mathematics

MDPI

Article

epSFEM: A Julia-Based Software Package of Parallel Incremental Smoothed Finite Element Method (S-FEM) for Elastic-Plastic Problems

Meijun Zhou [1], Jiayu Qin [1,*], Zenan Huo [1], Fabio Giampaolo [2] and Gang Mei [1,*]

[1] School of Engineering and Technology, China University of Geosciences (Beijing), Beijing 100083, China; meijun.zhou@cugb.edu.cn (M.Z.); zenan.huo@email.cugb.edu.cn (Z.H.)
[2] Consorzio Interuniversitario Nazionale per l'Informatica (CINI), 80100 Naples, Italy; fabio.giampaolo@consorzio-cini.it
* Correspondence: jiayu.qin@cugb.edu.cn (J.Q.); gang.mei@cugb.edu.cn (G.M.)

Abstract: In this paper, a parallel Smoothed Finite Element Method (S-FEM) package epSFEM using incremental theory to solve elastoplastic problems is developed by employing the Julia language on a multicore CPU. The S-FEM, a new numerical method combining the Finite Element Method (FEM) and strain smoothing technique, was proposed by Liu G.R. in recent years. The S-FEM model is softer than the FEM model for identical grid structures, has lower sensitivity to mesh distortion, and usually produces more accurate solutions and a higher convergence speed. Julia, as an efficient, user-friendly and open-source programming language, balances computational performance, programming difficulty and code readability. We validate the performance of the epSFEM with two sets of benchmark tests. The benchmark results indicate that (1) the calculation accuracy of epSFEM is higher than that of the FEM when employing the same mesh model; (2) the commercial FEM software requires 10,619 s to calculate an elastoplastic model consisting of approximately 2.45 million triangular elements, while in comparison, epSFEM requires only 5876.3 s for the same computational model; and (3) epSFEM executed in parallel on a 24-core CPU is approximately 10.6 times faster than the corresponding serial version.

Keywords: elastic-plastic problems; incremental theory; Smoothed Finite Element Method (S-FEM); Julia language; parallel programming

MSC: 35-04

Citation: Zhou, M.; Qin, J.; Huo, Z.; Giampaolo, F.; Mei, G. epSFEM: A Julia-Based Software Package of Parallel Incremental Smoothed Finite Element Method (S-FEM) for Elastic-Plastic Problems. *Mathematics* **2022**, *10*, 2024. https://doi.org/10.3390/math10122024

Academic Editors: Fajie Wang and Ji Lin

Received: 12 May 2022
Accepted: 9 June 2022
Published: 11 June 2022

1. Introduction

Currently, numerical methods are the most important tools for solving various scientific and engineering problems [1]. For example, the Finite Element Method (FEM), one of the most successful numerical methods, has been widely employed in different scientific and engineering fields because of its mathematically rigorous proof and satisfactory efficiency [2–4]. However, the shortcomings and deficiencies of FEM are becoming increasingly significant [2,5–8]. (1) The FEM applies the problem domain of finite degrees of freedom to the problem domain of infinite degrees of freedom, which makes the system stiffness matrix "too rigid". (2) The conventional FEM has high requirements for mesh quality and cannot deal with distorted meshes. (3) When the conventional FEM uses simple and low-order elements to calculate large and complex structures, the calculation accuracy is often unsatisfactory, while when higher-order elements with higher accuracy are used, the computational cost is quite expensive.

To cope with the above deficiencies of FEM or decrease the computational cost of generating meshes, meshfree methods have emerged, such as Radial Point Interpolation Method (RPIM), Element Free Galerkin (EFG) and Meshless Local Petrov–Galerkin

(MLPG) [2,5,9,10]. The mesh-free methods can be used to analyze crack problems and large deformation problems because mesh-free methods employ a group of scattered nodes in the discrete problem domain, avoiding the requirement for continuity of the problem domain. However, the more complex computational process of mesh-free methods leads to the desire to achieve higher computational accuracy, which is not only computationally time-consuming but also inefficient [7].

In recent years, the Smoothed Finite Element Method (S-FEM), a new numerical method combining the FEM and strain smoothing technique was proposed by Liu G.R. et al. [7,11]. The system stiffness matrix of S-FEM model is softer than the FEM model for identical grid structures, has lower sensitivity to mesh distortion, and usually produces more accurate solutions and a higher convergence speed [7]. Due to the above characteristics, S-FEM is frequently used in the fields of material mechanics [12,13], dynamics [14,15], fracture mechanics [16], plate and shell mechanics [17], fluid structure interaction [18], acoustics [19], heat transfer [20] and biomechanics [21].

Typical S-FEM models include cell-based S-FEM (CS-FEM) [15,22], node-based S-FEM (NS-FEM) [23,24], edge-based S-FEM (ES-FEM) [14,16] for 2D and 3D problems and face-based S-FEM (FS-FEM) [25] for 3D problems. In addition, there are hybrid and modified types of S-FEM. For example, Chen et al. [26] proposed an edge-based smoothed extended finite element method, ESm-XFEM, for the analysis of linear elastic fracture mechanics. An improved ES-FEM method, bES-FEM, was proposed by Nguyen-Xuan et al. [27]. bES-FEM can be applied to almost incompressible and incompressible problems. Xu et al. [28] proposed a hybrid smoothed finite element method (H-SFEM) for solving solid mechanics problems by combining FEM and NS-FEM based on triangular meshes. Zeng et al. [29] proposed a beta finite element method (βFEM) based on the smooth strain technique applied to the modeling of crystalline materials.

Compared with FEM, the calculation of the S-FEM has the following two differences. First, we need to construct the smoothing domains and modify or reconstruct the strain field in the S-FEM. Moreover, because the smoothing domain may involve a portion of adjacent elements, the memory requirements for S-FEM will be larger [7]. The two differences mentioned above may lead to a higher computational cost for the S-FEM than the FEM for the same grid structure. However, given the calculation cost, the results calculated by the S-FEM model are more accurate than the FEM, and thus, achieve higher efficiency. To make S-FEM more applicable to large-scale engineering problems, parallel strategies of multicore CPUs and/or multicore GPUs are usually used to improve and optimize the computational power of S-FEM.

Currently, there are many software packages developed for utilizing FEM to solve various scientific and engineering problems, while the development of software and library packages for the S-FEM is still in progress [30]. Current S-FEM software packages are mostly implemented in C++ and Fortran. However, static languages such as Fortran and C/C++ have more complex language structures, are more difficult to learn, and require high programming skills. Although high-level dynamic languages, such as MATLAB and Python, are easy to learn, highly visual and interactive, the computing speed of dynamic language is slow and there are expensive licensing fees associated with the use of commercial software such as MATLAB. Julia is an efficient, user-friendly, open-source programming language, developed by MIT in 2009 [31]. Furthermore, it balances the problems of computing performance, programming difficulty and code legibility [32].

Many researchers have used the Julia language to develop software packages related to numerical computation. For example, Frondelius et al. [33] designed an FEM structure by using the Julia language, which enables large-scale FEM models to be processed by using distributed simple programming models across a cluster of computers. Sinaie et al. [34] implemented the Material Point Method (MPM) in the Julia language. In the large strain solid mechanics simulation, only Julia's built-in characteristics are used, which has better performance than the MPM code based on MATLAB. Zenan Huo et al. [35] implemented a package of S-FEM for linear elastic static problems by using Julia lan-

guage. Pawar et al. [36] developed a one-dimensional solver for the Euler equation, and an arakawa spectral solver and pseudo-spectral solver for the two-dimensional incompressible Navier–Stokes equation for the analysis of computational fluid dynamics using the Julia language. Heitzinger et al. [37] used the Julia language to implement numerical stochastic homogeneity of elliptic problems and discussed the advantages of using Julia to solve multiscale problems involving partial differential equations. Kemmer et al. [38] designed a finite element and boundary element solver using Julia to calculate the electrostatic potential of proteins in structural solvents. Fairbrother et al. [39] developed a package for Gaussian processes, GaussianProcesses.jl, using the Julia language. GaussianProcesses.jl takes advantage of the inherent computational benefits of the Julia language, including multiple assignments and just-in-time compilation, to generate fast, flexible and user-friendly packages for Gaussian processes.

In this paper, a parallel incremental S-FEM software package epSFEM for elastic-plastic problems is designed and implemented by utilizing the Julia language on a multi-core CPU. Distributed parallelism and partitioned parallelism are used for the assembly of the stiffness matrix, allowing multiple cells to be assembled simultaneously, avoiding excessive for loops and saving computation time. The system of equations is solved using the PARDISO [40] parallel sparse matrix solver. epSFEM applies to more common and complex elastic-plastic mechanical problems in practical engineering. Moreover, epSFEM adopts an incremental theory suitable for most load cases to solve elastic-plastic problems, and the calculation results are more reliable and accurate.

The contributions of this paper can be summarized as follows:

(1) A parallel S-FEM package epSFEM using incremental theory to solve elastic-plastic problems is developed by Julia language.

(2) The computational efficiency of epSFEM is improved by using distributed and partitioned parallel strategy on a multi-core CPU.

(3) epSFEM features a clear structure and legible code and can be easily extended.

The rest of this paper is organized as follows. The theory related to S-FEM and Julia language are presented in Section 2. The detailed implementation steps of the software package epSFEM are described in Section 3. Two sets of numerical examples are used to assess the correctness of the epSFEM and to evaluate its efficiency in Section 4. The performance, strengths and weaknesses of the epSFEM and the future direction of work are discussed in Section 5. Section 6 presents the main conclusions.

2. Background

In this section, the theoretical basis of the S-FEM and parallelization strategy of the Julia language on a multicore CPU are introduced.

2.1. Smoothed Finite Element Method (S-FEM)

2.1.1. Overview of the S-FEM

The S-FEM is the implementation of the FEM by employing the strain smoothing technique to modify or reconstruct the strain field such that more accurate or special performance solutions can be obtained. NS-FEM, for example, has an upper bound solution to the model because of its weak super-convergence, insensitivity to mesh deformation and an overly soft system stiffness matrix. In the ES-FEM and FS-FEM models, there are no unphysical modes, so both methods give good results for dynamic and static problems. In the S-FEM, the most important goal is the modification of the compatible strain field or reconstruction of the strain field only from the displacement field [11]. To guarantee the stable and convergent properties of the established S-FEM model, this strain modification or reconstruction needs to be conducted in an appropriate way to obtain special characteristics. Strain modification or reconstruction can be implemented within the element, but it is generally conducted across the element to obtain more information from adjacent elements. Different modification or reconstruction methods correspond to separate S-FEMs, that is, CS-FEM, NS-FEM, ES-FEM and FS-FEM.

For two-dimensional static problems, ultra accurate numerical solutions can be obtained using ES-FEM, and the calculation results of ES-FEM based on T3 elements are even more accurate than traditional FEM with Q4 elements (same number of nodes) [11,14]. Therefore, the ES-FEM is employed to solve the two-dimensional elastic-plastic problem in this paper, and the implementation steps are introduced as follows.

2.1.2. Workflow of the ES-FEM

The ES-FEM calculation process is similar to that of the FEM, except that the ES-FEM needs to construct a smoothing domain on the basis of the FEM model and modify or reconstruct the strain field. As shown in Figure 1, many techniques designed for FEM can be adapted for ES-FEM. In short, the difference between the ES-FEM and FEM is that all calculations of the FEM are based on elements, while all calculations of the ES-FEM are conducted on smoothing domains.

