarrow Nurbs2Dders([\eta, \zeta], p, q, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}^{\mathrm{T}})$
11: $J \Leftarrow \boldsymbol{cp}^{\mathrm{T}}{}_*[d\boldsymbol{\eta}^{\mathrm{T}}\ d\boldsymbol{\zeta}^{\mathrm{T}}]$
12: $d\boldsymbol{R} \Leftarrow [d\boldsymbol{\eta}^{\mathrm{T}}\ d\boldsymbol{\zeta}^{\mathrm{T}}]*\mathcal{J}^{-1}$
13: $\boldsymbol{B} \Leftarrow \mathrm{convert}(d\boldsymbol{R})$
14: $\boldsymbol{tm}^{*} \Leftarrow \boldsymbol{B}_*\boldsymbol{U}_{ed*}\sqrt{\mathrm{DN}_{\mathrm{ey,\ ex}}}$
15: $\boldsymbol{ES}_{ij} \Leftarrow (\boldsymbol{tm}^{*})^{\mathrm{T}}$;
16: $\boldsymbol{sd}_{ij} \Leftarrow -\boldsymbol{sn}_{ij}{}^{\mathrm{T}}*\boldsymbol{D}*\boldsymbol{sn}_{ij}$
17: $\boldsymbol{LT}_{ij} \Rightarrow ti$
18: $dJ_{ti} \Leftarrow \boldsymbol{sd}_{ij}*\boldsymbol{mx}_{ij}$
19: $d\boldsymbol{C}_{ti} \Leftarrow \boldsymbol{mx}_{ij}$
20: *end*

Table 4 Parallel algorithm for the update scheme

Segment 1 – Kernel 1, Update scheme

Λ_M, Λ_R, Λ_L: bound of Lagrange multiplier Λ
ne_x, ne_y: number of elements

1: $\Lambda_M = 0.5*(\Lambda_R+\Lambda_L)$
2: while $(\Lambda_R\text{-}\Lambda_L)\ \ (\Lambda_R+\Lambda_L) > tol_1$ && $\Lambda_R > tol_2$
 # segement 1 – **kernel** 1
3: $i \leftarrow blockIdx.x*blockDim.x+threadIdx.x$
4: *if* $i<n*m\text{-}1$
5: $\boldsymbol{x}_i \Leftarrow normalization(\varphi_i)$
6: $tm_1 \Leftarrow max(tol_3, -dJ_i\ \ (d\boldsymbol{C}_i*\Lambda_M)$
7: $tm_2 \Leftarrow min(\boldsymbol{x}_i + d, \boldsymbol{x}_i * tm_1\ \hat{}\ dp)$
8: $tm_3 \Leftarrow min(tol_4, tm_2)$
9: $tm_4 \Leftarrow min(\boldsymbol{x}_i - d, tm_3)$
10: $\boldsymbol{x}_i^{\#} \Leftarrow max(tol_5, tm_4)$
11: $\varphi_i \Leftarrow anti\text{-}normalization(\boldsymbol{x}_i^{\#})$
12: *end*
 # end segement 1
13: $\Phi^{\#}=\boldsymbol{B}_*\varphi$
 # segement 2 – **kernel** 2
14: $i \leftarrow blockIdx.x*blockDim.x+threadIdx.x$
15: *if* $i<(ne_x+1)*(ne_y+1)\text{-}1$
16: $\boldsymbol{a}_i \Leftarrow BouldElement(\Phi^{\#})$
17: $\boldsymbol{A}_{\boldsymbol{0}} \Leftarrow \boldsymbol{A}_{\boldsymbol{0}}+\boldsymbol{a}_i$
18: end
 # end segement 2
19: *if* $\boldsymbol{A}_0 - vf * ne_x*ne_y > 0$ $\Lambda_L = \Lambda_M$
20: *else* $\Lambda_R = \Lambda_M$
21: *end*
22: $\Lambda_M \Leftarrow 0.5*(\Lambda_R+\Lambda_L)$
23: *end*

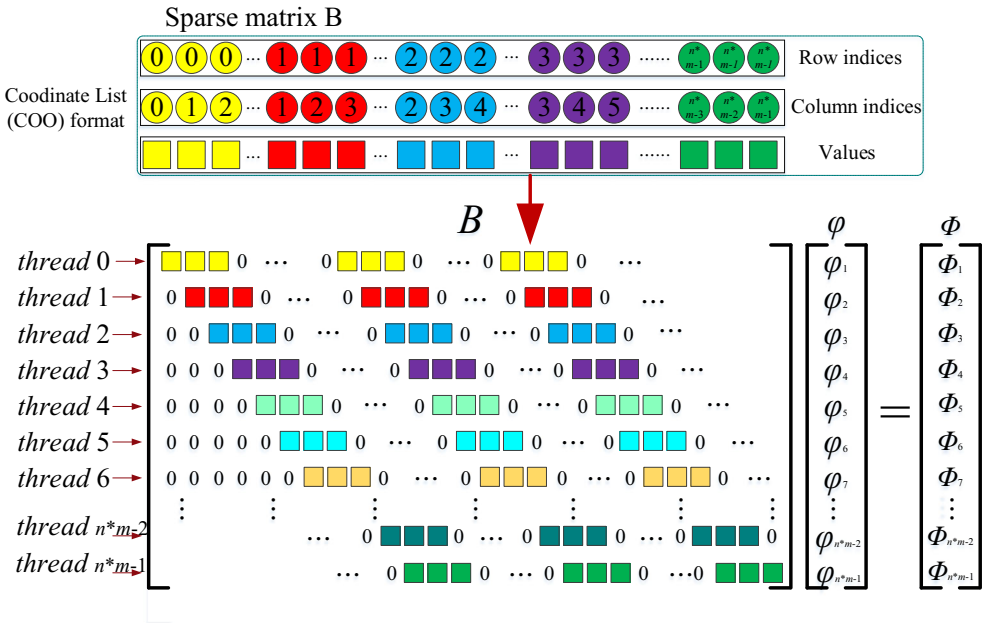


Fig. 17 COO format matrix multiplication for $\Phi = B^{*}\varphi$

execute the add statement during the first iteration while threads 128–255 do not. The pairwise sums are stored in elements 0–127 after the first iteration. Since the warps consist of 32 threads with continuous thread identities, all threads in warps 0–3 execute the add operations, whereas warps 4–7 all skip the add operations (Kirk and Wen-mei 2012).

