



ARTICLE

## A Parallel Computing Schema Based on IGA

Jinggang Deng<sup>1,2</sup>, Bingquan Zuo<sup>1,2,\*</sup>, Huixin Luo<sup>1,2</sup>, Weikang Xie<sup>1,2</sup> and Jiashu Yang<sup>1,2</sup>

<sup>1</sup>Key Laboratory of Metallurgical Equipment and Control Technology Ministry of Education, Wuhan University of Science and Technology, Wuhan, 430081, China

<sup>2</sup>Hubei Key Laboratory of Mechanical Transmission and Manufacturing Engineering, Wuhan University of Science and Technology, Wuhan, 430081, China

\*Corresponding Author: Bingquan Zuo. Email: zuobingquan@gmail.com

Received: 03 December 2021 Accepted: 27 January 2022

### ABSTRACT

In this paper, a new computation scheme based on parallelization is proposed for Isogeometric analysis. The parallel computing is introduced to the whole progress of Isogeometric analysis. Firstly, with the help of the “tensor-product” and “iso-parametric” feature, all the Gaussian integral points in particular element can be mapped to a global matrix using a transformation matrix that varies from element. Then the derivatives of Gauss integral points are computed in parallel, the results of which can be stored in a global matrix. And a middle layer is constructed to assemble the final stiffness matrices in parallel. The numerical example results show that: the method presented in the paper can reduce calculation time and improve the use rate of computing resources.

### KEYWORDS

Isogeometric analysis; parallel; Gaussian integration; middle layer

## 1 Introduction

FEM usually require a fine, often multidimensional, discretization, resulting into several doffs and associated unknowns. The research on FEM acceleration has been developed in many aspects. One important way is the fast mesh generation method [1–3], because the meshing process takes up 75% of the time for FEM [4,5]. In order to keep the discrete accuracy using fewer elements, the adaptive mesh generation methods [6,7] are constructed. Besides, researchers also tried to increase the speed of discretization [8] integration [9,10], stiffness matrix assembling [11] and solving [12,13]. And most of these methods used parallel technology, especially the GPU [14,15]. In particular, A parallel FEM [16] framework based on the matrix-free technique has drawn much more attention than before, compared with the element-by-element (EBE) [17] and node-by-node (NBN) [18,19].

Isogeometric analysis(IGA) is a new numerical method for solving partial differential equations(PDE) [20], which was proposed by Hughes et al. in 2005 [4]. Different from finite element method (FEM) [21], in IGA, models from CAD do not need to be transformed into meshes by approximating to the geometric model which can be a time-consuming progress. But more computation time is still required for complex models because the size of the final matrix from both IGA and FEM



grows. Obviously, an efficient computing framework is important for them. On behalf of its meshfree characteristic, IGA saved much time on mesh generation, but some efforts still have been done to accelerate the computing process. Researchers firstly take account of different splines to optimize the mesh, such as B-spline, NURBS [4], T-splines [22,23] and so on. And based these splines, local-refine supported splines (e.g., HB/THB [24–26], LR-Spline [27–29], AST [30,31], PHT [29,32]...) are used to control the non-essential subdivision areas. As done in FEM, the integration is also considered in [33–35]. It should be noted that: the Gaussian integral points are quite important for integration, so reducing the points number [36] and optimizing the points value [37] are two main approach in integration for IGA. And a new way of integrating in [38] In any event, assembly remains a critical part of acceleration, so GPU was used for matrix assembling in IGA [39].

As shown above, the parallel computation shows its huge potential in accelerating for both FEM and IGA. Especially, as a newcomer in numerical methods, there are fewer research about parallel computation on IGA than others. Related research is mainly focused on the theory and application of IGA [20–22], and the advantages and compare with FEM. It is only in recent years that some work has been done to improve computing efficiency, and it is still in its infancy. Reference [40] proposed a new level set-based topology optimization (TO) method using a parallel strategy of Graphics Processing Units (GPUs) and IGA. Reference [41] proposed and investigate new robust preconditioners for space-time IGA of parabolic evolution problems. An innovative family of solution schemes based on domain decomposition methods (DDM) is proposed, that significantly reduces the computational cost in [42]. Reference [43] proposed a solution strategy for rIGA for parallel distributed memory machines and compare the computational costs of solving rIGA vs IGA discretizations. Since most parallel strategies are only for one or two steps in IGA, it is particularly important to propose a complete parallel framework. In this paper, the process of IGA is reconstructed by combining the parallel algorithm of finite element, so that the whole process of IGA can be processed in parallel to save the time of calculation. The presented paper structure is as follows:

The first chapter mainly introduce related research background, such as IGA, calculate acceleration.

The second chapter introduces the basic concept of IGA based on NURBS.

The third chapter is the content of the Gaussian integral point mapping, parallel derivation and stiffness matrix of parallel assembly based on the middle layer.

The fourth chapter has carried on the contrast experiment in parallel acceleration and traditional geometric analysis such as time-consuming.

The fifth chapter is the summary.

## 2 NURBS and Isogeometric Analysis

This section mainly introduces some concepts of Non-Uniform Rational B-Splines (NURBS) and IGA with Poisson equation as an example.

### 2.1 NURBS Theory

B-spline is defined as  $\xi := (\xi_i)$  non-decreasing sequence  $\xi := (\xi_i)$  where  $\xi_i$  is called the base point [44]. The one-dimensional B-spline basis of order 0 is composed of

$$N_{i,0}(u) = \begin{cases} 1 & \xi_i \leq u \leq \xi_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The point sequence  $u$  may contain repeated points. Defines a specific point sequence with repeated points in the point sequence. The  $i$ th B-spline basis with order  $p > 0$  is defined as

$$N_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{i,p-1}(\xi) + \frac{\xi - \xi_{i+1}}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,p-1}(\xi) \quad (2)$$

The 2-d B spline basis is defined as the tensor product of two 1-d B spline basis

$$N_{ij}(\xi, \zeta) = N_{i,p}(\xi) N_{j,q}(\zeta) \quad (3)$$

where  $p$  and  $q$  are the orders on  $u$  and  $v$  directions. The NURBS basis is defined as

$$R_{ij}(\xi, \zeta) = \frac{W_{ij} N_{i,p}(\xi) N_{j,q}(\zeta)}{\sum_{ij} W_{ij} N_{i,p}(\xi) N_{j,q}(\zeta)} \quad (4)$$

where  $W_{ij}$  is the weight of  $R_{ij}$ .

## 2.2 Isogeometric Analysis

Both FEM and IGA have the same theoretical basis, which is the weak form of partial differential equations. As a typical elliptic partial differential equation, Poisson equation is commonly used in gravitational field, electromagnetic field, and thermodynamic field.

$$\begin{cases} -\Delta U(X) = f(X), & \text{in } \Omega \\ U(X) = U_0(X), & \text{on } \partial\Omega \end{cases} \quad (5)$$

where  $\Omega$  is a two-dimensional field bounded by  $\partial\Omega$ ,  $f(X) \in L^2(\Omega) : \Omega \rightarrow R$  is the source function,  $U(X)$  is the unknown function. Transform Eq. (6) into a weak form

$$\int_{\Omega} \nabla U(X) \cdot \nabla \varphi(X) d\Omega = \int_{\Omega} f(X) \varphi(X) d\Omega \quad (6)$$

In which  $\varphi(X)$  is zero boundary partial  $\Omega$  experiment function. In IGA geometry,  $U(X)$  is estimated by  $\hat{U}(X)$ .  $\hat{U}(X)$  is a function of the constituent space of a NURBS basis. Define the mapping  $F$  from  $\Omega$  to parameter domain  $\Omega_0$

$$X = F(\xi, \zeta) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} R_{ij}(\xi, \zeta) C_{ij} \quad (7)$$

where  $R_{ij}$  is the tensor product of NURBS bases, and  $C_{ij}$  is the corresponding control point,  $V = (\xi, \zeta) \in \Omega_0$ , since  $F$  is a mapping matrix,  $F$  inverse exists, defined as

$$\hat{U}(V) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} R_{ij}(F^{-1}(X)) Q_{ij} \quad (8)$$

Now transform Eq. (7) to the parameter field

$$\begin{aligned} & \int_{\Omega_0} (DF^{-T}(\xi, \zeta) \nabla U(\xi, \zeta)) \cdot (DF^{-T}(\xi, \zeta) \nabla \varphi(\xi, \zeta)) |\det DF|(\xi, \zeta) d\Omega_0 \\ &= \int_{\Omega_0} (f\varphi)(F(\xi, \zeta)) |\det DF(\xi, \zeta)| d\Omega_0 \end{aligned} \quad (9)$$

$DF(\xi, \zeta)$  is the Jacobian matrix and is defined as

$$|\det DF(\xi, \zeta)| = \begin{vmatrix} \partial F_1 / \partial \xi & \partial F_1 / \partial \zeta \\ \partial F_2 / \partial \xi & \partial F_2 / \partial \zeta \end{vmatrix}$$

With  $\hat{U}(X)$  instead of  $U(X)$ , then test letter  $\varphi_{ij}(X) = R_{ij}(F^{-1}(X))$ ,  $1 \leq i \leq n_i$ ,  $1 \leq j \leq n_j$  substitute in type (10), and then get linear equations

$$\mathbf{KQ} = \mathbf{b} \quad (10)$$

where

$$K_{i_1j_1, i_2j_2} = \int_{\Omega} \nabla \varphi_{i_1j_1}(X) \cdot \nabla \varphi_{i_2j_2}(X) d\Omega, \quad (11)$$

$$b_{ij} = \int_{\Omega} f(X) \varphi_{ij}(X) d\Omega \quad (12)$$

And  $Q := (Q_{ij})$ . Once the linear equation has been solved, substitute  $Q$  into Eq. (9) and the numerical solution is obtained.