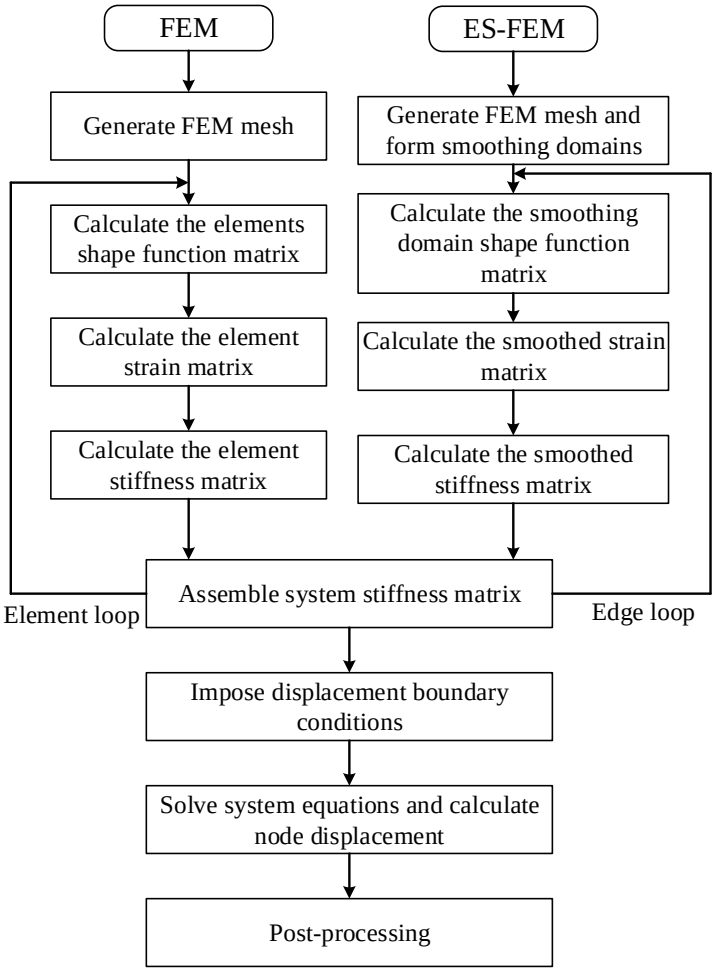


Figure 1. Flow chart of the FEM and ES-FEM.

The two-dimensional solid mechanics problem with problem domain Ω and boundary $\Gamma = \Gamma_u \cup \Gamma_t$ are considered, where Γ_u is the essential boundary where displacement conditions are prescribed and Γ_t is the natural or force boundary.

The calculation procedure of ES-FEM is as follows [7,11,14]:

(1) Discretization of the problem domains and construction of the smoothing domains

In the ES-FEM, general polygonal elements are used to divide the problem domain, mainly T3 elements suitable for solving two-dimensional problems. When the T3 element is used, the meshing can be the same as the standard FEM, such as the widely used Delaunay triangulation method.

As shown in Figure 2, on the basis of the polygonal element mesh, the smoothing domain is constructed. The problem domain is divided into N_e polygonal elements, including N_{eg} edges. The edge-based smoothing domain is composed of two nodes connecting one edge and the centroid of its adjacent elements. The two nodes A and B connecting edge AB and centroid D of the triangle element form the smoothing domain (ABD), see Figure 2. The construction of the smoothing domain, such as the discrete problem domain, must follow the principle of no gap and no overlap, that is, $\Omega = \Omega_1^s \cup \Omega_2^s \cup \ldots \cup \Omega_{N_e}^s$, $\Omega_i^s \cap \Omega_j^s = \varnothing$, and $i \neq j$.

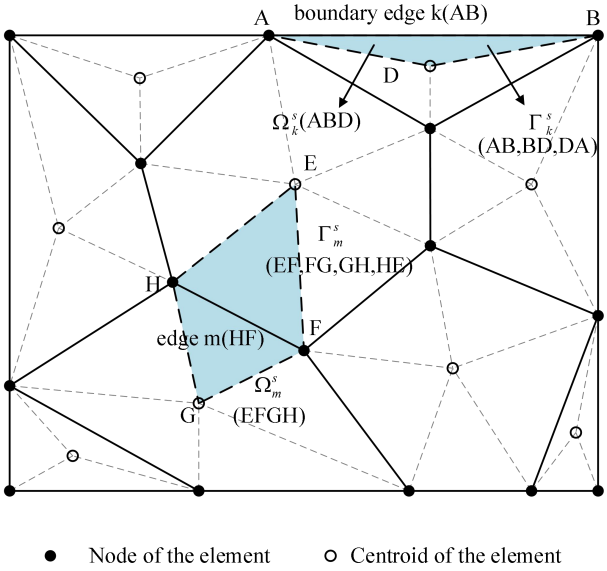


Figure 2. Polygon element mesh and the edge-based smoothing domain in ES-FEM.

(2) Creation of the displacement field

The generalized displacement field $\tilde{\mathbf{u}}$ at any point in the triangular element is approximated as:

$$\tilde{\mathbf{u}} = \sum_{i=1}^{N_n} \mathbf{N_i}(\mathbf{x})\tilde{\mathbf{d}}_i \tag{1}$$

where N_n is the number of smoothing domain nodes, $\tilde{\mathbf{d}}_i$ is the nodal displacement at node i, and $\mathbf{N_i}(\mathbf{x})$ is the shape function:

$$\mathbf{N_i}(\mathbf{x}) = \begin{bmatrix} N_i(x) & & \\ & \ddots & \\ & & N_i(x) \end{bmatrix}_{n \times n} \tag{2}$$

where n is the degree of freedom of the smoothing domain nodes.

The Gauss integration point interpolation distribution of the ES-FEM shape function is illustrated in Figure 3. As shown in Figure 3, the commonly used linear triangular elements are employed to divide the mesh. Here, the shape function values at the Gauss integral point are calculated in two cases: boundary edge and internal edge. The results are shown in Tables 1 and 2.

(3) Construction of the smoothed strain field

For triangular, quadrilateral and polygonal elements, strain smoothing techniques can be used to construct the strain field directly from the boundary integrals of the shape function without the need for coordinate mapping.

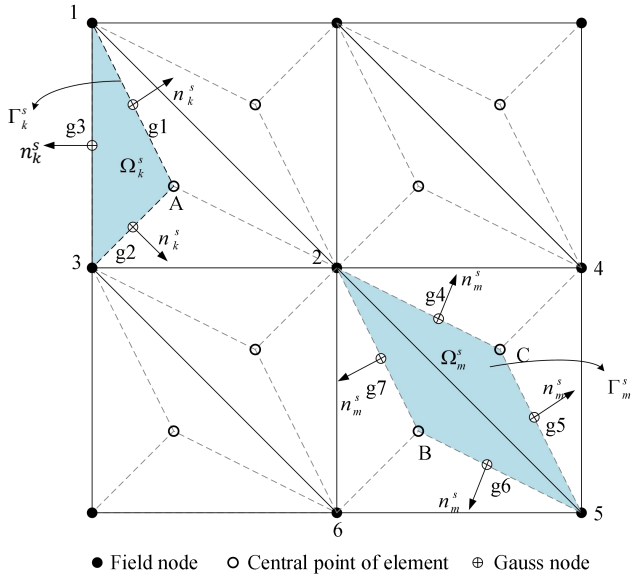


Figure 3. Illustration of the interpolation distribution of the Gaussian integration points of the ES-FEM shape function.

Table 1. The shape function entries at different points on the boundary of the smoothing domain connected to the outer edge 1–2 in Figure 3.

Node Number	1	2	3	Node Attributes
1	1.0	0.0	0.0	Field node
2	0.0	1.0	0.0	Field node
3	0.0	0.0	1.0	Field node
A	1/3	1/3	1/3	Centroid of element
g1	1/2	1/2	0.0	Gauss point
g2	1/6	4/6	1/6	Gauss point
g3	4/6	1/6	1/6	Gauss point

Table 2. The shape function entries at different points on the internal smoothing domain connected with the inner edges 3–5 in Figure 3.

Node Number	3	4	5	6	Node Attributes
3	1.0	0.0	0.0	0.0	Field node
4	0.0	1.0	0.0	0.0	Field node
5	0.0	0.0	1.0	0.0	Field node
6	0.0	0.0	0.0	1.0	Field node
B	1/3	1/3	1/3	0.0	Centroid of element
C	1/3	0.0	1/3	1/3	Centroid of element
g4	4/6	1/6	1/6	0.0	Gauss point
g5	1/6	1/6	4/6	0.0	Gauss point
g6	1/6	0.0	4/6	1/6	Gauss point
g7	4/6	0.0	1/6	1/6	Gauss point

In the ES-FEM, the smoothed strain $\bar{\varepsilon}$ is computed as follows:

$$\bar{\varepsilon} = \int_{\Omega_k^s} \tilde{\varepsilon}(\mathbf{x}) \Phi(x) d\Omega \tag{3}$$

where $\tilde{\boldsymbol{\varepsilon}}(\mathbf{x}) = \mathbf{L_d u}$ is the strain that satisfies the compatibility condition in the traditional FEM, $\Phi(x)$ is the smoothing function, and Ω_k^s is the smoothing domain, which can be defined as follows:

$$\Phi(x) = \begin{cases} \frac{1}{A_k^s}, x \in \Omega_k^s \\ 0, x \notin \Omega_k^s \end{cases} \tag{4}$$

where A_k^s is the area of the smoothing domain.

Combining the Gaussian divergence theorem, the domain integral is transformed into the edge integral to obtain the following smoothed strain calculation equation:

$$\bar{\boldsymbol{\varepsilon}} = \int_{\Omega_k^s} \tilde{\boldsymbol{\varepsilon}}(\mathbf{x}) d\Omega = \int_{\Omega_k^s} \mathbf{L_d} \tilde{\mathbf{u}}(\mathbf{x}) d\Omega = (1/A_k^s) \int_{\Gamma_k^s} \mathbf{L_n}(\mathbf{x}) \tilde{\mathbf{u}}(\mathbf{x}) d\Gamma, x \in \Omega_k^s \tag{5}$$

where $\mathbf{L_d}$ is the partial differential matrix operator, $\mathbf{L_n}$ is the outward unit normal vector and Γ_k^s is the boundary of the edge-based smoothing domain.

$$\mathbf{L_n}(\mathbf{x}) = \begin{bmatrix} n_x & 0 \\ 0 & n_y \\ n_y & n_x \end{bmatrix} \tag{6}$$

where n_x and n_y are the x-axis and y-axis components of the normal vector outside the unit, respectively.