5 Parallel strategy for sensitivity analysis and update scheme

5.1 Parallel strategy for sensitivity analysis

In the NURBS-based parameterized TO, the expansion coefficients (φ_i in (16)) are used as the design variables instead of the LSF values in the conventional LSM-based TO. For the minimum compliance problem, the mathematical model (Wang and Wang 2006a, b) may be defined as

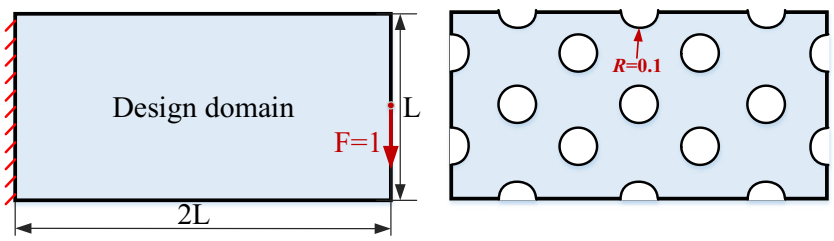
$$\begin{aligned} &\text{Minimize}: \quad J(\boldsymbol{u}, \Phi) = \int_{\Omega} \varepsilon^{T}(\boldsymbol{u}) \boldsymbol{E} \varepsilon(\boldsymbol{u}) H(\Phi) d\Omega, \\ &\text{Subject to}: \quad a(\boldsymbol{u}, \boldsymbol{v}, \Phi) = l(\boldsymbol{v}, \Phi), \quad \boldsymbol{u}\big|_{\partial\Omega} = \boldsymbol{u}_0, \quad \forall \boldsymbol{v} \in \boldsymbol{U}, \\ &V(\Omega) = \int_{\Omega} H(\Phi) d\Omega \leq V_{\max}, \end{aligned} \tag{18}$$

where $H(\Phi)$ is the Heaviside function (Wang and Wang 2004). $J(\boldsymbol{u}, \Phi)$ is the objective function, $\boldsymbol{E}$ is the elastic tensor and ε is the strain. $\boldsymbol{u}$ is the displacement, $\boldsymbol{u}_0$ is the prescribed displacement on the admissible Dirichlet boundary and $\boldsymbol{v}$ is the virtual displacement belonging to the space $\boldsymbol{U}$. The inequality $V(\Omega) \leq V_{max}$ represents the volume constraint. $a(\boldsymbol{u}, \boldsymbol{v}, \Phi)$ is the bilinear form for the strain energy, $l(\boldsymbol{v}, \Phi)$ is the linear for for the load. Based on the methods in (Allaire et al. 2004; Luo et al. 2007, 2008; Wang et al. 2003), the design sensitivities with respect to the expansion coefficients of the NURBS-based interpolation can be written as:

$$\frac{\partial J(\boldsymbol{u}, \Phi)}{\partial \varphi_i(t)} = \int_{\Omega} -\varepsilon^{T}(\boldsymbol{u}) \boldsymbol{E} \varepsilon(\boldsymbol{u}) \delta(\Phi) N_i d\Omega, \tag{19}$$

$$\frac{\partial V(\Phi)}{\partial \varphi_i(t)} = \int_{\Omega} \delta(\Phi) N_i d\Omega, \tag{20}$$

where $\delta(\Phi) = (1/\pi)^{*}(\gamma/(\Phi^2 + \gamma^2))$, and γ should be chosen as 2–4 times as the element size based on the numerical experiences in (Luo et al. 2009). The parallel algorithm for the sensitivity analysis with CUDA is given in Table 3. Similarly as the previous parallel strategy, "one-thread-one-design variable" mode of parallelization is adopted herein.



(a) Design domain. (b) Distribution of initial holes.

Fig. 18 Design domain and initial holes of the cantilever beam problem. **a** Design domain. **b** Distribution of initial holes

Table 5 Average time-consuming for one iteration of TO process in the cantilever problem (unit: s)

Type	Elements	DOFs	Stage	CPU_M	CPU_c	CPU_{MP}	GPU_{cuda}
1	32*16	1,224	S_1	0.43	0.37	0.47	0.06
			S_2	0.73	0.44	0.74	0.09
			S_3	0.04	0.06	0.06	0.03
			S_4	0.09	0.07	0.07	0.04
2	64*32	4,488	S_1	1.63	1.14	1.25	0.15
			S_2	4.15	2.17	2.73	0.29
			S_3	0.21	0.22	0.21	0.06
			S_4	0.32	0.32	0.36	0.07
3	128*64	17,160	S_1	6.4	3.57	3.53	0.34
			S_2	27.7	11.3	10.6	0.95
			S_3	1.1	0.9	0.8	0.14
			S_4	1.5	1.4	1.4	0.17
4	256*128	67,080	S_1	44.9	14.6	13.2	0.79
			S_2	213.9	63.4	64.5	3.2
			S_3	5.7	3.9	3.3	0.32
			S_4	8.2	7.2	6.5	0.42
5	512*256	265,224	S_1	318	75.3	69.7	2.4
			S_2	1963	378	457	10.8
			S_3	32.4	21.6	16.4	0.81
			S_4	52.3	36.4	31.9	1.2
6	1024*512	1,054,728	S_1	2329	413	389	7.9
			S_2	22,638	2344	3564	35.7
			S_3	176	113	93.4	2.3
			S_4	384	183	169	3.7

S_1 represents the stage of initial design domain. S_2 indicates the process of assembling the stiffness matrix $\boldsymbol{K}$ for IGA. S_3 is the stage of sensitivity analysis. S_4 represents the update scheme procedure

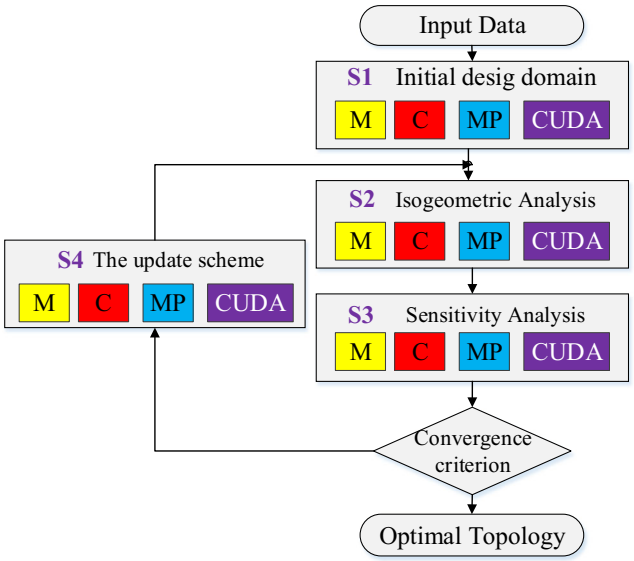


Fig. 19 TO implementing with different language

5.2 Parallel strategy for the update scheme

The optimality criteria (OC) methods attempt to satisfy a set of criteria related to the behavior of the structure to solve structural optimization problems (Hassani and Hinton 1998). It is efficient for the optimization problems with a large number of design variables and a few constraints (Bendsøe and Sigmund 2003), which is exactly the case in continuous TO with a global material volume constraint. For more complicated boundary conditions, other optimization schemes, e.g., the method of moving asymptotes (MMA) (Luo et al. 2007; Zuo et al. 2007), can be adopted instead of the OC method. The updating method for design variables can be written as (Luo et al. 2008) is

$$\varphi_i^{(k+1)} = \left(-\frac{\partial J(\boldsymbol{u},\,\Phi)}{\partial \varphi_i^{(k)}} \Big/ \Lambda^k \frac{\partial V(\Phi)}{\partial \varphi_i^k}\right)\varphi_i^{(k)} \qquad = D_i^{(k)}\varphi_i^{(k)} \tag{21}$$

