Front. Mech. Eng. 2025, 20(1): 5
https://doi.org/10.1007/s11465-025-0821-y

RESEARCH ARTICLE

Efficient blocked symmetric compressed sparse column method for finite element analysis

Yingjun WANG[a], Shijie LUO[a], Jinyu GU (✉)[a], Yuanfang ZHANG (✉)[b]

[a] *National Engineering Research Center of Novel Equipment for Polymer Processing, The Key Laboratory of Polymer Processing Engineering of the Ministry of Education, Guangdong Provincial Key Laboratory of Technique and Equipment for Macromolecular Advanced Manufacturing, South China University of Technology, Guangzhou 510641, China*
[b] *Shien-Ming Wu School of Intelligent Engineering, South China University of Technology, Guangzhou 511442, China*
✉ *Corresponding authors. Emails: jinyugu08@163.com (Jinyu GU); zhangyuanfang@scut.edu.cn (Yuanfang ZHANG)*

ABSTRACT In finite element analysis (FEA), optimizing the storage requirements of the global stiffness matrix and enhancing the computational efficiency of solving finite element equations are pivotal objectives. To address these goals, we present a novel method for compressing the storage of the global stiffness matrix, aimed at minimizing memory consumption and enhancing FEA efficiency. This method leverages the block symmetry of the global stiffness matrix, hence named the blocked symmetric compressed sparse column (BSCSC) method. We also detail the implementation scheme of the BSCSC method and the corresponding finite element equation solution method. This approach optimizes only the global stiffness matrix index, thereby reducing memory requirements without compromising FEA computational accuracy. We then demonstrate the efficiency and memory savings of the BSCSC method in FEA using 2D and 3D cantilever beams as examples. In addition, we employ the BSCSC method to an engine connecting rod model to showcase its superiority in solving complex engineering models. Furthermore, we extend the BSCSC method to isogeometric analysis and validate its scalability through two examples, achieving up to 66.13% memory reduction and up to 72.06% decrease in total computation time compared to the traditional compressed sparse column method.

KEYWORDS finite element analysis, global stiffness matrix, blocked symmetric property, memory reduction, isogeometric analysis

1 Introduction

Finite element analysis (FEA) is widely used in the field of science and engineering as a numerical simulation method applied across various domains such as aerospace [1,2], maritime engineering [3,4], automobile engineering [5,6], electronics [7], and civil engineering [8]. FEA breaks down complex structures or physical problems into a finite number of elements, modeling the interactions between these elements to represent complex systems or physical phenomena [9–11]. This method offers numerous advantages: it can handle irregular structures, accommodate various boundary and loading conditions, and effectively solve linear and nonlinear problems, as well as complex multiphysics problems [12–14].

Received May 21, 2024; accepted July 30, 2024; online February 20, 2025

Despite the significant advantages of FEA in simulating complex engineering problems, achieving accurate results still require solving large-scale finite element equations [15]. Thus, the efficiency of problem solving and the scale of addressable problems depend on how these equations are stored and solved. At present, three main strategies are used to manage large-scale FEA. The first approach involves using an adaptive mesh method, which refines the mesh in areas with complex boundaries or significant changes while employing coarser meshes in other regions to minimize the scale of finite element equations and computational costs [16,17]. Bellenger and Coorevits [18] proposed an alternative mesh refinement criterion, known as the h-adaptive procedure for 3D plastics problems, which achieves an optimally accurate grid based on preset parameters such as the number of elements, CPU time, and memory size. You et al. [19] introduced an adaptive mesh generation scheme for FEA of diverse heterogeneous materials. Their results show

that this method can significantly reduce mesh complexities and computational resources in heterogeneous objects, achieving substantial mesh reduction without compromising quality.

The second approach, which employs the assembly-free method, greatly reduces the storage requirements for the global stiffness matrix while ensuring solution accuracy. Yadav and Suresh [20] proposed the assembly-free deflated conjugate gradient method for large-scale FEA, which requires only 3 GB of memory to solve systems with degrees of freedom (DoFs) on a single GPU. Prabhune and Suresh [21] proposed an elastoplastic solver that eliminates the need for assembling stiffness matrices, addressing the high computational costs associated with residual stress and deformation prediction using selective laser melting.

The third approach, utilizing the compression storage method for the global stiffness matrix, minimizes storage space demands and enhances solving efficiency. Willcock and Lumsdaine [22] described two methods for accelerating the multiplication of a sparse matrix by compressing the matrix indices, thereby reducing memory bandwidth requirements during multiplication. They achieved compression ratios and speedups of up to 30% for various large sparse matrices. Chen et al. [23] proposed a cell sparse storage scheme that reduces memory requirements during the finite element equation-solving process and improves the speed of sparse matrix–vector multiplication (SpMV) through loop unrolling. Ribeiro and Ferreira [24] presented an MPI-based SBS parallel implementation of FEA for coarse-grain distributed memory architectures, using compressed data structures for storing stiffness matrices. Kawamura et al. [25] proposed an improved compressed sparse row (CSR) and ELLPACK (ELL) method, compressing continuous indices into two integers using maximum and minimum values, thereby reducing storage usage and access processes. When applied to the pwtk matrix, this method reduced storage space by 26.6% compared to traditional CSR. Ramírez-Gil et al. [26] proposed a CPU–GPU implementation that utilizes compressed sparse column (CSC) storage for the global stiffness matrix, achieving efficient assembly of stiffness matrices.

Although the methods described above can reduce the computational costs of FEA, they still have certain limitations. First, the adaptive grid method uses fine grids in areas with significant changes in the physical field and coarse grids in smoother areas, achieving a coupling between grid point distribution and the physical field to reduce computational costs [27]. However, it requires a considerable number of computational resources for grid generation and adjustment and may sacrifice some analytical accuracy in coarse grids. Second, the assembly-free method saves space by not explicitly storing the global stiffness matrix [28]. However, it inevitably prolongs the solution time; if the global stiffness matrix needs to be repeatedly used, the element stiffness matrix must be recalculated. The stiffness matrix compression storage method can achieve a balance between storage space and solution time, improving solution efficiency while saving storage space. However, current sparse matrix compression storage methods have not fully considered the relationship between the positions of nonzero values in the global stiffness matrix. Therefore, based on the nonzero values of the finite element stiffness matrix, we present a new blocked symmetric CSC (BSCSC) method for storing the global stiffness matrix. Compared to traditional CSC and CSR methods, the proposed method reduces memory requirements in FEA and achieves more efficient SpMV.

The organization of the following text is as follows. Section 2 introduces the basic theory of FEA and the assembly of the global stiffness matrix. Section 3 elaborates on the specific scheme of the BSCSC method and the corresponding algorithmic procedure. In Section 4, we employ a 2D FEA example to verify the performance of the BSCSC method in storing and accessing the global stiffness matrix. A 3D FEA example is also used to verify the feasibility and superiority of the algorithm in solving large-scale problems. Furthermore, we apply the presented method to isogeometric analysis (IGA) and validate the generality and superiority of the algorithm using a quadratic annulus and an L-shaped beam in Section 5. Finally, we conclude this study and outline some perspectives for future work in Section 6.

2 FEA

The storage of the global stiffness matrix is a crucial factor that affects the efficiency of FEA solutions [20]. To investigate methods for storing the global stiffness matrix, we describe the basic theory of FEA in solid mechanics and the process of assembling the global stiffness matrix in this section.

The expression for the static balance equation for the general conditions is given by [29]:

$$\boldsymbol{KU} = \boldsymbol{F}, \tag{1}$$

where $\boldsymbol{K}$ is the global stiffness matrix, which is obtained by assembling the element stiffness matrix $\boldsymbol{k}_e$. $\boldsymbol{F}$ and $\boldsymbol{U}$ denote the magnitude of the load and the displacement, respectively.

$$\boldsymbol{k}_e = \int_{\Omega} \boldsymbol{B}_e^{\mathrm{T}} \boldsymbol{D} \boldsymbol{B}_e \, \mathrm{d}\Omega, \tag{2}$$

where $\boldsymbol{B}_e$ is the strain displacement matrix of the element e, and $\boldsymbol{D}$ is the solid material elasticity matrix.

For a 2D problem, $\boldsymbol{D}$ is defined as follows [30]:

$$\boldsymbol{D} = \frac{E}{1-\upsilon^2} \begin{bmatrix} 1 & \upsilon & 0 \\ & 1 & 0 \\ \mathrm{Sym} & & \dfrac{1-\upsilon}{2} \end{bmatrix}, \tag{3}$$

where E is the elastic modulus of the material, and υ is the Poisson's ratio.

The strain matrix $\boldsymbol{B}_e$ of the element is

$$\boldsymbol{B}_e = \begin{bmatrix} \boldsymbol{B}_1 & \boldsymbol{B}_2 & \cdots & \boldsymbol{B}_i \end{bmatrix}, \tag{4}$$

where i is the number of nodes of the element e. $\boldsymbol{B}_i$ is the block submatrix of the strain matrix, whose expression is given below:

$$\boldsymbol{B}_i = \begin{bmatrix} \dfrac{\partial N_i}{\partial x} & 0 \\ 0 & \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial y} & \dfrac{\partial N_i}{\partial x} \end{bmatrix}, \tag{5}$$

where N_i denotes the shape function of the isoparametric element. In this section, we consider a four-node quadrilateral element as an example. Thus, the scale of the strain matrix $\boldsymbol{B}_e$ of a four-node element is 3 × 8. The scale of the element stiffness matrix $\boldsymbol{k}_e$ is 8 × 8.

We will illustrate the assembly process of the global stiffness matrix using a simple finite element mesh as an example. As shown in Fig. 1, the finite element mesh contains 4 elements, 9 nodes, and 18 DoFs. The numbers in the middle of the elements indicate the element numbering, the numbers near the nodes indicate the node numbering, and the numbers in parentheses indicate the DoF numbering. All numbering in this paper starts from 0. The rows and columns are also numbered from 0 when defining the numbering system. The scale of the global stiffness matrix is 18 × 18. Figure 1 shows that shared nodes (1 and 4) and shared DoFs (1, 10, 4, and 13) exist in the stiffness matrix of elements 0 and 1. The data for the repeated DoFs are assembled at the same locations in the global stiffness matrix. Nodes 0 and 2 do not belong to the same element, so the value at positions (0, 2) and (2, 0) in the global stiffness matrix is 0. This results in the inevitable presence of many zeros in the global stiffness matrix, making it a sparse matrix. Employing sparse matrix storage methods to store the global stiffness matrix can effectively save memory and improve efficiency.

3 Implementation of the BSCSC method

In this section, we elaborate on the implementation of the BSCSC method for storing the global stiffness matrix and the corresponding finite element solution algorithm. First, the foundational principles of the basic sparse matrix storage method are introduced in Section 3.1. Based on this, we elaborate on the detailed procedure of the BSCSC method in Section 3.2. In addition, we describe in detail the finite element equation solving algorithm based on BSCSC in Section 3.3.

3.1 Sparse matrix storage algorithm

In the solution of finite element equations, the global stiffness matrix contains far fewer nonzero values than zero values, and the zero values are not involved in arithmetic operations [31]. As a result, compressed storage of sparse matrices can effectively save memory space and improve computational efficiency. Typical matrix compression storage methods include the coordinate (COO) method, CSC, and CSR [32,33].

The COO method stores only the nonzero values along with their corresponding row and column indices. A schematic of this storage format is shown in Fig. 2. When using the COO method to store the global stiffness matrix, three vectors, ***val***, ***row_index***, and ***col_index***, need to be established. The length of each vector is of size N_z, where N_z denotes the number of nonzero values.

