Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2021.017321

ARTICLE

A GPU-Based Parallel Algorithm for 2D Large Deformation Contact Problems Using the Finite Particle Method

Wei Wang[1,2], **Yanfeng Zheng**[1,3], **Jingzhe Tang**[1], **Chao Yang**[1] **and Yaozhi Luo**[1,*]

[1]College of Civil Engineering and Architecture, Zhejiang University, Hangzhou, 310058, China

[2]Center for Balance Architecture, Zhejiang University, Hangzhou, 310028, China

[3]Architectural Design and Research Institute of Zhejiang University Co., Ltd., Hangzhou, 310028, China

[*]Corresponding Author: Yaozhi Luo. Email: luoyz@zju.edu.cn

Received: 01 May 2021 Accepted: 26 July 2021

ABSTRACT

Large deformation contact problems generally involve highly nonlinear behaviors, which are very time-consuming and may lead to convergence issues. The finite particle method (FPM) effectively separates pure deformation from total motion in large deformation problems. In addition, the decoupled procedures of the FPM make it suitable for parallel computing, which may provide an approach to solve time-consuming issues. In this study, a graphics processing unit (GPU)-based parallel algorithm is proposed for two-dimensional large deformation contact problems. The fundamentals of the FPM for planar solids are first briefly introduced, including the equations of motion of particles and the internal forces of quadrilateral elements. Subsequently, a linked-list data structure suitable for parallel processing is built, and parallel global and local search algorithms are presented for contact detection. The contact forces are then derived and directly exerted on particles. The proposed method is implemented with main solution procedures executed in parallel on a GPU. Two verification problems comprising large deformation frictional contacts are presented, and the accuracy of the proposed algorithm is validated. Furthermore, the algorithm's performance is investigated via a large-scale contact problem, and the maximum speedups of total computational time and contact calculation reach 28.5 and 77.4, respectively, relative to commercial finite element software Abaqus/Explicit running on a single-core central processing unit (CPU). The contact calculation time percentage of the total calculation time is only 18% with the FPM, much smaller than that (50%) with Abaqus/Explicit, demonstrating the efficiency of the proposed method.

KEYWORDS

Finite particle method; graphics processing unit (GPU); parallel computing; contact algorithm; large deformation

1 Introduction

The numerical simulation of large deformation contact problems plays an important role in engineering fields. These problems generally involve geometric nonlinearity, material nonlinearity, and contact nonlinearity. Several numerical methods are available for these problems, among which the finite element method (FEM) is one of the most widely used methods. Computational contact mechanics using the FEM has been developed by many scholars since the 1970s and can be found in the literature [1,2]. Recently, some elaborated contact algorithms in the FEM context, such as the mortar method [3] and the surface smoothing method [4], have been developed, mainly to improve the accuracy of contact constraints; the computational efficiency of these algorithms, however, has drawn less attention. In fact, contact algorithms generally account for a considerable proportion of the overall computational cost [5]. Specifically, according to Wriggers [1], the time complexities of contact search algorithms are generally of the order from $O(N)$ to $O(N \log N)$, where N denotes the number of contact nodes, which is unacceptable when millions of elements come into contact. In addition, the strong nonlinearities involved in large deformation contact problems may cause difficulties when using implicit techniques [4], while the time step sizes for these contact problems are usually very small when using explicit techniques [5], resulting in time-consuming calculation processes. Other methods, such as the discrete element method (DEM), are available for solving contact problems, but the DEM is more suitable for modeling contact in discrete particulate systems [6,7] and contact between rocks [8,9]. Therefore, a more efficient and stable method is strongly needed in large deformation contact problems.

The finite particle method (FPM) [10], derived from vector mechanics [11–13], is another feasible method for solving large deformation contact problems. In the FPM, a physical body is discretized into a number of particles and elements in space, and the motion path of each particle in time is modeled by a sequence of time steps, as illustrated in Fig. 1. The particles in the FPM are assumed to carry structural variables such as mass, velocity, and displacement. Within each time step, Newton's second law is adopted to formulate the motion of particles, and explicit time integration schemes are used to solve the equations of motion. In recent years, numerical methods based on vector mechanics have been successfully applied to various types of structural analyses, such as the analysis of shell structures [14], multiple body kinematic movements involving large displacements and rotations [15,16], progressive collapse simulations of structures [17], shape analysis of tensile structures [18], train-bridge interaction simulation [19], and fluid-solid interaction simulation [20].

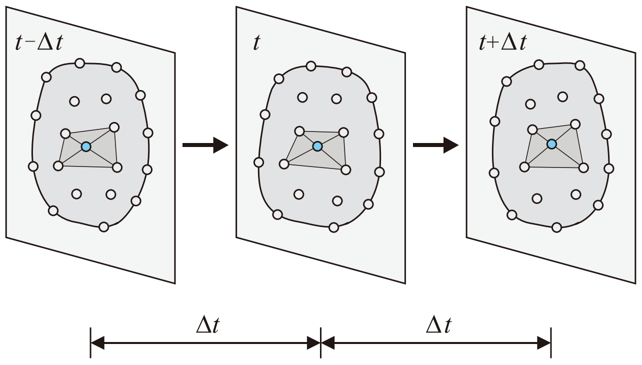

Figure 1: Discrete model of the FPM in space and time (only elements connected to the highlighted particle are depicted for brevity)

When dealing with large deformation problems, the FPM is effective in separating the pure deformation of elements from their total motion by using the fictitious reverse motion technique [10], which makes it suitable for solving large deformation contact problems. In addition, contact algorithms based on the explicit FEM, such as the one proposed by Hallquist et al. [21], can easily be adopted by the FPM due to the similar mechanisms of these two methods. However, contact calculation via the serially computed FPM is still time-consuming, mainly due to the small time step size for ensuring conditional stability of the method and the time-consuming contact search process. This problem can be effectively alleviated by using parallel acceleration for the FPM. In fact, no global stiffness matrix is assembled in the FPM, and the main solution procedures of the FPM are decoupled in nature; thus, the FPM is intrinsically suitable for parallel implementation with a high degree of parallelism.

Many parallel architectures are based on central processing unit (CPU) technology, e.g., open multiprocessing (OpenMP) [22,23] for multiple CPUs and message passing interface (MPI) for multiple computer hosts. However, the hardware costs of CPU-based parallel architectures are very high, and the quantitative restriction of CPU cores in personal computers makes concurrently processing a large number of procedures a difficult task. Recently, thanks to the development of graphics processing units (GPUs) with unified architectures and the introduction of specialized programming models, such as Compute Unified Device Architecture (CUDA) designed by NVIDIA, general-purpose computing on GPU (GPGPU) has become an increasingly adopted computing technique in engineering simulations [24–26]. While a CPU is designed to excel at executing a single thread as fast as possible, a GPU is designed to excel at executing thousands of threads in parallel [27]. As a result, GPUs provide higher instruction throughput and memory bandwidth at a lower cost than CPUs [28]. Therefore, the GPU architecture is more appropriate for parallel implementation of the FPM.

In this paper, a two-dimensional GPU-based parallel contact algorithm using the FPM is proposed and implemented based on the GPU-accelerated software developed in our previous study [24]. The algorithm takes full advantage of the parallel architecture of GPUs, and the main computational procedures of the algorithm, including evaluating elemental internal forces, searching contact pairs, detecting contact states, calculating contact forces, and solving the equations of motion, are executed in parallel on a GPU. There are four contributions of the current work compared to our previous study [24]: (1) two new quadrilateral elements FPM-Q4 and FPM-Q4R are developed for 2D large deformation problems; (2) a contact algorithm for 2D large deformation problems is proposed and incorporated into FPM; (3) The proposed contact algorithm is parallelized based on GPU; (4) the parallel contact algorithm is verified and its performance is fully tested.

This paper is structured as follows. The fundamentals of the parallelized FPM for planar solids are briefly introduced in Section 2. The parallel contact algorithm is discussed in Section 3, including the contact detection process and calculation of the contact forces. The GPU implementation of the proposed algorithm is presented in Section 4. Two verification examples and two performance tests are presented in Section 5 to demonstrate the accuracy and efficiency of the proposed contact algorithm. Finally, conclusions are given in Section 6.

2 Fundamentals of the Parallelized FPM for Planar Solids

The fundamentals of the FPM for planar solids are briefly described in this section, including the equations of motion of particles and the internal forces of quadrilateral elements. Readers are referred to the literature [12] for more details. The parallel algorithm for planar solids is presented in Section 2.3.

2.1 Equations of Motion of Particles

A planar solid, as shown in Fig. 2a, can be discretized into 20 particles and 12 quadrilateral elements. For an arbitrary particle α, the equation of motion can be formulated based on Newton's second law:

$$m_\alpha \ddot{\mathbf{d}}_\alpha = \mathbf{F}_\alpha^{\text{ext}} - \mathbf{F}_\alpha^{\text{int}} - \mathbf{F}_\alpha^{\text{dmp}} + \mathbf{F}_\alpha^{c} \tag{1}$$

where m_α is the mass of particle α, $\ddot{\mathbf{d}}_\alpha$ represents the acceleration vector, $\mathbf{F}_\alpha^{\text{ext}}$ denotes the external forces applied to particle α, $\mathbf{F}_\alpha^{\text{int}}$ represents the internal forces of particle α passed from the elements connected to it (see Section 2.2), $\mathbf{F}_\alpha^{c}$ denotes the contact forces (derived in Section 3.2), and $\mathbf{F}_\alpha^{\text{dmp}}$ represents the mass-proportional damping force, which is given by

$$\mathbf{F}_\alpha^{\text{dmp}} = \xi m_\alpha \dot{\mathbf{d}}_\alpha \tag{2}$$

where ξ is the damping coefficient and $\dot{\mathbf{d}}_\alpha$ denotes the velocity vector of particle α. The particle forces are illustrated in Fig. 2b.

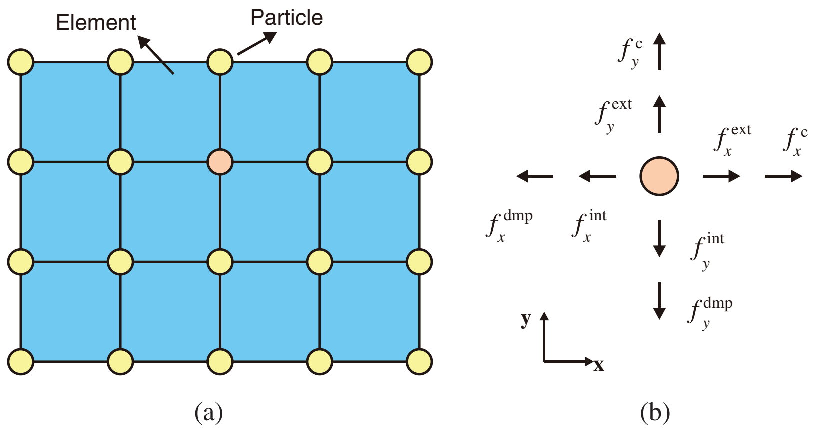

Figure 2: Discretization of a planar solid: (a) Particles and elements; (b) Particle forces

The central difference method is adopted to solve Eq. (1) because of its low computational cost and sufficient accuracy. According to the literature [29], the central difference method has the highest accuracy and maximum stability limit for any second-order accurate explicit method. Other explicit time integration algorithms such as generalized-α algorithm [30] can also be utilized in the FPM, but more computational time will be taken when using those algorithms. Thus, the central difference method is adopted in this paper to solve the equations of motion of particles. Given the particle displacements at times $t-\Delta t$ and t, where Δt is the time step size, the particle displacement at time $t+\Delta t$ can be explicitly expressed as follows:

$$^{t+\Delta t}\mathbf{d}_\alpha = c_1 \Delta t^2 \frac{1}{m_\alpha}\left({}^{t}\mathbf{F}_\alpha^{\text{ext}} - {}^{t}\mathbf{F}_\alpha^{\text{int}} + {}^{t}\mathbf{F}_\alpha^{c}\right) + 2c_1\, {}^{t}\mathbf{d}_\alpha - c_2\, {}^{t-\Delta t}\mathbf{d}_\alpha \tag{3}$$

where $c_1 = (1+\xi\Delta t/2)^{-1}$ and $c_2 = c_1(1-\xi\Delta t/2)$.