To simplify this update method, the variable vector $\boldsymbol{\varphi}$ is normalized to a vector $\boldsymbol{x}$ ranging from 0 to 1, and the minimal and maximal limits of $\boldsymbol{x}$ are set to 0.0001 and 1. The final form for the update method (Wang and Benson 2015a, b) is

$$\overline{\varphi}_i^{(k+1)} = \begin{cases} \max\left\{(1-m)\overline{\varphi}_i^{(k)}, \overline{\varphi}_{\min}\right\}, & if\ \left(D_i^{(k)}\right)^{\zeta}\overline{\varphi}_i^{(k)} \le \max\left\{(1-m)\overline{\varphi}_i^{(k)},\ \overline{\varphi}_{\min}\right\} \\ \left(D_i^{(k)}\right)^{\zeta}\overline{\varphi}_i^{(k)}, & if \max\left\{(1-m)\overline{\varphi}_i^{(k)}, \overline{\varphi}_{\min}\right\} < \left(D_i^{(k)}\right)^{\zeta}\overline{\varphi}_i^{(k)} < \min\left\{(1+m)\overline{\varphi}_i^{(k)}, \overline{\varphi}_{\max}\right\} \\ \min\left\{(1+m)\overline{\varphi}_i^{(k)},\ \overline{\varphi}_{\max}\right\}, & if\ \left(D_i^{(k)}\right)^{\zeta}\overline{\varphi}_i^{(k)} \ge \min\left\{(1+m)\overline{\varphi}_i^{(k)},\ \overline{\varphi}_{\max}\right\} \end{cases} \tag{22}$$

Where ζ is the damping factor and m is the move limit. In this paper, ζ is set to 0.3, while m should be determined by experiment (Bendsøe and Sigmund 2003). Similarly, "one-thread-one-design variable" mode of parallelization is adopted herein, and the procedure of the update method is given in Table 4.

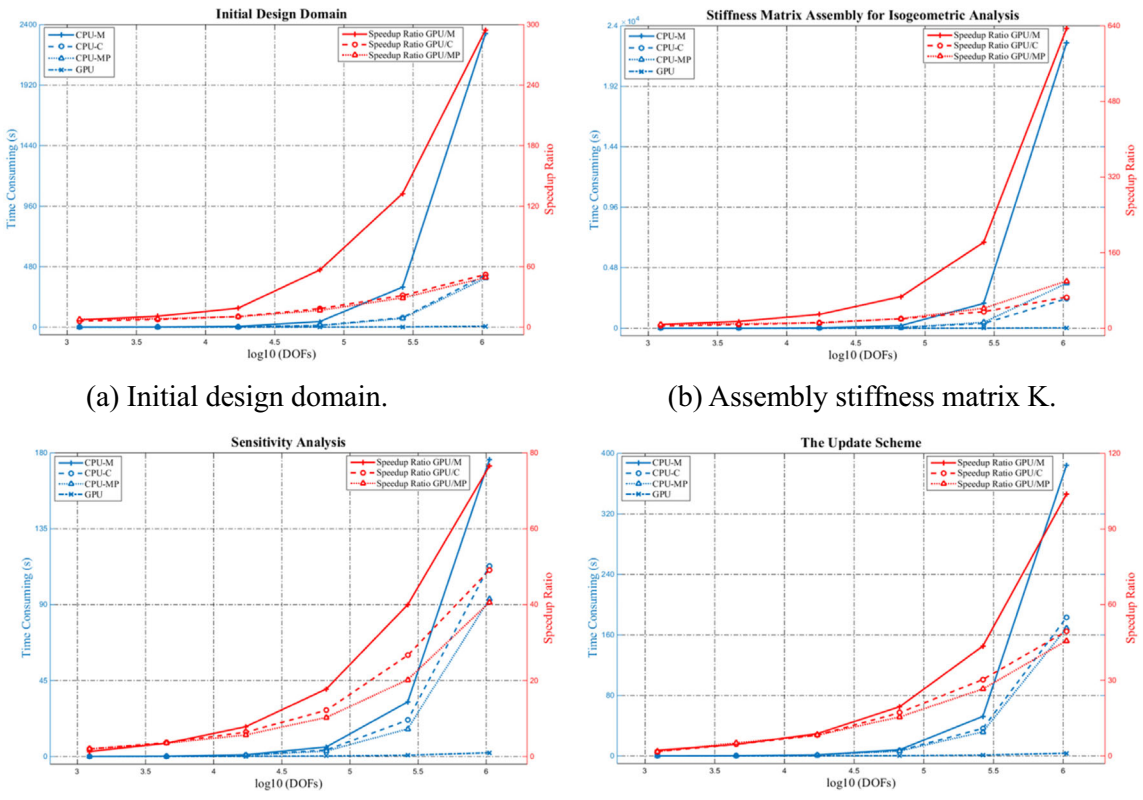
Table 6 Speedup for one iteration of the topology optimization in the cantilever problem

	GPU/CPU-MATLAB				GPU/CPU-C				GPU/CPU-MP			
	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4	S_1	S_2	S_3	S_4
1	7.17	8.11	1.33	2.25	6.17	4.89	2.00	1.75	7.83	8.22	2.00	1.75
2	10.87	14.31	3.50	4.57	7.60	7.48	3.67	4.57	8.33	9.41	3.50	5.14
3	18.82	29.16	7.86	8.82	10.50	11.89	6.43	8.24	10.38	11.16	5.71	8.24
4	56.84	66.84	17.81	19.52	18.48	19.81	12.19	17.14	16.71	20.16	10.31	15.48
5	132.50	181.76	40.00	43.58	31.38	35.00	26.67	30.33	29.04	42.31	20.25	26.58
6	294.81	634.12	76.52	103.78	52.28	65.66	49.13	49.46	49.24	99.83	40.61	45.68

The multiplication calculation in Line 13 of Table 4 can be parallelized with "one-thread-one-row" mode as shown in Fig. 17, then the vector of LSFs Φ can be obtained.

6 Numerical examples

The performance of the proposed parallel TO algorithm integrating IGA and NURBS-based parameterized LSM is verified through three benchmarks: the cantilever beam problem, the Messerschmidt-Bölkow-Blohm (MBB) beam problem, and the quarter annulus problem. The default parameters of these tests are as follows: the elastic modulus for the solid material is 1.0 and for the weak material, 0.0001. The Poisson's ratio is 0.3. The terminal criterion is the relative difference of the objective function values between two iterations, which is set to 0.0001. The 3×3 Gauss quadrature rule is used for the quadratic isogeometric elements. The volume ratio, i.e., material usage ratio, is limited to 0.5. All examples were implemented on a machine with the following hardware: the CPU is an Intel core E5 2630 v2 2.6GHz, the RAM is



(a) Initial design domain. (b) Assembly stiffness matrix K.

(c) Sensitivity analysis. (d) The update scheme.