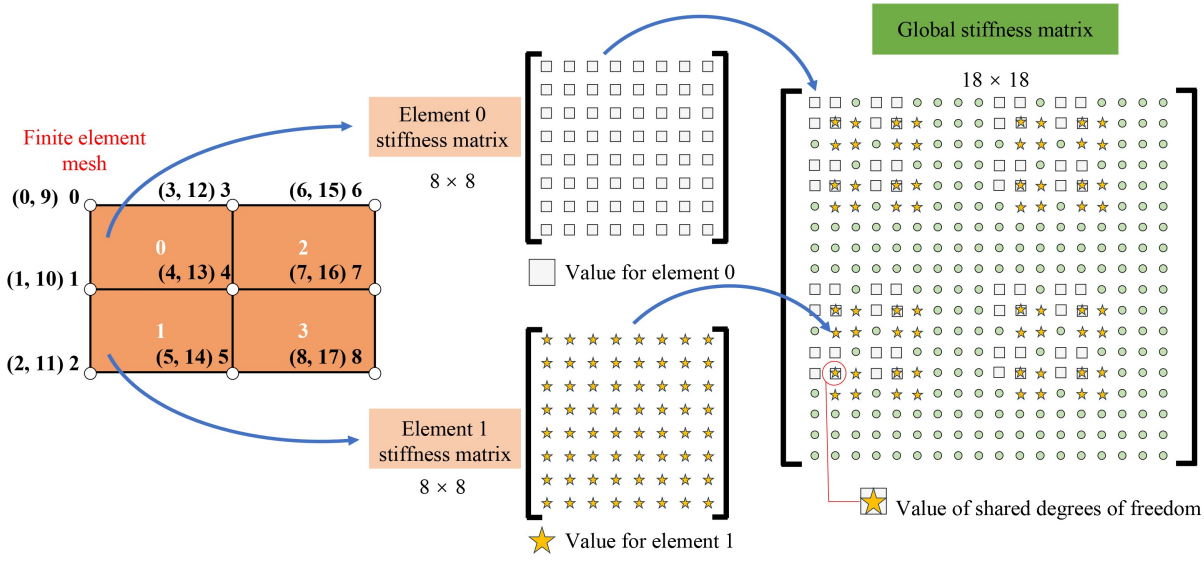


Fig. 1 Assembly of the global stiffness matrix.

val represents the values of the nonzero in the sparse matrix, ***row_index*** stores the row indices of the locations of the nonzero values, and ***col_index*** stores the column indices of the locations of the nonzero values. Therefore, the memory space required to store the sparse stiffness matrix employing the COO method is

$$\mathrm{MemUsage}_{\mathrm{COO}} = 4 \times N_z + 4 \times N_z + 8 \times N_z. \quad (6)$$

The CSR method utilizes row-wise compressed storage for the nonzero values of sparse matrices, whereas the CSC method employs column-wise compressed storage. Both methods operate similarly, enabling efficient access to nonzero values in any row or column of the sparse matrix. As an example, we describe the CSC method, which is the default storage method used in MATLAB. The storage format for the CSC method is depicted in Fig. 3. Assume that the size of a sparse matrix is $m \times n$ and contains N_z number of nonzero values. When storing this sparse matrix, we must set up three vectors of different sizes, ***val***, ***row_index***, and ***col_ptr***. ***val*** has length N_z and is utilized to store the values of the nonzero values of the sparse matrix. ***row_index*** has length N_z and is utilized to store the row indices of the locations of the nonzero values. ***col_ptr*** has length $n + 1$ and is utilized to store the location of the first nonzero value in each column of the sparse matrix in the vector ***val***, and $\boldsymbol{col\_ptr}[n+1] = N_z + 1$. Therefore, the memory space required to store the sparse stiffness matrix employing the CSC method is

$$\mathrm{MemUsage}_{\mathrm{CSC}} = 4 \times (n+1) + 4 \times N_z + 8 \times N_z. \quad (7)$$

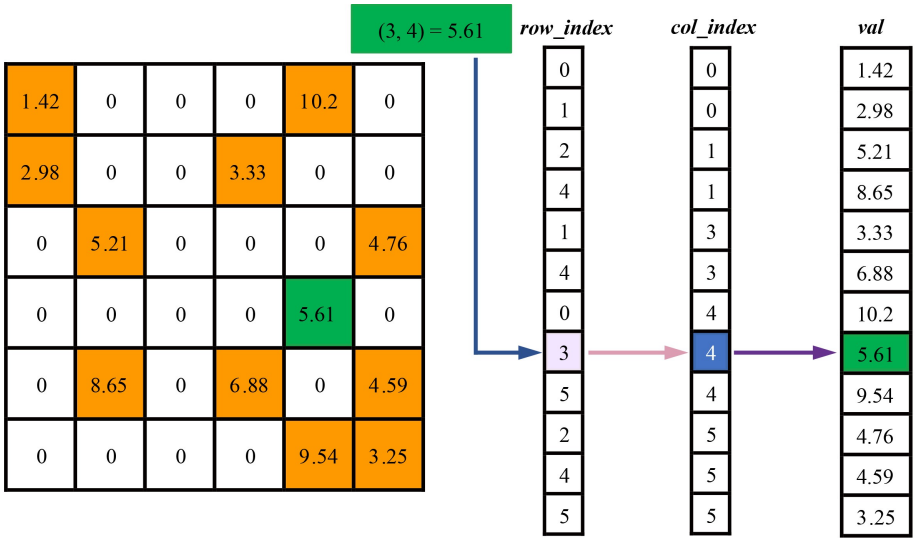


Fig. 2 Storage format of the COO method.

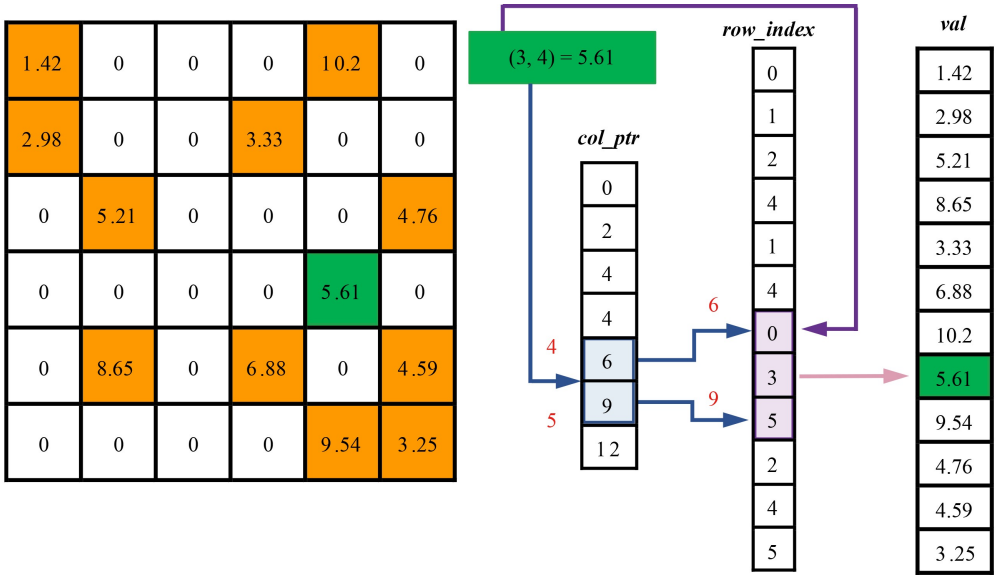


Fig. 3 Storage format of the CSC method.

3.2 BSCSC method

In FEA, the global stiffness matrix $\boldsymbol{K}$ is usually symmetric and block-structured. Therefore, we only need to store information from only the upper or lower triangular part of the matrix, along with details of the partial blocks, to reconstruct the global stiffness matrix. We introduce the BSCSC method to fully utilize these characteristics. This method is based on the concept of blocking the global stiffness matrix. In a 2D context, the global stiffness matrix can be divided into four blocks, whereas in a 3D context, it can be divided into nine blocks. Therefore, the stored variables can be divided into vectors of diagonal blocks ($\boldsymbol{val}_{\mathrm{d}}$, $\boldsymbol{row}_{\mathrm{d}}$, and $\boldsymbol{col\_ptr}_{\mathrm{d}}$) and vectors of nondiagonal blocks ($\boldsymbol{val}_{\mathrm{nd}}$, $\boldsymbol{row}_{\mathrm{nd}}$, and $\boldsymbol{col\_ptr}_{\mathrm{nd}}$). Particularly, vectors $\boldsymbol{row}_{\mathrm{d}}$ and $\boldsymbol{row}_{\mathrm{nd}}$ are used to store the row indices of the nonzero values; vectors $\boldsymbol{col\_ptr}_{\mathrm{d}}$ and $\boldsymbol{col\_ptr}_{\mathrm{nd}}$ are used to store the position of the first nonzero value in each column of the sparse matrix in vectors $\boldsymbol{val}_{\mathrm{d}}$ and $\boldsymbol{val}_{\mathrm{nd}}$; and $\boldsymbol{val}_{\mathrm{d}}$ and $\boldsymbol{val}_{\mathrm{nd}}$ store the values of the nonzero values of the sparse matrix.

The principle of the BSCSC method is demonstrated using a 2D FEA problem as an example. We discretize a 2D design domain into 16 rectangular elements, as shown in Fig. 4. The discretized structure consists of 25 nodes and 50 DoFs. The numbering methods for elements, nodes, and DoFs in the finite element mesh are similar to those shown in Fig. 1. For matrix definition, the rows and columns are indexed starting from 0. Therefore, the finite element equations for this discrete design domain can be expressed as

$$\begin{bmatrix} K_{0,0} & K_{0,1} & \cdots & \cdots & K_{0,48} & K_{0,49} \\ K_{1,0} & K_{1,1} & \cdots & \cdots & K_{1,48} & K_{1,49} \\ \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \vdots & \vdots \\ K_{48,0} & K_{48,1} & \cdots & \cdots & K_{48,48} & K_{48,49} \\ K_{49,0} & K_{49,1} & \cdots & \cdots & K_{49,48} & K_{49,49} \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_i \\ \vdots \\ U_{48} \\ U_{49} \end{bmatrix} = \begin{bmatrix} F_0 \\ F_1 \\ F_i \\ \vdots \\ F_{48} \\ F_{49} \end{bmatrix}, \tag{8}$$

where $K_{i,j}$ (i,j = 0,1, …,49) denotes the elements in the stiffness matrix $\boldsymbol{K}$, U_i denotes the elements in the displacement vector $\boldsymbol{U}$, F_i denotes the elements in the force vector $\boldsymbol{F}$, i denotes the rows, and j denotes the columns.

To illustrate the compression of the BSCSC method, consider node 1 (DoF numbers 1 and 21) in Fig. 4. As shown in Fig. 5(a), when assembling the global stiffness matrix $\boldsymbol{K}$, the elements corresponding to node 1 are located in rows (columns) 1 and 26. Due to the symmetry of the global stiffness matrix $\boldsymbol{K}$, the rows are symmetrical with respect to the nonzero values in the columns. Therefore, we use columns to describe all nonzero values. According to the properties of the global stiffness matrix, the nonzero values in columns 1 and 21 are positioned at indices 0, 1, 2, 5, 6, 7, 25, 26, 27, 30, 31, and 32. The data at positions 0, 1, 2, 5, 6, and 7 are identical to the data at positions 25, 26, 27, 30, 31, and 32. Thus, for node 1, only the data corresponding to positions 0, 1, 2, 5, 6, and 7 need to be stored in the global stiffness matrix $\boldsymbol{K}$, as shown in Fig. 5(b).

Therefore, the assembly of the global stiffness matrix corresponding to Node 1 is as follows.