Similar to the FEM, the smoothed strain field is divided into:

$$\bar{\boldsymbol{\varepsilon}}(\mathbf{x}) = \sum_I^{N_n} \bar{\mathbf{B}}_\mathbf{I}(\mathbf{x_k}) \mathbf{d_I} \tag{7}$$

where $\mathbf{B_I}$ is the smoothed strain matrix:

$$\bar{\mathbf{B}}_\mathbf{I}(\mathbf{x_k}) = \begin{bmatrix} \bar{b}_{Ix}(x_k) & 0 \\ 0 & \bar{b}_{Iy}(x_k) \\ \bar{b}_{Iy}(x_k) & \bar{b}_{Ix}(x_k) \end{bmatrix} \tag{8}$$

where $\bar{b}_{Ix}(x_k)$ and $\bar{b}_{Iy}(x_k)$ is defined as shown in Equation (9). The boundary integral method is used to solve the smoothed strain matrix. This method is applicable to any polygonal geometry in the smoothing domain.

$$\begin{cases} \bar{b}_{Ix} = (1/A_k^s) \int_{\Gamma_k^s} n_x N_I(x) d\Gamma = (1/A_k^s) \sum_{i=1}^{N_I} n_{i,x} N_I(x_i^G) l_i \\ \bar{b}_{Iy} = (1/A_k^s) \int_{\Gamma_k^s} n_y N_I(x) d\Gamma = (1/A_k^s) \sum_{i=1}^{N_I} n_{i,y} N_I(x_i^G) l_i \end{cases} \tag{9}$$

where N_I is the number of segments of Γ_k^s, $n_{i,x}$ and $n_{i,y}$ are the outer normal vectors of the Ith integration segment, x_i^G is the midpoint of each segment of the boundary, that is, the Gauss integration point, and $N_I(x_i^G)$ is the shape function value at the Gauss integration point.

(4) Establishment system of equations

The smoothed Galerkin weak form is utilized to establish the system equation in the ES-FEM. During this process, only a simple summation calculation of the relevant parameters of the smoothing domain is required.

The linear system of equations of ES-FEM is:

$$\bar{\mathbf{K}}^{\mathbf{ES-FEM}} \bar{\mathbf{d}} = \tilde{\mathbf{f}} \tag{10}$$

where $\bar{\mathbf{d}}$ is the displacement vector of all nodes in the S-FEM and $\tilde{\mathbf{f}}$ is the vector of all loads. $\bar{\mathbf{K}}^{\mathbf{ES-FEM}}$ is the system stiffness matrix of the ES-FEM and defined as Equation (11):

$$\bar{\mathbf{K}}_{\mathbf{IJ}}^{\mathbf{ES-FEM}} = \sum_{k=1}^{N_{eg}} \int_{\Omega_k^s} \bar{\mathbf{B}}_\mathbf{I}^\mathbf{T} \mathbf{c} \bar{\mathbf{B}}_\mathbf{J} d\Omega = \sum_{k=1}^{N_{eg}} \bar{\mathbf{B}}_\mathbf{I}^\mathbf{T} \mathbf{c} \bar{\mathbf{B}}_\mathbf{J} A_k^s \tag{11}$$

where **c** is the matrix of elasticity coefficients.

(5) Imposition of the boundary conditions

In the ES-FEM, the application process of displacement boundary conditions is similar to that of the FEM because the shape function used in the S-FEM has the same delta property as the FEM. The main methods include the direct method, set "1", multiple large numbers, Lagrange multiplier and penalty function. The force boundary conditions are added directly to the corresponding nodes.

(6) Postprocessing

The weighted average rule is used to obtain the equivalent nodal stress in the smoothing domain, and the shape function interpolation technique is used to obtain the continuous stress field in the problem domain. The process is similar to the FEM. Finally, the accuracy of the results is assessed in relation to the actual problem.

2.2. The Julia Language

Julia, officially released in 2012, is a flexible dynamic language for scientific and numerical computation [41]. To solve large-scale numerical computation problems, parallel computing is considered essential. There are useful built-in features in Julia that make it easier for developers to design efficient parallel code. Three of the parallel strategies, that is, coroutine, multithreaded and distributed computing, are dependent on a multicore CPU. Developers can select the appropriate parallelism method for their needs. Parallel computing on many-core GPUs can be conducted by using specific packages or utilizing the built-in function of Julia and parallel arrays [1].

In this paper, parallelism on a multicore CPU is applied to effectively improve the calculation and assembly efficiency of the global stiffness matrix. In the Julia language, distributed computing based on a multicore CPU first redistributes tasks according to the number of CPU cores of the computer and then dynamically allocates computing tasks to each process so that multiple processes can be calculated at the same time, thus improving the computing efficiency. In the parallel computing of Julia language, "SharedArray" is used to reduce memory usage and improve computational efficiency. Moreover, when a "SharedArray" is employed, multiple processes are allowed to operate on the same array in the meantime [42,43].

3. The Implementation of Package epSFEM

3.1. Overview

A parallel S-FEM package using incremental theory to solve elastic-plastic problems is developed on a multicore CPU. This package contains the following three components:

Preprocessing: The preprocessing includes mesh generation and the construction of smoothing domains based on the mesh. After the preprocessing is completed, the model details of constructing the smoothing domain can be obtained, and stored in sparate five files: nodes, elements, internal edges, external edges and the centroids of mesh elements.

Solver: The solver uses incremental S-FEM to solve the elastoplastic problems, which is the main part of the whole software package. It is mainly categorized into: (1) assembly of the elastic stiffness matrix and (2) incremental loading and semismooth Newton method iterations to solve the system of equations. The calculation procedure of the incremental loading and semismooth Newton method iterations of the solver is illustrated in Figure 4.

Postprocessing: ParaView [44] is utilized to visualize the numerical calculation results. The WriteVTK.jl package in Julia is used to write the "vtu" format file needed for ParaView visualization.

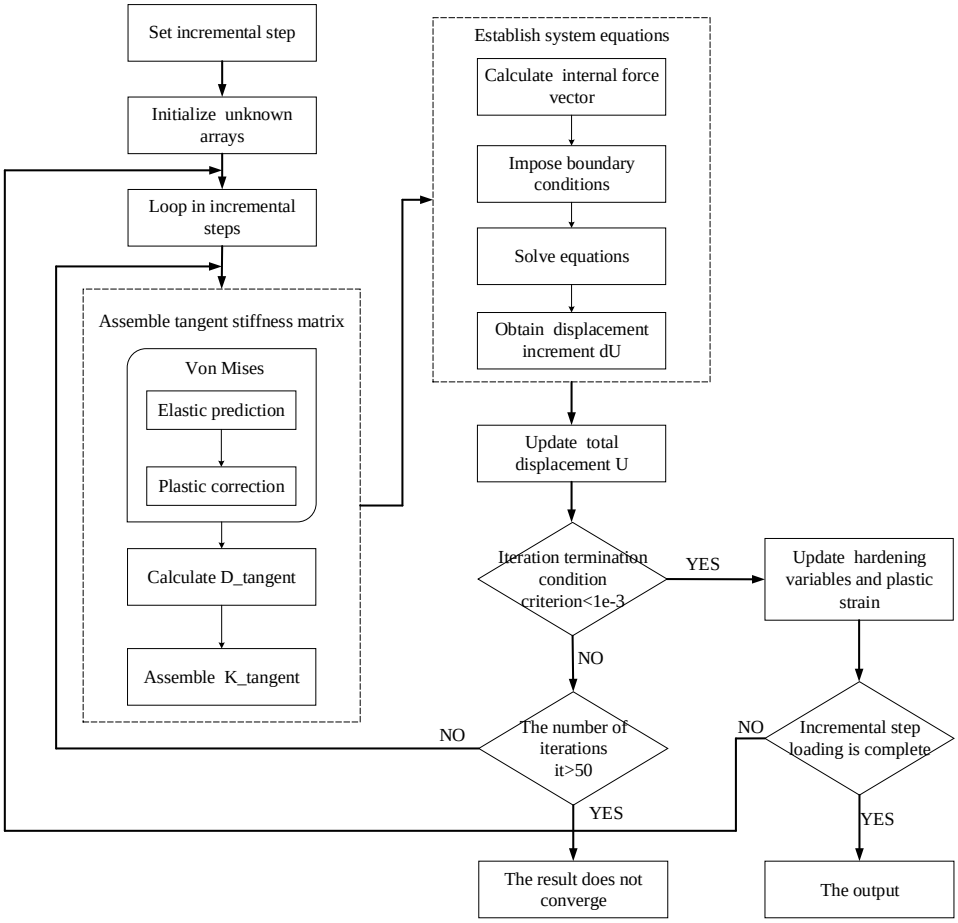


Figure 4. Illustration of the calculation procedure of incremental loading and semismooth Newton method iterations.

3.2. Preprocessing

3.2.1. Mesh Generation

Mesh generation is the first step of the numerical analysis, which also affects the accuracy and efficiency of the numerical analysis. Currently, there are many mature mesh generation algorithms and software. The focus of this paper is on the solver section, so a simple direct generation method is used to generate the mesh and then divide the smoothing domain on this basis. Because T3 elements have good adaptability and are most used in science and engineering practice, we choose T3 elements to divide the problem domain.

3.2.2. Construction of the Smoothing Domain

Constructing the smoothing domain based on the meshing of the problem domain is one of the key tasks of the S-FEM. According to the methods of constructing the smoothing domain and storing model information in Refs. [35,45], the smoothing domain of the mesh is constructed, and the model information after dividing the smoothing domain is output. To get the best performance out of the Julia language, the following calculations can be looped in the unit of column, and the model information is stored based on the column. In this paper, we address the mesh details by integrating the features of ES-FEM and Julia parallel computation and then utilize five matrices to save the mesh details in an appropriate way; see Figure 5.

The "Node" matrix stores the x and y coordinates of the mesh nodes. The "Centroid" matrix stores the x and y coordinates of the center of the cell. The node numbers corresponding to the mesh cells are stored in the "Element". The three node numbers of the

triangular cells are stored in the three rows of "Element" in a counterclockwise order, and the number of columns is the number of mesh cells.

In the ES-FEM, the smoothing domain is constructed by using edges as the basis. We divide all the edges of the model into two categories: the outer edges are saved in the "Edge_out" matrix, and the inner edges are saved in the "Edge_in" matrix. For the matrix "Edge_out", the two node numbers of the outer edge are stored in the first two rows, the serial number of the triangle is appended to the third row, and the rest point of the triangle is appended to the fourth row. Because one inner edge belongs to two triangles, the first two rows store the node numbers of the inner edges, the third and fourth rows of "Edge_in" are the serial numbers of neighboring triangles and the last two rows are the numbers of the other points in the triangle.

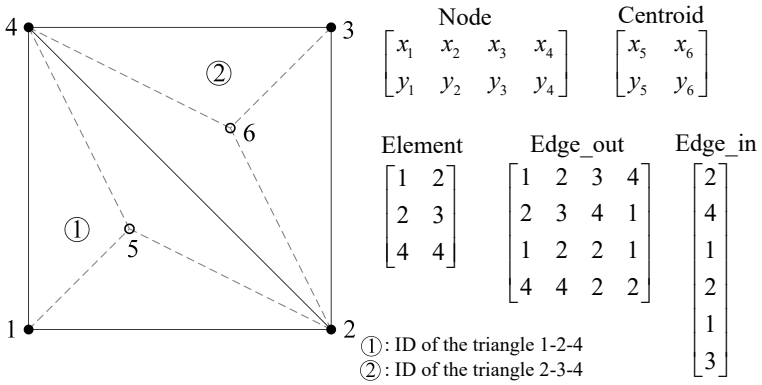


Figure 5. Illustration of the matrices "Node", "Centroid", "Element", "Edge_out" and "Edge_in", using two adjacent triangles as examples.