Fig. 20 Time consuming and speedup ration in TO processes. **a** Initial design domain. **b** Assembly stiffness matrix K. **c** Sensitivity analysis. **d** The update scheme

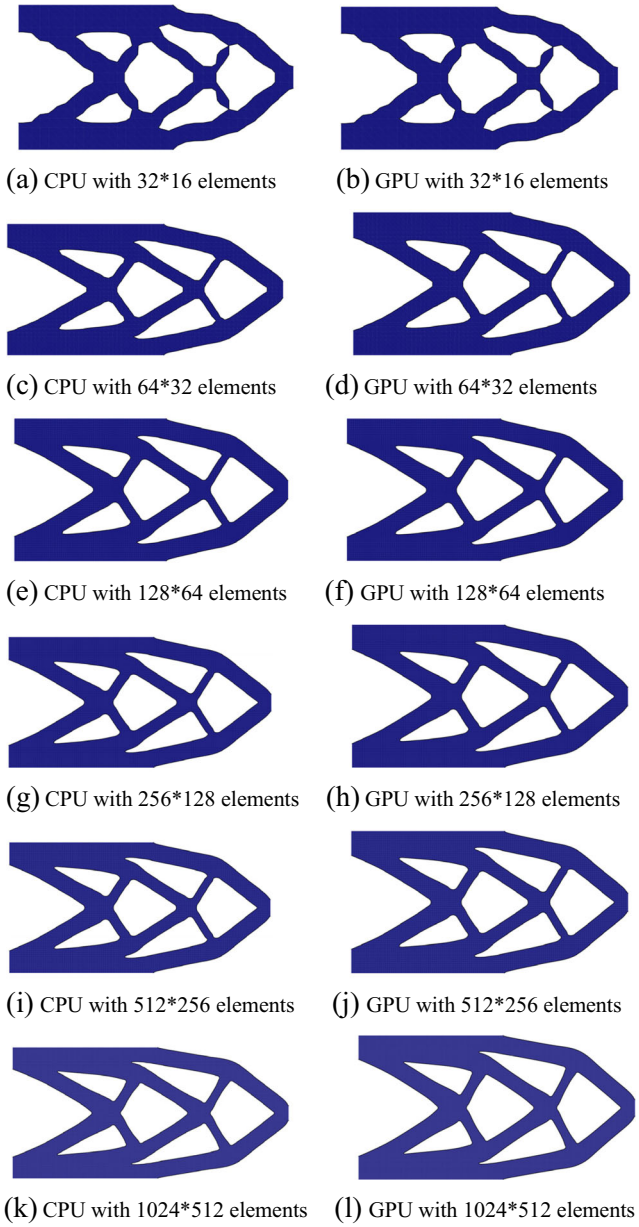


Fig. 21 TO results of cantilever beam problem with different NURBS elements. **a** CPU with 32*16 elements. **b** GPU with 32*16 elements. **c** CPU with 64*32 elements. **d** GPU with 64*32 elements. **e** CPU with 128*64 elements. **f** GPU with 128*64 elements. **g** CPU with 256*128 elements. **h** GPU with 256*128 elements. **i** CPU with 512*256 elements. **j** GPU with 512*256 elements. **k** CPU with 1024*512 elements. **l** GPU with 1024*512 elements

DDR3 SDRAM (16GB), the OS is Windows 8.1 64-bit. The GPU is NVIDIA Quadro K5000, which contains 8 streaming multiprocessors and 1536 CUDA cores. The compilers for CPU codes are Mathworks MATLAB 2014 (Natick, Massachusetts, USA) and Microsoft Visual Studio 2010 (Redmond, Washington, USA), and the compiler for GPU code is NVIDIA CUDA 7.0. All CUDA kernel routines of the proposed approach are developed from scratch by the authors.

Table 7 Final convergent objetive function values for different DOFs of the cantilever problem

Number of NURBS elements	Final convergent objective function value	
	CPU	GPU
32*16	58.992	58.997
64*32	58.991	58.719
128*64	59.946	59.864
256*128	60.871	60.559
512*256	60.589	60.845
1024*512	60.723	60.962

6.1 Cantilever beam

The cantilever beam, as a benchmark problem, is commonly used to evaluate TO methods (Wang and Benson 2015a, b). The design domain of the cantilever beam and its size are shown in Fig. 18a. The dimensionless quantity calculation is adopted, and L is set to 1. The distribution of initial holes is shown in Fig. 18b, which are same as that used by other researchers (Luo et al. 2007, 2009; Wang and Benson 2015a, b).

To demonstrate the speedup of the GPU implementation, the cantilever beam problem is solved by modifying discretization with different quadratic NURBS elements. Table 5 shows the results for different sizes of the cantilever beam problem. Since no reinitialization is needed, the initial design domain will just be executed once in the TO processes, while IGA, sensitivity analysis and the update scheme will be iteratively executed in the solving processes. To compare efficiency among different environments, MATLAB (M), C, MATLAB with parallel (MP) and CUDA C are used in this work as shown in Fig. 19.

The average computational time of one iteration is given in Table 5. When the scale of the problem is small (e.g., 32*16), the consuming time of MATLAB with built-in parallel is larger than MATLAB, since the time cost in data transmission may be larger than that in computation. Similarly, the time spending in CUDA are more than C language when the calculating problem is very small. When the scale is more than 1 million, each step will take several hours with CPU computing, while it just needs tens of seconds under CUDA environment.

The speedup ratios among GPU computing and the others are listed in Table 6. The speedups of CUDA to MATLAB are from 1.44 to 691.44, while CUDA to C from 0.66 to 109.32, CUDA to MP from 4.11 to 145.26 for the different scale problems, which demonstrates the high efficiency of the proposed parallel isogeometric TO. Compared with C, MATLAB has a slower speed to compute in this work. Therefore, the speedup ratio of CUDA to MATLAB can up to nearly 700

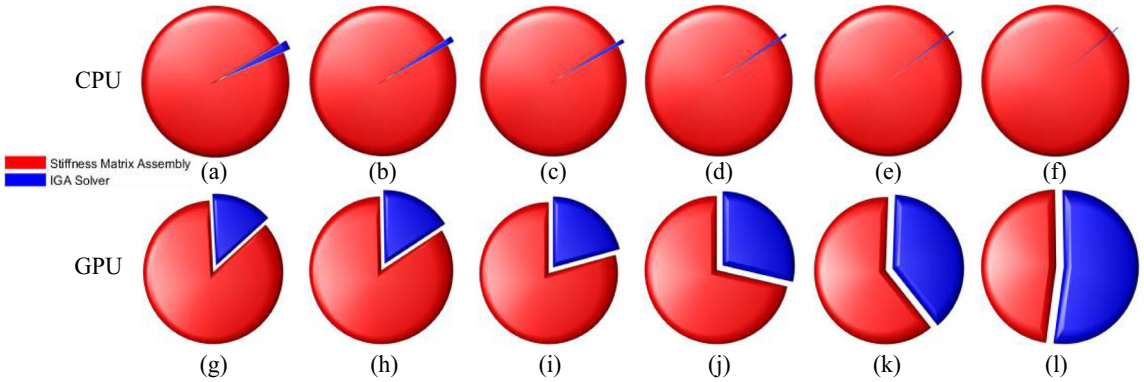


Fig. 22 Consuming time ratio in IGA process with different number of elements: (**a**) CPU with mesh 32*16, (**b**) CPU with mesh 64*32, (**c**) CPU with mesh 128*64, (**d**) CPU with mesh 256*128, (**e**) CPU with mesh 512*256, (**f**) CPU with mesh 1024*512, (**g**) GPU with mesh 32*16, (**h**) GPU with mesh 64*32, (**i**) GPU with mesh 128*64, (**j**) GPU with mesh 256*128, (**k**) GPU with mesh 512*256, (**l**) GPU with mesh 1024*512, (**m**) GPU with mesh 1024*512

times. Meanwhile, there are some scholars contributing efforts to parallel algorithms for TO processes (Challis et al. 2014; Duarte et al. 2015; Zegard and Paulino 2013) with speedups less than 14, and IGA computation (Karatarakis et al. 2014) with speedup of nearly 90.