2.2 Internal Forces of Quadrilateral Elements

The internal forces of quadrilateral elements are derived in this section. First, the pure deformation is approximated by using fictitious reverse motion, and a local coordinate system

(LCS) is introduced based on the pure deformation vector. Subsequently, the strain and stress increments are computed in the LCS, and the internal forces are derived using the principle of virtual work.

2.2.1 Pure Deformation

As shown in Fig. 3a, a quadrilateral element moves from time t_a to t_b, where $t_b = t_a + \Delta t$. The corresponding configurations are denoted as Ω^a and Ω^b, respectively, and the position vectors of each particle are represented as $\mathbf{X}_i^a$ and $\mathbf{X}_i^b$ $(i = 1, 2, 3, 4)$, respectively. The incremental displacement vectors of the particles from time t_a to t_b are denoted as

$$\Delta\mathbf{d}_i = \mathbf{X}_i^b - \mathbf{X}_i^a, \quad (i = 1, 2, 3, 4) \tag{4}$$

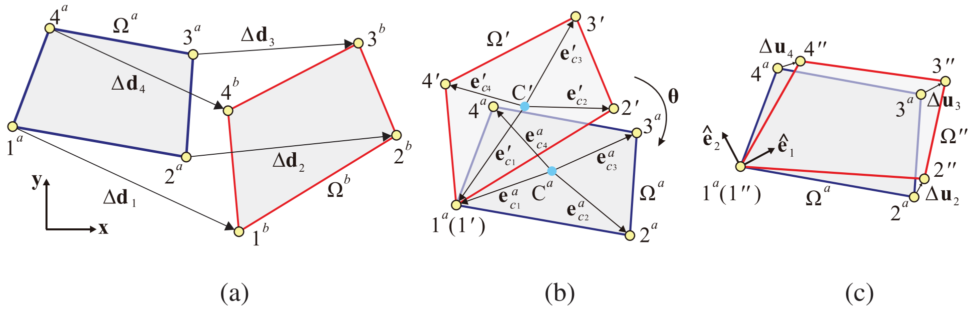

Figure 3: Fictitious reverse motion for calculating pure deformation: (a) Configurations Ω^a and Ω^b; (b) Fictitious configuration Ω'; (c) Fictitious configuration Ω'' and the local coordinate system

To separate the rigid-body motion from the total displacement, a fictitious reverse motion technique based on vector mechanics [12] is adopted, and the procedures are described as follows:

Step 1: Fictitious reverse translation. In the following, particle 1 is chosen as a reference particle. The configuration Ω^b is translated to a fictitious configuration Ω', as shown in Fig. 3b, by a vector of $-\Delta\mathbf{d}_1$, and the corresponding position vector $\mathbf{X}_i'$ is expressed as

$$\mathbf{X}_i' = \mathbf{X}_i^b - \Delta\mathbf{d}_1, \quad (i = 1, 2, 3, 4) \tag{5}$$

Step 2: Fictitious reverse rotation. The configuration Ω' is rotated to another fictitious configuration Ω'', as shown in Fig. 3c, where the rotation axis is orthogonal to the plane containing the elements, and the rotation angle is obtained by

$$\theta = \frac{1}{4}\sum_{i=1}^{4} \operatorname{asin}\frac{\mathbf{e}_{ci}' \times \mathbf{e}_{ci}^a}{\mathbf{e}_{ci}' \times \mathbf{e}_{ci}^a}, \quad (i = 1, 2, 3, 4) \tag{6}$$

where $\mathbf{e}_{ci}^a$ and $\mathbf{e}_{ci}'$ are the unit vectors pointing from the centroid of the element to particle i based on configurations Ω^a and Ω', respectively. The position vector of the particles in configuration Ω'' is obtained by

$$\mathbf{X}_i'' = \mathbf{X}_1' + \mathbf{R}_1(\mathbf{X}_i' - \mathbf{X}_1'), \quad (i = 1, 2, 3, 4) \tag{7}$$

where $\mathbf{R}_1$ is the matrix representing the rotation from Ω' to Ω'', which is calculated using Rodrigues' rotation formula [31]:

$$\mathbf{R}_1 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{8}$$

Step 3: Pure deformation approximation. Since the rigid motion is separated from the total displacement through the fictitious reverse motion, the pure deformation matrix of the element is approximated as

$$\Delta\mathbf{U} = \begin{bmatrix} \Delta\mathbf{u}_1 & \Delta\mathbf{u}_2 & \Delta\mathbf{u}_3 & \Delta\mathbf{u}_4 \end{bmatrix}^{\mathrm{T}} \tag{9}$$

with

$$\Delta\mathbf{u}_i = \mathbf{X}_i'' - \mathbf{X}_i^a, \ \ (i = 1, 2, 3, 4) \tag{10}$$

Step 4: Definition of the LCS. An LCS with axes $\hat{\mathbf{e}}_1$, $\hat{\mathbf{e}}_2$, and $\mathbf{e}_3$, as shown in Fig. 3c, is introduced to reduce the total number of independent variables and simplify the process of internal force calculation; here, the unit vector $\mathbf{e}_3$ is orthogonal to the plane, $\hat{\mathbf{e}}_1 = [\hat{e}_{1x}, \hat{e}_{1y}]$ is parallel to $\Delta\mathbf{u}_i$ with the greatest magnitude (e.g., $\Delta\mathbf{u}_3$ in Fig. 3c), and $\hat{\mathbf{e}}_2 = [\hat{e}_{2x}, \hat{e}_{2y}] = \mathbf{e}_3 \times \hat{\mathbf{e}}_1$. The corresponding pure deformation matrix in the LCS, denoted as $\Delta\hat{\mathbf{U}}$, is obtained by

$$\Delta\hat{\mathbf{U}} = \mathbf{R}_2 \Delta\mathbf{U} \tag{11}$$

where $\mathbf{R}_2$ is the transformation matrix from the global coordinate system (GCS) to the LCS:

$$\mathbf{R}_2 = \begin{bmatrix} \hat{e}_{1x} & \hat{e}_{1y} \\ \hat{e}_{2x} & \hat{e}_{2y} \end{bmatrix} \tag{12}$$

2.2.2 Strain and Stress Increments

After obtaining the pure deformation, the concepts of shape function and isoparametric transformation developed in the conventional FEM are introduced to the FPM to describe the strain and stress distributions within each quadrilateral element. The strain increment vector at each integration point in the LCS is given as

$$\Delta\hat{\boldsymbol{\varepsilon}}_i = \hat{\mathbf{B}}_i \Delta\hat{\mathbf{U}}, \quad (i = 1, 2, \ldots, N_{\mathrm{int}}) \tag{13}$$

where N_{int} is the number of integration points and $\hat{\mathbf{B}}_i$ denotes the strain-displacement matrix at the ith integration point in the LCS. The details of isoparametric transformation for quadrilateral elements are omitted here for brevity but can be found in the literature [32].

The stress increment vector at each integration point in the LCS can be evaluated as

$$\Delta\hat{\boldsymbol{\sigma}}_i = \mathbf{D}^e \Delta\hat{\boldsymbol{\varepsilon}}_i, \ \text{or} \ \Delta\hat{\boldsymbol{\sigma}}_i = \mathbf{D}^{ep} \Delta\hat{\boldsymbol{\varepsilon}}_i \quad (i = 1, 2, \ldots, N_{\mathrm{int}}) \tag{14}$$

where $\mathbf{D}^e$ and $\mathbf{D}^{ep}$ denote the elastic and elastoplastic constitutive matrices, respectively. For plastic materials, the combined isotropic-kinematic hardening is utilized in the FPM based on the J_2 flow theory [33]. The yield function is

$$f(\boldsymbol{\sigma}, \alpha, \kappa) = \|\xi\| - \kappa \le 0, \ \ \xi = \mathbf{s} - \alpha$$

where $\boldsymbol{\sigma}$ is a Cauchy stress tensor, $\mathbf{s}$ is a deviatoric stress tensor, α is a deviatoric back stress tensor, ξ is called the effective stress, and κ is the plastic internal variable. The radial return algorithm for J_2 theory is adopted to determine the actual form of $\mathbf{D}^{ep}$ in Eq. (14), which can be found in the literature [34].

2.2.3 Internal Forces

The principle of virtual work is adopted to derive the internal forces of elements. For each element, the variation of the external work, δU^{ext}, done by the elemental internal force $\hat{\mathbf{F}}^e$ is

$$\delta U^{\text{ext}} = (\delta \Delta \hat{\mathbf{U}})^{\text{T}} \hat{\mathbf{F}}^e \tag{15}$$

and the variation of the internal work, δU^{int}, done by the element stress $\hat{\boldsymbol{\sigma}}$ is

$$\delta U^{\text{int}} = \int_V (\delta \Delta \hat{\boldsymbol{\varepsilon}})^{\text{T}} \hat{\boldsymbol{\sigma}} \, dV = (\delta \Delta \hat{\mathbf{U}})^{\text{T}} \int_V \hat{\mathbf{B}}^{\text{T}} \hat{\boldsymbol{\sigma}} \, \text{d}V \tag{16}$$

Considering that $\delta U^{\text{ext}} = \delta U^{\text{int}}$, the elemental internal force in the LCS can be evaluated as

$$\hat{\mathbf{F}}^e = \int_V \hat{\mathbf{B}}^{\text{T}} \hat{\boldsymbol{\sigma}} \, \text{d}V \tag{17}$$

Note that the internal forces of elements in the LCS should be transformed back to the GCS by

$$\mathbf{F}^e = \mathbf{R}_2^{-1} \mathbf{R}_1^{-1} \hat{\mathbf{F}}^e \tag{18}$$

The equivalent internal force of particle α is a sum of the internal forces of the elements connected to it, i.e.,

$$\mathbf{F}_\alpha^{\text{int}} = \sum_{i=1}^{n} \mathbf{F}_i^e \tag{19}$$

where n denotes the number of elements connected to particle α.

2.2.4 Integration Schemes

Two integration schemes of elemental internal forces are adopted in this paper. The first one is the selective integration technique (also called the B bar method) proposed by Hughes [35], which provides good accuracy for both compressible and nearly incompressible media by using a onepoint integration for volumetric stresses and a twopoint quadrature in each direction for the deviatoric stresses. The corresponding elements of the B bar method are called FPM-Q4 and used in Section 5.1. The other integration scheme is the uniform reduced integration with hourglass control developed by Flanagan et al. [36]. The corresponding elements with this reduced integration technique are called FPM-Q4R and used in Section 5.2.

2.3 Parallel Algorithm for Planar Solids

The parallel algorithm for planar solids is presented in Algorithm 1. Note that the keyword pair "**parallel for** … **end parallel**" indicates that the codes within it are executed in parallel on GPU threads. The equation of motion of each particle is solved individually, and the evaluations of the elemental internal forces are self-reliant between elements; thus, the FPM can be accelerated by GPGPU techniques with a high degree of parallelism. The corresponding GPU implementation for planar solids is given in Section 4.