3.3. Solver

The incremental S-FEM is utilized to address the elastoplastic problem, choosing the implicit constitutive integration algorithm of the linear kinematic hardening Von Mises constitutive model and the corresponding consistent tangent modulus. First, the elastic predicted stress is calculated according to the strain of the equilibrium iteration, and then the modified stress is calculated according to a certain direction to make the stress return to the updated yield surface [46,47]. The nonlinear equations are solved by employing the semismooth Newton method.

The solution process is composed of two major procedures: (1) assembly of the elastic stiffness matrix and (2) incremental loading and semismooth Newton method iterations. The second procedure is composed of multiple incremental step cyclic calculations. Each incremental step can be divided into three steps: (1) assembly of tangent stiffness matrix, (2) solving of equations and (3) updating of hardening variables and plastic strain. According to the characteristics of parallel computing, the calculation of the latter step cannot be dependent on the previous step, so when assembling the elastic stiffness matrix, the tangent stiffness matrix can be calculated in parallel to improve efficiency. Distributed computing is used in Julia to calculate the elastic stiffness matrix and the tangent stiffness matrix for multiple elements in parallel. When solving the overall nonlinear system equations, we utilize the semismooth Newton method for each iteration. For the set of equations in each iteration, a parallel sparse equation solver, PARDISO, is used [40]. The detailed procedure of the solver in epSFEM will be presented in the subsequent sections.

3.3.1. Assembly of Elastic Stiffness Matrix

After the model is preprocessed, it needs to be assembled with an elastic stiffness matrix first. In epSFEM, we calculate the stiffness matrix of the associated smoothing domain by dividing the outer edge and the inner edge, and the calculation process is basically the same. Taking the internal edge as an example:

(1) The areas of the two triangle elements that share the inner edge is attached are computed. This process is conducted by procedure "area.jl" according to Equations (12)–(16):

$$a = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{12}$$

$$b = \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2} \tag{13}$$

$$c = \sqrt{(x_3 - x_1)^2 + (y_3 - y_1)^2} \tag{14}$$

$$p = (a + b + c)/2 \tag{15}$$

$$A = \sqrt{(p(p-a)(p-b)(p-c)} \tag{16}$$

where xi and yi is the coordinate of the node i.

(2) The length of each edge of the smoothing domain is calculated in Equation (17). For the smoothing domain of the inner side, there are four edges. This process is realized in the file "lp.jl":

$$lp = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{17}$$

(3) The normal outward vector $v1$ is calculated for each side of the smoothing domain. This is performed in "vectorin.jl", according to Equations (18)–(20):

$$y = y_2 - y_1 \tag{18}$$

$$x = x_1 - x_2 \tag{19}$$

$$v1 = \left[\frac{y}{\sqrt{(y^2 + x^2)}} \frac{x}{\sqrt{(y^2 + x^2)}} \right] \tag{20}$$

(4) Assembly of the global elastic stiffness matrix and the global smoothed strain matrix. First, the stiffness matrix of the smoothing domain element is computed and then assembled according to the smooth domain nodes. Due to the large number of zero elements in the matrix, to reduce the memory occupation, the matrix is stored in a sparse form. There are three common ways to construct sparse arrays: Compressed Sparse Row (CSR), Compressed Sparse Column (CSC) and COOrdinate (COO).

First, the COO format is used to construct the global elastic stiffness matrix, since the multi-dimensional arrays in Julia are stored according to column-based sequence. Then, for the convenience of solving the subsequent system equations, we replace it with the CSC format. To construct a sparse array according to COO format, we first need to construct three one-dimensional arrays, that is, IK_elast, JK_elast and VK_elast.

IK_elast, JK_elast and VK_elast denote the row number, column number and value of each entry in the global stiffness matrix according to the order of each row. Since the sparse functions can accumulate the entries at the same position automatically, the magnitudes of IK_elast, JK_elast and VK_elast can be predetermined.

The assembly method of the global smoothed strain matrix is basically the same as the stiffness matrix, except that when it is assembled, the rows are carried out according to the elements, and the columns are carried out according to the nodes. Three one-dimensional arrays, IB, JB and VB, are constructed in advance.

For parallel computing, the six arrays of IK_elast, JK_elast, VK_elast, IB, JB and VB need to be converted to "SharedArrays" in advance, and the elastic stiffness matrix is assembled in parallel using the "@distributed" macro in Julia. Because the stiffness matrix calculation of each element has no data dependence, there will be no data interference when performing parallel computing.

The number of processes needs to be added using the function "addprocs" before all parallel computing starts. In the parallel elastic stiffness matrix assembly, we use the "@distributed" macro to automatically allocate tasks to each process according to the

number of processes and the total number of tasks for parallel computing of the loop. The total number of tasks currently is equal to the total number of smoothing domains. The "@distributed" macro is executed asynchronously on the loop; it will generate independent tasks on all available processes and return immediately without waiting for the computing to complete. To wait for the computing task to complete, the "@sync" macro must be used before the call. The procedure of assembling the elastic stiffness matrix for the internal edges by distributed parallel computation is illustrated in Algorithm 1. After the global stiffness matrix and the global smoothed strain matrix are assembled, the "Sparse" function is used to convert them into the CSC format.

Algorithm 1 Parallel calculation and assembly of the elastic stiffness matrix

Input: Node, Centroid, Element, Edge_in, shear, bulk
Output: K_elast, B
Set the number of CPU cores for the Julia program.
Set IK_elast, JK_elast, VK_elast, IB, JB and VB to SharedArrays.
@sync @distributed **begin**
for every internal edge **do**
  Compute the area of the interior quadrilateral.
  Compute the side lengths of the interior quadrilateral.
  Compute the normal unit vectors of the four sides of the interior quadrilateral.
  Compute the smoothed strain matrix of an interior quadrilateral.
  Compute the stiffness matrix of an interior quadrilateral.
  Compute the elastic coefficient matrix.
  Assemble global stiffness matrix and smoothed strain matrix.
end for
end
K_elast = sparse (IK_elast, JK_elast, VK_elast)
B = sparse (IB, JB, VB)

3.3.2. Assembly of the Tangent Stiffness Matrix

The tangent stiffness matrix of the model needs to be calculated when solving the elastic-plastic problem using incremental theory. Equation (21) is used instead of Equation (22) to calculate the global tangent stiffness matrix. Among them, elastic stiffness matrix $\mathbf{K_{elast}}$, smoothed strain matrix $\mathbf{B}$ and elastic matrix $\mathbf{D_{elast}}$ can be obtained in advance at the stage of assembling the elastic stiffness matrix; only elastoplastic matrix $\mathbf{D_{tangent}}$ depends on the plastic model, and must be partially reorganized or modified in each Newton iteration. When most portions of the model are in the elastic stage, $\mathbf{D_{tangent}} - \mathbf{D_{elast}}$ is more sparse than $\mathbf{D_{tangent}}$ [48,49].

$$\mathbf{K_{tangent}} = \mathbf{K_{elast}} + \mathbf{B^T}(\mathbf{D_{tangent}} - \mathbf{D_{elast}})\mathbf{B} \tag{21}$$

$$\mathbf{K_{tangent}} = \mathbf{B^T}\mathbf{D_{tangent}}\mathbf{B} \tag{22}$$

$\mathbf{D_{tangent}}$ is calculated by the constitutive integral. The implicit discrete method is used to solve the constitutive integral, that is, elastic prediction and plastic correction. For the constitutive relation, the linear kinematic hardening Von Mises model is employed.

The steps to calculate the tangent stiffness matrix are as follows:

(1) Calculation of the smoothed strain field. Since the global smoothed strain matrix $\mathbf{B}$ has been calculated and assembled in the stage of assembling the elastic stiffness matrix, the smoothed strain field $\boldsymbol{\varepsilon}$ can be acquired according to the strain coordination Equation $\boldsymbol{\varepsilon} = \mathbf{Bu}$.

(2) The implicit Von Mises constitutive integral algorithm is used to obtain the stress **S** and tangent operator **DS** of the model, by procedure "constitutive_problem1.jl". The formula, according to [48–51], is:

$$T_k(\varepsilon_k) = \begin{cases} \sigma_k^{tr}, \left|\mathbf{s}_k^{tr}\right| \le Y, \\ \sigma_k^{tr} - \frac{2G}{2G+a}(\left|\mathbf{s}_k^{tr}\right| - Y)\mathbf{n}_k^{tr}, \left|\mathbf{s}_k^{tr}\right| > Y \end{cases} \tag{23}$$

where $T_k(\varepsilon_k)$ represents the stress–strain operator, $\sigma_k^{tr} = C(\varepsilon_k - \varepsilon_{k-1}^p)$, $\mathbf{s}_k^{tr} = \mathrm{I}_D\sigma_k^{tr} - \beta_{k-1}$, $\mathbf{n}_k^{tr} = \frac{\mathbf{s}_k^{tr}}{\left|\mathbf{s}_k^{tr}\right|}$, a is the hardening parameters and Y is the yield stress.

$$T_k^0(\varepsilon_k) = \begin{cases} C, \left|\mathbf{s}_k^{tr}\right| \le Y, \\ C - \frac{4G^2}{2G+a}\mathrm{I}_D + \frac{4G^2}{2G+a}\frac{Y}{\left|\mathbf{s}_k^{tr}\right|}(\mathrm{I}_D - \mathbf{n}_k^{tr} \otimes \mathbf{n}_k^{tr}), \left|\mathbf{s}_k^{tr}\right| > Y \end{cases} \tag{24}$$

where $T_k^0(\varepsilon_k)$ is the derivative of the stress–strain operator, $C = K\mathrm{I} \otimes \mathrm{I} + 2G\mathrm{I}_D$, $\mathrm{I} \otimes \mathrm{I}$ is the unit second-order tensor, $\mathrm{I}_D = \mathrm{I} - \frac{\mathrm{I}\otimes\mathrm{I}}{3}$, $K = E/3(1-2\mu)$ is the bulk modulus and $G = E/2(1+\mu)$ is the shear modulus.

The modification of hardening variable β_k and plastic strain ε_k^p is:

$$\beta_k = \begin{cases} \beta_{k-1}, \left|\mathbf{s}_k^{tr}\right| \le Y, \\ \beta_{k-1} + \frac{a}{2G+a}(\left|\mathbf{s}_k^{tr}\right| - Y)\mathbf{n}_k^{tr}, \left|\mathbf{s}_k^{tr}\right| > Y \end{cases} \tag{25}$$

$$\varepsilon_k^p = \begin{cases} \varepsilon_{k-1}^p, \left|\mathbf{s}_k^{tr}\right| \le Y, \\ \varepsilon_{k-1}^p + \frac{1}{2G+a}(\left|\mathbf{s}_k^{tr}\right| - Y)\mathbf{n}_k^{tr}, \left|\mathbf{s}_k^{tr}\right| > Y \end{cases} \tag{26}$$

where β_{k-1} hardening tensor from the previous incremental step and ε_{k-1}^p plastic strain tensor from the previous incremental step.

To check whether plastic correction is needed, the array CRIT of $1 \times s\_n\_e$ is defined representing the yield criterion, that is, $\left|\mathbf{s}_k^{tr}\right| - Y$, and the corresponding logical array IND_p of $1 \times s\_n\_e$ with the smoothing domain of plastic behavior, where s_n_e represents the total number of smoothing domains. The parallel implementation of the implicit Von Mises constitutive integral is shown in Algorithm 2.