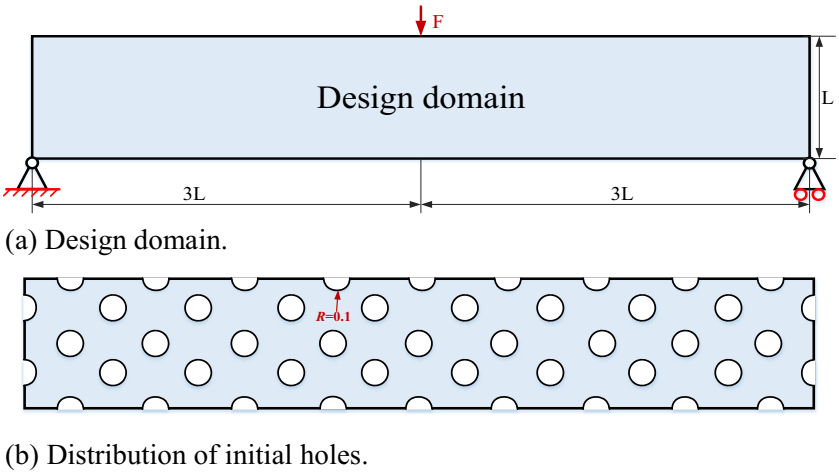
Figure 20 shows the time consuming and speedup ratios for initial design domain, assembling the stiffness matrix $\boldsymbol{K}$ of IGA, sensitivity analysis and the update scheme. It is observed that the performance of the GPU implementation with a small scale problem is not superior, especially compared with C. However, increasing the number of elements, the powerful computational ability is gradually reflected by the increasing speedups. When the DOFs is large enough, the computing capability of GPU can be fully utilized.

The TO results of the cantilever beam problem are shown in Fig. 21 with different mesh scales, and the objective function values are a little different when the TO algorithm converges, which are listed in Table 7. The accuracy of the objective function values under GPU parallel computing are consistent with CPU computing. With the scale increasing, the topology is basically unchanged, but it is not sufficient to represent the precise structure when the mesh number is small.

The comparisons of the time-consuming ratios in IGA processes are shown in Fig. 22. The time of the assembling the stiffness matrix processes is much more than solving in CPU. However, the solving time in GPU is just a little less than assembling the stiffness matrix since the serial solver and CUDA-parallel matrix assembly are used in the GPU scheme. The comparisons show the significant acceleration of CUDA parallel strategy for IGA process.

6.2 MBB beam

Figure 23 defines the design domain of a typical Messerschmidt-Bölkow-Blohm (MBB) beam, which is a widely used benchmark in TO. The allowed material usage



(a) Design domain.

(b) Distribution of initial holes.

Fig. 23 Design domain and initial holes of the MBB beam problem. **a** Design domain. **b** Distribution of initial holes

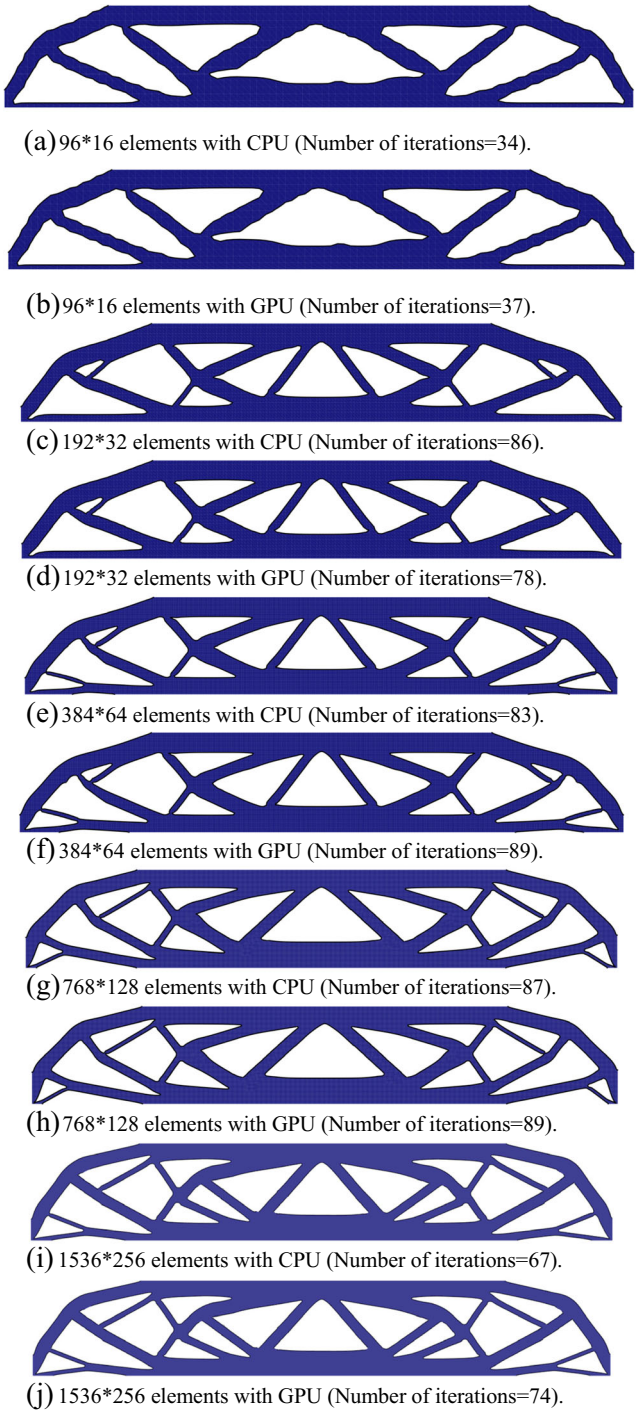


Fig. 24 Design domain and initial holes of the MBB Beam problem. **a** 96*16 elements with CPU (Number of iterations = 34). **b** 96*16 elements with GPU (Number of iterations = 37). **c** 192*32 elements with CPU (Number of iterations = 86). **d** 192*32 elements with GPU (Number of iterations = 78). **e** 384*64 elements with CPU (Number of iterations = 83). **f** 384*64 elements with GPU (Number of iterations = 89). **g** 768*128 elements with CPU (Number of iterations = 87). **h** 768*128 elements with GPU (Number of iterations = 89). **i** 1536*256 elements with CPU (Number of iterations = 67). **j** 1536*256 elements with GPU (Number of iterations = 74)

is limited to 50%. The beam length and width size ratio is 6:1 with a width of L = 1. The domain is initialized with 47 holes.