The elements of  $K$  and  $b$  are calculated by Eqs. (8) and (9), respectively. Transform Eqs. (8) and (9) into the parameter field, and get

$$K_{i_1j_1, i_2j_2} = \int_{\Omega_0} (DF^{-T}(\xi, \zeta) \nabla U(\xi, \zeta)) \cdot (DF^{-T}(\xi, \zeta) \nabla \varphi(\xi, \zeta)) |det DF|(\xi, \zeta) d\Omega_0 = \sum_{E_k} K_{i_1j_1, i_2j_2 E_k} \quad (13)$$

and

$$b_{ij} = \int_{\Omega_0} (f \varphi_{ij})(F(\xi, \zeta)) |det DF| d\Omega_0 = \sum_{E_k} b_{ij, E_k} \quad (14)$$

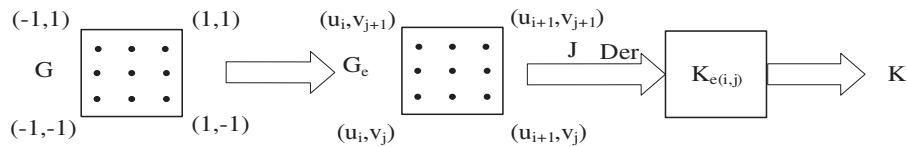
where  $E_k = [u_{k_1}, u_{k_1+1}] \times [v_{k_2}, v_{k_2+1}]$  is a Nurbs parameter space,

and

$$b_{ij, E_k} = \int_{E_k} (f \varphi_{ij})(F(\xi, \zeta)) |det DF| dE_k \quad (15)$$

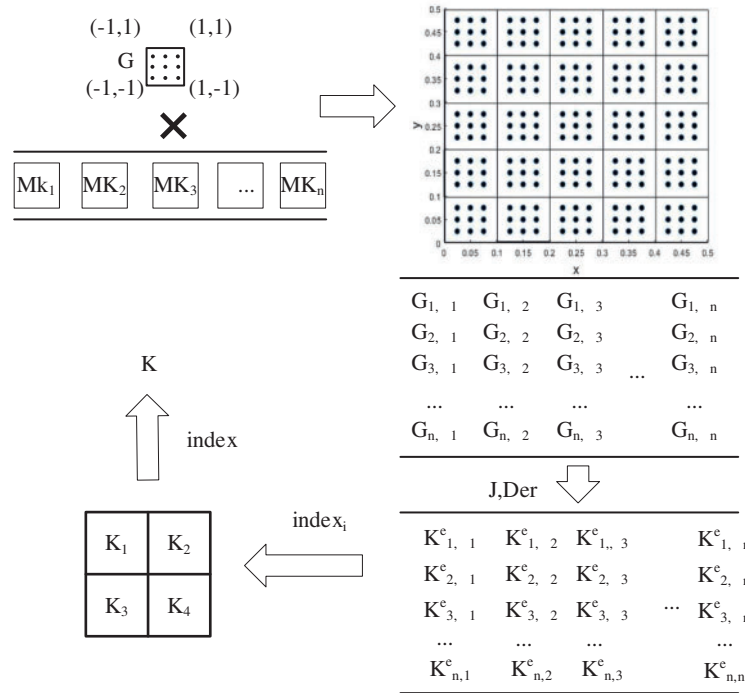
### 3 Process Refactoring Based on Parallel

The traditional solution process of IGA divides the calculation of Gaussian integral points into elements (see in Fig. 1), which is not suitable for parallel calculation. There are multi-layer cycles in the calculation of elements, which has a great impact on the calculation efficiency. In this chapter, the traditional IGA process is broken, the stiffness matrix is assembled into non-zero elements concentrated in the form of diagonal to accelerate the IGA solution process.



**Figure 1:** The traditional stiffness matrix solution process

Not all NURBS models are highly continuous, in order not to affect the accuracy of the experiment and increase the universality of the experiment, this paper does not adopt any rule for the time being. The chart of the algorithm in this paper is shown in Fig. 2.

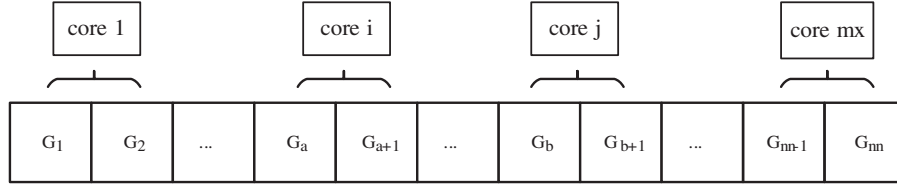


**Figure 2:** Parallel algorithm flow chart

The segmentation proposed in this paper is based on the node vector features and kernel number of each direction of the model. In order to facilitate the subsequent calculation, the amount of computation for each kernel should be roughly the same. The first thing to consider is whether the number of node vector segments ( $npart_u$ ,  $npart_v$ , and  $npart_w$ ) in each direction can be divisible by the kernel number  $mx$ . If it can be divisible, the number of segment segments in each direction is the kernel number  $mx$ . If it cannot be divisible, the number of segments is considered to have a common factor with the kernel number, and the kernel number is factorized. The number of segments in each direction is the common factor obtained by factorization. If it is not divisible and has no common factors, the element traversal is carried out in the directions of  $u$ ,  $v$  and  $w$ , and each unit traversal is assigned to a core, so as to ensure that the calculation amount of each core is consistent. Each processor's assigned cell is stored in a cell called *gelement*.

### 3.1 Parallel Calculation of the Derivative of Gaussian Integral Points

At first, the number of Gaussian integral points needed to be calculated for each kernel should be allocated. The Gaussian integral points should be divided into  $mx$  (Number of cores) regions of roughly the same size based on the number of cores in the computer.



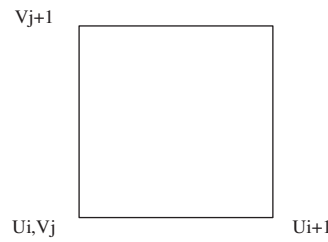
**Figure 3:** Distribution of computing

According to the properties of NURBS, it is more efficient to calculate the values and derivatives of all non-zero basis functions at a point. It can be found that the calculation process at different points only depends on the interval value of NURBS. Because the interval values of NURBS are fixed, the calculations at different points are independent. Moreover, the number of Gaussian points in each non-zero computation domain is the same, and the computation amount of each Gaussian point is the same. The traditional IGA calculation of Gaussian integral points is carried out in elements, which cannot be carried out in parallel due to the inclusion of many other parameters in element calculation. Therefore, we separate the solution of Gaussian integral points from the calculation steps of element stiffness matrix, so that the calculation can be carried out in parallel.

The set of Gaussian points obtained from the above is stored in an element called rule, when calculating the Gaussian integral derivative points need to be in the element coordinates and weight projection to the corresponding physical element domain, then be able to establish a matrix  $Gauss_{nquad \times 3}$  to store the data and the coordinates of the Gaussian integral points as the result of the Gaussian integral points.  $Gauss_{nquad \times 3}$  includes two directions of information  $u$ ,  $v$  and the weight  $W$ ,

$$Gauss_{nquad \times 3} = [u \ v \ W] \quad (16)$$

The Gaussian integral point can be mapped to an element in the form of matrix.  $GP$  can be obtained by multiplying the value of  $[-1, 1]$  by the element mapping matrix  $R$ , and the weight can be updated and stored in  $GP$ . For the below element, its mapping region in the  $u$  direction  $ulen_i = u_{i+1} - u_i$ ,  $v$  direction  $vlen_j = v_{j+1} - v_j$ .



**Figure 4:** Element area

When the Gaussian integral point is mapped to the whole world, the principle of graphics can be adopted to stretch and translate the low dimension by using the high-dimensional matrix, so the mapping matrix of the original Gaussian integral point is

$$MK_{i,j} = \begin{bmatrix} (u_{i+1} - u_i)/2 & 0 & 0 & 0 \\ 0 & (v_{j+1} - v_j)/2 & 0 & 0 \\ 0 & 0 & (u_{i+1} - u_i) * (v_{j+1} - v_j)/4 & 0 \\ (u_{i+1} + u_i)/2 & (v_{j+1} + v_j)/2 & (w_{k+1} + w_k)/2 & 1 \end{bmatrix} \quad (17)$$

The same is true for 3D models,

$$MK_{i,j,k} = \begin{bmatrix} (u_{i+1} - u_i)/2 & 0 & 0 & 0 & 0 \\ 0 & (v_{j+1} - v_j)/2 & 0 & 0 & 0 \\ 0 & 0 & (w_{k+1} - w_k)/2 & 0 & 0 \\ 0 & 0 & 0 & (u_{i+1} - u_i) * (v_{j+1} - v_j) * (w_{k+1} - w_k)/8 & 0 \\ (u_{i+1} + u_i)/2 & (v_{j+1} + v_j)/2 & (w_{k+1} + w_k)/2 & 0 & 1 \end{bmatrix} \quad (18)$$

The information of Gaussian integral points  $GP_{n \times n_{quad} \times 4}$  can be obtained by mapping Gaussian integral points to elements through mapping matrix.