Step 1. Store the number of nonzero values before the ith column in the diagonal blocks (denoted as $nz_{\mathrm{d},i}$) in $\boldsymbol{col\_ptr}_{\mathrm{d}}$, and in the nondiagonal blocks (denoted as $nz_{\mathrm{nd},i}$) in $\boldsymbol{col\_ptr}_{\mathrm{nd}}$.

$$\boldsymbol{col\_ptr}_{\mathrm{d}} = [\ldots, nz_{\mathrm{d},1}, nz_{\mathrm{d},1}+2, \ldots]. \tag{9}$$

$$\boldsymbol{col\_ptr}_{\mathrm{nd}} = [\ldots, nz_{\mathrm{nd},1}, nz_{\mathrm{nd},1}+6, \ldots]. \tag{10}$$

Step 2. Store row number of nonzero values $\boldsymbol{row}_{\mathrm{d}}$ in diagonal blocks and row number of nonzero values $\boldsymbol{row}_{\mathrm{nd}}$ in nondiagonal blocks.

$$\boldsymbol{row}_{\mathrm{d}} = [\ldots, 0, 1, \ldots]. \tag{11}$$

$$\boldsymbol{row}_{\mathrm{nd}} = [\ldots, 0, 1, 2, 5, 6, 7, \ldots]. \tag{12}$$

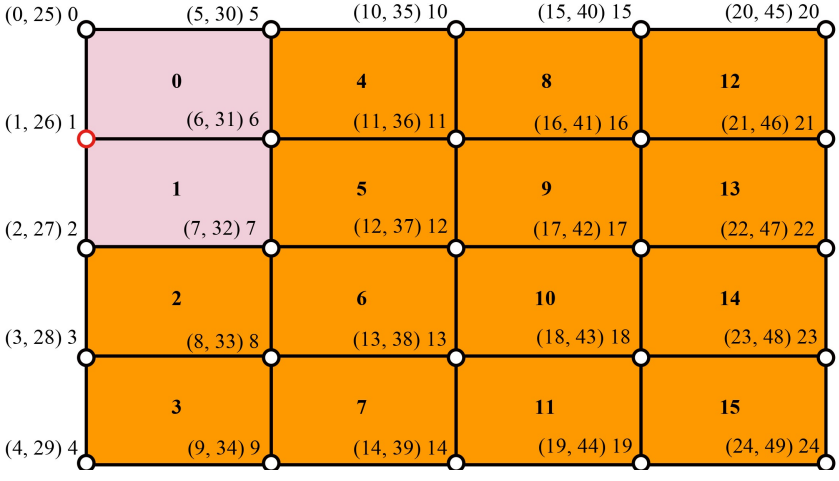


Fig. 4 2D design domain.

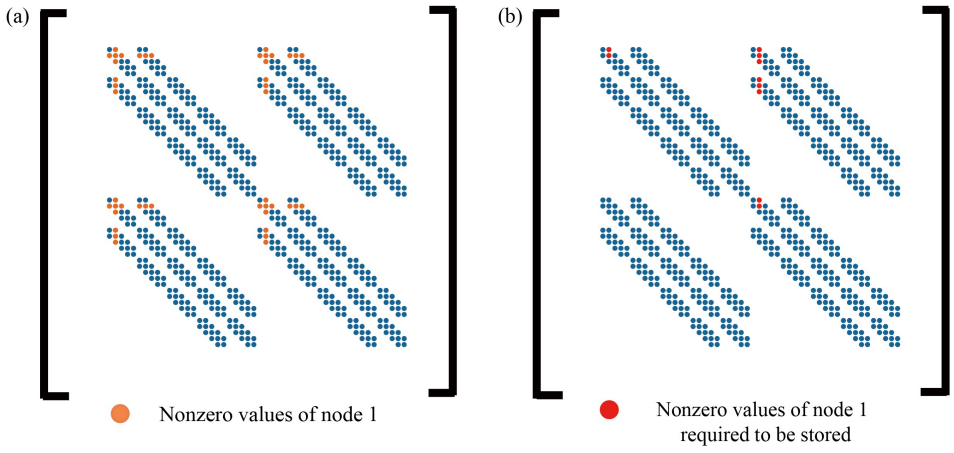


Fig. 5 Distribution of nonzero values of the global stiffness matrix $\boldsymbol{K}$. (a) Nonzero values of node 1; (b) nonzero values of node 1 required to be stored.

Step 3. Store the $\boldsymbol{val}_{\mathrm{d}}$ and $\boldsymbol{val}_{\mathrm{nd}}$ of nonzero values in diagonal and nondiagonal blocks, respectively.

$$\boldsymbol{val}_{\mathrm{d}} = [..., K_{0,1}, K_{1,1}, ..., K_{25,26}, K_{26,26}, ...]. \tag{13}$$

$$\boldsymbol{val}_{\mathrm{nd}} = [..., K_{0,26}, K_{1,26}, K_{2,26}, K_{5,26}, K_{6,26}, K_{7,26}, ...]. \tag{14}$$

When applying the Dirichlet boundary condition to the structure, the diagonal values are set to one, and the corresponding rows and columns of the global stiffness matrix are set to zero. These zero values are stored in the global stiffness matrix to retain its block symmetry. By contrast, when Neumann boundary conditions are applied, the global stiffness matrix remains unchanged and maintains its block symmetry. Thus, the BSCSC method stores the same amount of data in the diagonal blocks as it does in the upper triangular portion of the nondiagonal blocks.

For the two-dimensional example shown in Fig. 4, the BSCSC method accounts for 676 nonzero values N_z in the global stiffness matrix $\boldsymbol{K}$. Therefore, we can determine the memory required to store the global stiffness matrix $\boldsymbol{K}$ containing 16 elements, as shown in Table 1.

According to the data in Table 1, the BSCSC method requires 4176 bytes of storage space. In comparison, storing the matrix using the CSC format requires 8316 bytes. Recall that $\mathrm{MemUsage}_{\mathrm{CSC}} = 4 \times (n + 1) + 4 \times N_z + 8 \times N_z = 8316$. Therefore, the BSCSC method reduces memory usage by 49.78% compared to the CSC method. This indicates that the BSCSC method can be more efficient in terms of memory storage. For the 2D example, the computational expression for the storage space required for the global stiffness matrix $\boldsymbol{K}$ is

$$\mathrm{MemUsage}_{\mathrm{BSCSC\_2D}} = 10 \times NZ_{\mathrm{d}} + 12 \times NZ_{\mathrm{nd}} + 9 \times n + 8, \tag{15}$$

where NZ_{d} is the number of nonzero values of the diagonal block, and NZ_{nd} is the number of stored elements of the nondiagonal block.

Table 1 Memory required to store the global stiffness matrix K containing 16 elements

Variable	Length	Type
$\boldsymbol{col\_ptr}_{\mathrm{d}}$	$\frac{1}{2}n + 1 = 26$	int
$\boldsymbol{col\_ptr}_{\mathrm{nd}}$	$\frac{1}{2}n + 1 = 26$	int
$\boldsymbol{row}_{\mathrm{d}}$	$\frac{1}{2}nz_{\mathrm{d},25} + \frac{1}{4}n = 97$	int
$\boldsymbol{row}_{\mathrm{nd}}$	$nz_{\mathrm{nd},25} = 169$	int
$\boldsymbol{val}_{\mathrm{d}}$	$nz_{\mathrm{d},25} + \frac{1}{2}n = 194$	double
$\boldsymbol{val}_{\mathrm{nd}}$	$nz_{\mathrm{nd},25} = 169$	double

For the 3D example, an $n \times n$ global stiffness matrix $\boldsymbol{K}$ can be divided into nine pieces due to the three DoFs x, y, and z. Therefore, the memory required for the global stiffness matrix $\boldsymbol{K}$ adopting the BSCSC method is

$$\mathrm{MemUsage}_{\mathrm{BSCSC\_3D}} = 14 \times NZ_{\mathrm{d}} + 28 \times NZ_{\mathrm{nd}} + \frac{22}{3} \times n + 8. \tag{16}$$

3.3 BSCSC-based algorithm for solving finite element equations

The solution methodologies for finite element equations can be broadly classified into two categories: direct solution methods [34] and iterative methods [35]. Direct solution methods involve factorizing the system of equations into upper and lower triangular systems, followed by solving these systems through backward substitution. Although this method has high computational complexity, it is primarily used for solving small-scale equation systems. By contrast, iterative methods progressively approximate the precise solution of the equation system by iteratively updating the solution vector. This approach is extensively used for solving large-scale linear equation systems. Commonly employed iterative techniques include the conjugate gradient

method, Jacobi iterative method, and Gauss–Seidel method [36]. Notably, the conjugate gradient method is known for its superior convergence rates, particularly for systems with symmetric positive definite matrices. Moreover, by integrating preconditioning methods, such as Jacobi preconditioning and multigrid methods, the convergence of the conjugate gradient method is significantly improved. This combination is referred to as the preconditioned conjugate gradient (PCG) method. Consequently, in this investigation, we employ the PCG method to effectively solve the finite element equation system. The procedure of this algorithm is shown in Table 2.

Table 2 PCG method algorithm

Algorithm 1 PCG

Input: global stiffness matrix $\boldsymbol{K}$, load vector $\boldsymbol{F}$.

Output: exact iteration solution of displacement vector $\boldsymbol{U}_k$. Function: $\boldsymbol{U}$ = PCG($\boldsymbol{K}$,$\boldsymbol{U}$,$\boldsymbol{F}$).

1. Initialize parameters: maximum iteration *maxiter* and tolerance *tol*
2. $k = 0$ // Iteration counter of Function
3. $\boldsymbol{r}_0 = \boldsymbol{F} - \text{SpMV}(\boldsymbol{K},\boldsymbol{U}_k)$ // Calculating residual
4. $\boldsymbol{p}_0, z_0 = \boldsymbol{M}^{-1}\boldsymbol{r}_0$ // Using the preconditioner
5. **while** $k <$ *maxiter* **do**
6. $\alpha_k = \boldsymbol{r}_k^{\mathrm{T}} z_k \left[\boldsymbol{p}_k^{\mathrm{T}} \cdot \text{SpMV}(\boldsymbol{K},\boldsymbol{p}_k)\right]^{-1}$ // Calculating step length
7. $\boldsymbol{U}_{k+1} = \boldsymbol{U}_k + \alpha_k \boldsymbol{p}_k$ // Updating solution vector
8. $\boldsymbol{r}_{k+1} = \boldsymbol{r}_{k+1} - \alpha_k \cdot \text{SpMV}(\boldsymbol{K},\boldsymbol{p}_k)$ // Updating residual
9. **if** $\|\boldsymbol{r}_{k+1}\| / \|\boldsymbol{r}_0\| < tol$ // Checking convergence criterion
10. **break**
11. **end**
12. $z_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{r}_{k+1}$ // Updating preconditioner
13. $\beta_k = \boldsymbol{r}_{k+1}^{\mathrm{T}} z_{k+1} / \boldsymbol{r}_k^{\mathrm{T}} z_k$ // Updating search direction
14. $\boldsymbol{p}_{k+1} = z_{k+1} + \beta_k \boldsymbol{p}_k$
15. $k = k + 1$// Updating iteration counter
16. **end**
17. **return** $\boldsymbol{U}_k$

Notably, all iterative methods require the execution of the SpMV during each iteration, and its speed directly affects the efficiency of solving the system of linear equations. The performance of the SpMV is influenced by how the global stiffness matrix is stored in a compressed manner. A good storage algorithm should minimize the SpMV computation time while reducing the storage requirements to achieve an efficient solution. Therefore, we develop a BSCSC-based SpMV method that improves CPU throughput and accelerates the SpMV process through loop unrolling. The process of the BSCSC-based SpMV method (for a 2D problem) is shown in Table 3.

Table 3 Procedure of the BSCSC-based SpMV method

Algorithm 2 BSCSC-based SpMV method (for a 2D problem)

Input: global stiffness matrix $\boldsymbol{K}$, displacement vector $\boldsymbol{U}$, dimension of model *dim*.

Output: load vector $\boldsymbol{F}$ = SpMV_2d($\boldsymbol{K}$,$\boldsymbol{U}$).