Algorithm 1: Parallel algorithm for planar solids

```
// Comments:
// $N_e$—the number of elements
// $N_p$—the number of particles
1: // Parallel computing of evaluating elemental internal forces
2: parallel for $0 \leq iElem \leq (N_e - 1)$ do
3:       Obtain pure deformation vector in the LCS, $\Delta\hat{\mathbf{U}}$, using Eq. (11)
4:       Compute strain increments $\Delta\hat{\boldsymbol{\varepsilon}}_i$ using Eq. (13)
5:       Compute stress increments $\Delta\hat{\boldsymbol{\sigma}}_i$ using Eq. (14)
6:       Evaluate elemental internal force $\mathbf{F}^e$ using Eq. (18)
7: end parallel
8: // Parallel computing of solving equations of motion of particles
9: parallel for $0 \leq iParticle \leq (N_p - 1)$ do
10:      Assemble equivalent internal force of particle, $\mathbf{F}_\alpha^{\text{int}}$, using Eq. (19)
11:      Solve the equation of motion of particle using Eq. (3)
12: end parallel
```

3 Parallel Contact Algorithm Based on the FPM

3.1 Contact Detection

The contact between two deformable bodies is considered. The contact surface on one body is selected as the master and the contact surface on the other body as the slave, as shown in Fig. 4, and this master-slave pair is called a contact surface pair. The node-to-segment (NTS) [21] technique is adopted for discretizing contact surfaces into contact particles and contact segments, as shown in Fig. 4. After discretizing contact bodies, the particles on the master surface are defined as master particles while the particles on the slave surfaces as slave particles. Note that proper selection of the master and slave surfaces is significant for NTS contact discretization. According to the literature [4], the selection of master and slave surfaces should satisfy that the slave nodes cannot penetrate into the master surface, while the master nodes are free to penetrate into the slave surface. Thus, the contact surface of the stiffer body should be the master surface. More details about choosing the master and slave surfaces can be found in the literature [4].

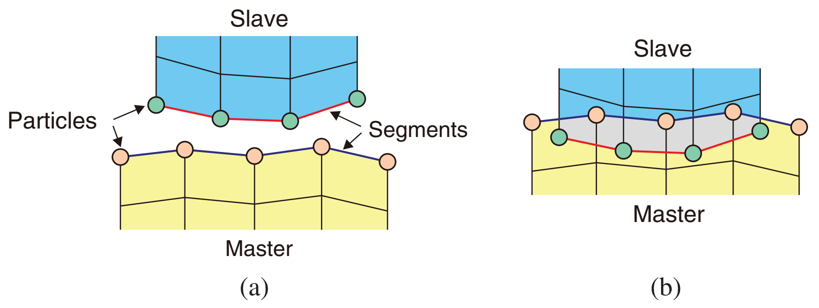

Figure 4: Discretization of contact surfaces: (a) Separation state; (b) Penetration state

The contact search algorithms of contact surfaces are typically decomposed into two distinct phases: global search and local search. The purpose of global search is to first collect master segment candidates for each slave particle, and the aim of local search is to choose the exact master segment for each slave particle and detect the contact state. Note that global search can be performed every n time steps to reduce the computational time, while local search needs to be performed at every time step. To make this contact search algorithm more suitable for parallel computing, a linked-list data structure represented by two arrays [37] is used.

3.1.1 Data Structure

For a two-body contact problem, the potential range of a contact surface pair in space is represented by a bounding box, as illustrated in Fig. 5. The coordinates of the bottom-left and top-right corners of the bounding box are denoted as $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$, respectively. The bounding box is divided into several square cells of the same size, and the size of each cell is determined by

$$L_c = \lambda \bar{L} \tag{20}$$

where $\bar{L}$ is the average length of all contact segments, λ is a cell size scale factor with a default value of 1.0. The number of cells in each direction is obtained by

$$N_x = \text{floor}\left(\frac{x_{\max} - x_{\min}}{L_c}\right) + 1 \tag{21}$$

$$N_y = \text{floor}\left(\frac{y_{\max} - y_{\min}}{L_c}\right) + 1 \tag{22}$$

where floor$(\cdot)$ is an operator used to round a number downwards to the nearest integer. The total number of cells is calculated by

$$N_c = N_x N_y \tag{23}$$

Subsequently, each particle of the contact surface pair can be mapped into a cell, and the cell index is calculated by

$$i_c = i_x + i_y N_x \tag{24}$$

where $i_x = \text{floor}((x - x_{\min})/L_c)$ and $i_y = \text{floor}((y - y_{\min})/L_c)$ are indices of the cell along the x and y directions, respectively, and (x, y) is the coordinate of the particle.

Algorithm 2: Parallel algorithm for constructing the data structure

```
1: parallel for $0 \le iCell \le (N_c - 1)$ do
2:     initialize the array d_head: $d\_head[iCell] = -1$
3: end parallel
4: parallel for $0 \le iParticle \le (N_p - 1)$ do
5:     initialize the array d_next: $d\_next[iParticle] = -1$
6: end parallel
7: parallel for $0 \le iParticle \le (N_p - 1)$ do
8:     compute the cell index, iCell, of this particle by Eq. (24)
9:     modify array d_head: $iHead =$ atomicExch(d_head + iCell, iParticle)
10:    update array d_next: $d\_next[iParticle] = iHead$
11: end parallel
```

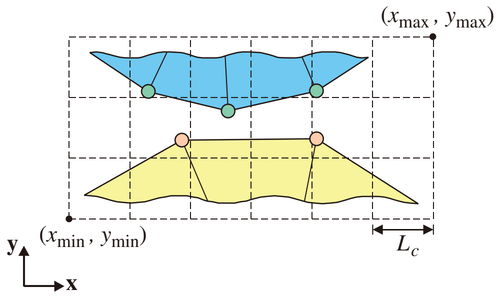

Figure 5: Space decomposition of the bounding box of a contact surface pair

A linked-list data structure [37] is adopted to store the particles in each cell. This data structure is beneficial to the parallel global search process and suitable for the coalescence access of GPU memory. A bounding box that is divided into 16 cells is taken as an example, as shown in Fig. 6, with 20 contact particles located within these cells. Two arrays are used to construct the linked list, as illustrated in Fig. 7a. The array *d_head* is used to store the first contact particle index in each cell, and the array *d_next* is used to link the remaining particle indices in the same cell [38]. Note that each element in array *d_next* indicates the index of the next particle until -1 is encountered. For example, particles 10, 8, and 7 are located in the 6th cell ($i_x = 2$, $i_y = 1$); therefore, particle 10 is stored in the 6th element of array *d_head*, while particles 8 and 7 are stored in the 10th and 8th elements of array *d_next*, respectively, and the 7th element of array *d_next* is set to -1 to indicate the end of the linked list. The lengths of these two arrays are N_c and N_p, respectively, where N_p denotes the number of contact particles.

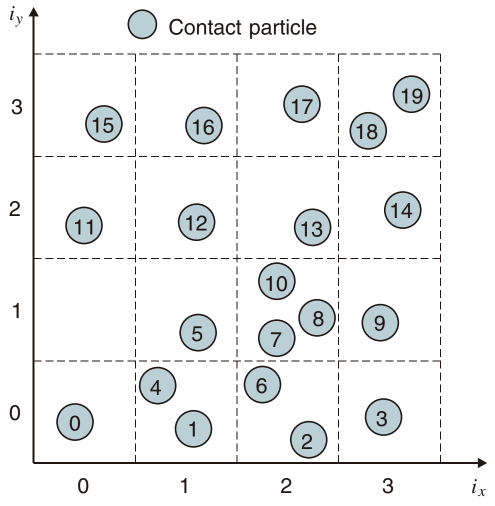

Figure 6: Mapping of contact particles into cells

The main steps for constructing the data structure for a contact surface pair are described in Algorithm 2. The elements of arrays *d_head* and *d_next* are initialized to -1. The indices of particles located in the same cell are stored in the linked list one by one. Note that the function $b =$ **atomicExch(***addr*, *a***)** is an atomic function in the CUDA toolkit to replace the value at the address *addr* with value *a* and then return the old value *b*, which performs a read-modify-write atomic operation in a thread and guarantees no interference with other threads [27]. For multiple surface pairs, the contact particles of all pairs can be stored in a single linked-list data structure, as shown in Fig. 7b. This data structure ensures the efficiency of parallel contact search for contact problems with multiple surface pairs, which is validated in Section 5.2.

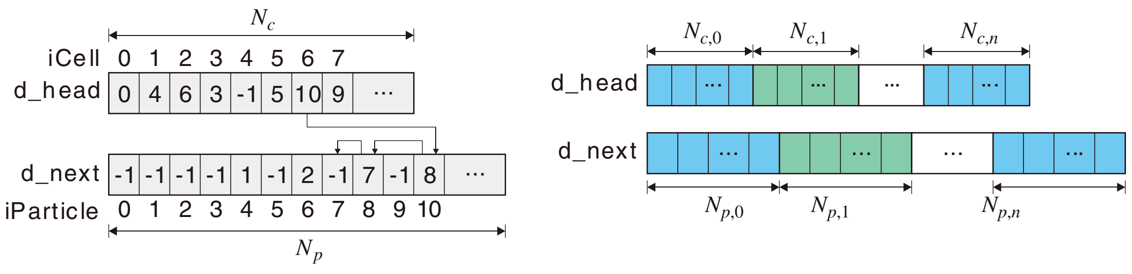

Figure 7: Linked-list data structure: (a) One surface pair; (b) Multiple surface pairs

3.1.2 Parallel Global Search

The global search consists of two steps: (1) find the closest master particle for each slave particle and (2) choose the segments connected to this master particle as master segment candidates. The closest master particle can be identified by looping for all master particles, which is time-consuming. To improve the efficiency, only particles located in the nine (3 × 3) neighboring cells are detected in this study [39]. The loop of the particles in each cell takes advantage of the well-organized data structure built above. Note that this approach will miss some segment candidates if the cell size is too small. As shown in Fig. 8, if the two particles of a segment are located outside of the nine cells, this segment cannot be detected. Therefore, the cell size should satisfy the following condition:

$$L_c \geq L_{\max}/3 \tag{25}$$

where $L_{\max}$ is the maximum length of all master contact segments. The parallel global search procedures for each slave particle are summarized in Algorithm 3.

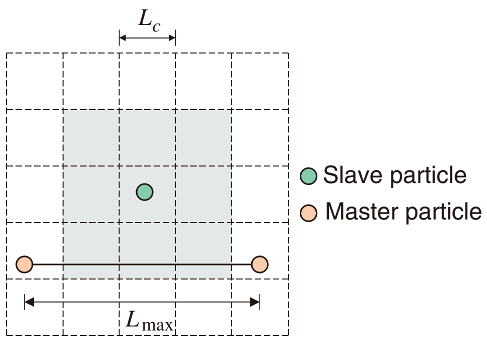

Figure 8: Minimum cell size length requirement

Algorithm 3: Parallel algorithm for global search

```
// Comments:
// $N_s$—the number of slave particles
// [$cell\_min\_x$, $cell\_max\_x$]—index range of detected cells along the $x$ direction
// [$cell\_min\_y$, $cell\_max\_y$]—index range of detected cells along the $y$ direction
1: parallel for $0 \le iSlave \le (N_s - 1)$ do
2:     initialize minimum distance, $d\_min$, to a large value
3:     initialize index of closest particle: $iClosest = -1$
4:     for $cell\_min\_y \le iCell\_y \le cell\_max\_y$ do
5:         for $cell\_min\_x \le iCell\_x \le cell\_max\_x$ do
6:             compute the cell index, $iCell$, using Eq. (24)
7:             get the first particle in this cell: $iParticle = d\_head[iCell]$
8:             while $iParticle \ge 0$ do
9:                 if $iParticle$ is a master particle then
10:                    compute distance, $d$, between $iSlave$ and $iParticle$
11:                    if $d < d\_min$ then
12:                        update the minimum distance: $d\_min = d$
13:                        update closest particle: $iClosest = iParticle$
14:                    end if
15:                end if
16:                get the next particle in this cell: $iParticle = d\_next[iParticle]$
17:            end while
18:        end for
19:    end for
20: end parallel
```

3.1.3 Parallel Local Search

To choose the exact master segment, projections of the slave particle onto each segment candidate should be performed first. Fig. 9 illustrates the projection of a slave particle S onto a master segment candidate CM. Particle C is the closest master particle to particle S. The position vectors of these three particles are denoted by $\mathbf{X}_S$, $\mathbf{X}_C$, and $\mathbf{X}_M$, respectively.