---

**Algorithm 1:** Stores Gaussian points data

---

**Input:** Gauss<sub>n<sub>quad</sub>×4</sub> Q<sub>n</sub>  
**Output:** GP  
ijk=1  
One=ones(nquad,1)  
Parfor n=1:mx  
**for** i=1:size(gelement{n},1) // Iterate over each element  
K=[Gauss1(:,1) Gauss1(:,2) Gauss1(:,3) Gauss1(:,4) OneA]MK<sub>i</sub>  
GP{n}(ijk:ijk+nnquad-1,1:4)=K(:,1:4)  
ijk=ijk+1  
**End**

---

After the corresponding data is stored, Gaussian integral point derivative calculation is carried out. At this time, different processors need to be allocated to calculate the values of Gaussian integral points of different blocks. Assuming that the computer has  $mx$  processors, the number of threads is  $m$ , and each processor computes a quarter of the total number of nodes,  $nn/mx$ . The scale of the derivative of each point in each direction is the product of the order in each direction,

$$nnel = order_u * order_v * order_w \quad (19)$$

Algorithm 2 is the flow that each processor executes.

---

**Algorithm 2:** Compute all values and derivatives at Gaussian points

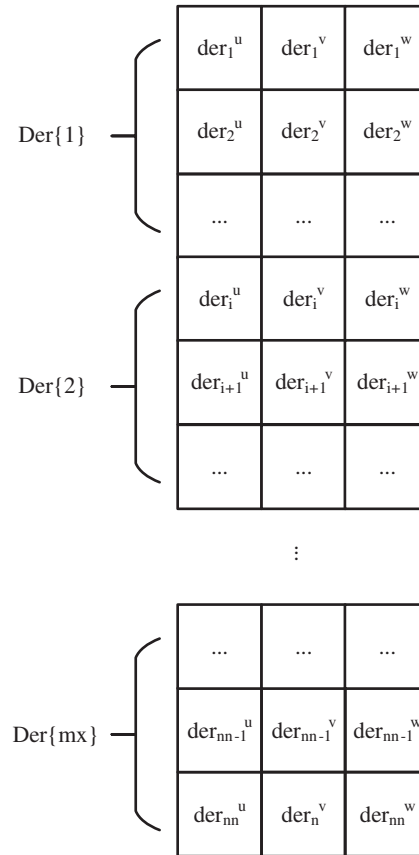
---

**Input:** mx,nnel,GP  
**Output:** Der  
parfor n=1:mx  
**for** i=1:size(GP{n},1)  
get deru, derv, derw at i;  
Der{n}((i-1)\*nnel+1:i\*nnel,1)=deru;  
Der{n}((i-1)\*nnel+1:i\*nnel,2)=derv;  
Der{n}((i-1)\*nnel+1:i\*nnel,3)=derw;  
**end**  
**end**

---

This paper combines the derivative of Gaussian integral points with parallel processing, and uses multi-threads to carry out parallel processing by using the characteristics of multi-core computer. The derivative of Gaussian integral points obtained by each thread is a vector product, the integral is stored as follows:

$$der = \begin{bmatrix} Der\{1\} \\ Der\{2\} \\ \vdots \\ Der\{mx\} \end{bmatrix} \quad (20)$$



**Figure 5: Derivative store**

### 3.2 Parallel of Stiffness Matrix Based on Intermediate Layer

Because model by blocking rules is divided into nuclear number block at first, each processor has stored the relevant elements of the information. Each element can be found in the location of the global and numbered, so local stiffness matrix can be assembled by the local index. The global stiffness matrix can also be assembled by the local stiffness matrix in this way.



**Algorithm 3:** Model chunking strategy

---

```

Input: nurbsdata
Output:  $h_1 h_2 \dots h_{A*B*C}$ 
Ascending order of  $u v w$ 
find  $\max\{l,m,n\}$ 
if the number of node vector segments is divisible by the number of cores
get  $u_1 \sim u_N$ 
else
    if the segment number and kernel number of node vector have common factors
        get A
         $N = A*B*C$ 
        get B and C
        get  $u_1 \sim u_A, v_1 \sim v_B, w_1 \sim w_C$ 
    else
        for  $i = 1:\text{totalelement}$ 
             $\text{element}_i \in \text{core}_n$ 
             $n = n+1$ 
            if  $n > mx$ 
                 $n = n-mx$ 
        end
    end
end

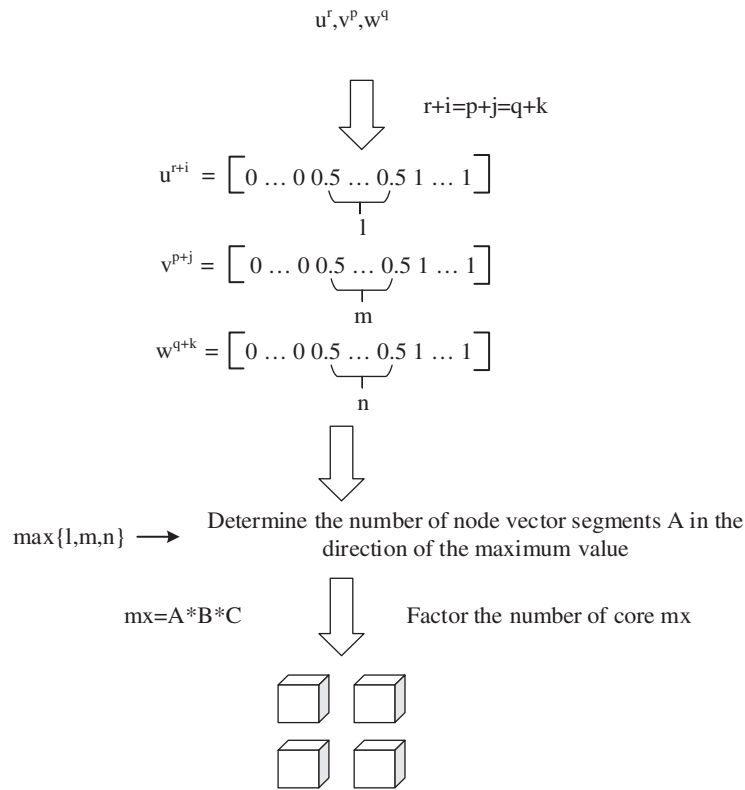
```

---

As is shown in [Fig. 6](#), in order to reduce the overlap degree of the boundary of each region after partitioning, we should first consider the order of each node vector. The smaller the order is, the worse the continuity is. In order to facilitate comparison, we can upgrade the three direction node vectors to the same order, and then calculate the number of specific gravity nodes. Then according to the number of segments of the node vector with multiple nodes, the number of segments in each direction is determined. To block the node vector is to find the point where the maximum degree of overlap is located on the node vector. That is to say, the nodes with high degree of overlap can be marked while the order is raised, and then regions can be partitioned according to the marked positions, so as to reduce the border overlap degree of each sub-region after partitioning to the greatest extent.

The larger the sub-region is, the closer the individual region is to the whole model; the smaller the sub-region is, the closer the individual region is to a single element. In this paper, the middle layer is adopted for block processing. Simple adjustment in the subsequent application can change the block strategy to make the block region larger or smaller, see in [Fig. 7](#).

Since the elements are numbered at the very beginning, a region index can be obtained according to the initial number when calculating regional stiffness, and regional stiffness can be assembled according to the region index, and then the obtained regional stiffness matrix is assembled into the overall stiffness matrix.



**Figure 6:** Decomposition strategy when the segment number and kernel number of node vector have common factors

When assembling each zone stiffness matrix, the Jacobian matrix is obtained by taking derivatives which greatly increased the workload, making computing time extended.