1. m = sizeof($\boldsymbol{U}$)/sizeof(double)
2. $\boldsymbol{F}$ = calloc(length, sizeof(double))
3. **for** j = 0 to m/2 − 1
4. **for** $i = \boldsymbol{col\_ptr}_{\mathrm{d}}[j]$ to $\boldsymbol{col\_ptr}_{\mathrm{d}}[j+1]-2$
5. $\boldsymbol{F}[\boldsymbol{row}_{\mathrm{d}}[i]] \mathrel{+}= \boldsymbol{val}_{\mathrm{d}}[i] \times U[j]$.
6. $\boldsymbol{F}[j] \mathrel{+}= \boldsymbol{val}_{\mathrm{d}}[i] \times \boldsymbol{U}[\boldsymbol{row}_{\mathrm{d}}[i]]$.
7. $\boldsymbol{F}[\boldsymbol{row}_{\mathrm{d}}[i]+m/2] \mathrel{+}= \boldsymbol{val}_{\mathrm{d}}[i+gap_{\mathrm{d}}] \times \boldsymbol{U}[j+m/2]$.
8. $\boldsymbol{F}[j+m/2] \mathrel{+}= \boldsymbol{val}_{\mathrm{d}}[i] \times \boldsymbol{U}[\boldsymbol{row}_{\mathrm{d}}[i]+m/2]$.
9. **end for**
10. $\boldsymbol{F}[j] \mathrel{+}= \boldsymbol{val}_{\mathrm{d}}[\boldsymbol{col\_ptr}_{\mathrm{nd}}[j+1]-1] \times \boldsymbol{U}[j]$.
11. **for** $i = \boldsymbol{col\_ptr}_{\mathrm{nd}}[j]$ to $\boldsymbol{col\_ptr}_{\mathrm{nd}}[j+1]-1$
12. $\boldsymbol{F}[\boldsymbol{row}_{\mathrm{nd}}[i]] \mathrel{+}= \boldsymbol{val}_{\mathrm{nd}}[i] \times \boldsymbol{U}[j+m/2]$.
13. $\boldsymbol{F}[j+m/2] \mathrel{+}= \boldsymbol{val}_{\mathrm{nd}}[i] \times \boldsymbol{U}[\boldsymbol{row}_{\mathrm{nd}}[i]]$.
14. **end for**
15. **end for**
16. **return** $\boldsymbol{F}$

4 Numerical examples and discussion

To evaluate the performance of the BSCSC method in FEA, we present a 2D and a 3D example in this section. Section 4.1 focuses on evaluating the performance of the BSCSC method for matrix–vector multiplication. Here, we compare the solution speed and storage efficiency of finite element equations for a 2D cantilever beam using the Jacobi–PCG method with BSCSC and CSC storage formats. In Section 4.2, we apply the BSCSC method to a 3D cantilever beam to investigate its universality. Moreover, in Section 4.3, we extend the BSCSC method to engine connecting rods to verify its excellent performance in complex structures. All computations are executed on a CPU Intel® Core™ i7-10700F using a C/C++ compiler. Besides, all variables are dimensionless unless otherwise specified.

4.1 FEA of 2D cantilever beam

To demonstrate the superior performance of the BSCSC method over the CSC method in matrix–vector multiplication, we use the finite element equation analysis of a cantilever beam as an example. The analysis domain, boundary conditions, and loads of the 2D cantilever beam are shown in Fig. 6. An external load ($\boldsymbol{F}$ = 1) is applied to the right center point in the negative y-axis direction. The material properties are modeled with dimensionless parameters: elasticity modulus (E = 1) and Poisson's ratio

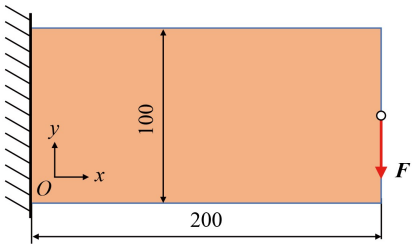


Fig. 6 Analysis domain, boundary conditions, and loads of the 2D cantilever beam.

($\upsilon = 0.3$).

The FEA mesh size of the 2D cantilever beam is set to 1, and the entire structure is discretized into 20000 four-node elements. The scale of the corresponding global stiffness matrix $\boldsymbol{K}$ is 40602 × 40602, with its distribution shown in Fig. 7. The displacement distribution of the 2D cantilever beam in the y-axis direction, obtained by the BSCSC and CSC methods, is shown in Fig. 8. The results indicate that the displacement distribution obtained by the BSCSC method matches that of the CSC method, as the BSCSC method only changes the storage mode while maintaining the same data as the CSC method. Therefore, the analysis results of the BSCSC method do not generate errors.

Figure 9 displays the time and memory required for conducting FEA on the 2D cantilever beam using BSCSC and CSC methods. The results indicate that the iteration time required by the BSCSC method is significantly less than that of the CSC method. Specifically, the average iteration time for the CSC method is 2.521 ms, whereas that for the BSCSC method is 1.387 ms, representing a 44.98% decrease. In addition, the total solution time and memory utilization for the global stiffness matrix $\boldsymbol{K}$ using the CSC method are 5.475 s and 8.436 MB, respectively. In comparison, the BSCSC method requires 2.034 s and 3.378 MB, representing reductions of 62.58% in total solution time and 59.95% in memory usage. These results demonstrate that the presented BSCSC method offers significant advantages in memory saving and high

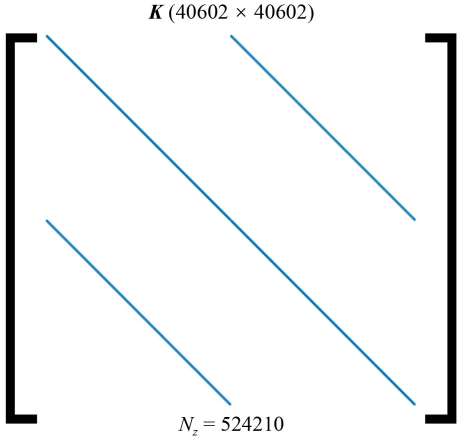


Fig. 7 Nonzero value distribution of the global stiffness matrix of the 2D cantilever beam.

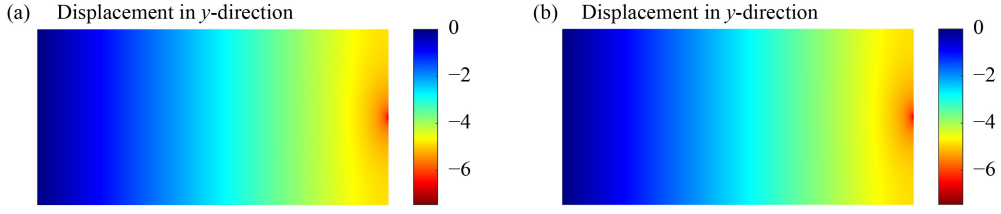


Fig. 8 Displacement distribution in the y-direction of the 2D cantilever beam. (a) BSCSC method. (b) CSC method.

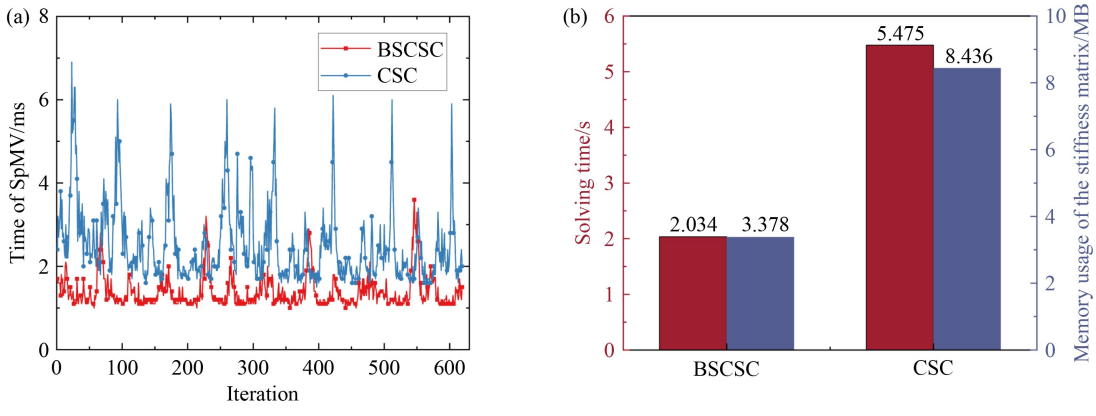


Fig. 9 Time and memory required for conducting FEA on the 2D cantilever beam using BSCSC and CSC methods. (a) Time for matrix–vector multiplication for different numbers of iteration. (b) Total time and memory required to solve the FEA.

solving efficiency in FEA.

4.2 FEA of 3D cantilever beam

In this section, we use the 3D cantilever beam as an example to verify the applicability of the BSCSC method for FEA of 3D structures. The analysis domain, boundary conditions, and loads of the 3D cantilever beam are shown in Fig. 10. An external load ($\boldsymbol{F}$ = 64) is applied at the upper right corner in the negative z-axis direction. The material properties are the same as those used for the 2D cantilever beam.

The minimum FEA mesh size for the 3D cantilever beam is set to 1, and the entire structure is discretized into 16384 eight-node ortho-hexahedral elements. The scale of the corresponding global stiffness matrix $\boldsymbol{K}$ is 57915 × 57915. The global stiffness matrix contains a total of 2745603 nonzero values, and its distribution is depicted in Fig. 11. The total displacement distribution of the 3D cantilever beam as obtained using the BSCSC and CSC methods are shown in Fig. 12. Similar to the 2D case, the displacement distributions from both methods are identical.

Figure 13 shows the time and memory required for

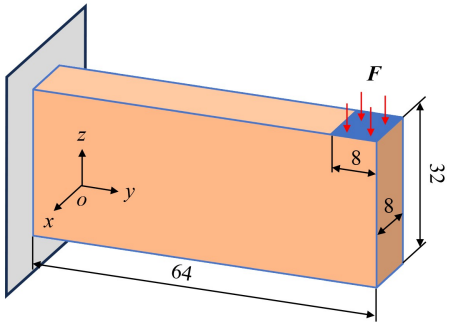


Fig. 10 Analysis domain, boundary conditions, and loads of the 3D cantilever beam.

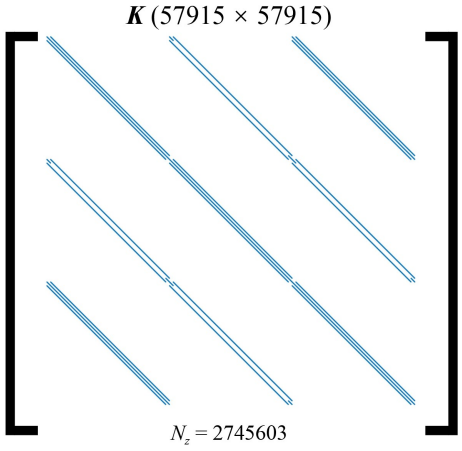


Fig. 11 Nonzero value distribution of the global stiffness matrix of the 3D cantilever beam.

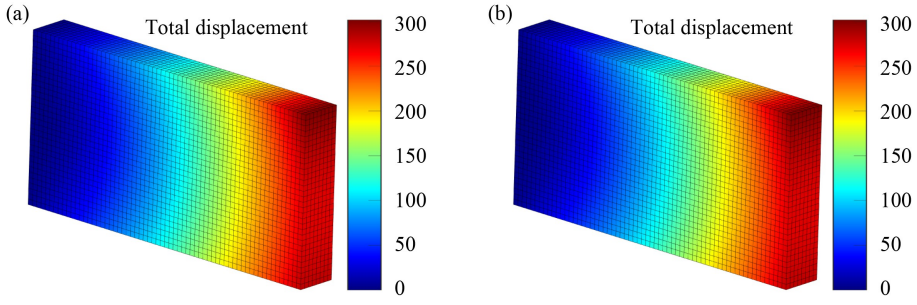


Fig. 12 Displacement distribution in the z-axis direction of the 3D cantilever beam. (a) BSCSC method. (b) CSC method.