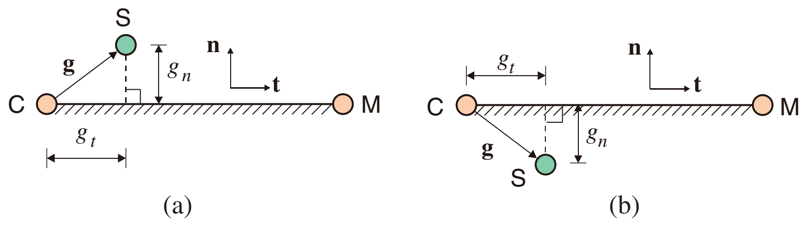

Figure 9: Projection of slave particle onto a master segment candidate: (a) Open gap ($g_n > 0$); (b) Closed gap ($g_n \le 0$)

An LCS with unit vectors $\mathbf{t}$, $\mathbf{n}$ and $\mathbf{e}_3$ is introduced. The tangent vector $\mathbf{t}$ is parallel to the master segment

$$\mathbf{t} = \frac{\mathbf{m}}{\|\mathbf{m}\|} \tag{26}$$

where $\mathbf{m} = \mathbf{X}_\mathrm{M} - \mathbf{X}_\mathrm{C}$ is the master segment vector, the vector $\mathbf{e}_3$ is orthogonal to the plane containing contact bodies, and the normal vector $\mathbf{n} = \mathbf{e}_3 \times \mathbf{t}$. The gap normal and tangential projections, g_n and g_t, are determined by

$$g_n = \mathbf{g} \cdot \mathbf{n}, \quad g_t = \mathbf{g} \cdot \mathbf{t} \tag{27}$$

where $\mathbf{g} = \mathbf{X}_S - \mathbf{X}_C$ is the gap vector. In addition, a normalized gap tangential projection, ζ, is introduced by

$$\zeta = \frac{g_t}{\|\mathbf{m}\|} \tag{28}$$

and the range of ζ is

$$-\zeta_\mathrm{tol} \le \zeta \le 1 + \zeta_\mathrm{tol} \tag{29}$$

where ζ_tol is a tolerance that can be set to 1/50 [40].

Generally, three projection cases exist, as illustrated in Fig. 10, i.e., a regular case and two special cases: the "in-of-both" case and "out-of-both" case. For the regular case shown in Fig. 10a, g_n is calculated by Eq. (27). For the other two special cases shown in Figs. 10b and 10c, the contact between the slave particle and its master segments is simplified to the contact between the slave particle and its closest master particle. Note that more accurate treatments for these two special cases can be found in the literature [40].

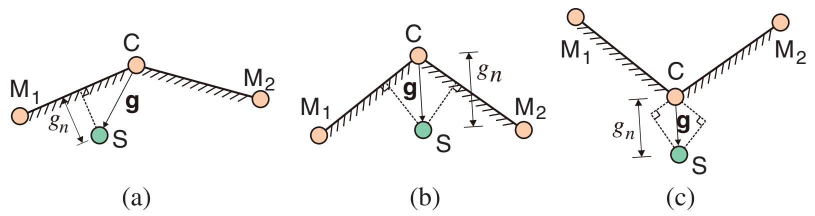

Figure 10: Projection cases: (a) Regular case; (b) "In-of-both" case; (c) "Out-of-both" case

Once the master segment has been identified, the gap status (open or closed) can be determined by checking the sign of g_n, as shown in Fig. 9. An open gap indicates that the contact surfaces are separated with no need to enforce contact constraints, while a closed gap indicates that the contact forces should be calculated and exerted on the contact particles, which is discussed in Section 3.2. The parallel local search procedures for each slave particle are summarized in Algorithm 4.

3.2 Contact Forces

The penalty method is adopted to calculate the contact forces of particles due to its simplicity and ease of implementation. The calculations of the normal contact force and tangential contact force (i.e., friction force) using the penalty method are discussed in Sections 3.2.1 and 3.2.2, respectively. Note that the contact forces are calculated in parallel for each slave particle, and then the reaction forces are applied in parallel to the corresponding master particles.

Algorithm 4: Parallel algorithm for local search

```
// Comments:
// $N_s$—the number of slave particles
1: parallel for $0 \leq iSlave \leq (N_s - 1)$ do
2:     read the index of closest master particle: $iClosest$
3:     get the number of segments, $nSeg$, connected to the particle $iClosest$
4:     if $nSeg = 1$ then
5:         calculate $\zeta$ using Eq. (28)
6:         if $\zeta$ satisfies Eq. (29) then
7:             regular projection case
8:         else then
9:             the master segment for this particle is not exist
10:        end if
11:    elseif $nSeg = 2$ then
12:        calculate $\zeta_0$ and $\zeta_1$ using Eq. (28)
13:        if both $\zeta_0$ and $\zeta_1$ satisfy Eq. (29) then
14:            in-of-both case
15:        elseif $\zeta_0$ satisfies Eq. (29) then
16:            regular projection case, the master segment is the first one
17:        elseif $\zeta_1$ satisfies Eq. (29) then
18:            regular projection case, the master segment is the second one
19:        else then
20:            out-of-both case
21:        end if
22:    end if
23: end parallel
```

3.2.1 Normal Contact Force

The magnitude of the normal contact force calculated by the penalty method is proportional to the amount of penetration, as shown in Fig. 11. For a contact pair composed of slave particle S and master segment CM, assuming that its projection case is regular (see Fig. 10a), the normal contact force is determined by

$$\mathbf{F}_n = \epsilon_n |g_n| \cdot \mathbf{n} \tag{30}$$

where ϵ_n is the normal penalty parameter of the master segment CM, g_n is the normal gap, and $\mathbf{n}$ is the normal unit vector of CM. The penalty parameter is obtained using the following equation proposed by Hallquist et al. [5,21]:

$$\epsilon_n = s_n \frac{KA^2}{V} \tag{31}$$

where K is the bulk modulus of segment CM, A is the area of the segment, V is the volume of the element that contains segment CM, s_n is a scale factor to adjust the normal penalty parameter. Note that appropriate penalty parameters are important for contact force calculation, because too large penalty parameter leads to non-convergent results, while too small penalty parameter results in unrealistic large penetration. In this paper, the scale factor s_n is initially set to default value of 1.0 and then adjusted via a trial-and-error process until a satisfying result is obtained.

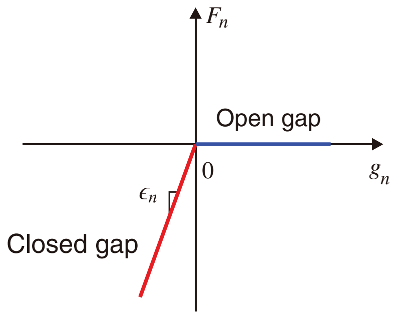

Figure 11: The relation between the normal contact force and normal gap

For the other two special projection cases, as shown in Figs. 10b and 10c, ϵ_n in Eq. (30) is set to the average value of the penalty parameters of segments CM_1 and CM_2. This approach is used to consider the contact stiffness contributions of both segments and reduce the variation of contact stiffness when the slave particle slides from one master segment to the other.

3.2.2 Friction Force

The most commonly used friction model is the classic Coulomb friction model [41]. However, the friction force of the Coulomb model is discontinuous at zero velocity, which may lead to oscillation results. A smoothed friction model [5] is adopted in this study. The friction force calculation is an adaptation of the radial return algorithm for elastic-perfectly plastic materials and can be described as follows:

Let ${}^t\mathbf{F}_n$ represent the normal contact force of slave particle S at time t calculated by Eq. (30), μ denote the coefficient of friction, and ${}^{t-\Delta t}\mathbf{F}_s$ represent the friction force at time $t-\Delta t$. Then, the friction force at time t is calculated by the following steps:

(1) Compute the magnitude of the maximum static friction force:

$$ {}^tF_{s,\max} = \mu \| {}^t\mathbf{F}_n \| \tag{32} $$

(2) Calculate the trial friction force:

$$ {}^t\mathbf{F}_s^* = {}^{t-\Delta t}\mathbf{F}_s - \epsilon_t \left({}^t\mathbf{d}_r - {}^{t-\Delta t}\mathbf{d}_r \right) \tag{33} $$

where ${}^{t-\Delta t}\mathbf{d}_r$ and ${}^t\mathbf{d}_r$ denote the relative displacements between slave particle S and master segment CM at times $t-\Delta t$ and t, respectively; ϵ_t is the tangential penalty parameter of master segment CM, which is determined by:

$$ \epsilon_t = s_t \frac{KA^2}{V} \tag{34} $$

where s_t is a scale factor with a defau value of 1.0 and is used to adjust this penalty parameter.

(3) If the trial force does not exceed the maximum friction force, the friction is identified as static, and the friction force at time t is equal to thitrial force; otherwise, the friction is identified as kinetic, and the magnitude of the friction force at time t should equal ${}^tF_{s,\max}$.

$$ {}^t\mathbf{F}_s = \begin{cases} {}^t\mathbf{F}_s^* & \text{if } \|{}^t\mathbf{F}_s^*\| \le {}^tF_{s,\max} \\ {}^tF_{s,\max}\frac{{}^t\mathbf{F}_s^*}{\|{}^t\mathbf{F}_s^*\|} & \text{if } \|{}^t\mathbf{F}_s^*\| > {}^tF_{s,\max} \end{cases} \tag{35} $$

Note that the friction force is considered only in the regular projection case (see Fig. 10a).

Once the normal and tangential contact forces of slave particle S have been obtained, the resultant contact force, as shown in Eq. (1), is cculated by

$$ \mathbf{F}_{\mathrm{S}}^c = \mathbf{F}_n + \mathbf{F}_s \tag{36} $$

As shown in Fig. 12a, the reaction contact forces applied to the particles of the master segment CM in the regular case can be calculated by

$$ \mathbf{F}_{\mathrm{C}}^c = -(1-\zeta)\mathbf{F}_{\mathrm{S}}^c,\ \mathbf{F}_{\mathrm{M}}^c = -\zeta\mathbf{F}_{\mathrm{S}}^c \tag{37} $$

In the other two special projection cases, the reaction force applied to the closest master particle C has the same magnitude but opposite direction compared to $\mathbf{F}_{\mathrm{S}}^c$, as shown in Figs. 12b and 12c.