It can be known from Eq. (7) that the NURBS basis function is obtained by multiplying the derivative of NURBS parameter points with the coordinates of control points

$$X = F(u, v) = \sum_{i=1}^{n_i} \sum_{j=1}^{n_j} R_{ij}(u, v) C_{ij}$$

and the Jacobian is

$$|det DF(\xi, \zeta)| = \begin{vmatrix} \partial F_1 / \partial u & \partial F_1 / \partial v \\ \partial F_2 / \partial u & \partial F_2 / \partial v \end{vmatrix}$$

	x			y			z					
Element 1	$x_1^1$	$x_2^1$		$x_p^1$	$y_1^1$	$y_2^1$		$y_p^1$	$z_1^1$	$z_2^1$		$z_p^1$
Element n	$x_1^n$	$x_2^n$		$x_p^n$	$y_1^n$	$y_2^n$		$y_p^n$	$z_1^n$	$z_2^n$		$z_p^n$

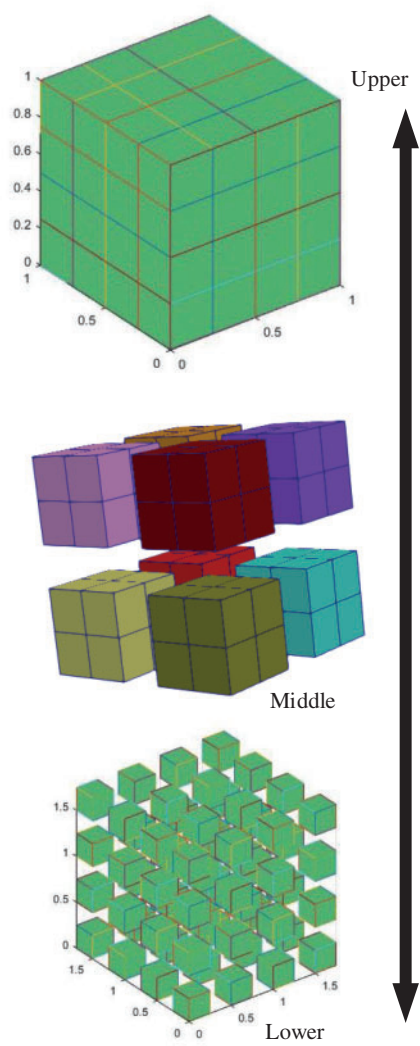


Figure 7: Model block

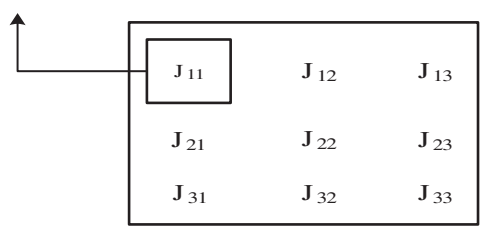


Figure 8: Calculation of jacobian

$$\begin{matrix} J_{11}^1 & x_1^1 & x_2^1 & x_3^1 & & x_p^1 \\ J_{11}^2 & x_1^2 & x_2^2 & x_3^2 & & x_p^2 \\ \dots & \dots & \dots & \dots & & \dots \\ J_{11}^n & x_1^n & x_2^n & x_3^n & & x_p^n \end{matrix} = \begin{matrix} \frac{\partial \varphi_1}{\partial u} & \frac{\partial \varphi_2}{\partial u} & \frac{\partial \varphi_3}{\partial u} & \dots & \frac{\partial \varphi_p}{\partial u} \end{matrix}$$

Substituting the Jacobian matrix into Eqs. (2)–(8) through the above equation, the Jacobian matrix can be written in the form of derivative and coordinate multiplication. In this way, the derivation process of the Jacobian matrix is avoided, and the calculation consumption in each traversal element is reduced, so that the calculation time is greatly reduced.

Algorithm 4 is to seek the overall stiffness matrix program.

---

**Algorithm 4:** Compute the local stiffness matrix

---

**Input:** Der, GP

**Output:**  $K_i$

```

for each subdomain
for ii = 1:nn //Iterate over each Gaussian integral point for subdomain
    rejac = coordinate*der(j:k,1:3)
    detjacob = det(rejac); //determinant of Jacobian
    invjacob=rejac-1; //inverse of Jacobian matrix
    Derivatives u ,v ,w physical coordinate [dhdx,dhdy,dhdz]
    Get ek
    Assembly local stiffness matrix with local index
    Get  $K_i$ 
end

```

---

Experiments show that the calculated stiffness matrix may exist bandwidth is larger, the overall stiffness matrix is discrete, calculating is not convenient to follow-up the right item, it will need to get the stiffness matrix of the matrix transformation, the elements, to funnel non-zero element stiffness matrix of the diagonal, enables the calculation to saves time.

Regional stiffness matrix assembly was conducted by regional index, by the middle tier block fixing the control points of each element, determine its position in the regional block, then the element stiffness matrix according to fill in the corresponding position stiffness matrix, finally will regional stiffness matrix through the global index assembly into the whole stiffness matrix. The assembly process is as follows:

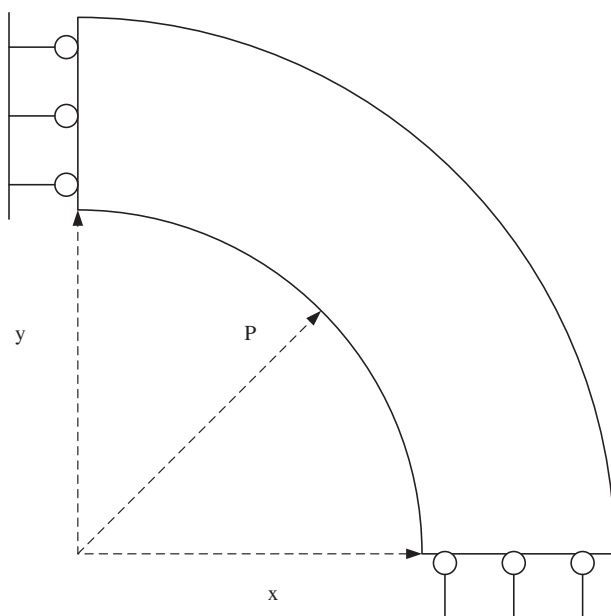
$$K = K_a^e * index_a + K_b^e * index_b + \dots K_n^e * index_n$$

## 4 Analysis of Experimental Results

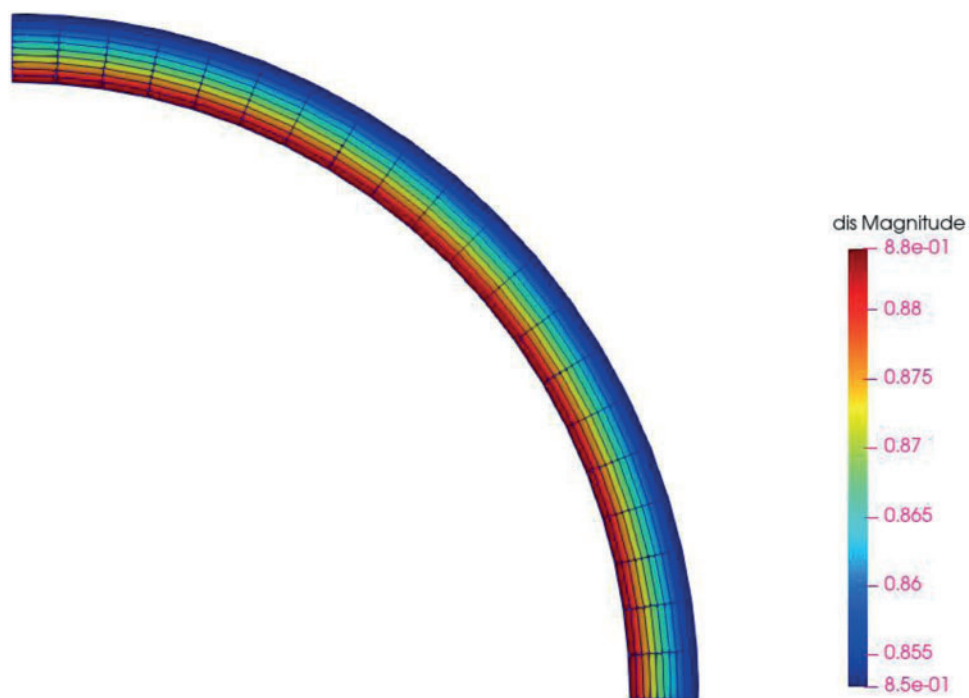
In this chapter, the algorithm is programed in Matlab2018b, and four different tests are carried out on one PC with i7-6700HQ (4 cores) and 16 GB memory.

### 4.1 2D Example

It is assumed that the inner wall of the thick-walled cylinder (inner diameter  $R_i$ , outer diameter  $R_o$ ) is subjected to uniform radial pressure  $P$ , and the upper and lower bottom surfaces are axially fixed. Due to its symmetry, this paper simplifies the model to 1/4.



**Figure 9:** The problem of uniform internal pressure in thick-walled cylinder



**Figure 10:** Ring calculation result

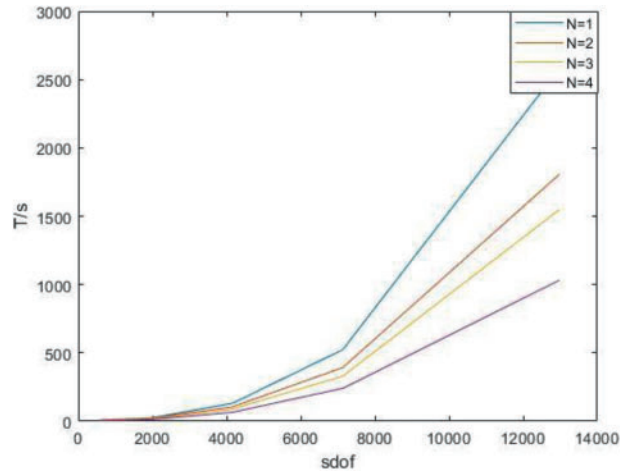
In order to compare the results under different degrees of freedom, P subdivision and H subdivision are used to process the original model. In this chapter, only the derivation of the parallel Gaussian integral point and only parallel stiffness matrix assembly are compared with the parallelism

of the whole process to show the effect brought by the parallelism of each process. The degree of freedom is controlled by inserting different number of nodes into the original node vector, 21 nodes are inserted initially.