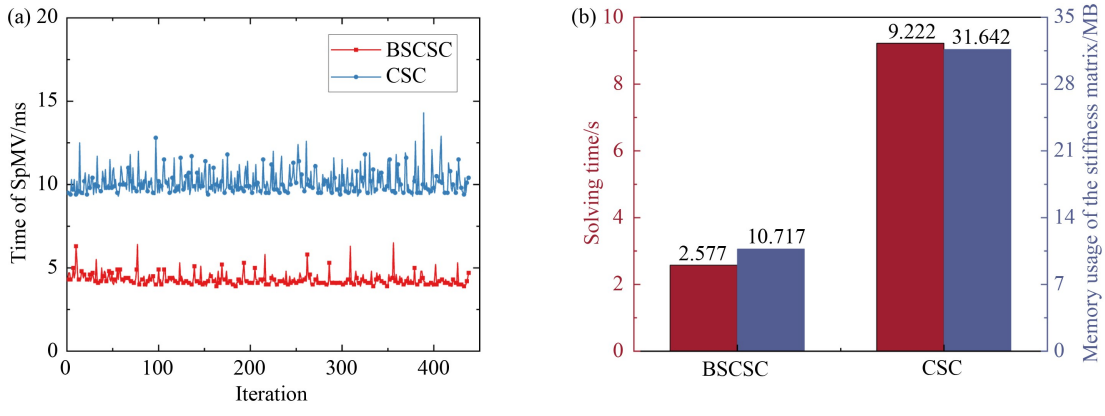


Fig. 13 Time and memory required for conducting FEA on the 3D cantilever beam using BSCSC and CSC methods. (a) Time for matrix–vector multiplication for different numbers of iteration. (b) Total time and memory required to solve the FEA.

conducting FEA on the 3D cantilever beams using both methods. The CSC method has an average iteration time of 10.121 ms, whereas the BSCSC method reduces this to 4.276 ms, making 57.75% reduction. Moreover, the total solution time and memory utilized by the CSC method for the global stiffness matrix $\boldsymbol{K}$ are 9.222 s and 31.642 MB, respectively. By contrast, the BSCSC method uses only 2.577 s and 10.717 MB, representing reductions of 72.06% in total solution time and 66.13% in memory usage. These analytical results demonstrate the feasibility and superiority of the BSCSC method for large-scale FEA of 3D cantilever beams.

4.3 FEA of engineering structure

In this section, we use the engine connecting rod model as an example to validate the BSCSC method's effectiveness in analyzing complex engineering structures. Figure 14(a) displays the design domain, boundary conditions, and applied loads for the engine connecting rod model, which is subject to an external homogeneous load $\boldsymbol{F}$ of 10 MPa. The material properties are defined with an elastic modulus and Poisson's ratio of E = 210 GPa and υ = 0.3, respectively. Four-node tetrahedral elements are adopted to discretize the engine connecting rod model, and the mesh model is shown in Fig. 14(b). The entire structure comprises 79005 elements and 16873 nodes, with the corresponding global stiffness matrix $\boldsymbol{K}$ having dimensions of 50619 × 50619. It includes 2008791 nonzero values, as depicted in Fig. 15.

The engine connecting rod undergoes analysis using the BSCSC and CSC methods. Figure 16 illustrates the displacement distributions of the engine connecting rod obtained by both methods. The results indicate that the displacement distributions in the x, y, and z directions, as well as the total displacement, are identical for both methods due to the consistent data storage.

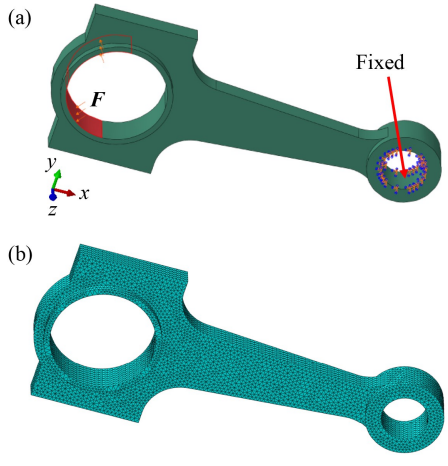


Fig. 14 Engine connecting rod. (a) Analysis domain, boundary conditions, and loads. (b) Mesh model.

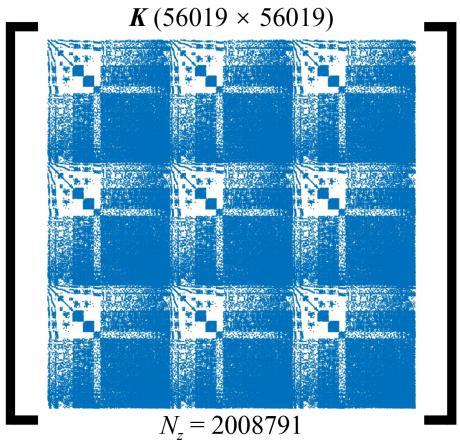


Fig. 15 Nonzero value distribution of the global stiffness matrix of the engine connecting rod.

The time and memory requirements for conducting FEA on the engine connecting rod using both methods are shown in Fig. 17. The average iteration time for the CSC method is 5.913 ms, whereas the BSCSC method achieves this in 3.215 ms, marking a 45.63% reduction. The CSC method consumes 33.233 s and 23.182 MB of memory to resolve the global stiffness matrix $\boldsymbol{K}$, whereas the BSCSC method reduces these to 10.419 s and 7.917 MB, respectively. Therefore, the BSCSC method decreases the total solution time by 68.65% and memory usage by 65.85%. These findings highlight the BSCSC method's advantages in memory efficiency and computational speed for FEA applications.

5 Extensions

As an alternative to traditional FEA, IGA achieves the unification of geometric and analytical models through the use of spline basis functions—such as non-uniform rational B-splines (NURBS) and T-splines—employed in CAD as shape functions for physical fields [37–39]. This integration is crucial for engineering design as it considerably reduces the time from design to analysis, significantly increasing efficiency. Therefore, exploring the application of the BSCSC method within IGA is essential to further enhance analytical efficiency. This section provides two examples to validate the scalability of the BSCSC method in IGA. In Section 5.1, we briefly introduce the IGA method. In Section 5.2, a quadratic annulus example is used to verify the superiority of the BSCSC method in IGA. In Section 5.3, an L-shaped beam example is presented to analyze the performance of the BSCSC method in IGA across different element scales.

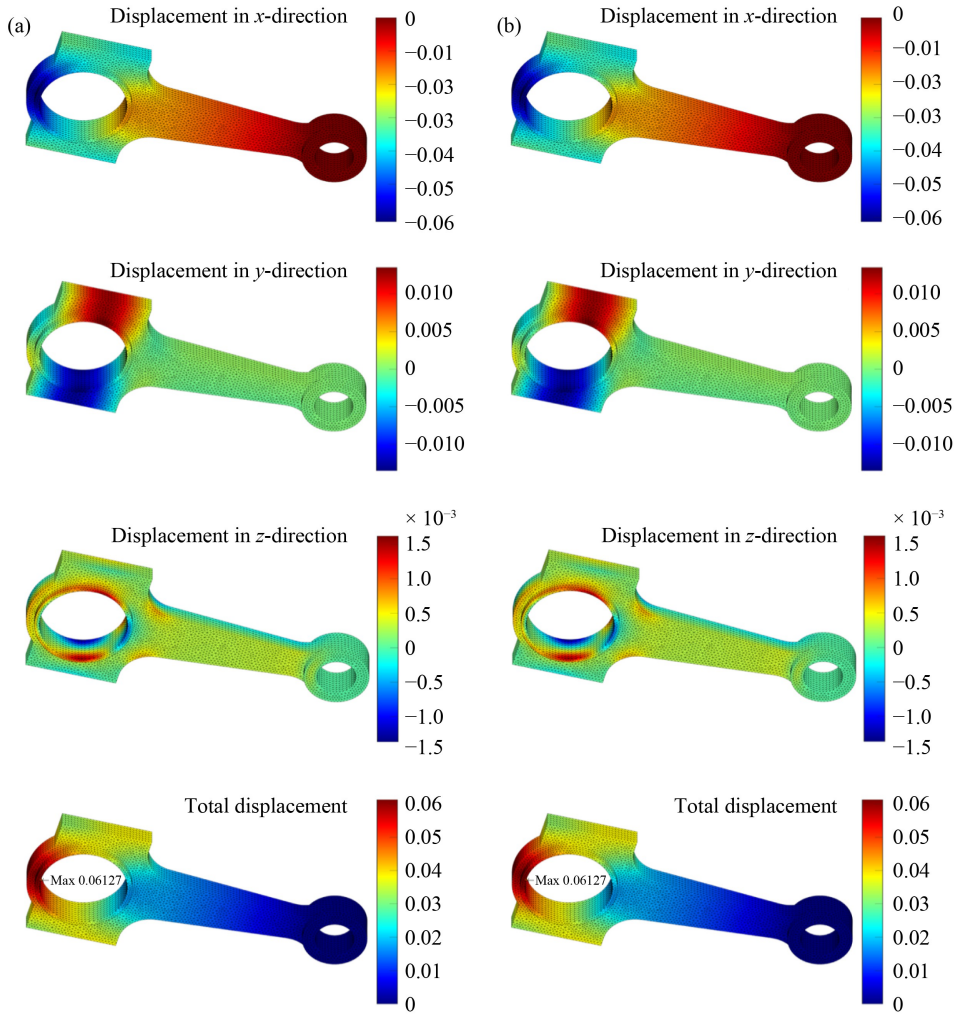


Fig. 16 Displacement distributions of the engine connecting rod. (a) BSCSC method. (b) CSC method.

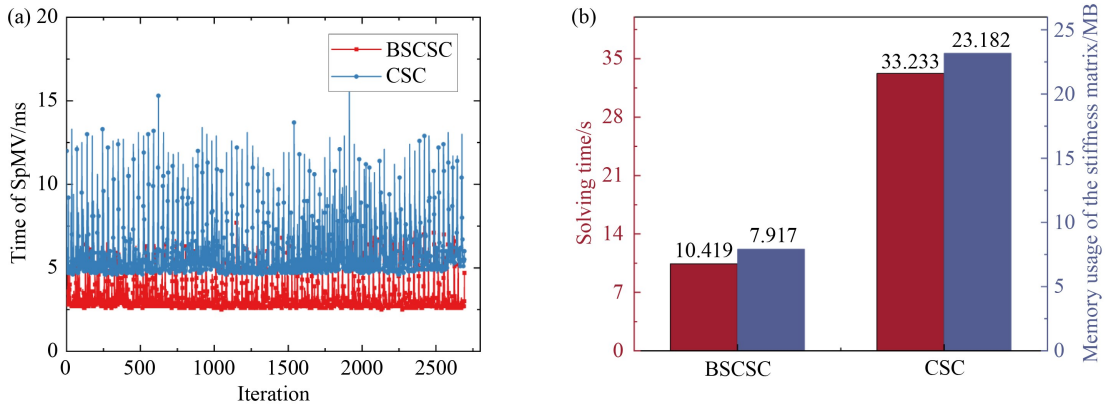


Fig. 17 Time and memory required for conducting FEA on the engine connecting rod using BSCSC and CSC methods. (a) Time for matrix–vector multiplication for different numbers of iteration. (b) Total time and memory required to solve the FEA.

5.1 IGA method

IGA leverages the NURBS basis function instead of the shape function used in traditional FEA for conducting analysis. The NURBS basis function is based on the B-spline basis function, which constructs complex curves or surfaces using node vectors, control point vectors, and weights. The B-spline basis function is defined by the

Cox-de Boor recursive equation, as shown as follows [40]:

$$B_{i,0}(\xi)=\begin{cases}1 & \text{if } \xi_i \leqslant \xi < \xi_{i+1},\\ 0 & \text{otherwise,}\end{cases} \tag{17}$$

$$B_{i,p}(\xi)=\frac{\xi-\xi_i}{\xi_{i+p}-\xi_i}B_{i,p-1}(\xi)+\frac{\xi_{i+p+1}-\xi}{\xi_{i+p+1}-\xi_{i+1}}B_{i+1,p-1}(\xi) \quad (p>0), \tag{18}$$

where ξ_i is a knot in the nondecreasing sequence knot vectors (denoted by $\boldsymbol{\Xi}=[\xi_1,\xi_2,...,\xi_{n_c+p+1}]$), and p represents the degree of the basis function.

The expression for the NURBS basis function is obtained by introducing the weighting coefficients w_i, as shown as follows:

$$N_{i,p}(\xi)=\frac{B_{i,p}(\xi)w_i}{\sum_{j=1}^{n_c}B_{j,p}(\xi)w_i}, \tag{19}$$

where n_c is the number of control points.