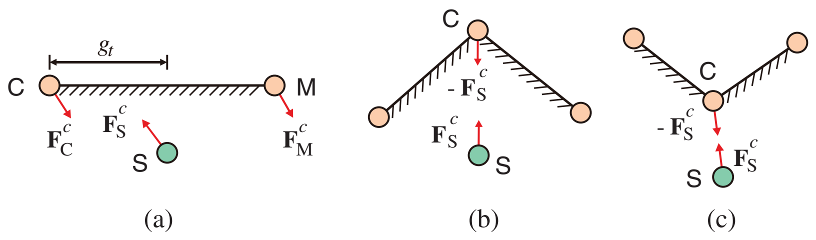
g_t
C
M
$\mathbf{F}_{\mathrm{C}}^c$
$\mathbf{F}_{\mathrm{S}}^c$
S
$\mathbf{F}_{\mathrm{M}}^c$
(a)
C
$-\mathbf{F}_{\mathrm{S}}^c$
$\mathbf{F}_{\mathrm{S}}^c$
S
(b)
C
$-\mathbf{F}_{\mathrm{S}}^c$
$\mathbf{F}_{\mathrm{S}}^c$
S
(c)

Figure 12: Reaction contact force of master particles: (a) Regular case; (b) “In-of-both” case; (c) “Out-of-both” case

4 GPU Implementation

The proposed contact algorithm based on the FPM is suitable for parallel implementation, as described in Sections 2 and 3. The GPU architecture is more appropriate than the CPU architecture for the parallel implementation of the FPM, as mentioned in Section 1. The parallel contact algorithm is implemented in the GPU-accelerated software developed in our previous study [24]. The algorithm is executed on NVIDIA GPUs powered by CUDA. The CUDA programming model is briefly introduced in Section 4.1, and the GPU implementation of the algorithm is presented in Section 4.2.

4.1 CUDA Programming Model

A generic CUDA application mainly consists of two parts: host code, running serially on a CPU, and device code, running in parallel on a GPU, as illustrated in Fig. 13. The host code is responsible for organizing host and device memories and managing the transfer of data between them. The device code consists of several device functions, named kernels, that are executed on GPU threads. The threads are organized into n-dimensional blocks, which are then organized into an n-dimensional grid (where n can be 1, 2, or 3), as depicted in Fig. 13. For a more detailed description of the CUDA programming model, the reader is referred to the literature [27].

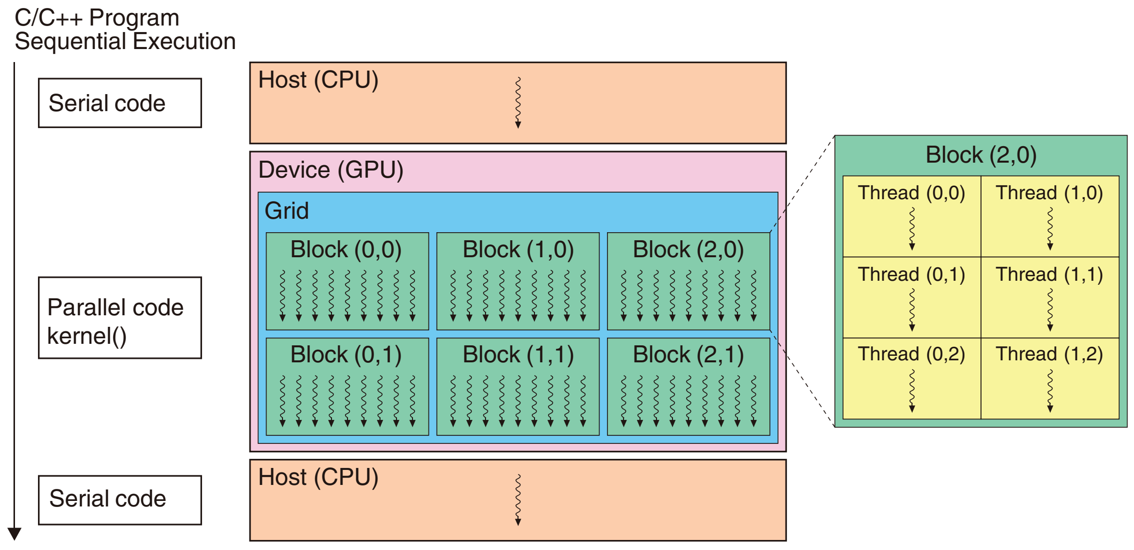

Figure 13: Schematic illustration of the CUDA programming model

4.2 GPU Implementation of the Algorithm

The flowchart of the parallel contact algorithm implemented in the FPM software is presented in Fig. 14. After preprocessing, the model data and variables are organized in the host memory and transferred to the device memory. A sequence of time steps is then performed for the analysis. Within each time step, the computational procedures are executed in a series of CUDA kernels managed by three FPM solvers: element solver, contact solver, and particle solver. Typically, the solution procedures consist of four main steps:

(1) **Element solver:** evaluate the elemental internal forces in parallel for each element, as described in Section 2.2.
(2) **Contact solver:**
    i. **Parallel global search:** search for master segment candidates for each slave particle, as discussed in Section 3.1.2.
    ii. **Parallel local search:** choose the master segment for each slave particle from the candidates, and calculate the gap vector, as described in Section 3.1.3.
    iii. **Parallel contact force calculation:** calculate the normal and tangential contact forces for each slave particle, and apply the reaction contact forces to the corresponding master particles, as discussed in Section 3.2.
(3) **Particle solver:** solve the equation of motion in parallel for each particle using Eq. (3).
(4) **Time step controller:** if the current time step is the last one, stop the calculation; otherwise, go to Step (1) and start the iteration for the next time step.

As described in our previous study [24], memory management is important to achieve satisfactory performance of the FPM solvers. Among all memory spaces available in CUDA, global device memory is the most widely used for all FPM solvers because it has the largest storage capacity. The global memory can be accessed by all threads using aligned memory transactions. The optimization of global memory throughput can be achieved if the memory access patterns are suitable for coalescence [25], which requires adjacent threads to access successive memory addresses. To achieve this coalescence of memory accesses, an adaptation of the structure-of-arrays (SoA) storage pattern [42] is adopted to manage all data buffers in global memory. In addition to memory management, the thread block size impacts the kernel performance. Based on performance tests for most of the kernels implemented in this study, 128 threads per block results in the best efficiency; thus, for convenience, the thread block size is set to a constant value of 128 for all kernels.

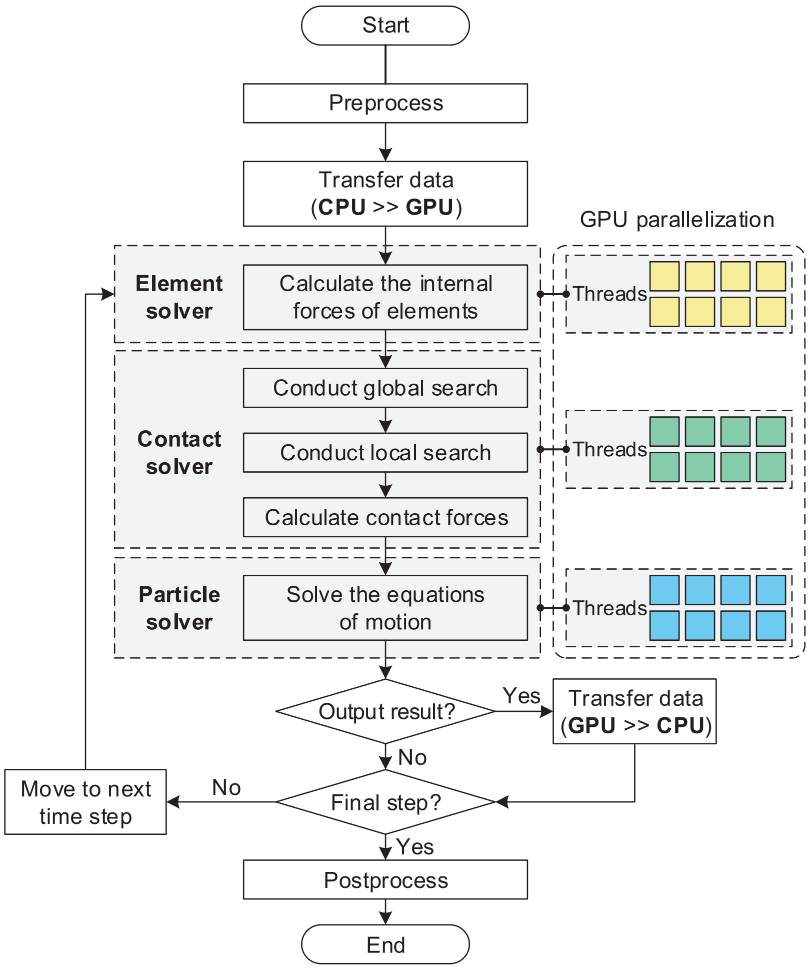

Figure 14: Flowchart of the GPU-based parallel contact algorithm

5 Numerical Examples

The effectiveness of the proposed contact algorithm is verified by two examples as presented in Section 5.1. Subsequently, the performance of the algorithm is tested in Section 5.2. All examples are computed under the assumption of a plane strain condition with a unit thickness of 1.0.

5.1 Verification

In this section, two verification examples are presented to demonstrate the accuracy of the proposed contact algorithm. These two examples focus on the frictional contact between two deformable bodies undergoing large sliding and large deformation. The first example involves elastic behavior, while the second example involves elastoplastic behavior. Fictitious mass damping is used in the verification examples to obtain static solutions. The quadrilateral element FPM-Q4, as described in Section 2.2.4, is used for the analyses in this section.

5.1.1 Contact in Elastic Problem

This numerical example is the so-called frictional ironing problem, which was previously studied in the literature [4,43]. The example comprises the sliding of a deformable block along an elastic slab, as illustrated in Fig. 15. The slab is fixed at its bottom surface, whereas the block is subjected to an imposed displacement at its upper surface. There are two load steps. From 0 to 1 s, a downward displacement $u_y = -10$ mm is applied to the block, while from 1 to 2 s, the block slides horizontally along the slab with a displacement $u_x = 250$ mm. The material parameters are given in Fig. 15, and the mass damping coefficient is set to $\xi = 1.0\times 10^5$. The friction coefficient between the contact surfaces is $\mu = 0.3$. The block and slab are discretized into 136 and 3,840 quadrilateral elements, respectively. The scale factors for the normal and tangential penalty parameters are set to $s_n = 20.0$ and $s_t = 0.7$, respectively.

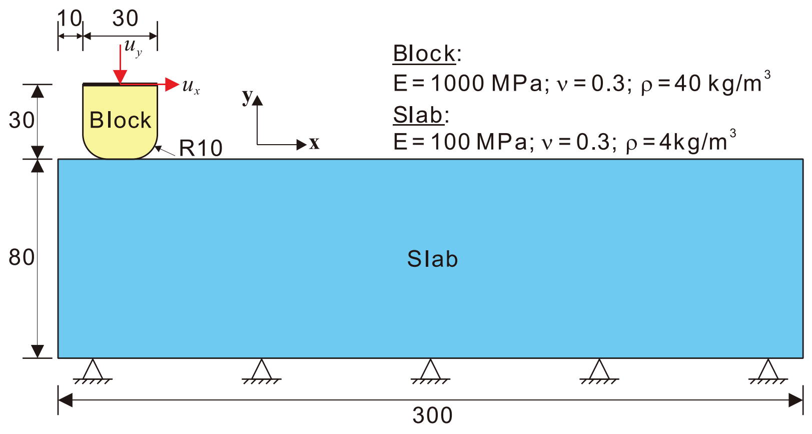
10
30
u_y
u_x
30
Block
R10
y
x
Block:
E = 1000 MPa; ν = 0.3; ρ = 40 kg/m^3
Slab:
E = 100 MPa; ν = 0.3; ρ = 4kg/m^3
80
Slab
300

Figure 15: Geometry and material properties of the ironing problem (unit of length: mm)

The deformed configurations with a shear stress distribution are presented in Fig. 16 for three typical instants. In addition, the evolution of the total contact force components between the contacting bodies is presented in Fig. 17. From 0 to 1 s, as the vertical displacement of the block increases, the normal contact force gradually increases. The deformed configuration of the slab is not symmetrical; thus, the horizontal force component is not equal to zero. Starting from 1 s, the block begins to slide on the slab. The normal contact force increases slightly and remains constant afterwards. The horizontal force increases sharply after 1 s due to the contribution of the friction force and remains constant afterwards. Both contact force components drop slightly when the block reaches the right end of the slab. The results calculated by the FPM agree well with those obtained by Neto et al. using the surface smoothing method [4]. Thus, this example demonstrates the effectiveness of the proposed contact algorithm in elastic problems.