The derivative time of the parallel Gaussian integral point is shown as [Table 1](#).

**Table 1:** Time of derivatives of parallel Gaussian integral points only under H subdivision

vector	node	sdof	1	2	3	4
$\xi_0$	21	598	3.91 s	10.46 s	7.35 s	5.11 s
$\xi_1$	41	1978	24.92 s	22.31 s	18.44 s	13.52 s
$\xi_2$	61	4158	131.23 s	104.98 s	89.24 s	65.61 s
$\xi_3$	81	7138	524.39 s	393.29 s	330.37 s	241.22 s
$\xi_4$	101	12978	2581.57 s	1807.11 s	1548.94 s	1032.63 s

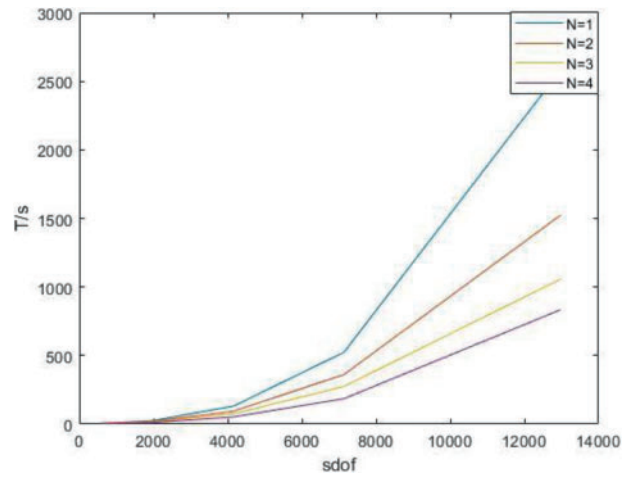


**Figure 11:** Time of derivatives of parallel Gaussian integral points only under H subdivision

Time of parallel stiffness matrix assembly under H subdivision is shown as [Table 2](#).

**Table 2:** Time of parallel stiffness matrix assembly under H subdivision

vector	node	sdof	1	2	3	4
$\xi_0$	21	598	3.91 s	8.61 s	6.37 s	5.11 s
$\xi_1$	41	1978	24.92 s	20.23 s	16.89 s	11.62 s
$\xi_2$	61	4158	131.23 s	93.57 s	75.11 s	52.36 s
$\xi_3$	81	7138	524.39 s	361.92 s	274.85 s	185.83 s
$\xi_4$	101	12978	2581.57 s	1524.68 s	1057.93 s	834.52 s

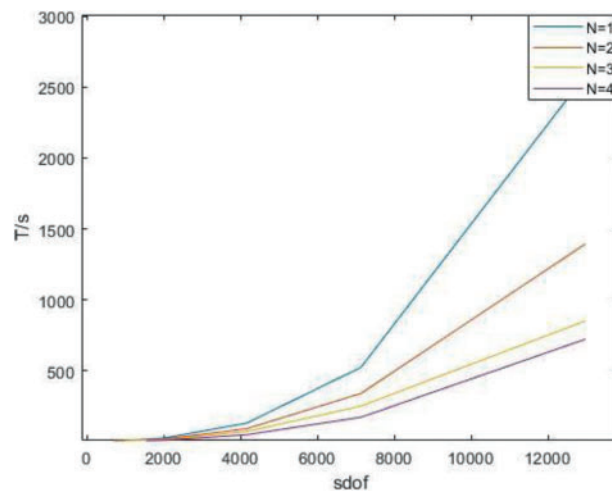


**Figure 12:** Time of parallel stiffness matrix assembly under H subdivision

Time of the whole process in parallel under H subdivision is shown as [Table 3](#).

**Table 3:** Time of the whole process in parallel under H subdivision

vector	node	sdof	1	2	3	4
$\xi_0$	21	598	3.91 s	7.52 s	5.68 s	4.21 s
$\xi_1$	41	1978	24.92 s	18.66 s	15.17 s	10.34 s
$\xi_2$	61	4158	131.23 s	91.86 s	72.18 s	49.87 s
$\xi_3$	81	7138	524.39 s	340.85 s	251.71 s	173.05 s
$\xi_4$	101	12978	2581.57 s	1394.04 s	851.92 s	722.93 s



**Figure 13:** Time of the whole process in parallel under H subdivision

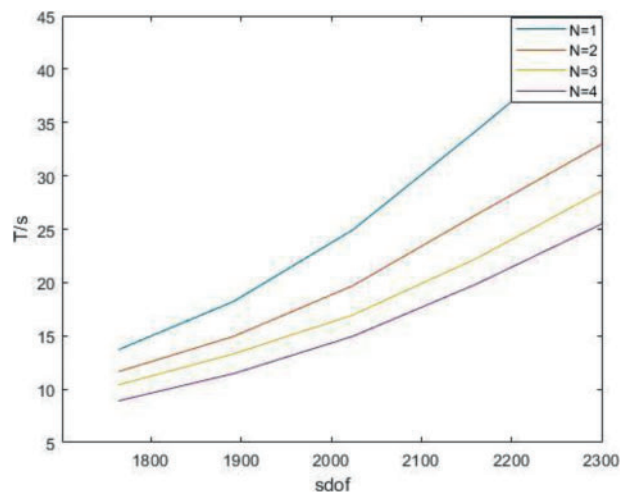
As can be seen from the above chart, the advantages of the parallel algorithm are not obvious when the degree of freedom is low, because it takes time to start the parallel algorithm. With the

increase of the degree of freedom, the advantages of the parallel algorithm gradually manifest, and the acceleration ratio tends to the number of processors.

Comparison of the two algorithms under different orders is made by statistics, the number of nodes inserted into the initial node vector is 41, and the order is 3, 2 as shown in [Table 4](#).

**Table 4:** Time of derivatives of parallel Gaussian integral points with different orders under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	3,2	1764	13.7 s	11.65 s	10.41 s	8.91 s
$\xi_1$	4,3	1892	18.24 s	14.96 s	13.32 s	11.47 s
$\xi_2$	5,4	2024	24.94 s	19.7 s	16.96 s	14.96 s
$\xi_3$	6,5	2160	34.17 s	26.31 s	22.21 s	19.82 s
$\xi_4$	7,6	2300	43.95 s	32.96 s	28.57 s	25.49 s



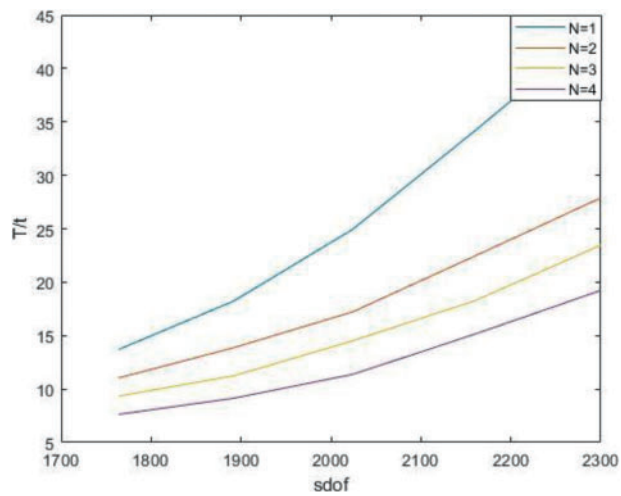
**Figure 14:** Time of derivatives of parallel Gaussian integral points with different orders under P subdivision

Time of parallel stiffness matrix assembly under P subdivision is shown as [Table 5](#).

**Table 5:** Time of parallel stiffness matrix assembly under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	3,2	1764	13.7 s	11.04 s	9.34 s	7.63 s
$\xi_1$	4,3	1892	18.24 s	13.87 s	11.24 s	9.14 s
$\xi_2$	5,4	2024	24.94 s	17.22 s	14.51 s	11.37 s
$\xi_3$	6,5	2160	34.17 s	22.45 s	18.27 s	15.14 s
$\xi_4$	7,6	2300	43.95 s	27.89 s	23.47 s	19.23 s



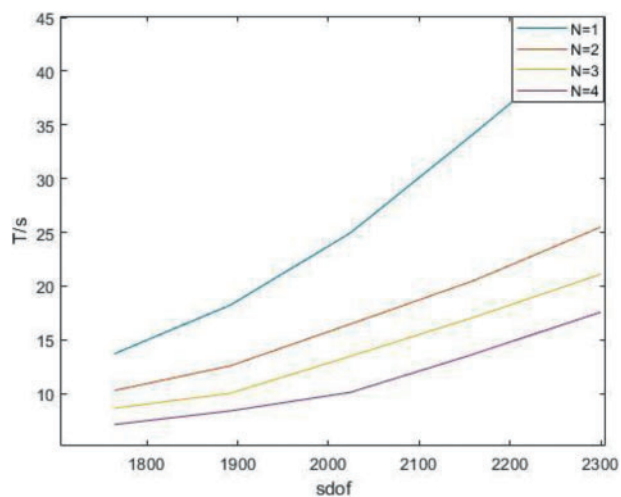


**Figure 15:** Time of parallel stiffness matrix assembly under P subdivision

Time of the whole process in parallel with different degree under P subdivision is shown as [Table 6](#).