Thus, a p-degree NURBS curve $C(\xi)$ can be expressed as a function related to the control point $\boldsymbol{Q}_i$ and the NURBS basis function $N_{i,p}$ as

$$C(\xi)=\sum_{i=1}^{n_c}N_{i,p}(\xi)\boldsymbol{Q}_i. \tag{20}$$

An NURBS surface can be defined as the product of p-degree NURBS curves in the ξ-direction and q-degree NURBS curves in the η-direction. The NURBS surface can be expressed as

$$S(\xi,\eta)=\sum_{i=1}^{n_c}\sum_{j=1}^{m_c}N_{i,p}(\xi)N_{j,q}(\eta)\boldsymbol{Q}_{i,j}, \tag{21}$$

where $\boldsymbol{Q}_{i,j}$ is the control points grid in ξ- and η-directions. $N_{i,p}(\xi)$ and $N_{j,q}(\eta)$ are the NURBS basis functions defined in both two directions.

Therefore, the element stiffness matrix $\boldsymbol{k}_e$ in FEA is transformed in IGA as [41]:

$$\boldsymbol{k}_e=\int_{\hat{\Omega}_e}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}|\boldsymbol{J}_1|\mathrm{d}\hat{\Omega}=\int_{\bar{\Omega}_e}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{B}|\boldsymbol{J}_1||\boldsymbol{J}_2|\mathrm{d}\bar{\Omega}, \tag{22}$$

where $\hat{\Omega}_e$ and $\bar{\Omega}_e$ represent the paracentric domain in NURBS parametric space $\boldsymbol{\Xi}$ and integration parametric space $\bar{\boldsymbol{\Xi}}$, respectively. $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$ are the Jacobian matrices denoting the transformation relation from NURBS parametric space to the physical space and the integration parametric space to the NURBS parametric space, respectively.

For the 2D structure, the knot vector $\boldsymbol{\Xi}$ includes $\boldsymbol{\Xi}^{\xi}=[\xi_1,\xi_2,\ldots,\xi_{n_c+p+1}]$ in the ξ-direction and $\boldsymbol{\Xi}^{\eta}=[\eta_1,\eta_2,\ldots,\eta_{m_c+q+1}]$ in the η-direction; and the strain–displacement matrix $\boldsymbol{B}$ is represented as

$$\boldsymbol{B}=\begin{bmatrix}\frac{\partial N_1}{\partial x} & 0 & \cdots & \frac{\partial N_{nmc}}{\partial x} & 0\\ 0 & \frac{\partial N_1}{\partial y} & \cdots & 0 & \frac{\partial N_{nmc}}{\partial y}\\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \cdots & \frac{\partial N_{nmc}}{\partial y} & \frac{\partial N_{nmc}}{\partial x}\end{bmatrix}, \tag{23}$$

where x and y represent the position parameter coordinates of the 2D structure. nmc is the product of the control points in $\boldsymbol{\Xi}$, i.e., $nmc=n_c\times m_c$.

The formulas for the Jacobi matrices $\boldsymbol{J}_1$ and $\boldsymbol{J}_2$ are as follows

$$\boldsymbol{J}_1=\begin{bmatrix}\frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi}\\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta}\end{bmatrix}, \tag{24}$$

$$\boldsymbol{J}_2=\begin{bmatrix}\frac{\partial \xi}{\partial \bar{\xi}} & \frac{\partial \eta}{\partial \bar{\xi}}\\ \frac{\partial \xi}{\partial \bar{\eta}} & \frac{\partial \eta}{\partial \bar{\eta}}\end{bmatrix}, \tag{25}$$

where $\bar{\xi}$ and $\bar{\eta}$ are the parameters defined in the Gaussian orthogonal domain.

The assembly of the element stiffness matrix into the global stiffness matrix in IGA is similar to that in traditional FEA. However, IGA utilizes high-order elements, leading to a denser stiffness matrix than that in traditional FEA. Thus, the global stiffness matrix in IGA remains blocked and symmetric.

Consider a simple 2D IGA model as an example. The NURBS parameter space is defined as [0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1] × [0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1], with degrees p, q = 2 and weight factor $w_{i,j}=1$. The IGA model and the distribution of nonzero values are shown in Fig. 18. Therefore, the BSCSC method can still be effectively applied to IGA.

5.2 IGA of quadratic annulus

To verify the scalability of the BSCSC method, we apply it to the IGA method, using the IGA analysis of a quadratic annulus as an example. The Jacobi PCG method is used to solve the IGA equations. The analysis domain, boundary conditions, and loads of the quarter annulus are shown in Fig. 19. An external uniform load ($\boldsymbol{F}$ = 1) is applied to the top left corner in the positive direction of the x-axis. The material properties are consistent with those specified in Section 4.1.

To obtain more accurate analysis results, we perform several h-refinements on the circular model, resulting in a final number of 4096 elements. The global stiffness matrix $\boldsymbol{K}$ scale is 8712 × 8712. The displacement distributions in the x-axis direction of the quadratic annulus obtained by the two stiffness matrix storage methods are shown in Fig. 20. The results clearly show that the displacement distribution obtained by the BSCSC

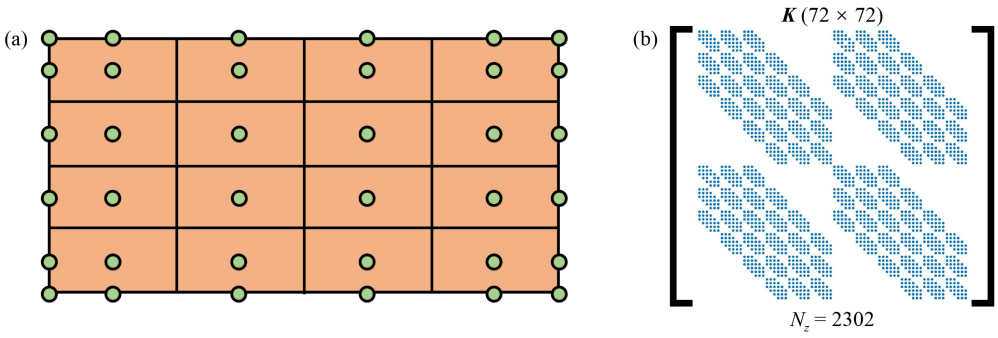


Fig. 18 IGA model and distribution of nonzero values.

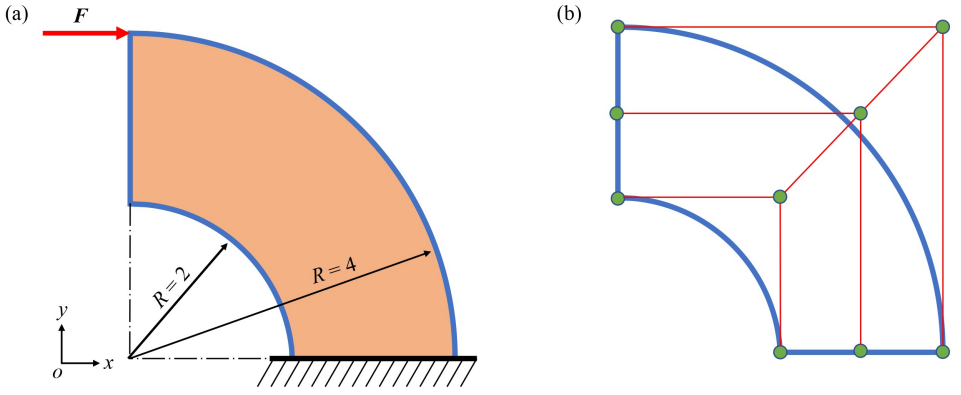


Fig. 19 Quadratic annulus. (a) Analysis domain, boundary conditions, and loads. (b) Initial control points.

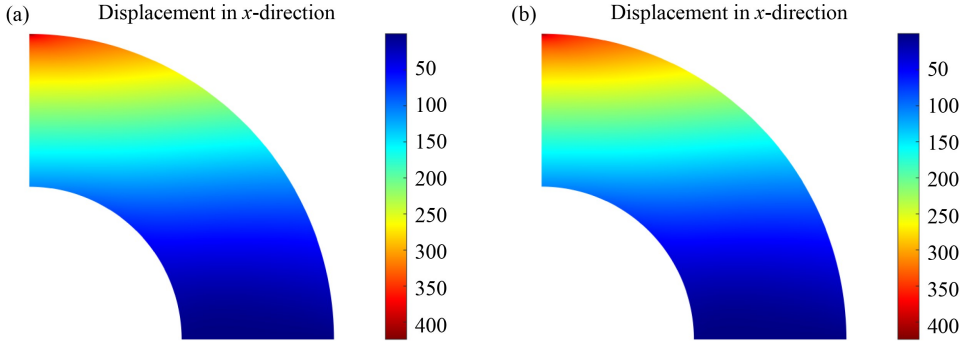


Fig. 20 Displacement distribution in the *x*-direction of the quadratic annulus. (a) BSCSC method. (b) CSC method.

method is identical to that obtained by the CSC method in the IGA. This demonstrates that the BSCSC method can guarantee analytical accuracy in IGA.

Figure 21 illustrates the time and memory requirements for IGA of the quadratic annulus using the BSCSC and CSC methods. It includes the time for matrix–vector multiplication for different iteration numbers and the total time needed to solve the IGA. According to the results in Fig. 21(a), the iteration time of the BSCSC method is significantly lower than that of the CSC method. The average iteration time of the CSC method is 0.915 ms, whereas the BSCSC method averages 0.413 ms, representing a 54.86% reduction in time compared to the CSC method. The results in Fig. 21(b) show that the total time and memory utilized by the CSC method for the global stiffness matrix $\boldsymbol{K}$ are 1.184 s and 4.839 MB; whereas the BSCSC method uses 0.383 s and 1.797 MB, respectively. As a result, the total time for solving the IGA equations is reduced by 67.65%, and the memory usage of the BSCSC method is reduced by 62.86% compared to the CSC method. These results demonstrate that the BSCSC method offers significant advantages in terms of memory saving and high solution efficiency in terms of IGA.

5.3 IGA of L-shaped beam

In this section, we analyze an L-shaped beam to further

verify the efficient performance of the BSCSC method in IGA across different scales. The analysis domain, boundary conditions, and loads of the L-shaped beam are shown in Fig. 22. An external load ($\boldsymbol{F}$ = 1) is uniformly distributed along the left boundary in the positive direction of the x-axis. The material properties are the same as those in Section 4.1.

We apply 5-, 6-, and 7-time h-refinements to the L-shaped beam, resulting in 2048, 8192, and 32768 elements and corresponding global stiffness matrix scales of 4488 × 4488, 17160 × 17160, and 67080 × 67080, respectively. Table 4 shows the displacement distribution of the L-shaped beam in the x-axis direction obtained by both methods. The results indicate that the displacement distributions achieved by the BSCSC method are identical to those obtained by the CSC method for different element scales in IGA. This confirms that the BSCSC method maintains accuracy across different element scales in IGA.