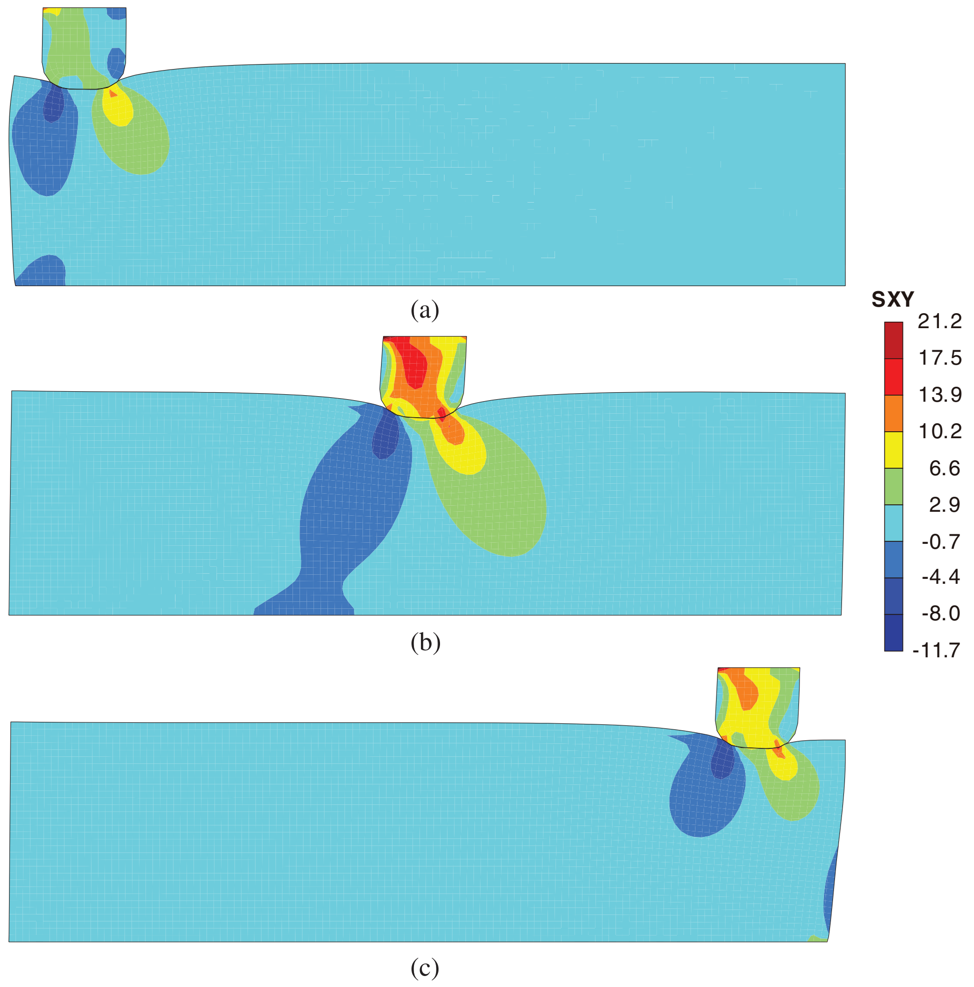

Figure 16: Contour plots of shear stress (unit: MPa) at three instants: (a) 1.0 s; (b) 1.5 s; (c) 2.0 s

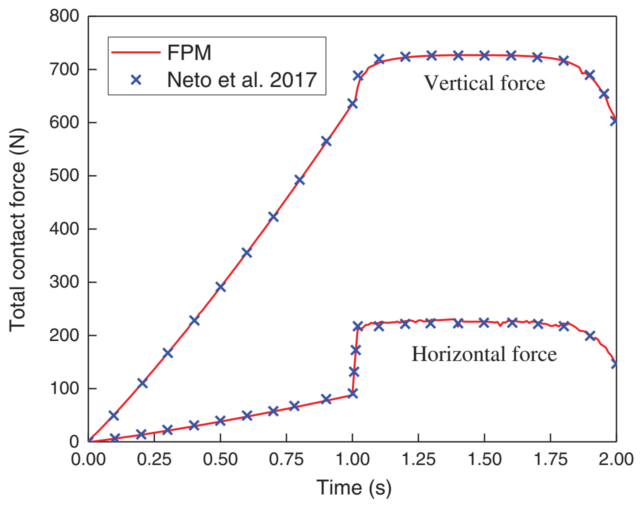

Figure 17: Contact forces obtained by the FPM and surface smoothing method [4]

5.1.2 Contact in Elastoplastic Problem

The second example was proposed by Yang et al. [44] and comprises contact between two curved beams, as shown in Fig. 18. The lower beam is fixed at its bottom surface, while the upper beam is subjected to a horizontal displacement $u_x = 31.5$ mm at its upper surface. Both beams are modeled using an elastoplastic material with isotropic hardening. The elastic material properties are given by Young's modulus $E = 689.56$ MPa and Poisson's ratio $\nu = 0.32$, while the plastic properties are given by the yield stress $\sigma_0 = 31$ MPa and the plastic modulus $h = 261.2$ MPa. The density of the material is $\rho = 50$ kg/m^3, and the mass damping coefficient is set to $\xi = 1.0 \times 10^4$. Each beam is discretized into 240 quadrilateral elements. Both frictionless and frictional cases are investigated, and two different values of the friction coefficient are considered: $\mu = 0.3$ and $\mu = 0.6$. The scale factors for the normal and tangential penalty parameters are set to $s_n = 10.0$ and $s_t = 1.0$, respectively.

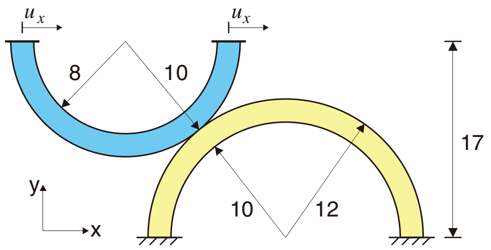

Figure 18: Geometry of the contact problem between two curved beams (unit of length: mm)

Deformed configurations of the beams for 15 mm of prescribed displacement on the upper beam are depicted in Fig. 19 for both frictionless and frictional cases, with the arrows illustrating the nodal contact forces subjected by the lower beam. The contact forces in the frictionless case are normal to the contact surface, while the contact forces in the other two frictional cases comprise tangential components. A higher friction coefficient leads to a higher friction force.

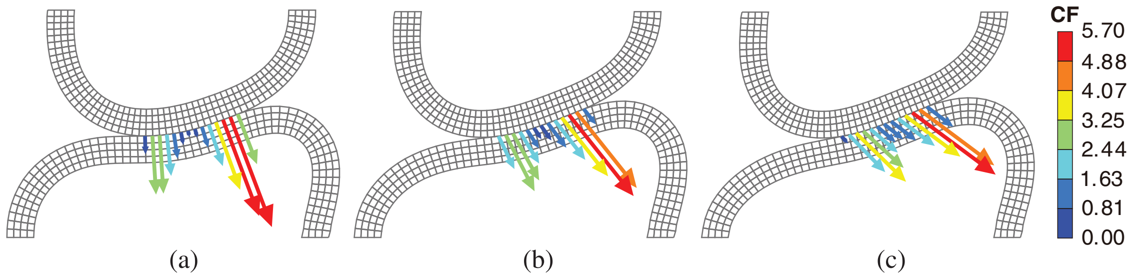

Figure 19: Deformed configurations of the beams and nodal contact forces (unit: N) of the lower beam: (a) $\mu = 0.0$; (b) $\mu = 0.3$; (c) $\mu = 0.6$

The evolution of the total contact force components for the upper beam is presented in Fig. 20. Both the horizontal and vertical contact forces calculated by the FPM agree well with the results obtained by Neto et al. using the surface smoothing method [45]. Given higher values of friction coefficient, the amplitude of contact forces increases, the inflection points of the curves occur later, and contact forces decrease faster. These phenomena are related to the deformation mode of the lower beam due to the friction force.

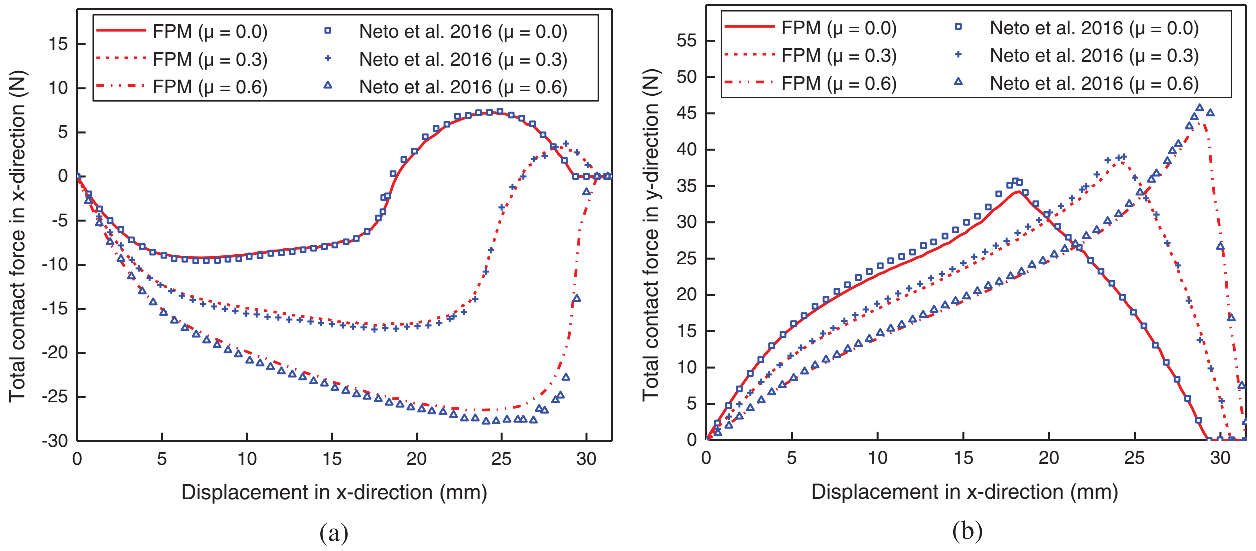

Figure 20: Contact forces obtained by the FPM and surface smoothing method [45]): (a) *x*-direction; (b) *y*-direction

The deformed configuration of the beams with contour plots of equivalent plastic strain is presented in Fig. 21, which is similar to the results in the literature [45,46]. The plastic regions mainly appear in the lower beam due to its larger diameter. Besides, the equivalent plastic strain in the region near the contact area is lower given higher values of friction coefficient. This can be explained by the different deformation shapes of the lower beam, as shown in Fig. 19.