Move limit controls the changes that can happen at each iteration step and are chosen by experiments. For example, a typical useful value of move limit is 0.2 for density topology optimization (Bendsøe and Sigmund 2003), but for level set topology optimization, the move limit usually is smaller (e.g., 0.01 in (Luo et al. 2008)). By experiments, the suitable move limit m is set to 0.02, 0.03, 0.05, 0.08 and 0.10 respectively with increased elements, and the filter radius is the same for all examples herein. The number of iterations and the TO results corresponding to Fig. 23 are shown in Fig. 24. The design domain is discretized with different quadratic NURBS elements. The optimal design is obtained after different iterations under CPU and GPU environments. When the mesh is less, it is not enough to describe a precise structure. When the mesh is adequate, the optimal result would be obtained, and then the

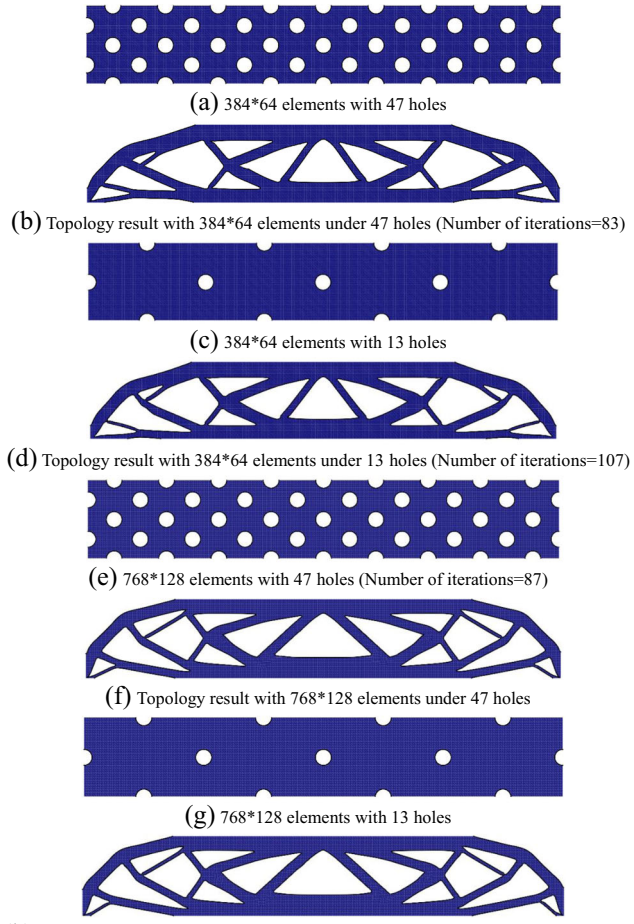
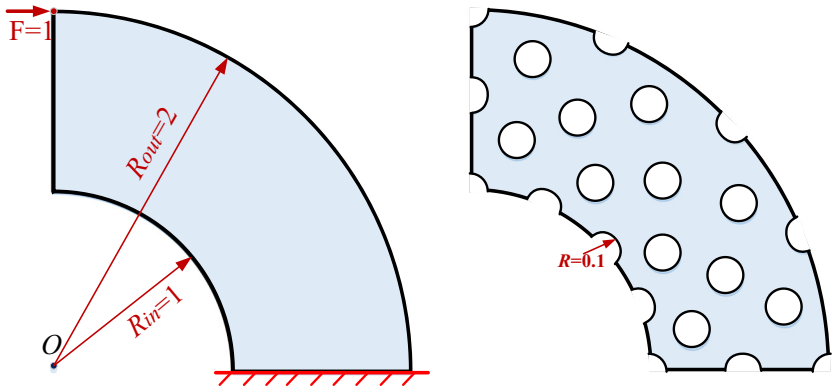
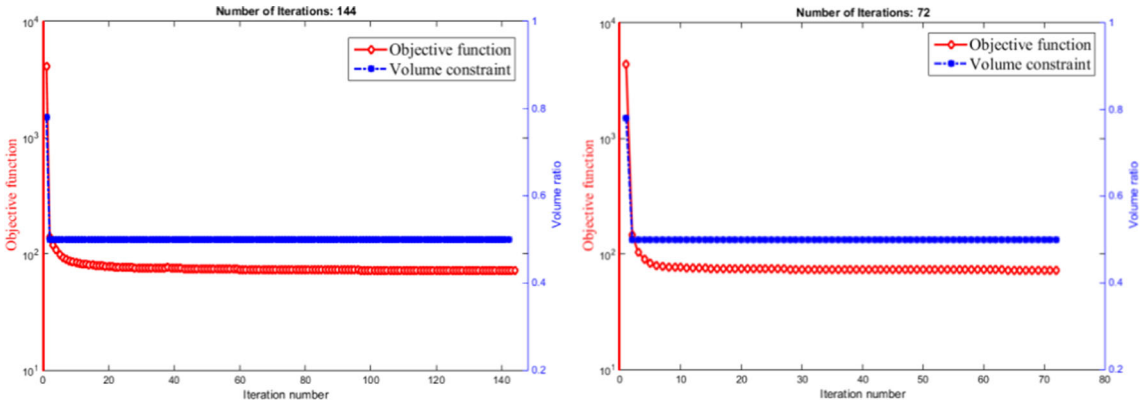


Fig. 25 Optimal results with different elements and initial holes. (a) 384*64 elements with 47 holes. **b** Topology result with 384*64 elements under 47 holes (Number of iterations = 83). **c** 384*64 elements with 13 holes. **d** Topology result with 384*64 elements under 13 holes (Number of iterations = 107). **e** 768*128 elements with 47 holes (Number of iterations = 87). **f** Topology result with 768*128 elements under 47 holes. **g** 768*128 elements with 13 holes. **h** Topology result with 768*128 elements under 13 holes (Number of iterations = 112)



(a) Design domain. (b) Distribution of initial holes.

Fig. 26 Design domain and initial holes of the quarter annulus. **a** Design domain. **b** Distribution of initial holes



(a) CPU (b) GPU

Fig. 27 Convergent histories of the quarter annulus. **a** CPU. **b** GPU

increased mesh would not change the result's shape, since there is no mesh dependency.

Different number of initial holes on the design domain would obtain the similar topology, but different convergence speed as shown in Fig. 25.

(a) CPU. (b) GPU.

Fig. 28 Optimization results of the quarter annulus. **a** CPU. **b** GPU

6.3 Quarter annulus

Figure 26 defines the design domain of a quarter annulus, and the proposed TO method is used to optimize a quarter annulus. The equal-interval initial holes are distributed in the design domain as shown in Fig. 26b. A mesh of 64×64 quadratic NURBS elements is used for both CPU and GPU computing, and the shape and size of the NURBS elements are different in this problem. When a mesh of 512*512 quadratic NURBS elements (DOFs are 528,392) is adopted to this problem, the consuming time of one iteration in the IGA-based TO process using CUDA is approximately 80 s, while it costs nearly ten hours for CPU implementation with MATLAB.

The objective function and the volume ratio over the iterations for both CPU and GPU are given in Fig. 27. The objective function values are very large at the beginning of the optimization, and then decreases to a normal value during the TO processes. The convergence criterion are the same for CPU and GPU computing in this problem. The iterative solution is stopped