Figure 23 shows the iteration time required to solve the IGA of the L-shaped beam for the three element scales. The results demonstrate that the iteration time of the BSCSC method is consistently lower than that of the CSC method across different element scales. The average iteration time of the CSC method for solving the three element scales is 0.451, 1.934, and 7.886 ms. In comparison, the average iteration time of the BSCSC method is 0.201, 0.862, and 3.661 ms, representing reductions of 55.43%, 55.43%, and 53.58%, respectively. Figure 24 illustrates the total time and memory required to solve the IGA for the L-shaped beam. The results show that the total solution time and memory usage for the global stiffness matrix $\boldsymbol{K}$ are lower for the BSCSC method compared to the CSC method across different element scales. Specifically, the total time for the CSC method to solve the three element scales is 0.561, 4.007, and 30.616. For the BSCSC method, the times are 0.278, 2.151, and 16.589 s, reducing the total solution time by 50.45%, 46.32%, and 45.82%, respectively. Furthermore, the CSC method utilizes 2.449, 9.617, and 38.108 MB of memory for solving the global stiffness matrix $\boldsymbol{K}$ for the three element scales. Conversely, the BSCSC method utilizes 0.911, 3.571, and 14.147 MB of memory, reducing memory usage by 62.80%, 62.87%, and

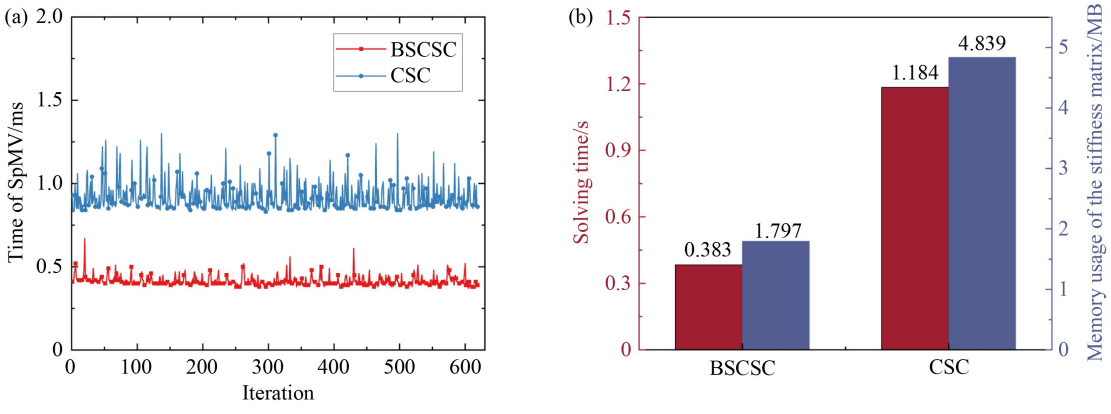


Fig. 21 Time and memory required for conducting IGA on the quadratic annulus using BSCSC and CSC methods. (a) Time for matrix–vector multiplication for different numbers of iteration. (b) Total time and memory required to solve the IGA.

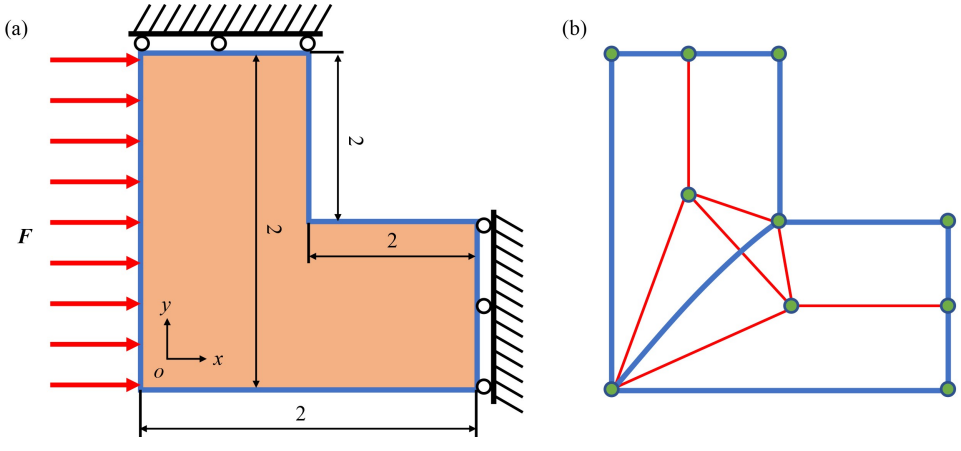
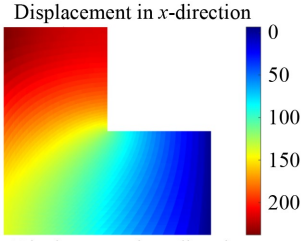
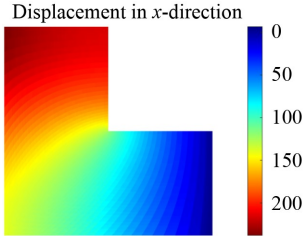
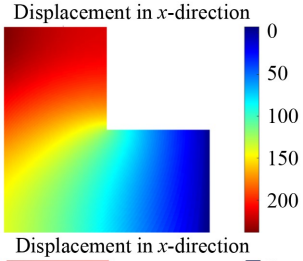
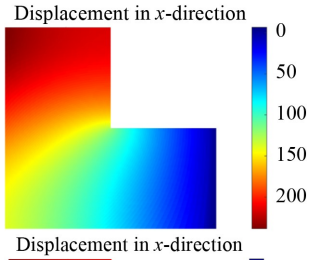
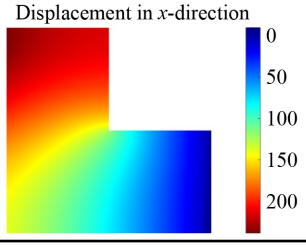
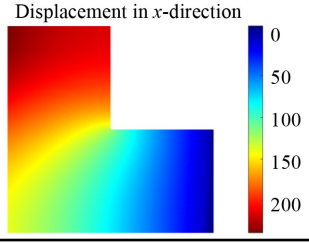


Fig. 22 L-shaped beam. (a) Analysis domain, boundary conditions, and loads. (b) Initial control points.

Table 4 Displacement distribution in the *x*-axis direction of the L-shaped beam under different element scales

Total elements	BSCSC method	CSC method
2048		
8192		
32768		

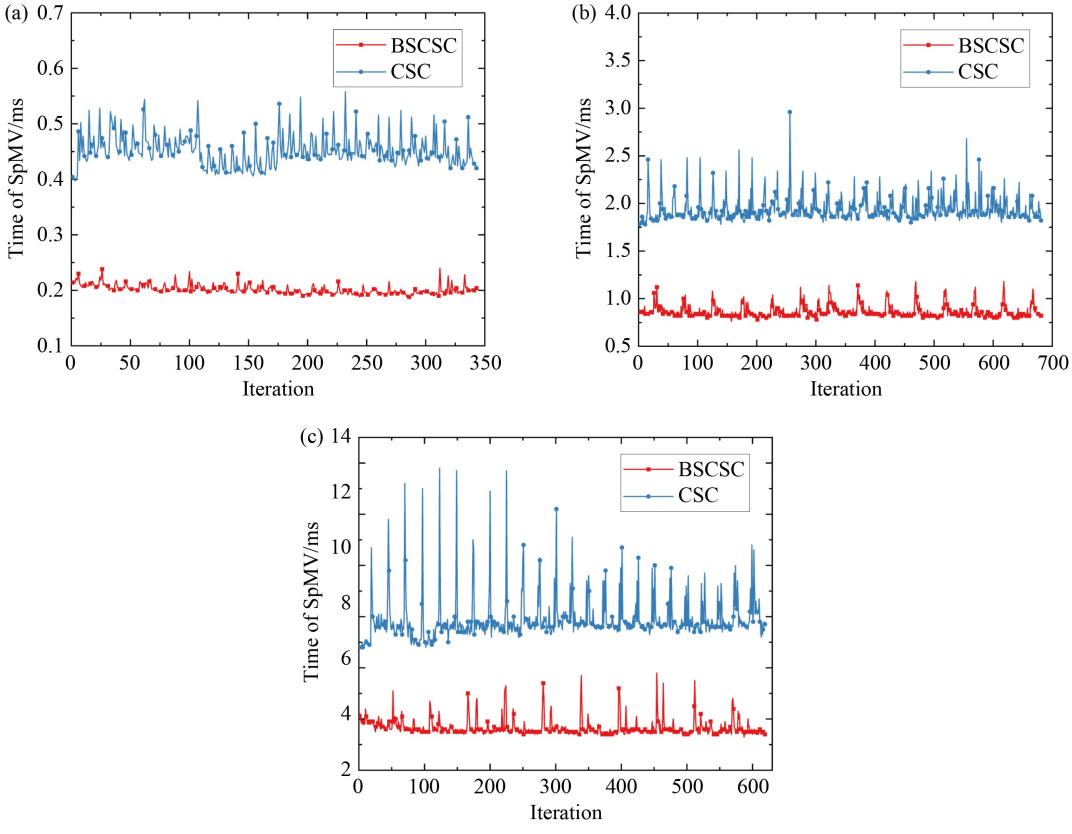


Fig. 23 Iteration time required to solve the IGA of the L-shaped beam for the three element scales: (a) 2048; (b) 8192; (c) 32768.

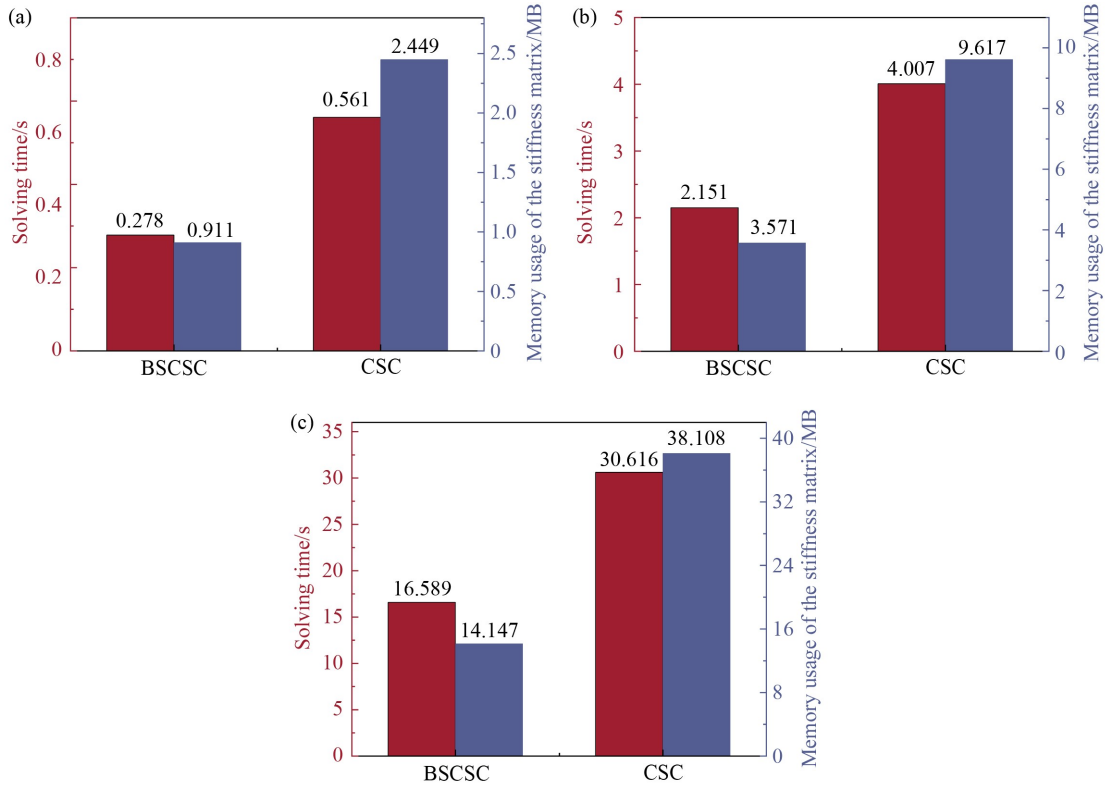


Fig. 24 Total time and memory required to solve the IGA for the L-shaped beam: (a) 2048; (b) 8192; (c) 32768.

62.88%, respectively. The findings above demonstrate the superiority of the BSCSC method over the conventional CSC method for IGA across different element scales.

6 Conclusions

In this study, we introduced a novel compressed storage method for the global stiffness matrix and elaborated on the corresponding algorithmic procedure for solving finite element equations. This method fully utilizes the blocked symmetry property of the global stiffness matrix, thereby reducing memory utilization and improving the computational efficiency of FEA. Notably, the BSCSC method changes the storage format without altering the numerical values within the global stiffness matrix, thereby maintaining the computational accuracy of FEA. To demonstrate the advantages of the BSCSC method in complex engineering structures, we employed engine connecting rods. We also extended the application of the BSCSC method to IGA to ascertain its scalability. By examining a quadratic annulus and an L-shaped beam, we verified that the BSCSC method offers comparable advantages in terms of memory conservation and efficiency improvements within the IGA domain.