In the parallel computing of constitutive integrals, all processes can access the underlying data. To avoid conflicts, we first construct a "myrange" function to assign tasks to each process according to the number of CPU cores added. Then, the main computing process is defined as a kernel function "assembly_tangent", and a wrapper "shared_constructive" is defined to encapsulate the kernel function. Finally, the function "constitutive_problem" is constructed to call the packaged kernel function for partition parallel computing. The "constitutive_problem" function minimizes the communication between the processes so that each process can continue to compute the allocated part for a period of time, and improve the efficiency of parallelism. The "@async" macro is used to wrap arbitrary expressions into tasks. For any content within its scope, Julia will start to run this task and then continue to execute the next code in the script without waiting for the current task to complete before executing it. The "@sync" macro means that the next task will not be executed until the dynamic closure defined by the macro "@async" is completed.

(3) Calculation of the global tangent stiffness matrix. First, the sparse elastoplastic matrix $\mathbf{D_{tangent}}$ is constructed according to the tangent operator **DS** obtained by the constitutive integral and then the global tangent stiffness matrix is calculated according to Equation (21).

Algorithm 2 Parallel implementation of implicit Von Mises constitutive integral algorithm

Input: E, Ep_prev, Hard_prev, shear, bulk, a, Y, S, DS, IND_p
Output: S, DS, IND_p

```
Set the number of CPU cores for the Julia program.
Set E, Ep_prev, Hard_prev, S, DS, IND_p to ShareArray.
Assign the number of tasks for each process according to the number of processes.
for number of tasks in each process do
    Check whether the smoothing domain yields according to the yield criterion.
    Elastic prediction of stress tensor.
    Calculate the consistent tangent operator.
    Plastic correction of the stress tensor.
    Plastic correction of the consistent tangent operator.
end for
Parallel computing using "remotecall" in Julia language.
```

3.3.3. Solution of System of Equations

In this process, the internal force of the model is calculated by using the stress obtained from the constitutive relationship and the smoothed strain matrix. Then, the displacement boundary conditions are applied by the direct method; that is, the corresponding rows and columns with displacement boundary conditions of "0" are deleted. A logical array Q is designed, which sets the displacement boundary condition of "0" to "0" and the rest to "1". Then, the stiffness matrix, displacement and force are calculated with a logical array index. After that, the Pardiso.jl package is added and the "MKLPardisoSolver" solver in the package is used to solve the system of equations. Finally, the node displacement increment "$\mathbf{dU}$" of one Newton iteration in an incremental step can be obtained.

In this paper, the semismooth Newton method is employed to solve nonlinear system of equations and check whether iteration is convergent according to Algorithm 3. "MKLPardisoSolver" is the solver in the Pardiso.jl package, Q is the logical array corresponding to the displacement boundary conditions, $\mathbf{f}$ is the external force vector, $\mathbf{F}$ is the internal force vector, the subscript k represents the kth incremental step and the superscript it represents the it-th iteration step, and $\|\mathbf{U}\|_e^2 = \mathbf{U}^{\mathbf{T}}\mathbf{K}_{\mathbf{elast}}\mathbf{U}$. In each Newton iteration, the tangent stiffness matrix $\mathbf{K}_{\mathbf{tangent}}$ is used to solve the linear problem, which corresponds to the system of linear equations:

$$\mathbf{K}_k^{it}\mathbf{dU}^{it} = \mathbf{f}_k - \mathbf{F}_k \tag{27}$$

Algorithm 3 Newton iteration terminates judgment

```
initialization U_k^0 = U_k
for it = 1, 2, 3 ... do
    ps = MKLPardisoSolver()
    dU^it[Q] = solve(ps, K_k^it[Q1, Q1], (f_k − F_k^it))
    U_k^it = U_k^{it−1} + dU^it
    ||dU^it||_e / (||U_k^{it−1}||_e + ||U_k^it||_e) ≤ criterion
end for
set U_k = U_k^it
```

3.3.4. Update of Hardening Variable and Plastic Strain

After each incremental step is calculated, the hardening variable and plastic strain need to be updated by using Equations (25) and (26). Based on the implicit constitutive integration algorithm of Algorithm 2, the modification of the hardening variable and plastic strain is added. The parallel strategy in this part is consistent with Algorithm 2.

3.4. Postprocessing

After the execution of the solver, the widely used visualization software ParaView is used to visualize the numerical computational results. The relevant package WriteVTK.jl in Julia can write VTK XML files and use ParaView to visualize multidimensional datasets [44]. The VTK format files support include straight line (.vtr), structured mesh (.vts), image data (.vti), unstructured mesh (.vtu) and polygon data (.vtp) [52].

An unstructured mesh "vtu" format file is designed. Its implementation steps are as follows: (1) we need to define a cell type, which is defined in this paper as "VTKCellTypes.VTK_TRIANGLE", representing the linear triangular element; (2) the "MeshCell" function is used to define the mesh model and obtain an array containing all mesh cells; (3) to generate a "vtu" format file, we need to initialize the file with mesh nodes and element information and then add node displacement data and other information to the file; (4) we can save the file as a "vtu" format file.

4. Validation and Evaluation of epSFEM

In this section, two sets of benchmark tests are performed on a powerful computational platform to evaluate the correctness and efficiency of epSFEM. The details of the workstation computer used are shown in Table 3.

Table 3. Specifications of the workstation computer for performing the benchmark tests.

Specifications	Details
CPU	Intel Xeon Gold 5118 CPU
CPU Cores	24
CPU Frequency	2.30 GHz
CPU RAM	128 GB
OS	Windows 10 professional
IDE	Visual studio Code
Julia	Version 1.5.2

4.1. Validation of the Accuracy of epSFEM

To validate the correctness of epSFEM, we use the model shown in Figure 6a to perform elastoplastic analysis and compare its calculation accuracy with traditional finite element software. In this example, a symmetric displacement boundary condition is set up on the left and bottom of the computational model. The traction force of F_t = 200 N/m acts on the top of the model along the normal direction, and the traction force is added in increments through the cyclic load shown in Figure 6b. The elastic parameters are: E = 206,900 (Young's modulus) and $\mu = 0.29$ (Poisson's ratio). The parameters related to plastic materials are specified as follows: $a = 1000$, $Y = 450\sqrt{(2/3)}$. The mesh computational model with 150 triangular elements is illustrated in Figure 7a, and the computational model after constructing the smoothing domain is shown in Figure 7b.

To demonstrate the accuracy of the calculation, the displacement calculation of the model in Figure 6a is conducted, and comparisons are made in the three following cases.

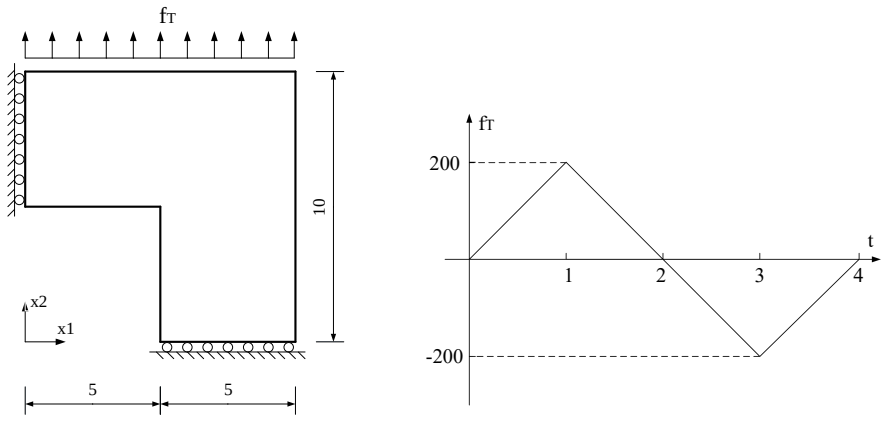
(1) epSFEM is employed to calculate the displacement of a mesh model, which includes 341 nodes and 600 triangular elements (T3 elements); see Figure 8a.

(2) According to Ref. [49], the conventional FEM is used to calculate the displacement of a mesh model, which includes 341 nodes and 600 triangular elements (T3 elements); see Figure 8b.

(3) According to Ref. [49], the conventional FEM is employed to calculate the displacement of a highly accurate mesh model that includes 231,681 nodes and 76,800 eight-node quadrilateral elements (Q8 elements).

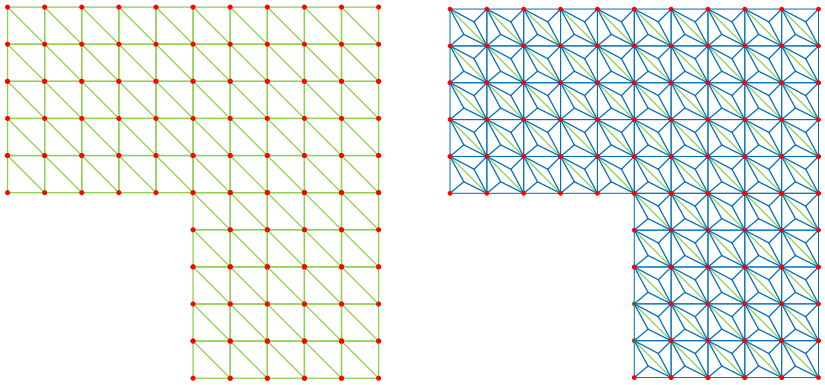
The displacements of the top node of the model calculated by the above three methods are compared in Figure 9. As shown in Figure 9, the displacement calculated by epSFEM

has higher accuracy than FEM-T3 and slightly lower accuracy than FEM-Q8. Hence, the correctness of epSFEM is proven.



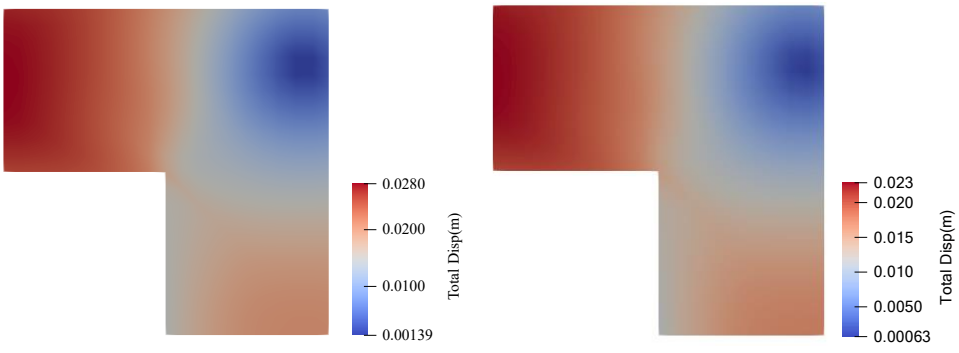
(**a**) (**b**)

Figure 6. (**a**) Simplified 2D geometry of the elastic-plastic problem and (**b**) history of the traction force.



(**a**) (**b**)

Figure 7. (**a**) A mesh computational model with 150 triangular elements and (**b**) a computational model after constructing the smoothing domain.



(**a**) (**b**)

Figure 8. (**a**) The contour of displacement calculated using epSFEM and (**b**) the contour of displacement calculated using FEM-T3.