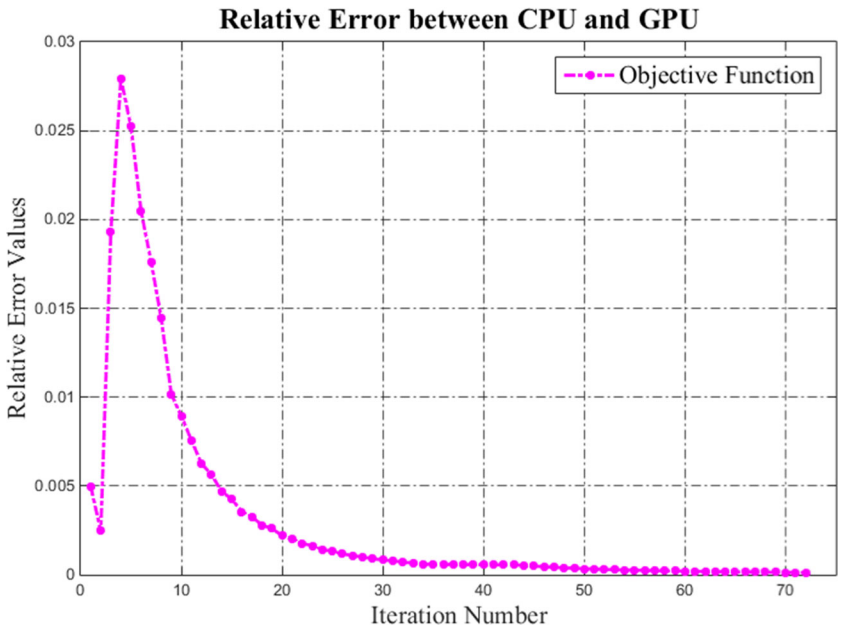


Fig. 29 Relative error between CPU and GPU computing

in the 144*th* iteration for CPU computing while 72*th* iteration for GPU computing with 64*64 NURBS elements. The TO results for both CPU and GPU computing are shown in Fig. 28. The difference between them is small.

To discuss solving error in different environments (CPU and GPU), the relative error of the objective function values between CPU and GPU is compared, which can be defined as:

$$\varepsilon = \frac{\|J_{\mathrm{CPU}} - J_{\mathrm{GPU}}\|_2}{\|J_{\mathrm{CPU}}\|_2}, \tag{22}$$

in which J_{CPU} represents the objective function obtained from CPU computing while J_{GPU} indicates the objective function from GPU parallel computing, ε is the relative error between CPU and GPU.

The relative errors of the first 72 times are shown in Fig. 29, which shows a decreased tendency in the first 2 times since the objective function of both CPU and GPU are very large at the beginning of the optimization. In the 2–4 iteration, the relative errors increases in a sharp tendency because the convergence of single precision (used in GPU) is faster than double precision (used in CPU), but this does not mean that a single-precision has a better convergence than double precision. When the shape reaching a stable line with only minor changes, the objective function of CPU and GPU will be approximative, and the relative error between CPU and GPU becomes smaller.

7 Discussion and conclusions

A GPU implementation is applied to the isogeometric TO method integrating the IGA and the parameterized LSM, which greatly improve the computational efficiency compared with CPU. The GPU parallel strategy shows significant speedups (more than 100 for some cases) for initial design domain, stiffness matrix assembly, sensitivity analysis and the update scheme in the TO processes.

A cantilever beam example with different scales of meshes is used to validate the high efficiency of GPU parallel strategy for parameterized LSM-based TO using IGA by comparing it to that of CPU. A MBB beam example was used to analyze the influence of the mesh scale on the optimal topology. Furthermore, a quarter annulus example with curve-boundary design domain was used to discuss the relative errors between CPU and GPU, and the consuming time of one iteration on only one GPU with over 500 thousand DOFs problem is within 80 s. The examples illustrate that GPU can effectively solve TO problems.

Although the focus of the present paper is the minimum compliance problem of TO, the proposed parallel strategies will not be restricted to this specific problem, and it can be extended to deal with other problems. Moreover, it can be further improved in many aspects: e.g., multiple GPU devices and distributed parallel computing with MPI. Besides that, another forthcoming work is to extend these strategies to three-dimensional problems.

Acknowledgements This research was supported by the National Natural Science Foundation of China (51475180).

References

Aage N, Lazarov BS (2013) Parallel framework for topology optimization using the method of moving asymptotes. Struct Multidiscip Optim 47:493–505

Allaire G, Jouve F, Toader A-M (2004) Structural optimization using sensitivity analysis and a level-set method. J Comput Phys 194: 363–393

Allaire G, Dapogny C, Frey P (2013) A mesh evolution algorithm based on the level set method for geometry and topology optimization. Struct Multidiscip Optim 48:711–715

Allaire G, Dapogny C, Frey P (2014) Shape optimization with a level set based mesh evolution method. Comput Methods Appl Mech Eng 282:22–53

Andrews T (2012) Computation time comparison between matlab and C++ using launch windows

Bazilevs Y, Beirao da Veiga L, Cottrell JA, Hughes TJR, Sangalli G (2006) Isogeometric analysis: approximation, stability and error estimates for h-refined meshes. Math Models Methods Appl Sci 16: 1031–1090

Bendsøe MP, Sigmund O (2003) Topology optimization: theory, methods, and applications. Springer Science & Business Media

Challis VJ, Roberts AP, Grotowski JF (2014) High resolution topology optimization using graphics processing units (GPUs). Struct Multidiscip Optim 49:315–325

Chen J, Shapiro V, Suresh K, Tsukanov I (2007) Shape optimization with topological changes and parametric control. Int J Numer Methods Eng 71:313–346

Chen J, Freytag M, Shapiro V (2008) Shape sensitivity of constructively represented geometric models. Comput Aided Geometric Des 25: 470–488

Cottrell JA, Hughes TJ, Bazilevs Y (2009) Isogeometric analysis: toward integration of CAD and FEA. Wiley

Davis T (2007) Creating sparse finite-element matrices in MATLAB. Guest blog in: Loren Shure: Loren on the Art of MATLAB

De Boor C (1972) On calculating with B-splines. J Approximation Theory 6:50–62

Dede L, Borden M, Hughes T (2012) Topology optimization with Isogeometric analysis in a phase field approach. Archives Comput Methods Eng 19:427–465

Duarte LS, Celes W, Pereira A, Menezes IF, Paulino GH (2015) PolyTop++: an efficient alternative for serial and parallel topology optimization on CPUs & GPUs. Struct Multidiscip Optim 52:845–859

Georgescu S, Chow P, Okuda H (2013) GPU acceleration for FEM-based structural analysis. Archives Comput Methods Eng 20:111–121

Guo X, Zhang W, Zhong W (2014) Explicit feature control in structural topology optimization via level set method. Comput Methods Appl Mech Eng 272:354–378

Hassani B, Hinton E (1998) A review of homogenization and topology optimization III—topology optimization using optimality criteria. Comput Struct 69:739–756

Herrero D, Martinez J, Marti P (2013) An implementation of level set based topology optimization using GPU 10th World Congress on Structural and Multidisciplinary Optimization

Hsu M-C, Akkerman I, Bazilevs Y (2011) High-performance computing of wind turbine aerodynamics using isogeometric analysis. Comput Fluids 49:93–100

Huang X, Xie Y-M (2010) A further review of ESO type methods for topology optimization. Struct Multidiscip Optim 41:671–683

Hughes TJR (2012) The finite element method: linear static and dynamic finite element analysis. Courier Corporation

Hughes TJR, Cottrell JA, Bazilevs Y (2005) Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput Methods Appl Mech Eng 194:4135–4195

Johnson RW (2005) Higher order B-spline collocation at the Greville abscissae. Appl Numer Math 52:63–75

Karatarakis A, Karakitsios P, Papadrakakis M (2014) GPU accelerated computation of the isogeometric analysis stiffness matrix. Comput Methods Appl Mech Eng 269:334–355

Kirk D (2007) NVIDIA CUDA software and GPU parallel computing architecture ISMM 103–104

Kirk DB, Wen-mei WH (2012) Programming massively parallel processors: a hands-on approach. Newnes