**Table 6:** Time of the whole process in parallel with different degree under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	3,2	1764	13.7 s	10.28 s	8.63 s	7.12 s
$\xi_1$	4,3	1892	18.24 s	12.59 s	10.03 s	8.39 s
$\xi_2$	5,4	2024	24.94 s	16.46 s	13.47 s	10.11 s
$\xi_3$	6,5	2160	34.17 s	20.5 s	17.09 s	13.67 s
$\xi_4$	7,6	2300	43.95 s	25.49 s	21.1 s	17.58 s



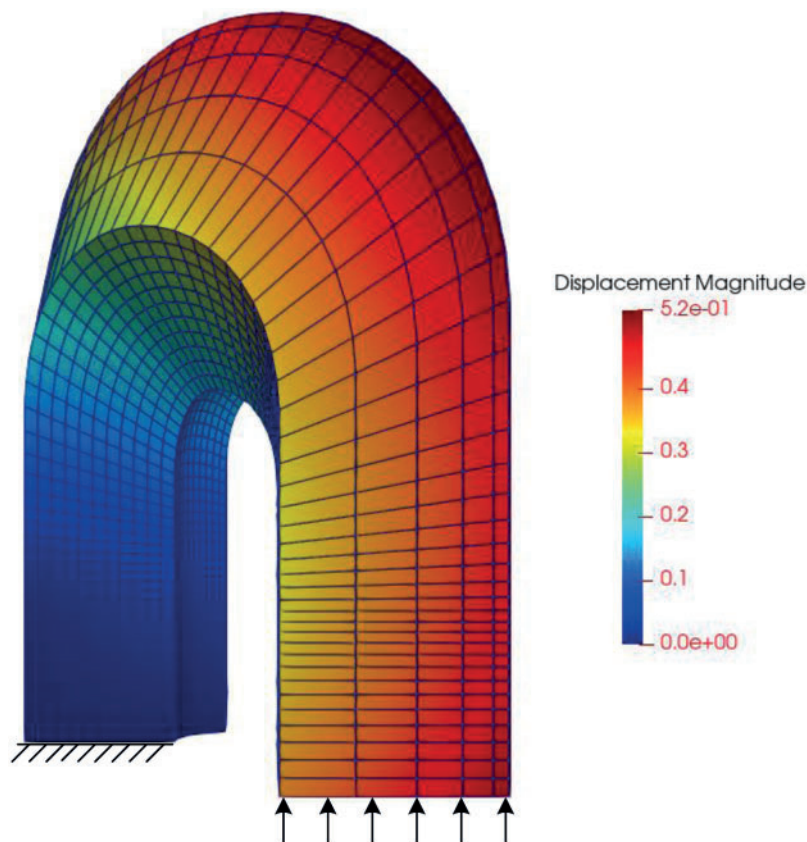
**Figure 16:** Time of the whole process in parallel with different degree under P subdivision

It can be clearly seen from the figure and table that as the order increases, the degree of freedom increases and the acceleration ratio approaches the kernel number.

After increasing the degree of freedom of the model, the two algorithms were compared, and it was found that whether the number of control points was increased by H subdivision or the order was increased by P subdivision, the parallel algorithm is always less time consuming than the traditional algorithm, the two algorithms are compared. It is found that the parallel algorithm takes less time than the traditional algorithm no matter the number of control points is increased by H subdivision or the order is increased by P subdivision. The calculation efficiency of stiffness matrix can be greatly improved by using multi-thread calculation.

#### 4.2 3D Example

Another example in this paper is the model of the shape of a horseshoe. A load is applied on one side to observe its stress and another side is fixed.

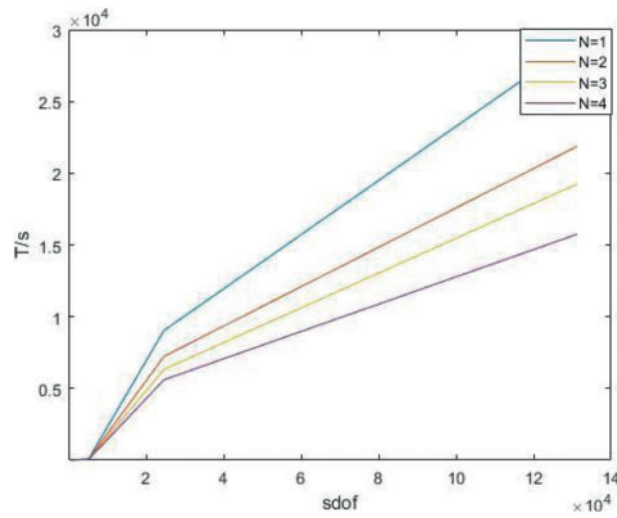


**Figure 17:** Displacement of horseshoe

The degree of freedom of the model is enhanced by H subdivision. In this example, the dichotomy method is adopted to insert nodes, shown as [Table 7](#).

**Table 7:** Time of derivatives of parallel Gaussian integral points only under H subdivision

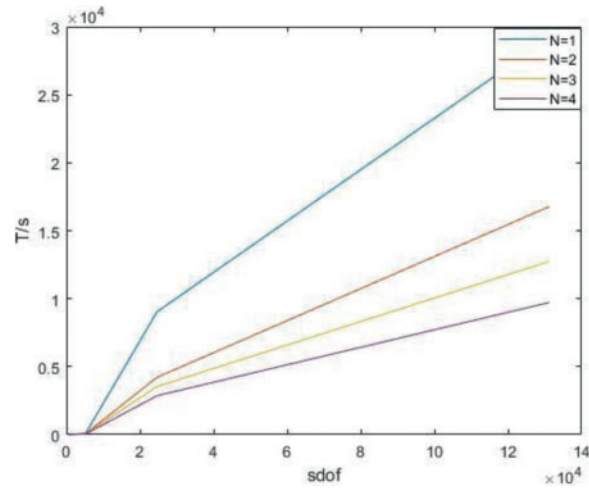
vector	node	sdof	1	2	3	4
$\xi_0$	[0.5]	600	1.82 s	5.62 s	4.67 s	3.29 s
$\xi_1$	[0.25 0.5 0.75]	1512	8.51 s	7.41 s	6.64 s	5.79 s
$\xi_2$	[0.125:0.125:0.875]	5280	118.66 s	99.67 s	87.81 s	78.32 s
$\xi_3$	[0.0625:0.0625:0.9375]	24624	9034.86 s	7227.89 s	6324.4 s	5601.61 s
$\xi_4$	[0.03125:0.03125:0.9685]	131274	29172.53 s	21879.39 s	19253.87 s	15753.16 s

**Figure 18:** Time of derivatives of parallel Gaussian integral points only under H subdivision

Time of parallel stiffness matrix assembly under H subdivision is shown as [Table 8](#).

**Table 8:** Time of parallel stiffness matrix assembly under H subdivision

vector	node	sdof	1	2	3	4
$\xi_0$	[0.5]	600	1.82 s	4.99 s	3.87 s	2.63 s
$\xi_1$	[0.25 0.5 0.75]	1512	8.51 s	7.12 s	5.88 s	4.71 s
$\xi_2$	[0.125:0.125:0.875]	5280	118.66 s	86.43 s	76.94 s	57.29 s
$\xi_3$	[0.0625:0.0625:0.9375]	24624	9034.86 s	4215.62 s	3542.78 s	2864.38 s
$\xi_4$	[0.03125:0.03125:0.9685]	131274	29172.53 s	16784.23 s	12743.69 s	9724.93 s

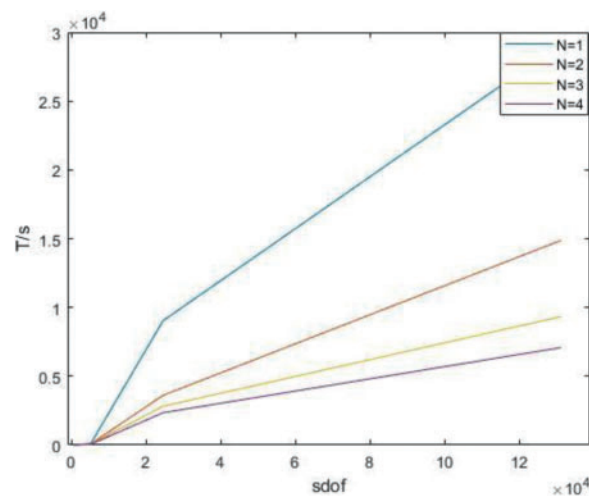


**Figure 19:** Time of parallel stiffness matrix assembly under H subdivision

Time of the whole process in parallel under H subdivision is shown as [Table 9](#).