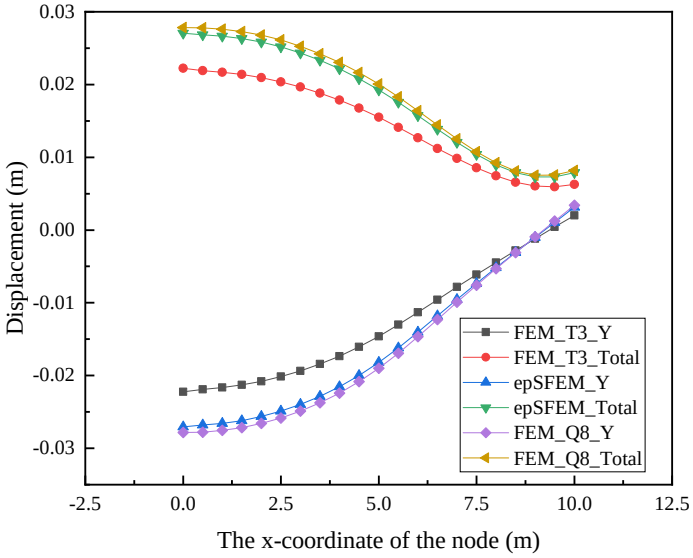


Figure 9. Comparison curves of node displacements at the top of the model calculated by different methods.

4.2. Evaluation of the Efficiency of epSFEM

To better analyze the computing efficiency of epSFEM, the computational efficiency of the serial and parallel versions of the epSFEM are recorded and compared. Five mesh models were created based on the same size model, shown in Figure 6a. Detailed information on the mesh is shown in Table 4.

Table 4. Details of the used five mesh models.

Mesh Models (T3)	Number of Nodes	Number of Elements
1	173,761	345,600
2	308,481	614,400
3	609,301	1,215,000
4	909,701	1,815,000
5	1,231,361	2,457,600

In epSFEM, the calculation procedure can be composed of two steps: (1) assembly of the elastic stiffness matrix and (2) incremental loading and semismooth Newton method iterations. In this paper, we focus on the solution of elastic-plastic problems, so the time consumption is predominantly in the second step, which is composed of multiple incremental cyclic loading steps. Each of the incremental steps can be composed of three stages: (1) assembly of tangent stiffness matrix, (2) solving of system of equations and (3) updating of hardening variables and plastic strain. Since the Pardiso.jl package is employed to solve equations in serial and parallel code, the efficiency of solving equations in serial and parallel ways are not discussed. For the assembly of the elastic stiffness matrix, its time consumption accounts for a small proportion in the whole elastic-plastic analysis, which is not discussed in this paper. The parallel method of the hardening variable and plastic strain update part is consistent with the parallel method of tangent stiffness matrix assembly. Therefore, we mainly evaluate the computing efficiency of assembling the tangent stiffness matrix in this paper.

As shown in Figure 10, the time to compute the parallelizable section of the tangent stiffness matrix in the serial and parallel versions for five different scale mesh models is compared. As shown in Figure 10, it takes only approximately 335 s to compute a mesh model, including 2.45 million elements on the parallel version, while it takes approximately

3537.6 s to compute the same model on the serial version. On the 24-core CPU, the parallel speedup can reach 10.6.

To reflect the computational efficiency of epSFEM, we also made a comparison between commercial software and epSFEM in terms of the time required to calculate the five scale models, as shown in Table 4. The total time required for the solver computing is recorded for comparison. As shown in Figure 11, for a model containing 2.45 million elements, ABAQUS requires 10,619 s to compute, while the parallel version of epSFEM needs only 5876.3 s to complete the computation. The parallel version of epSFEM is approximately 1.8 times faster than ABAQUS.

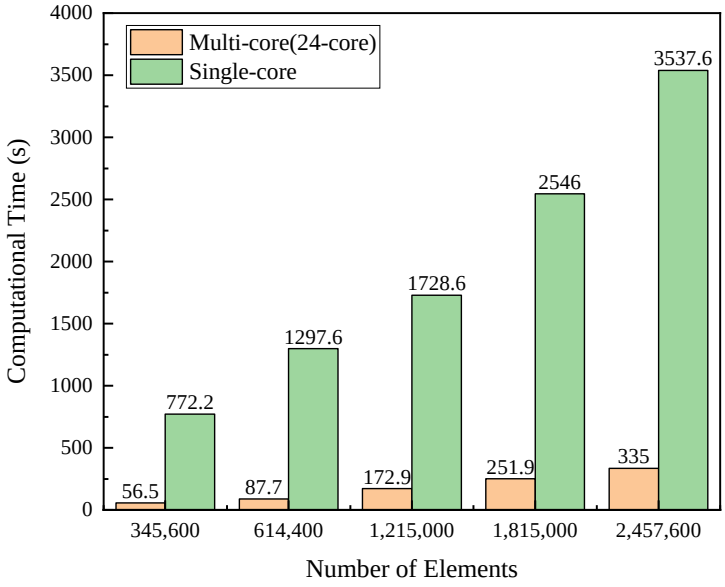


Figure 10. Comparison of serial and parallel epSFEM computing time of the parallelizable section of the tangent stiffness matrix.

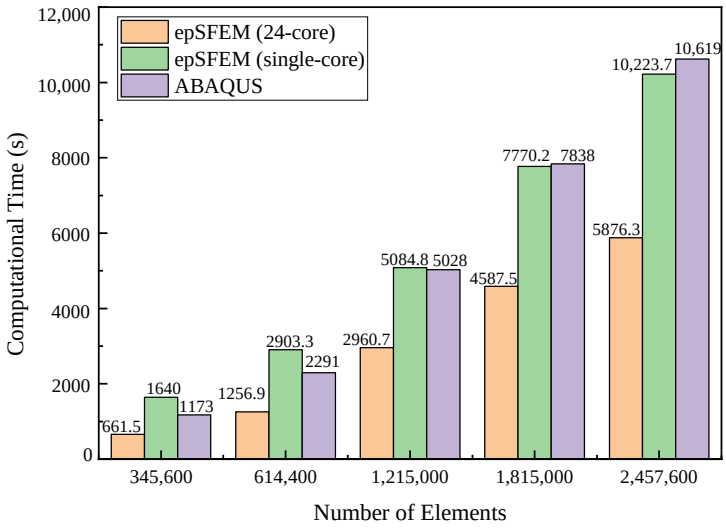


Figure 11. Comparison of the computation time of serial and parallel epSFEM and ABAQUS solvers for elastic-plastic problems.

ABAQUS was also used to calculate the displacements for a mesh model with 341 nodes and 600 triangular cells and to compare the displacements obtained by ABAQUS with those obtained by epSFEM_T3 and FEM_Q8 in Section 4.1. Using the displacement solution of FEM_Q8 as the reference solution, it can be seen that the displacement calculation accuracy of epSFEM is higher than that of ABAQUS; see Figure 12. It can be seen from the

above results that the calculation time of epSFEM is shorter than that of ABAQUS when calculating the same mesh model, and the calculation accuracy of epSFEM is higher than that of ABAQUS, so the calculation efficiency of epSFEM is higher than that of ABAQUS.

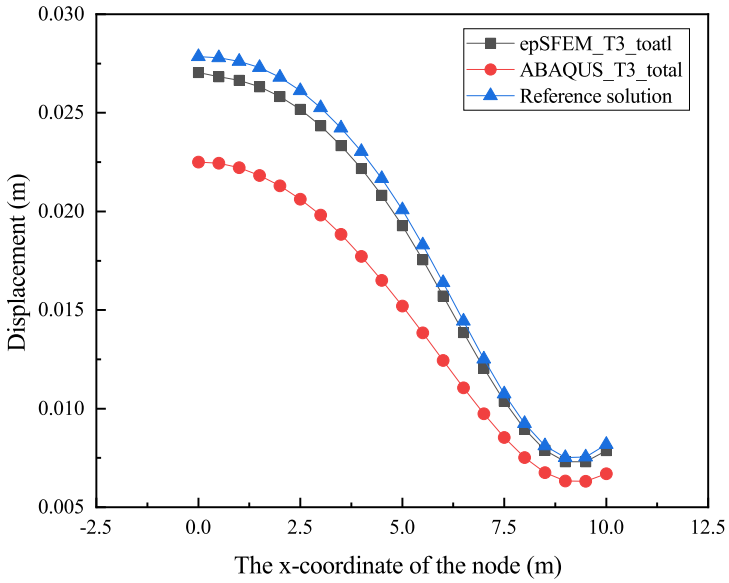


Figure 12. Comparison curves of node displacements at the top of the model calculated with ABAQUS and epsFEM.

5. Discussion

In this section, the capability, strengths and weaknesses of the epSFEM software package, as well as the future direction of work, are discussed.

5.1. Comprehensive Evaluation of epSFEM

5.1.1. Computational Accuracy

The accuracy of the calculation is the first guarantee for whether a software package can be used. To verify the correctness of the epSFEM calculation, a numerical example is used in Section 4.1. As listed in Table 5, the total displacement results of the six nodes at the top $y = 10$ m of the model are selected for comparison. Taking the displacements of FEM-Q8 as the baseline and comparing the displacements of epSFEM-T3 and FEM-T3 with them, it can be shown that the displacements of epSFEM-T3 are significantly closer to the baseline. The result difference is expressed by relative error. As seen in Table 5, for the node displacement at $x = 0$ and $y = 10$ m, the error of FEM-T3 compared with FEM-Q8 is 25.24%, while the error of epSFEM-T3 compared with FEM-Q8 is only 2.96%. This is because the S-FEM is based on the smoothing domain calculation that optimizes the system stiffness matrix and enables the displacements to be closer to the reference values.

Table 5. Validation of the accuracy of the epSFEM by comparison of calculated displacements.

Position	Method			Relative Error	
	FEM-T3	epSFEM-T3	FEM-Q8	FEM-T3	epSFEM-T3
0.0 m	0.02223 m	0.02704 m	0.02784 m	25.24%	2.96%
2.0 m	0.02095 m	0.02583 m	0.02680 m	27.92%	3.76%
4.0 m	0.01787 m	0.02217 m	0.02423 m	35.59%	9.29%
6.0 m	0.01267 m	0.01572 m	0.01640 m	29.44%	4.33%
8.0 m	0.00744 m	0.00896 m	0.00924 m	24.19%	3.13%
10.0 m	0.00626 m	0.00788 m	0.00819 m	30.83%	3.93%

5.1.2. Computational Efficiency

In this paper, the efficiency of computation is contrasted in two aspects: parallel speedup of parallelizable code and solver computation time; see Figures 10 and 11.

In this paper, we recorded the time required to compute the parallelizable portion of the tangent stiffness matrix, that is, constitutive integral algorithm, for seven different size mesh models using serial and parallel epSFEM. As shown in Table 6, the parallel speedup is 10.2 for the computing model with 38,400 elements, increases to 14.8 for the computing model with 0.6 million elements and decreases to 10.0 for the computational model with 1.2 million elements, after which the parallel speedup increases slightly with the increase of the computational model size and basically stabilizes.

Table 6. The parallel speedup of the parallelizable section of the tangential stiffness matrix.