Kuźnik K, Paszyński M, Calo V (2012) Graph grammar-based multi-frontal parallel direct solver for two-dimensional isogeometric analysis. Proc Comput Sci 9:1454–1463

Li K, Qian X (2011) Isogeometric analysis and shape optimization via boundary integral. Comput Aided Des 43:1427–1437

Luo Z, Tong L, Wang MY, Wang S (2007) Shape and topology optimization of compliant mechanisms using a parameterization level set method. J Comput Phys 227:680–705

Luo Z, Wang MY, Wang S, Wei P (2008) A level set-based parameterization method for structural shape and topology optimization. Int J Numer Methods Eng 76:1–26

Luo Z, Tong L, Kang Z (2009) A level set method for structural shape and topology optimization using radial basis functions. Comput Struct 87:425–434

MATLAB (2014) version 8.4.0. The MathWorks Inc. Natick, Massachusetts

Microsoft Visual Studio (2010) Microsoft Corp. Redmond, Washington

Mukherjee S, Moore N, Brock J, Leeser M (2012) CUDA and OpenCL implementations of 3D CT reconstruction for biomedical imaging High Performance Extreme Computing (HPEC), 2012 I.E. Conference on. IEEE 1–6

Nvidia C (2007) Compute unified device architecture programming guide

Osher S, Fedkiw RP (2001) Level set methods: an overview and some recent results. J Comput Phys 169:463–502

Osher S, Sethian JA (1988) Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. J Comput Phys 79:12–49

Otomori M, Yamada T, Izui K, Nishiwaki S, Andkjær J (2012) A topology optimization method based on the level set method for the design of negative permeability dielectric metamaterials. Comput Methods Appl Mech Eng 237:192–211

Piegl L, Tiller W (2012) The NURBS book. Springer Science & Business Media

Ploskas N, Samaras N (2014) GPU accelerated pivoting rules for the simplex algorithm. J Syst Softw 96:1–9

Rozvany G (2001) Aims, scope, methods, history and unified terminology of computer-aided topology optimization in structural mechanics. Struct Multidiscip Optim 21:90–108

Schmidt S, Schulz V (2011) A 2589 line topology optimization code written for the graphics card. Comput Vis Sci 14:249–256

Seo Y-D, Kim H-J, Youn S-K (2010) Isogeometric topology optimization using trimmed spline surfaces. Comput Methods Appl Mech Eng 199:3270–3296

Sigmund O, Maute K (2013) Topology optimization approaches. Struct Multidiscip Optim 48:1031–1055

Suresh K (2013) Efficient generation of large-scale pareto-optimal topologies. Struct Multidiscip Optim 47:49–61

Tavakkoli S, Hassani B, Ghasemnejad H (2013) Isogeometric topology optimization of structures by using MMA. Int J Optim Civil Eng 3: 313–326

van Dijk NP, Maute K, Langelaar M, Van Keulen F (2013) Level-set methods for structural topology optimization: a review. Struct Multidiscip Optim 48:437–472

Wadbro E, Berggren M (2009) Megapixel topology optimization on a graphics processing unit. SIAM Rev 51:707–721

Wang Y, Benson DJ (2015a) Multi-patch nonsingular isogeometric boundary element analysis in 3D. Comput Methods Appl Mech Eng 293:71–91

Wang Y, Benson DJ (2015b) Isogeometric analysis for parameterized LSM-based structural topology optimization. Computat Mech 1–17

Wang Y, Benson DJ (2016) Geometrically constrained isogeometric parameterized level-set based topology optimization via trimmed elements. Front Mech Eng. doi:10.1007/s11465-016-0403-0

Wang MY, Wang X (2004) PDE-driven level sets, shape sensitivity and curvature flow for structural topology optimization. Comput Model Eng Sci 6:373–396

Wang S, Wang MY (2006a) Structural shape and topology optimization using an implicit free boundary parametrization method. Comput Model Eng Sci 13:119

Wang S, Wang MY (2006b) Radial basis functions and level set method for structural topology optimization. Int J Numer Methods Eng 65: 2060–2090

Wang MY, Wang X, Guo D (2003) A level set method for structural topology optimization. Comput Methods Appl Mech Eng 192: 227–246

Wang Y, Benson DJ, Nagy AP (2015a) A multi-patch nonsingular isogeometric boundary element method using trimmed elements. Computat Mech 1–19

Wang Y, Wang Q, Deng X, Xia Z, Yan J, Xu H (2015b) Graphics processing unit (GPU) accelerated fast multipole BEM with level-skip M2L for 3D elasticity problems. Adv Eng Softw 82:105–118

Wang Y, Xu H, Pasini D (2016) Multiscale isogeometric topology optimization for lattice materials. Comput Methods Appl Mech Eng

Wei P, Wang MY, Xing X (2010) A study on X-FEM in continuum structural optimization using a level set model. Comput Aided Des 42:708–719

Wei Y, Wang Q, Huang Y, Wang Y, Xia Z (2015) Acceleration of free-vibrations analysis with the Dual Reciprocity BEM based on $\mathcal{H}$-matrices and CUDA. Eng Comput 32:211–233

Wong J, Kuhl E, Darve E (2015) A new sparse matrix vector multiplication graphics processing unit algorithm designed for finite element problems. Int J Numer Methods Eng 102:1784–1814

Woźniak M (2015) Fast GPU integration algorithm for isogeometric finite element method solvers using task dependency graphs. J Comput Sci

Woźniak M, Kuźnik K, Paszyński M, Calo V, Pardo D (2014) Computational cost estimates for parallel shared memory isogeometric multi-frontal solvers. Comput Math Appl 67:1864–1883

Wu J, Dick C, Westermann R (2016) A system for high-resolution topology optimization. IEEE Trans Vis Comput Graph 22:1195–1208

Xia Q, Wang MY, Shi T (2014a) A level set method for shape and topology optimization of both structure and support of continuum structures. Comput Methods Appl Mech Eng 272:340–353

Xia Z, Wang Q, Huang Y, Yixiong W, Yingjun W (2014b) Parallel Strategy of FMBEM for 3D Elastostatics and its GPU Implementation Using CUDA ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Am Soc Mech Eng V01AT02A060-V001AT002A060

Xia Z, Wang Q, Liu Y, Wang Y, Wei Y (2014c) M2L optimization in FMBEM and its GPU implementation. WIT Trans Model Simulat 56:307–319. doi:10.2495/BEM360261

Xia Z, Wang Q, Wang Y, Yu C (2015) A CAD/CAE incorporate software framework using a unified representation architecture. Adv Eng Softw 87:68–85

Xia Z, Wang Q, Liu Q, Wang Y, Liu J, Chen G (2016) A novel approach for automatic reconstruction of boundary condition in structure analysis. Adv Eng Softw 96:38–57

Yamada T, Izui K, Nishiwaki S, Takezawa A (2010) A topology optimization method based on the level set method incorporating a fictitious interface energy. Comput Methods Appl Mech Eng 199:2876–2891

Zegard T, Paulino GH (2013) Toward GPU accelerated topology optimization on unstructured meshes. Struct Multidiscip Optim 48:473–485

Zuo K-T, Chen L-P, Zhang Y-Q, Yang J (2007) Study of key algorithms in topology optimization. Int J Adv Manuf Technol 32:787–796