Currently, the BSCSC method addresses linear FEA problems in serial operations and has not been extended to nonlinear FEA or parallel computation. Our future research will focus on two main areas: (1) integrating the BSCSC method with parallel computing techniques to solve large-scale finite element equations and (2) extending the application of the BSCSC method to solve nonlinear equation systems within FEA, thereby addressing more complex engineering challenges.

Nomenclature

Abbreviations

BSCSC	Blocked symmetric compressed sparse column
COO	Coordinate
CSC	Compressed sparse column
CSR	Compressed sparse row
DoF	Degree of freedom
ELL	ELLPACK
FEA	Finite element analysis
IGA	Isogeometric analysis
NURBS	Non-uniform rational B-splines
PCG	Preconditioned conjugate gradient
SpMV	Sparse matrix–vector multiplication

Variables

$\boldsymbol{B}_e$	Strain displacement matrix of element e
$B_{i,p}(\xi)$	B-spline basis function
col_index	Stores the column indices of the locations of the nonzero values
$\boldsymbol{col\_ptr}_{\mathrm{d}}$	Store the diagonal blocks position of the first nonzero value in each column
$\boldsymbol{col\_ptr}_{\mathrm{nd}}$	Store the nondiagonal blocks position of the first nonzero value in each column
$C(\xi)$	NURBS curve
$\boldsymbol{D}$	Solid material elasticity matrix
E	Elastic modulus of the material
$\boldsymbol{F}$	Load magnitude
$\boldsymbol{J}_1$	Jacobian matrices denoting the transformation relation from NURBS parametric space to the physical space
$\boldsymbol{J}_2$	Jacobian matrices denoting the integration parametric space to the NURBS parametric space
$\boldsymbol{k}_e$	Element stiffness matrix
$\boldsymbol{K}$	Global stiffness matrix
$n_{\mathrm{c}},m_{\mathrm{c}}$	Number of control points in different directions
$nz_{\mathrm{d},i}$	Number of nonzero values before the ith column in the diagonal blocks
$nz_{\mathrm{nd},i}$	Number of nonzero values before the ith column in the nondiagonal blocks
N_i	Shape function of the isogeometric element
$N_{i,p},N_{j,q}$	NURBS basis function of the NURBS curve in different directions
N_z	Number of nonzero values
NZ_{d}	Number of nonzero values of the diagonal block
NZ_{nd}	Number of nonzero values of the nondiagonal block
p,q	Degree of the basis function
$\boldsymbol{Q}_i$	Control point
$S(\xi,\eta)$	NURBS surface
row_index	Stores the row indices of the locations of the nonzero values
$\boldsymbol{row}_{\mathrm{d}}$	Store the diagonal blocks row indices of the nonzero values in BSCSC
$\boldsymbol{row}_{\mathrm{nd}}$	Store the nondiagonal blocks row indices of the nonzero values in BSCSC
$\boldsymbol{U}$	Displacement vector
val	Stores the values of the nonzero in the sparse matrix
$\boldsymbol{val}_{\mathrm{d}}$	Stores the diagonal blocks values of the nonzero in the sparse matrix
$\boldsymbol{val}_{\mathrm{nd}}$	Stores the nondiagonal blocks values of the nonzero in the sparse matrix
$\boldsymbol{\Xi}$	Nondecreasing sequence knot vector
ξ_i,η_i	A knot in the nondecreasing sequence knot vectors in different directions
υ	Poisson's ratio
$\hat{\Omega}_e$	Paracentric domain in NURBS parametric space
$\bar{\Omega}_e$	Integration parametric space

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 52075184) and Guangdong Basic and Applied Basic Research Foundation, China (Grant No. 2024A1515011786). These supports are gratefully acknowledged.

Conflict of Interest The authors declare no conflict of interest.

References

1. Kilimtzidis S, Kotzakolios A, Kostopoulos V. Efficient structural optimisation of composite materials aircraft wings. Composite Structures, 2023, 303: 116268
2. Al-Haddad L A, Mahdi N M. Efficient multidisciplinary modeling of aircraft undercarriage landing gear using data-driven Naïve Bayes and finite element analysis. Multiscale and Multidisciplinary Modeling, Experiments and Design, 2024, 7(4): 3187–3199
3. Lee J H, Yoon J S, Ryu C H, Kim S H. Springback compensation based on finite element for multi-point forming in shipbuilding. Advanced Materials Research, 2007, 26–28: 981–984
4. Kefal A, Oterkus E. Displacement and stress monitoring of a Panamax containership using inverse finite element method. Ocean Engineering, 2016, 119: 16–29
5. Abdullah S, Al-Asady N A, Ariffin A K, Rahman M M. A review on finite element analysis approaches in durability assessment of automotive components. Journal of Applied Sciences, 2008, 8(12): 2192–2201
6. Wang T, Li Y G. Design and analysis of automotive carbon fiber composite bumper beam based on finite element analysis. Advances in Mechanical Engineering, 2015, 7(6): 1687814015589561
7. Ma L Q, Yu X C, Yang Y Y, Hu Y G, Zhang X Y, Li H Y, Ouyang X, Zhu P L, Sun R, Wong C P. Highly sensitive flexible capacitive pressure sensor with a broad linear response range and finite element analysis of micro-array electrode. Journal of Materiomics, 2020, 6(2): 321–329
8. Hanganu A D, Oñate E, Barbat A H. A finite element methodology for local/global damage evaluation in civil engineering structures. Computers & Structures, 2002, 80(20–21): 1667–1687
9. Lu X Z, Kim C W, Chang K C. Finite element analysis framework for dynamic vehicle-bridge interaction system based on ABAQUS. International Journal of Structural Stability and Dynamics, 2020, 20(3): 2050034
10. Zhang K H, Wang J X, Ban Y H, Sun C X, Gao P J, Jin D. Multi-field coupling simulation of gear: a review. Journal of Failure Analysis and Prevention, 2020, 20(4): 1323–1332
11. Cao Y, Zhao B, Ding W F, Huang Q. Vibration characteristics and machining performance of a novel perforated ultrasonic vibration platform in the grinding of particulate-reinforced titanium matrix

composites. Frontiers of Mechanical Engineering, 2023, 18(1): 14
12. Khechai A, Tati A, Guettala A. Finite element analysis of stress concentrations and failure criteria in composite plates with circular holes. Frontiers of Mechanical Engineering, 2014, 9(3): 281–294
13. Sobisch L, Kaiser T, Furlan T, Menzel A. A user material approach for the solution of multi-field problems in Abaqus: theoretical foundations, gradient-enhanced damage mechanics and thermo-mechanical coupling. Finite Elements in Analysis and Design, 2024, 232: 104105
14. Tian Y, Shi T L, Xia Q. Buckling optimization of curvilinear fiber-reinforced composite structures using a parametric level set method. Frontiers of Mechanical Engineering, 2024, 19(1): 9
15. Shioya R, Ogino M, Kanayama H, Tagami D. Large scale finite element analysis with a balancing domain decomposition method. Key Engineering Materials, 2003, 243–244: 21–26
16. Lo S H. Finite element mesh generation and adaptive meshing. Progress in Structural Engineering and Materials, 2002, 4(4): 381–399
17. Baiges J, Codina R, Castañar I, Castillo E. A finite element reduced-order model based on adaptive mesh refinement and artificial neural networks. International Journal for Numerical Methods in Engineering, 2020, 121(4): 588–601
18. Bellenger E, Coorevits P. Adaptive mesh refinement for the control of cost and quality in finite element analysis. Finite Elements in Analysis and Design, 2005, 41(15): 1413–1440
19. You Y H, Kou X Y, Tan S T. Adaptive meshing for finite element analysis of heterogeneous materials. Computer-Aided Design, 2015, 62: 176–189
20. Yadav P, Suresh K. Large scale finite element analysis via assembly-free deflated conjugate gradient. Journal of Computing and Information Science in Engineering, 2014, 14(4): 041008
21. Prabhune B C, Suresh K. A fast matrix-free elasto-plastic solver for predicting residual stresses in additive manufacturing. Computer-Aided Design, 2020, 123: 102829
22. Willcock J, Lumsdaine A. Accelerating sparse matrix computations via data compression. In: Proceedings of the 20th annual international conference on Supercomputing. New York: Association for Computing Machinery, 2006, 307–316
23. Chen P, Zheng D, Sun S L, Yuan M W. High performance sparse static solver in finite element analyses with loop-unrolling. Advances in Engineering Software, 2003, 34(4): 203–215
24. Ribeiro F L B, Ferreira I A. Parallel implementation of the finite element method using compressed data structures. Computational Mechanics, 2007, 41(1): 31–48
25. Kawamura T, Kazunori Y, Yamazaki T, Iwamura T, Watanabe M, Inoguchi Y. A compression method for storage formats of a sparse matrix in solving the large-scale linear systems. In: Proceedings of 2017 IEEE International Parallel and Distributed Processing Symposium Workshops. Lake Buena Vista: IEEE, 2017, 924–931
26. Ramírez-Gil F J, de Sales Guerra Tsuzuki M, Montealegre-Rubio W. Global finite element matrix construction based on a CPU-GPU implementation. 2015, arXiv preprint arXiv: 1501.04784
27. Paulino G H, Menezes I F M, Cavalcante Neto J B, Martha L F. A methodology for adaptive finite element analysis: towards an integrated computational environment. Computational Mechanics, 1999, 23(5–6): 361–388
28. Bian X, Fang Z D. Large-scale buckling-constrained topology optimization based on assembly-free finite element analysis. Advances in Mechanical Engineering, 2017, 9(9): 1687814017715422
29. Le-Duc T, Nguyen-Xuan H, Lee J. A finite-element-informed neural network for parametric simulation in structural mechanics. Finite Elements in Analysis and Design, 2023, 217: 103904
30. Chen S H, Liang P, Han W Z. A new method of sensitivity analysis of static responses for finite element systems. Finite Elements in Analysis and Design, 1998, 29(3–4): 187–203
31. Ribeiro F L B, Coutinho A L G A. Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses. International Journal for Numerical Methods in Engineering, 2005, 63(4): 569–588
32. Unnikrishnan N K, Gould J, Parhi K K. SCV-GNN: sparse compressed vector-based graph neural network aggregation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(12): 4803–4816
33. Noble G, Nalesh S, Kala S, Kumar A. Configurable sparse matrix-matrix multiplication accelerator on FPGA: a systematic design space exploration approach with quantization effects. Alexandria Engineering Journal, 2024, 91: 84–94
34. Zhong W X, Williams F W. On the direct solution of wave propagation for repetitive structures. Journal of Sound and Vibration, 1995, 181(3): 485–501
35. Zhu S R, Wu L Z, Song X L. An improved matrix split-iteration method for analyzing underground water flow. Engineering with Computers, 2023, 39(3): 2049–2065
36. Khrapov P, Volkov N. Comparative analysis of Jacobi and Gauss-Seidel iterative methods. International Journal of Open Information Technologies, 2024, 12: 23–34
37. Hughes T J R, Cottrell J A, Bazilevs Y. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Computer Methods in Applied Mechanics and Engineering, 2005, 194(39–41): 4135–4195
38. Bazilevs Y, Calo V M, Cottrell J A, Evans J A, Hughes T J R, Lipton S, Scott M A, Sederberg T W. Isogeometric analysis using T-splines. Computer Methods in Applied Mechanics and Engineering, 2010, 199(5–8): 229–263
39. Nguyen V P, Anitescu C, Bordas S P A, Rabczuk T. Isogeometric analysis: an overview and computer implementation aspects. Mathematics and Computers in Simulation, 2015, 117: 89–116
40. Gupta V, Jameel A, Verma S K, Anand S, Anand Y. An insight on NURBS based isogeometric analysis, its current status and involvement in mechanical applications. Archives of Computational Methods in Engineering, 2023, 30(2): 1187–1230
41. Wang Y J, Wang Z P, Xia Z H, Hien Poh L. Structural design optimization using isogeometric analysis: a comprehensive review. Computer Modeling in Engineering & Sciences, 2018, 117(3): 455–507