Number of Nodes	Number of Elements	Computing Time (s)		
		Single-Core	Multi-Core (24-Core)	Parallel Speedup
19,521	38,400	69.7	6.83	10.2
77,441	153,600	283.3	23.5	12.05
173,761	345,600	772.2	56.5	13.7
308,481	614,400	1297.6	87.7	14.8
609,301	1,215,000	1728.6	172.9	10.0
909,701	1,815,000	2546	251.9	10.1
1,231,361	2,457,600	3537.6	335	10.6

The reasons why the parallel speedup shows a pattern of increasing then decreasing and finally converging as the mesh scale increases are analyzed are as follows: (1) Parallel computing includes the time to allocate tasks; the amount of computation allocated to each process cannot be exactly the same, and there is the problem of load imbalance for each process, so the parallel speedup cannot reach the ideal parallel speedup. (2) When the mesh scale is small, such as 38,400 to 614,400, the total computation time increases as the mesh scale increases, the percentage of assigned tasks in the total time decreased, and the parallel speedup increases. (3) When the mesh scale increases to 1.2 million, the performance of the code decreases due to the larger memory allocation required and the increased garbage collection time during the code run. In the benchmark tests of this paper, the above effects do not have a significant impact on the overall performance of epSFEM as the scale continues to increase. On the contrary, it tends to a steady state.

TimerOutputs.jl package is used to test the time consumption and memory allocation in each part of the calculation process and generate the formatted table to output [53]. As listed in Table 7, the allocation of time and memory for each part of the parallel epSFEM solver when the number of elements is 600,000. Table 7 shows that the time proportion of the elastic stiffness matrix is very small, which is only 0.07% when the number of elements is 600,000. Therefore, we focus on the time and memory consumption of each part of the incremental loading and the semismooth Newton iteration, which is the plastic section in Table 7. Figure 13 presents the time occupancy of the tangent stiffness matrix assembly, solving equations, hardening variables and plastic strain updating when calculating the model with 2.45 million elements using the serial and parallel versions of epSFEM. Because the hardening variable and plastic strain only need to be updated once for each incremental step, the time proportion is the smallest. The tangent stiffness matrix assembly and solving equations need to be calculated not only for each incremental step, but also for each iteration, so the time proportion is longer. As shown in Figure 13, the proportion of time spent solving the equations in parallel computing is considerably larger than in serial computing, accounting for approximately 80%.

Table 7. Time and memory allocation of each part of the parallel epSFEM solver when the number of elements is 600,000.

Tot /% Measured		Time			Allocation		
		1256.9 s/100%			337.83 GiB/100%		
Section	ncalls	Time	%tot	avg	alloc	%tot	avg
solver	1	1256.9 s	100%	1256.9 s	337.83 GiB	100%	337.83 GiB
elastic	1	0.9 s	0.07%	0.9 s	1.83 GiB	0.54%	1.83 GiB
plastic	1	1256 s	99.93%	1256 s	336 GiB	99.46%	336 GiB
solving	132	946 s	75.3%	7.16 s	62.6 GiB	18.53%	486 MiB
assembly	132	246 s	19.6%	1.86 s	256 GiB	75.78%	1.94 GiB
constitutive	132	87.7 s	6.98%	664 ms	62.0 MiB	0.02%	481 KiB
K_tangent	132	150.1 s	11.95%	1.14 s	252 GiB	74.6%	1.91 GiB
hardening and strain	40	33.0 s	2.62%	824 ms	1.03 GiB	0.3%	26.3 MiB

In summary, epSFEM combines the features of incremental theory and the parallel strategy of the Julia language to achieve a parallel and efficient incremental S-FEM for solving the elastoplastic problem. Although epSFEM can take full advantage of multicore processors, it still requires a considerable amount of time to solve linear system equations for large sparse matrices. Moreover, due to the use of incremental theory, the calculation of the latter incremental step depends on the previous incremental step, and multiple incremental steps cannot be calculated in parallel, which also limits the computational efficiency of the code.

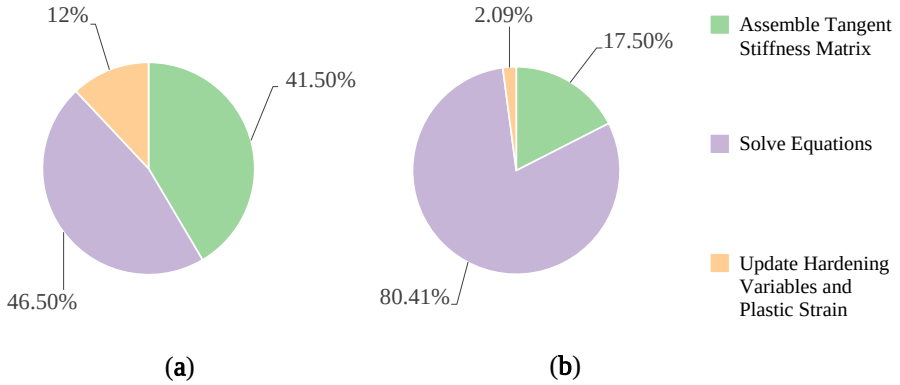


Figure 13. The proportion of time in each part of the epSFEM solver when calculating the model with 2.45 million elements using (**a**) the serial version of epSFEM and (**b**) the parallel version of epSFEM.

5.2. Comparison with Other S-FEM Programs

Compared with the S-FEM packages implemented with C++, epSFEM code is more readable and convenient for further development, and has lower requirements for programming ability. In contrast with the S-FEM packages implemented by MATLAB, epSFEM does not require the payment of licensing fees for the Julia language; additionally, the computational efficiency of the Julia language is higher than that of MATLAB. Moreover, epSFEM has a clear structure and modular implementation, and each calculation step is highly customized and has the characteristics of high efficiency and simplicity.

The epSFEM is suitable to more common and complex elastoplastic mechanical problems in practical engineering and has a wider range of applications than the elastic S-FEM package implemented using the Julia language. In contrast with the elastic-plastic S-FEM package with total strain theory realized by the Julia language, epSFEM uses incremental theory suitable for most loading situations to solve elastic-plastic problems, and the calculation results are more reliable and accurate.

5.3. Outlook and Future Work

epSFEM is an incremental ES-FEM to solve two-dimensional elastoplastic problems. The next step is to expand it to an incremental FS-FEM to solve three-dimensional elastoplastic problems. Currently, the S-FEM has been commonly utilized in material mechanics and biomechanics, but it is still less applied in the field of geotechnical mechanics [54,55]. We plan to extend epSFEM to use the Mohr-Coulomb criterion combined with the strength reduction method to analyze the deformation and failure of slopes. With the maturity of artificial intelligence technology such as machine learning and deep learning, mechanical analysis and numerical simulation methods can be well integrated with machine learning, which provides a new direction for computational mechanics [56–59]. In the future, the authors wish is to use machine learning combined with epSFEM to solve partial differential Equations (PDEs), or study parameter inversion.

6. Conclusions

In this paper, a parallel incremental S-FEM package epSFEM for elastic-plastic problems has been designed and implemented by the Julia language on a multicore CPU. epSFEM has a clear structure and legible code and can be easily developed further. epSFEM utilizes incremental S-FEM to solve elastic-plastic mechanics problems for complex load cases more common in practical engineering, and the calculation results are more accurate and reliable. A partitioned parallel strategy was designed to improve the computational efficiency of epSFEM. This strategy can avoid conflicts when accessing the underlying data in parallel computing. To demonstrate the correctness of epSFEM and assess its efficiency, two sets of benchmark tests were performed in this paper. The results indicated that (1) when calculating the same mesh model, the calculation accuracy of epSFEM is higher than that of the traditional FEM; (2) it requires only 5876.3 s to calculate an elastoplastic model, consisting of approximately 2.45 million T3 elements using the parallel epSFEM software package, while it needs 10,619 s to calculate the same model using the commercial FEM software ABAQUS; (3) on a 24-core CPU, the parallel execution of epSFEM is approximately 10 times faster than the corresponding serial version.

Author Contributions: Conceptualization, M.Z., J.Q. and G.M.; methodology, M.Z. and J.Q.; software, M.Z. and Z.H.; validation, M.Z. and G.M.; formal analysis, J.Q., Z.H. and F.G.; investigation, J.Q., Z.H. and F.G.; resources, M.Z.; data curation, M.Z.; writing—original draft preparation, M.Z. and G.M.; writing—review and editing, M.Z. and G.M.; visualization, M.Z.; supervision, J.Q. and G.M.; project administration, G.M.; funding acquisition, G.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was jointly supported by the Fundamental Research Funds for China Central Universities (Grant Numbers: 2652018091) and the National Natural Science Foundation of China (Grant Numbers: 11602235).

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Not applicable

Acknowledgments: The authors would like to thank the editor and the reviewers for their valuable comments.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

COO	COOrdinate
CPU	Central Processing Unit
CSC	Compressed Sparse Column
CS-FEM	Cell-based Smoothed Finite Element Method
CSR	Compressed Sparse Row
EFG	Element Free Galerkin
ES-FEM	Edge-based Smoothed Finite Element Method
FEM	Finite Element Method
FS-FEM	Face-based Smoothed Finite Element Method
GPU	Graphics Processing Unit
MLPG	Meshless Local Petrov-Galerkin
MPM	Material Point Method
NS-FEM	Node-based Smoothed Finite Element Method
PDEs	Partial Differential Equations
RPIM	Radial Point Interpolation Method
S-FEM	Smoothed Finite Element Method