**Table 9:** Time of the whole process in parallel under H subdivision

vector	node	sdof	1	2	3	4
$\xi_0$	[0.5]	600	1.82 s	4.35 s	3.27 s	2.04 s
$\xi_1$	[0.25 0.5 0.75]	1512	8.51 s	6.4 s	5.62 s	4.26 s
$\xi_2$	[0.125:0.125:0.875]	5280	118.66 s	83.06 s	72.38 s	55.77 s
$\xi_3$	[0.0625:0.0625:0.9375]	24624	9034.86 s	3613.94 s	2823.39 s	2349.06 s
$\xi_4$	[0.03125:0.03125:0.9685]	131274	29172.53 s	14877.72 s	9335.21 s	7084.63 s

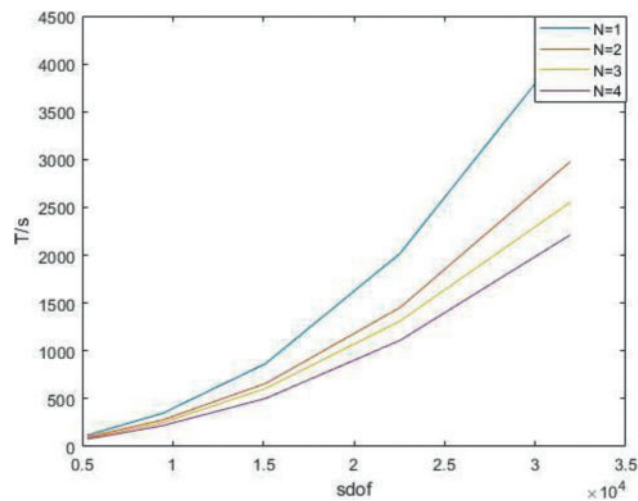


**Figure 20:** Time of the whole process in parallel under H subdivision

The degree of freedom of the model is enhanced by P subdivision,  $\xi = [0.125:0.125:0.875]$ , shown as [Table 10](#).

**Table 10:** Time of derivatives of parallel Gaussian integral points with different orders under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	[3,3,3]	5280	118.66 s	99.67 s	87.81 s	78.32 s
$\xi_1$	[4,4,4]	9438	348.29 s	278.63 s	250.77 s	219.42 s
$\xi_2$	[5,5,5]	15120	867.63 s	659.4 s	607.34 s	503.11 s
$\xi_3$	[6,6,6]	22542	2018.46 s	1453.29 s	1311.99 s	1110.15 s
$\xi_4$	[7,7,7]	31920	4251.39 s	2975.97 s	2550.83 s	2210.72 s

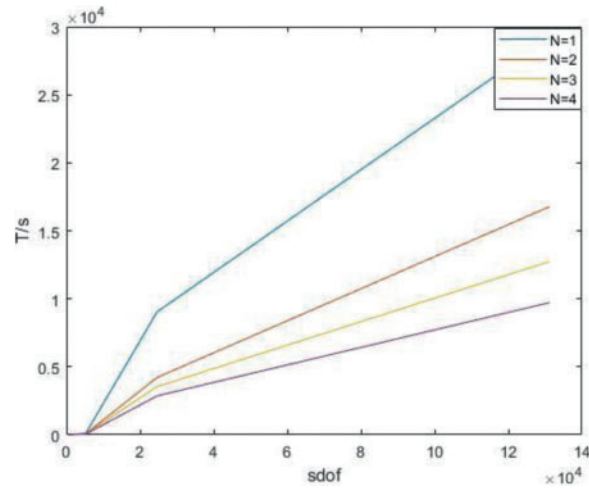


**Figure 21:** Time of derivatives of parallel Gaussian integral points with different orders under P subdivision

Time of parallel stiffness matrix assembly under P subdivision is shown as [Table 11](#).

**Table 11:** Time of parallel stiffness matrix assembly under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	[3,3,3]	5280	118.66 s	85.33 s	74.34 s	57.24 s
$\xi_1$	[4,4,4]	9438	348.29 s	251.24 s	196.48 s	152.33 s
$\xi_2$	[5,5,5]	15120	867.63 s	587.52 s	448.91 s	320.41 s
$\xi_3$	[6,6,6]	22542	2018.46 s	1241.87 s	772.59 s	634.52 s
$\xi_4$	[7,7,7]	31920	4251.39 s	2375.17 s	1542.74 s	1374.36 s

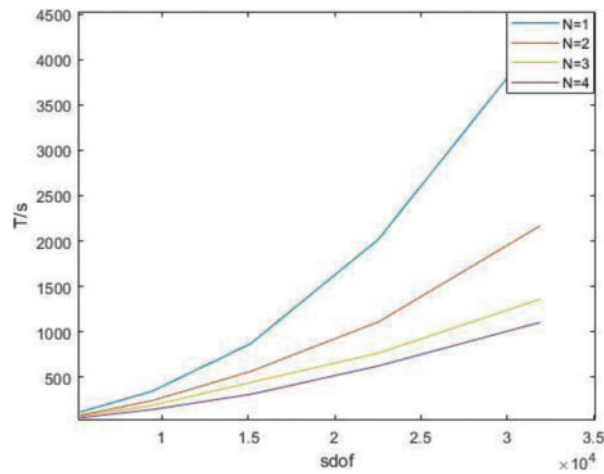


**Figure 22:** Time of parallel stiffness matrix assembly under P subdivision

Time of the whole process in parallel with different degree under P subdivision is shown as [Table 12](#).

**Table 12:** Time of the whole process in parallel with different degree under P subdivision

vector	order	sdof	1	2	3	4
$\xi_0$	[3,3,3]	5280	118.66 s	83.06 s	72.38 s	55.77 s
$\xi_1$	[4,4,4]	9438	348.29 s	243.8 s	191.56 s	146.28 s
$\xi_2$	[5,5,5]	15120	867.63 s	563.96 s	442.49 s	312.36 s
$\xi_3$	[6,6,6]	22542	2018.46 s	1110.15 s	767.07 s	625.76 s
$\xi_4$	[7,7,7]	31920	4251.39 s	2168.21 s	1360.45 s	1105.36 s



**Figure 23:** Time of the whole process in parallel with different degree under P subdivision

It can be seen from the above results that with the increase of degrees of freedom, the advantages of the parallel algorithm become more obvious, and when the degrees of freedom are similar, the advantages of the parallel algorithm become greater at high orders. It is found that the stiffness matrix takes up a large proportion of the calculation time in IGA.

## 5 Conclusion

In this paper, the solution of IGA is accelerated from three aspects. In the first step, the traditional isogeometric solution process is broken, and the Gaussian integral points are mapped to the global model through matrix by using the characteristics of model parameter elements. In the second step, the parallel algorithm PARFOR is used to calculate the derivatives of Gaussian integral points when calculating the derivative of Gaussian integral point, and the derivatives obtained by each kernel are stored in a single matrix. Thirdly, in the assembly process of the matrix, the model is divided into blocks according to the number of the cores and node vectors in each direction, and then different cores are assigned to each region for parallel calculation. Finally, the regional stiffness matrix is assembled by regional index, and the regional stiffness matrix is assembled by global index.

For the future work, it is necessary to focus on the reduction of Gaussian integral points combined with the existing parallel algorithm to further reduce the solution scale in the process of IGA calculation. In the solving process, the stiffness matrix can be divided into blocks and solved in parallel to further improve the solving efficiency.

**Funding Statement:** This research was supported by the Natural Science Foundation of Hubei Province (CN) (Grant No. 2019CFB693), the Research Foundation of the Education Department of Hubei Province (CN) (Grant No. B2019003) and the open Foundation of the Key Laboratory of Metallurgical Equipment and Control of Education Ministry (CN) (Grant No. 2015B14). The support is gratefully acknowledged.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the present study.

## References

1. Thompson, J. F., Thames, F. C., Mastin, C. W. (1974). Automatic numerical generation of body-fitted curvilinear coordinate system for field containing any number of arbitrary two-dimensional bodies. *Journal of Computational Physics*, 15(3), 299–319. DOI 10.1016/0021-9991(74)90114-4.
2. Thompson, J. F., Warsi, Z., Mastin, C. W. (1982). Boundary-fitted coordinate systems for numerical solution of partial differential equations—A review. *Journal of Computational Physics*, 47(1), 1–108. DOI 10.1016/0021-9991(82)90066-3.
3. Thacker, W. C., Gonzalez, A., Putland, G. E. (1980). A method for automating the construction of irregular computational grids for storm surge forecast models. *Journal of Computational Physics*, 37(3), 371–387. DOI 10.1016/0021-9991(80)90043-1.
4. Hughes, T., Cottrell, J. A., Bazilevs, Y. (2005). Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics Engineering*, 194(1), 39–41, 4135–4195. DOI 10.1016/j.cma.2004.10.008.
5. Wang, L., Hou, S., Shi, L. (2019). A simple FEM for solving two-dimensional diffusion equation with nonlinear interface jump conditions. *Computer Modeling in Engineering & Sciences*, 119(1), 73–90. DOI 10.32604/cmescs.2019.04581.