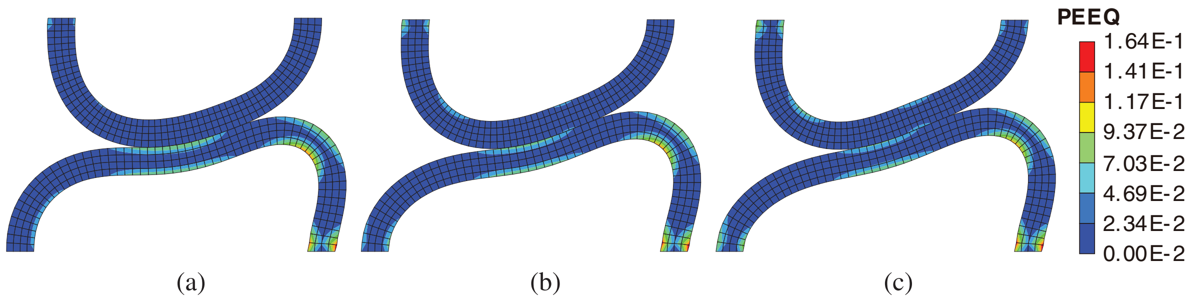

Figure 21: Contour plots of equivalent plastic strain: (a) $\mu = 0.0$; (b) $\mu = 0.3$; (c) $\mu = 0.6$

In conclusion, the proposed contact algorithm is also effective in analyzing elastoplastic problems.

5.2 Performance Test

The performance of the proposed method is tested in this section. The performances of parallel element and particle solvers are studied first via a benchmark problem without contact calculation in Section 5.2.1. Then the performance of the parallel contact solver is investigated via a large-scale contact problem in Section 5.2.2. The performances of the parallel FPM solvers are compared with the performances of Abaqus/Explicit solvers and our serial FPM solvers. Note that the kernel codes in the serial FPM solvers are almost identical to the codes in the parallel solvers, but they run serially on a CPU. The efficiency of the parallel implementation is quantified by introducing speedup ratio S, which is defined as

$$S = \frac{T_S}{T_P} \tag{38}$$

where T_S is the computational time of a serial algorithm, and T_P is the computational time of a parallel algorithm.

The quadrilateral element FPM-Q4R in the FPM as described in Section 2.2.4 is used in the performance test, while the 4-node quadrilateral element with reduced integration CPE4R in Abaqus/Explicit is used for comparison. The performance test is carried out on a PC equipped with an Intel® Core™ i7-4790 K quad-core 4.00 GHz CPU, an NVIDIA Titan V GPU with 5120 CUDA cores and 12 GB HBM2 memory, and a Windows 10 (64-bit platform) operating system.

5.2.1 *Test on Parallel Element and Particle Solvers*

As shown in Fig. 22, a cantilever beam is fixed at one edge and subjected to a tip load. The load ($P = 100$ N) is applied at time $t = 0$ and then kept constant during the whole simulation. The beam is 100 mm long and 20 mm height. It is made of an elastic material characterized by Young's modulus $E = 2000$ MPa, Poisson's ratio $\nu = 0.3$, and density $\rho = 3 \times 10^3$ kg/m^3. A dynamic analysis of the beam is carried out for 8 ms of simulation. The deformed configuration of the cantilever beam at time $t = 8$ ms obtained by FPM is presented in Fig. 23. The evolution of the vertical displacement of the cantilever tip obtained by FPM agrees well with that obtained by Abaqus, as shown in Fig. 24.

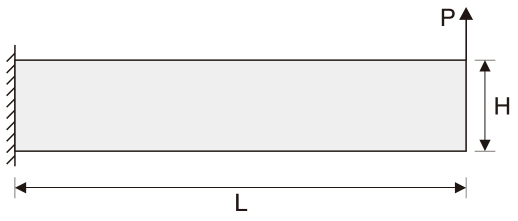

Figure 22: Geometry of cantilever beam

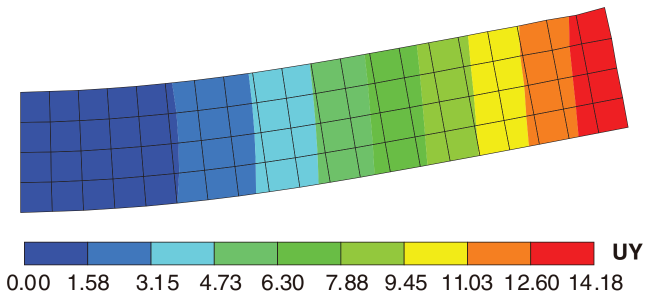

Figure 23: Vertical displacement contour of cantilever beam (unit of length: mm)

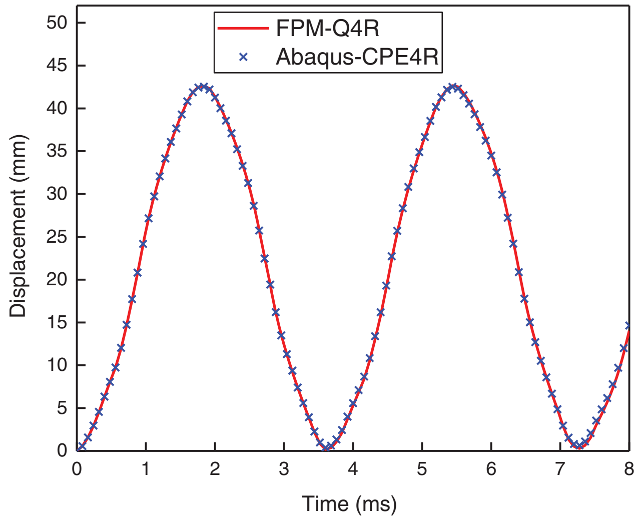

Figure 24: Vertical displacement of the tip node of the cantilever beam

To compare the performances of parallel FPM solvers relative to serial FPM solvers and the Abaqus/Explicit solver, different meshes of the cantilever are tested. Each mesh is performed with a simulation of 0.1 ms with a fixed time step size of 1.0×10^{-5} ms, resulting in 10,000 time increments in total. The computational times taken by Abaqus and FPM are listed in Tab. 1 and compared in Fig. 25. As the number of elements increases, the computational time required by parallel FPM solvers grows at a much lower rate than that of serial FPM solvers and Abaqus solvers. The speedup of parallel FPM solvers relative to serial FPM solvers reaches 35.0 when the number of elements approaches 3,000,000, and the speedup relative to Abaqus reaches 21.7 when the number of elements approaches 4,000,000. The maximum GPU utilization of the parallel element and particle solvers is obtained when the speedup reaches its saturation point, which means that all GPU threads cannot be concurrently processed due to the limitation of GPU memories and cores, and the time consumed by synchronization of threads and other processes leads to the slight decrease of the speedup ratio. If a more advanced GPU with larger memories and more CUDA cores is used, the speedup can be further improved.

Table 1: Computational time for calculating the cantilever beam (unit of time: s)

Elements	Abaqus (CPU)	FPM (CPU)	FPM (GPU)
20,480	24.0	43.5	6.7
81,920	104.0	167.1	12.2
163,840	207.0	374.7	16.9
327,680	412.0	744.6	27.4
655,360	829.0	1528.9	47.7
1,310,720	1636.0	3019.2	87.1
2,621,440	3253.0	5538.6	158.0
3,932,160	4913.0	8409.4	241.7
5,242,880	6437.0	10864.2	320.0

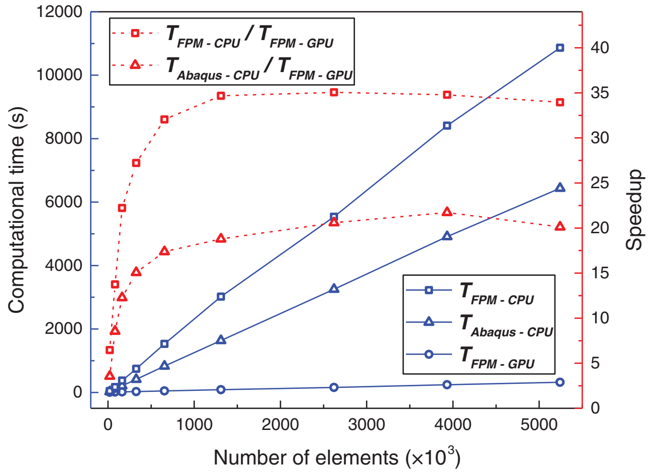

Figure 25: Comparison of computational time for calculating the cantilever beam

5.2.2 *Test on Parallel Contact Solver*

A two-dimensional simplification of the dynamic impact problem proposed by [38] is considered in this test, as shown in Fig. 26. A half-tube is moving toward a multilayer sheet with a prescribed displacement u_y. The sheet is divided into n layers with the same thickness t_l, and the gap between two adjacent layers is denoted by g_l. The nodes on both the leftmost and rightmost sides of the sheet are fixed only along the vertical direction.

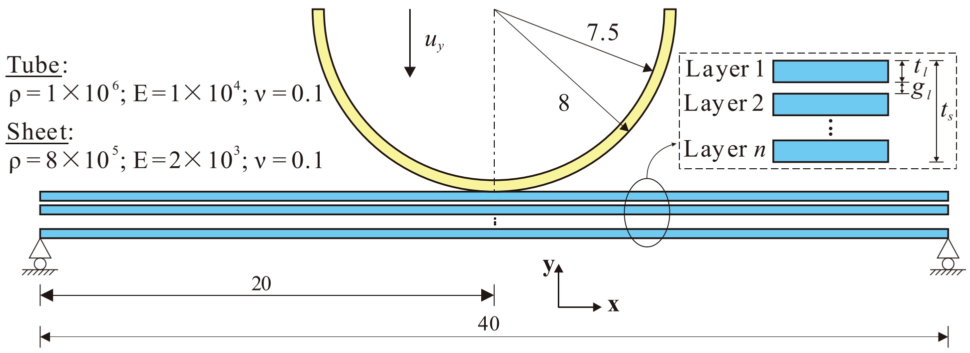
Tube:
ρ = 1×10^6; E = 1×10^4; ν = 0.1
Sheet:
ρ = 8×10^5; E = 2×10^3; ν = 0.1
u_y
7.5
8
Layer 1
Layer 2
Layer n
t_l
g_l
t_s
20
40
y
x

Figure 26: Contact between a half-tube and a multilayer sheet: Geometry and material properties (unit of length: mm; unit of density: kg/m^3; unit of modulus: MPa)

A three-layer sheet is taken as an example to demonstrate the results, where $t_l = 0.4$ mm and $g_l = 0.02$ mm. The scale factor for the normal penalty parameter is set to $s_n = 1.0$. The deformed shapes at three instants obtained by FPM are shown in Fig. 27, and they are very similar to the results obtained by Abaqus/Explicit, which are omitted here for brevity.

To test the performance and efficiency of the proposed contact algorithm, sheets with different numbers of layers are tested. The layer thickness is set to $t_l = 0.01$ mm and the gap is set to $g_l = 0.001$ mm. The tube is discretized with a fixed mesh size of 0.02 mm, and each layer of the sheet is discretized with a fixed mesh size of 0.01 mm. Each case is performed with a simulation time of 1.0 ms with a fixed time step size of 1.0×10^{-4} ms, resulting in 10,000 time increments in total.

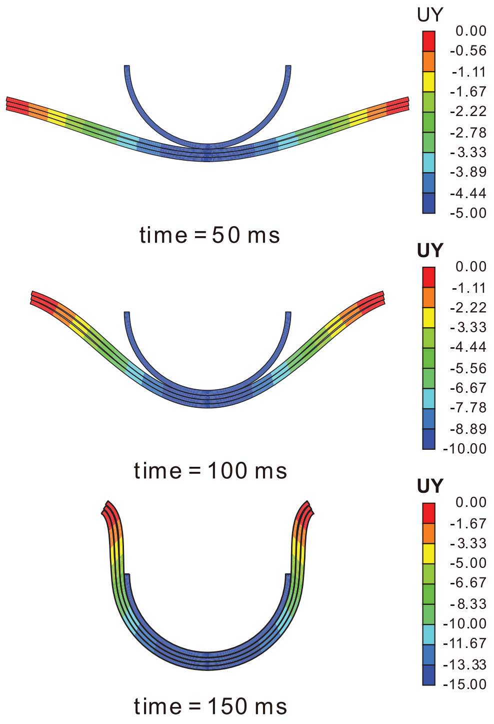

Figure 27: Vertical displacement contour at three instants (unit of length: mm)

The times of both total calculation and contact calculation taken by Abaqus/Explicit and both serial and parallel FPM solvers are listed in Tab. 2. Note that the time of contact calculation taken by Abaqus/Explicit cannot be extracted directly, and it is approximated by subtracting the time of calculation without contact from the time of calculation with contact.