References

1. Xiao, L.; Mei, G.; Xi, N.; Piccialli, F. Julia Language in Computational Mechanics: A New Competitor. *Arch. Comput. Methods Eng.* **2021**, *29*, 1713–1726. [CrossRef]
2. Xu, N.; Mei, G.; Qin, J.; Li, Y.; Xu, L. GeoMFree^{3D}: A package of meshfree local Radial Point Interpolation Method (RPIM) for geomechanics. *Comput. Math. Appl.* **2021**, *81*, 113–132. [CrossRef]
3. Vizjak, J.; Bekovic, M.; Jesenik, M.; Hamler, A. Development of a Magnetic Fluid Heating FEM Simulation Model with Coupled Steady State Magnetic and Transient Thermal Calculation. *Mathematics* **2021**, *9*, 2561. [CrossRef]
4. Li, Y.C.; Dang, S.N.; Li, W.; Chai, Y.B. Free and Forced Vibration Analysis of Two-Dimensional Linear Elastic Solids Using the Finite Element Methods Enriched by Interpolation Cover Functions. *Mathematics* **2022**, *10*, 456. [CrossRef]
5. Liu, G.R. *Meshfree Methods: Moving Beyond the Finite Element Method*, 2nd ed.; CRC Press: Boca Raton, FL, USA, 2009.
6. Liu, G.R. An Overview on Meshfree Methods: For Computational Solid Mechanics. *Int. J. Comput. Methods* **2016**, *13*, 1630001. [CrossRef]
7. Zeng, W.; Liu, G.R. Smoothed Finite Element Methods (S-FEM): An Overview and Recent Developments. *Arch. Comput. Methods Eng.* **2018**, *25*, 397–435. [CrossRef]
8. Liu, G.R.; Zhang, G.Y. *Smoothed Point Interpolation Methods: G Space Theory and Weakened Weak Forms*; World Scientific: Singapore, 2013.
9. Ding, R.; Shen, Q.; Yao, Y. The element-free Galerkin method for the dynamic Signorini contact problems with friction in elastic materials. *Appl. Math. Comput.* **2022**, *415*, 126696. [CrossRef]
10. Liu, Z.; Wei, G.; Qin, S.; Wang, Z. The elastoplastic analysis of functionally graded materials using a meshfree RRKPM. *Appl. Math. Comput.* **2022**, *413*, 126651. [CrossRef]
11. Liu, G.R.; Trung, N.T. *Smoothed Finite Element Methods*; CRC Press: Boca Raton, FL, USA, 2016.
12. Cui, X.Y.; Liu, G.R.; Li, G.Y.; Zhang, G.Y.; Sun, G.Y. Analysis of elastic-plastic problems using edge-based smoothed finite element method. *Int. J. Press. Vessel. Pip.* **2009**, *86*, 711–718. [CrossRef]
13. Cazes, F.; Meschke, G. An edge-based smoothed finite element method for 3D analysis ofsolid mechanics problems. *Int. J. Numer. Methods Eng.* **2013**, *94*, 715–739. [CrossRef]
14. Liu, G.R.; Nguyen-Thoi, T.; Lam, K.Y. An edge-based smoothed finite element method (ES-FEM) for static, free and forced vibration analyses of solids. *J. Sound Vib.* **2009**, *320*, 1100–1130. [CrossRef]
15. Nguyen-Thoi, T.; Phung-Van, P.; Rabczuk, T.; Nguyen-Xuan, H.; Le-Van, C. Free and forced vibration analysis using the n-sided polygonal cell-based smoothed finite element method (nCS-FEM). *Int. J. Comput. Methods* **2013**, *10*, 1340008. [CrossRef]
16. Tian, F.; Tang, X.; Xu, T.; Li, L. An adaptive edge-based smoothed finite element method (ES-FEM) for phase-field modeling of fractures at large deformations. *Comput. Methods Appl. Mech. Eng.* **2020**, *372*, 113376. [CrossRef]
17. Cui, X.Y.; Liu, G.R.; Li, G.Y.; Zhao, X.; Nguyen, T.T.; Sun, G.Y. A smoothed finite element method (SFEM) for linear and geometrically nonlinear analysis of plates and shells. *Comput. Model. Eng. Sci.* **2008**, *28*, 109–125.
18. Zhang, Z.Q.; Liu, G.R.; Khoo, B.C. Immersed smoothed finite element method for two dimensional fluid–structure interaction problems. *Int. J. Numer. Methods Eng.* **2012**, *90*, 1292–1320. [CrossRef]
19. He, Z.C.; Liu, G.R.; Zhong, Z.H.; Zhang, G.Y.; Cheng, A.G. Coupled analysis of 3D structural-acoustic problems using the edge-based smoothed finite element method/finite element method. *Finite Elem. Anal. Des.* **2010**, *46*, 1114–1121. [CrossRef]
20. Li, E.; Zhang, Z.; He, Z.C.; Xu, X.; Liu, G.R.; Li, Q. Smoothed finite element method with exact solutions in heat transfer problems. *Int. J. Heat Mass Transf.* **2014**, *78*, 1219–1231. [CrossRef]
21. Jiang, C.; Zhang, Z.Q.; Liu, G.R.; Han, X.; Zeng, W. An edge-based/node-based selective smoothed finite element method using tetrahedrons for cardiovascular tissues. *Eng. Anal. Bound. Elem.* **2015**, *59*, 62–77. [CrossRef]

22. Lee, K.; Lim, J.H.; Sohn, D.; Im, S. A three-dimensional cell-based smoothed finite element method for elasto-plasticity. *J. Mech. Sci. Technol.* **2015**, *29*, 611–623. [CrossRef]
23. Liu, G.R.; Nguyen-Thoi, T.; Nguyen-Xuan, H.; Lam, K.Y. A node-based smoothed finite element method (NS-FEM) for upper bound solutions to solid mechanics problems. *Comput. Struct.* **2009**, *87*, 14–26. [CrossRef]
24. Li, Y.; Liu, G. A novel node-based smoothed finite element method with linear strain fields for static, free and forced vibration analyses of solids. *Appl. Math. Comput.* **2019**, *352*, 30–58. [CrossRef]
25. Nguyen-Thoi, T.; Liu, G.R.; Lam, K.Y.; Zhang, G.Y. A face-based smoothed finite element method (FS-FEM) for 3D linear and geometrically non-linear solid mechanics problems using 4-node tetrahedral elements. *Int. J. Numer. Methods Eng.* **2009**, *78*, 324–353. [CrossRef]
26. Chen, L.; Rabczuk, T.; Bordas, S.P.A.; Liu, G.R.; Zeng, K.Y.; Kerfriden, P. Extended finite element method with edge-based strain smoothing (ESm-XFEM) for linear elastic crack growth. *Comput. Methods Appl. Mech. Eng.* **2012**, *209*, 250–265. [CrossRef]
27. Nguyen-Xuan, H.; Liu, G.R. An edge-based smoothed finite element method softened with a bubble function (bES-FEM) for solid mechanics problems. *Comput. Struct.* **2013**, *128*, 14–30. [CrossRef]
28. Xu, X.; Gu, Y.; Liu, G. A Hybrid smoothed finite element method (H-SFEM) to solid mechanics problems. *Int. J. Comput. Methods* **2013**, *10*, 1340011. [CrossRef]
29. Zeng, W.; Liu, G.R.; Li, D.; Dong, X.W. A smoothing technique based beta finite element method (beta FEM) for crystal plasticity modeling. *Comput. Struct.* **2016**, *162*, 48–67. [CrossRef]
30. Dudzinski, M.; Rozgiè, M.; Stiemer, M. oFEM: An object oriented finite element package for Matlab. *Appl. Math. Comput.* **2018**, *334*, 117–140. [CrossRef]
31. Gao, K.; Mei, G.; Piccialli, F.; Cuomo, S.; Tu, J.; Huo, Z. Julia language in machine learning: Algorithms, applications, and open issues. *Comput. Sci. Rev.* **2020**, *37*, 100254. [CrossRef]
32. Bezanson, J.; Edelman, A.; Karpinski, S.; Shah, V.B. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* **2017**, *59*, 65–98. [CrossRef]
33. Frondelius, T.; Aho, J. JuliaFEM-open source solver for both industrial and academia usage. *Raken. Mek.* **2017**, *50*, 229–233. [CrossRef]
34. Sinaie, S.; Nguyen, V.P.; Nguyen, C.T.; Bordas, S. Programming the material point method in Julia. *Adv. Eng. Softw.* **2017**, *105*, 17–29. [CrossRef]
35. Huo, Z.; Mei, G.; Xu, N. juSFEM: A Julia-based open-source package of parallel Smoothed Finite Element Method (S-FEM) for elastic problems. *Comput. Math. Appl.* **2021**, *81*, 459–477. [CrossRef]
36. Pawar, S.; San, O. CFD Julia: A Learning Module Structuring an Introductory Course on Computational Fluid Dynamics. *Fluids* **2019**, *4*, 159. [CrossRef]
37. Heitzinger, C.; Tulzer, G. Julia and the numerical homogenization of PDEs. In Proceedings of the 1st Workshop on High Performance Technical Computing Dynamic Languages, New Orleans, LA, USA, 17 November 2014; pp. 36–40. [CrossRef]
38. Kemmer, T.; Rjasanow, S.; Hildebrandt, A. NESSie.jl–Efficient and intuitive finite element and boundary element methods for nonlocal protein electrostatics in the Julia language. *J. Comput. Sci.* **2018**, *28*, 193–203. [CrossRef]
39. Fairbrother, J.; Nemeth, C.; Rischard, M.; Brea, J.; Pinder, T. GaussianProcesses.jl: A Nonparametric Bayes Package for the Julia Language. *J. Stat. Softw.* **2022**, *102*, 1–36. [CrossRef]
40. Pardiso.jl. 2021. Available online: https://github.com/JuliaSparse/Pardiso.jl (accessed on 10 February 2021).
41. The Julia Programming Language. 2021. Available online: https://julialang.org/ (accessed on 5 January 2021).
42. Huo, Z.; Mei, G.; Casolla, G.; Giampaolo, F. Designing an efficient parallel spectral clustering algorithm on multi-core processors in Julia. *J. Parallel Distrib. Comput.* **2020**, *138*, 211–221. [CrossRef]
43. Julia 1.6 Documentation. 2021. Available online: https://docs.julialang.org/en/v1/ (accessed on 10 May 2021).
44. Paraview. 2019. Available online: https://www.paraview.org/ (accessed on 28 May 2021).
45. Li, Y.; Yue, J.H.; Niu, R.P.; Liu, G.R. Automatic mesh generation for 3D smoothed finite element method (S-FEM) based on the weaken-weak formulation. *Adv. Eng. Softw.* **2016**, *99*, 111–120. [CrossRef]
46. Dodds, R.H., Jr. Numerical techniques for plasticity computations in finite element analysis. *Comput. Struct.* **1987**, *26*, 767–779. [CrossRef]
47. Blaheta, R. Convergence of Newton-type methods in incremental return mapping analysis of elasto-plastic problems. *Comput. Methods Appl. Mech. Eng.* **1997**, *147*, 167–185. [CrossRef]
48. De, Souza Neto, E.A.; Peri, D.; Owen, D.R.J. *Computational Methods for Plasticity*; Wiley: Hoboken, NJ, USA, 2008.
49. Čermák, M.; Sysala, S.; Valdman, J. Efficient and flexible MATLAB implementation of 2D and 3D elastoplastic problems. *Appl. Math. Comput.* **2019**, *355*, 595–614. [CrossRef]
50. Carstensen, C.; Klose, R. Elastoviscoplastic finite element analysis in 100 lines of Matlab. *J. Numer. Math.* **2002**, *10*, 157–192. [CrossRef]
51. Sysala; S. Properties and simplifications of constitutive time-discretized elastoplastic operators. *ZAMM-J. Appl. Math. Mech./Z. fÃ¼r Angew. Math. Und Mech.* **2014**, *94*, 233–255. [CrossRef]
52. WriteVTK.jl. 2021. Available online: https://github.com/jipolanco/WriteVTK.jl (accessed on 10 June 2021).
53. TimerOutputs.jl. 2021. Available online: https://github.com/KristofferC/TimerOutputs.jl (accessed on 10 August 2021).

54. Ma, Z.; Mei, G. Deep learning for geological hazards analysis: Data, models, applications, and opportunities. *Earth-Sci. Rev.* **2021**, *223*, 103858. [CrossRef]
55. Mei, G.; Xu, N.; Qin, J.; Wang, B.; Qi, P. A Survey of Internet of Things (IoT) for Geohazard Prevention: Applications, Technologies, and Challenges. *IEEE Internet Things J.* **2020**, *7*, 4371–4386. [CrossRef]
56. Rudy, S.; Alla, A.; Brunton, S.; Kutz, J. Data-driven identification of parametric partial differential equations. *SIAM J. Appl. Dyn. Syst.* **2019**, *18*, 643–660. [CrossRef]
57. Raissi, M.; Perdikaris, P.; Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
58. Haghighat, E.; Raissi, M.; Moure, A.; Gomez, H.; Juanes, R. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Comput. Methods Appl. Mech. Eng.* **2021**, *379*, 113741. [CrossRef]
59. Jacobs, B.; Celik, T. Unsupervised document image binarization using a system of nonlinear partial differential equations. *Appl. Math. Comput.* **2022**, *418*, 126806. [CrossRef]