6. Velho, L., Zorin, D. (2001). 4-8 subdivision. *Computer Aided Geometric Design*, 18(5), 397–427. DOI 10.1016/S0167-8396(01)00039-5.
7. Deng, C., Weiyin, M. A. (2013). A unified interpolatory subdivision scheme for quadrilateral meshes. *ACM Transactions on Graphics*, 32(3), 1–11. DOI 10.1145/2487228.2487231.
8. Lions, J. L., Maday, Y., Turinici, G. (2001). A “parareal” in time discretization of PDE’s. *Comptes Rendus de l’Académie des Sciences-Series I-Mathematics*, 332(7), 661–668. DOI 10.1016/S0764-4442(00)01793-6.
9. Zhang, J., Shen, D. (2014). GPU-Based implementation of finite element method for elasticity using CUDA. *IEEE International Conference on High Performance Computing & Communications & IEEE International Conference on Embedded & Ubiquitous Computing, Marriott Vancouver Pinnacle Hotel and the Renaissance Vancouver Harbourside Hotel* 1128 West Hastings Street Vancouver, vol. 2014, pp. 1003–1008. BC, Canada.
10. Wozniak, M. (2015). Fast GPU integration algorithm for isogeometric finite element method solvers using task dependency graphs. *Journal of Computational Science*, 11(2), 145–152. DOI 10.1016/j.jocs.2015.02.007.
11. Karatarakis, A., Karakitsios, P., Papadrakakis, M. (2014). GPU accelerated computation of the isogeometric analysis stiffness matrix. *Computer Methods in Applied Mechanics Engineering*, 269, 334–355. DOI 10.1016/j.cma.2013.11.008.
12. Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., Liu, J. W. (1999). A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3), 720–755. DOI 10.1137/S0895479895291765.
13. Liu, J. (1992). The multifrontal method for sparse matrix solution: Theory and practice. *Siam Review*, 34(1), 82–109. DOI 10.1137/1034004.
14. Sintorn, E., Assarsson, U. (2008). Fast parallel GPU-sorting using a hybrid algorithm. *Journal of Parallel Distributed Computing*, 68(10), 1381–1388. DOI 10.1016/j.jpdc.2008.05.012.
15. Shams, R., Sadeghi, P., Kennedy, R. A., Hartley, R. I. (2010). A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine*, 27(2), 50–60. DOI 10.1109/MSP.2009.935387.
16. Cuvelier, F., Japhet, C., Scarella, G. (2015). An efficient way to assemble finite element matrices in vector languages. *BIT Numerical Mathematics*, 56(3), 833–864. DOI 10.1007/s10543-015-0587-4.
17. Hughes, T., Levit, I., Winget, J. (1983). An element-by-element solution algorithm for problems of structural and solid mechanics. *Computer Methods in Applied Mechanics Engineering*, 36(2), 241–254. DOI 10.1016/0045-7825(83)90115-9.
18. Yagawa, G. (2004). Node-by-node parallel finite elements: A virtually meshless method. *International Journal for Numerical Methods in Engineering*, 60(1), 69–102. DOI 10.1002/nme.955.
19. Fujisawa, T., Inaba, M., Yagawa, G. (2003). Parallel computing of high-speed compressible flows using a node-based finite-element method. *International Journal for Numerical Methods in Engineering*, 58(3), 481–511. DOI 10.1002/(ISSN)1097-0207.
20. Smith, G. D. (1965). *Numerical solution of partial differential equations: With exercises and worked solutions*. Oxford University Press.
21. Clough, R. W. (1960). The finite element method in plane stress analysis. *ASCE Conference on Electronic Computation*, America.
22. Bazilevs, Y., Calo, V. M., Cottrell, J. A., Evans, J. A., Hughes, T. J. R. et al. (2015). Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics Engineering*, 199(5–8), 229–263. DOI 10.1016/j.cma.2009.02.036.
23. Scott, M. A., Borden, M. J., Verhoosel, C. V., Sederberg, T. W., Hughes, T. J. (2011). Isogeometric finite element data structures based on Bézier extraction of T-splines. *International Journal for Numerical Methods in Engineering*, 88(2), 126–156. DOI 10.1002/nme.3167.
24. Kiss, G., Giannelli, C., Zore, U., Jüttler, B., Großmann, D. (2014). Adaptive CAD model (re-)construction with THB-splines. *Graphical Models*, 76(1), 273–288. DOI 10.1016/j.gmod.2014.03.017.



25. Giannelli, C., Bertjuttler, Speleers, H. (2012). THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7), 485–498. DOI 10.1016/j.cagd.2012.03.025.
26. Kiss, G., Giannelli, C., Jüttler, B. (2021). *AlgoriThms and data structures for truncated hierarchical B-splines*. Berlin Heidelberg: Springer.
27. Bressan, A. (2013). Some properties of LR-splines. *Computer Aided Geometric Design*, 30(8), 778–794. DOI 10.1016/j.cagd.2013.06.004.
28. Dokken, T., Lyche, T., Pettersen, K. F. (2013). Polynomial splines over locally refined box-partitions. *Computer Aided Geometric Design*, 30(3), 331–356. DOI 10.1016/j.cagd.2012.12.005.
29. Deng, J., Chen, F., Feng, Y. (2006). Dimensions of spline spaces over T-meshes. *Journal of Computational and Applied Mathematics*, 194(2), 267–283. DOI 10.1016/j.cam.2005.07.009.
30. Scott, M. A., Li, X., Sederberg, T. W., Hughes, T. J. (2012). Local refinement of analysis-suitable T-splines. *Computer Methods in Applied Mechanics Engineering*, 213–216, 206–222. DOI 10.1016/j.cma.2011.11.022.
31. Beirao da Veiga, L., Buffa, A., Sangalli, G., Vázquez, R. (2013). Analysis-suitable T-splines of arbitrary degree: Definition, linear independence and approximation properties. *Mathematical Models Methods in Applied Sciences*, 23(11), 1979–2003. DOI 10.1142/S0218202513500231.
32. Deng, J., Chen, F., Li, X., Hu, C., Tong, W. et al. (2008). Polynomial splines over hierarchical T-meshes. *Graphical Models*, 70(1), 76–86. DOI 10.1016/j.gmod.2008.03.001.
33. Bartoň, M., Ait-Haddou, R., Calo, V. M. (2017). Gaussian quadrature rules for C1 quintic splines with uniform knot vectors. *Journal of Computational and Applied Mathematics*, 322(1), 57–70. DOI 10.1016/j.cam.2017.02.022.
34. Ait-Haddou, R., Bartoň, M., Calo, V. M. (2015). Explicit Gaussian quadrature rules for C1 cubic splines with symmetrically stretched knot sequences. *Journal of Computational and Applied Mathematics*, 290(1), 543–552. DOI 10.1016/j.cam.2015.06.008.
35. Bartoň, M., Calo, V. M. (2016). Gaussian quadrature for splines via homotopy continuation: Rules for C2 cubic splines. *Journal of Computational and Applied Mathematics*, 296(1), 709–723. DOI 10.1016/j.cam.2015.09.036.
36. Hughes, T. J. R., Reali, A., Sangalli, G. (2010). Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5–8), 301–313. DOI 10.1016/j.cma.2008.12.004.
37. Auricchio, F., Calabro, F., Hughes, T. J., Reali, A., Sangalli, G. (2012). A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 249–252(11), 15–27. DOI 10.1016/j.cma.2012.04.014.
38. Wu, Z., Wang, S., Shao, W., (2020). Reusing the evaluations of basis functions in the integration for isogeometric analysis. *Computer Modeling in Engineering & Sciences*, 122(2), 459–485. DOI 10.32604/cmescs.2020.08697.
39. Karatarakis, A., Metsis, P., Papadrakakis, M. (2013). GPU-acceleration of stiffness matrix calculation and efficient initialization of EFG meshless methods. *Computer Methods in Applied Mechanics and Engineering*, 258(1), 63–80. DOI 10.1016/j.cma.2013.02.011.
40. Xia, Z., Wang, Y., Wang, Q., Mei, C. (2017). GPU parallel strategy for parameterized LSM-based topology optimization using isogeometric analysis. *Structural and Multidisciplinary Optimization*, 56(2), 413–434. DOI 10.1007/s00158-017-1672-x.
41. Hofer, C., Langer, U., Neumüller, M. (2018). Parallel and robust preconditioning for space-time isogeometric analysis of parabolic evolution problems. *SIAM Journal on Scientific Computing*, 41(3), A1793–A1821. DOI 10.1137/18M1208794.
42. Stavroulakis, G., Tsapetis, D., Papadrakakis, M. (2018). Non-overlapping domain decomposition solution schemes for structural mechanics isogeometric analysis. *Computer Methods in Applied Mechanics Engineering*, 341(41–43), 695–717. DOI 10.1016/j.cma.2018.07.011.

43. Siwik, L., Woźniak, M., Trujillo, V., Pardo, D., Calo, V. M. et al. (2018). Parallel refined isogeometric analysis in 3D. *IEEE Transactions on Parallel Distributed Systems*, 30(5), 1134–1142. DOI 10.1109/TPDS.2018.2879664.
44. Piegl, L., Tiller, W. (1997). *The nurbs book*. Berlin, Germany: Springer-Verlag.