Table 2: Contact between tube and sheet: total computational time and time of contact calculation (unit of time: s)

Layers	Elements	Contact segments	Abaqus (CPU)		FPM (CPU)		FPM (GPU)	
			Contact	Total	Contact	Total	Contact	Total
10	70,450	77,218	56.0	152.0	24.4	144.1	3.0	12.9
50	230,450	397,218	300.0	622.0	127.3	511.5	5.7	28.2
100	430,450	797,218	595.0	1174.0	252.2	963.4	8.8	45.3
150	630,450	1,197,218	866.0	1711.0	380.0	1410.1	12.3	66.8
200	830,450	1,597,218	1154.0	2255.0	495.2	1861.9	15.0	82.9
250	1,030,450	1,997,218	1432.0	2803.0	644.2	2332.4	19.1	103.8
300	1,230,450	2,397,218	1674.0	3351.0	758.1	2778.3	21.6	117.6
350	1,430,450	2,797,218	1949.0	3900.0	893.5	3263.7	25.2	138.3
400	1,630,450	3,197,218	2198.0	4462.0	1031.4	3782.5	28.5	158.2

The computational time of contact calculation and the corresponding speedups are shown in Fig. 28. As the number of contact segments increases, the speedup of the parallel FPM contact solver relative to the serial one increases accordingly and reaches 36.1 when the number of contact segments exceeds 3,000,000, and the speedup relative to the Abaqus contact solver reaches 77.4

when the number of contact segments exceeds 2,000,000. It can be concluded that the serial FPM contact solver is more efficient than the Abaqus contact solver due to the proposed contact algorithm that consists of efficient global search and local search processes. The efficiency of the parallel FPM contact solver is further dramatically improved due to its high degree of parallelism.

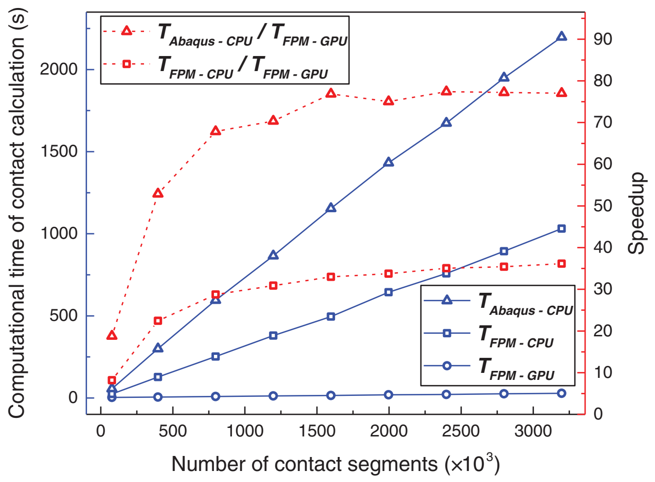

Figure 28: Contact between tube and sheet: computational time of contact calculation

The total computational time and the corresponding speedups are shown in Fig. 29. The speedup of the parallel FPM solvers relative to the serial ones reaches 23.9, and that relative to Abaqus reaches 28.5. Figs. 28 and 29 also indicate that when the number of contact segments exceeds 3,000,000, the GPU cores are fully used. Note that the maximum speedup depends heavily on the GPU hardware, and higher speedup can be achieved by using more advanced GPUs.

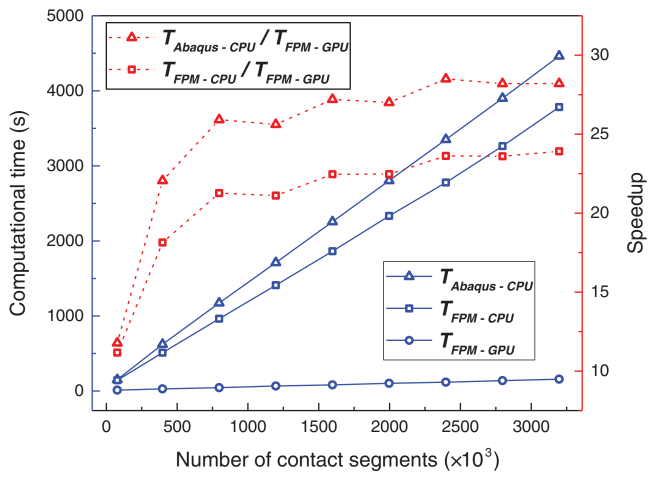

Figure 29: Contact between tube and sheet: total computational time

The percentages of contact calculation time in total calculation time are compared in Fig. 30. As the number of contact segments increases, the three curves remain nearly constant. Contact calculation accounts for approximately 50% of the total computational time in Abaqus, while the percentages in serial and parallel FPMs are only 27% and 18%, respectively. This indicates that the proposed GPU-based parallel contact algorithm can remarkably decrease the time consumption of contact calculation compared to the CPU-based serial contact algorithm.

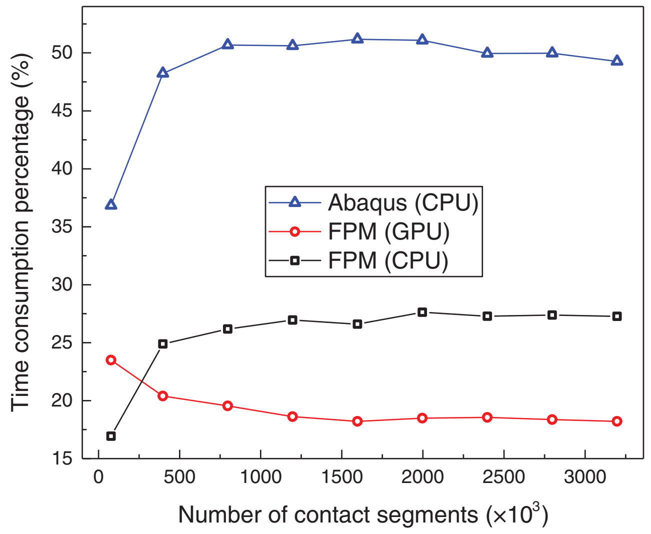

Figure 30: Contact between tube and sheet: time consumption percentages of contact calculation

The time consumption percentages of both serial and parallel FPM solvers are given in Fig. 31. In both serial and parallel cases, the element solvers are the most time-consuming solvers, while the particle solvers account for the smallest proportion. The serial contact solver accounts for a larger proportion than the parallel contact solver, demonstrating the effectiveness of the proposed parallel contact algorithm.

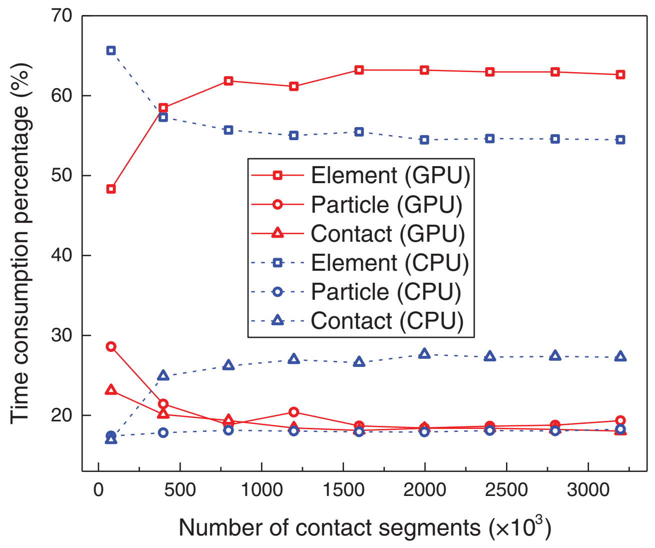

Figure 31: Contact between tube and sheet: time consumption percentages of serial and parallel FPM solvers

6 Conclusions

This paper presents a 2D GPU-based parallel contact algorithm using the FPM. A data structure suitable for parallel contact detection is introduced, and parallel global search and local search procedures are proposed. The main computational procedures, including elemental internal forces calculation, contact calculation, and solution of equations of motion, are executed in parallel on a GPU. Two verification examples and two performance tests are demonstrated, and the following conclusions can be made:

- Two typical verification examples involving large deformation frictional contacts and both elastic and elastoplastic behaviors are presented, and the results obtained by FPM agree well with those in the literature, indicating the accuracy of the FPM quadrilateral elements and the proposed contact algorithm.
- The performances of parallel element and particle solvers are studied first via a benchmark problem without contact calculation, and a maximum speedup of approximately 21.7 is achieved relative to Abaqus/Explicit when the number of elements approaches 4,000,000, and a maximum speedup of approximately 35.0 is achieved relative to serial FPM solvers when the number of elements approaches 3,000,000. Thus, the efficiency of the parallel element and particle solvers is proven.
- A large-scale contact problem is investigated. The speedup of the parallel FPM contact solver relative to the serial FPM contact solver reaches 36.1 when the number of contact segments exceeds 3,000,000, and that relative to the Abaqus contact solver reaches 77.4 when the number of segments exceeds 2,000,000. Thus, the efficiency of the parallel FPM contact solver has been proven. With respect to the total computational time, the speedup of parallel FPM solvers relative to serial FPM solvers reaches 23.9, and that relative to Abaqus reaches 28.5, demonstrating the efficiency of the parallel FPM solvers for solving large-scale contact problems.
- The percentages of contact calculation time in the total calculation time are 18% in parallel FPM, 27% in serial FPM and 50% in Abaqus/Explicit, which indicates the effectiveness of the proposed GPU-based parallel contact algorithm in reducing the time consumption proportion of contact calculation in the total calculation.

Funding Statement: This work was supported by the National Key Research and Development Program of China [Grant No. 2016YFC0800200]; the National Natural Science Foundation of China [Grant Nos. 51778568, 51908492, and 52008366]; and Zhejiang Provincial Natural Science Foundation of China [Grant Nos. LQ21E080019 and LY21E080022]. This work was also supported by the Key Laboratory of Space Structures of Zhejiang Province (Zhejiang University) and the Center for Balance Architecture of Zhejiang University.

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

References

1. Wriggers, P. (2006). *Computational contact mechanics.* Berlin: Springer-Verlag.
2. Laursen, T. A. (2003). *Computational contact and impact mechanics: Fundamentals of modeling interfacial phenomena in nonlinear finite element analysis.* Berlin, Heidelberg: Springer-Verlag.
3. Puso, M. A., Solberg, J. M. (2020). A dual pass mortar approach for unbiased constraints and self-contact. *Computer Methods in Applied Mechanics and Engineering, 367(1),* 113092. DOI 10.1016/j.cma.2020.113092.

4. Neto, D. M., Oliveira, M. C., Menezes, L. F. (2017). Surface smoothing procedures in computational contact mechanics. *Archives of Computational Methods in Engineering, 24(1),* 37–87. DOI 10.1007/s11831-015-9159-7.
5. Benson, D. J., Hallquist, J. O. (1990). A single surface contact algorithm for the post-buckling analysis of shell structures. *Computer Methods in Applied Mechanics and Engineering, 78(2),* 141–163. DOI 10.1016/0045-7825(90)90098-7.
6. Zhang, R., Pang, H., Dong, W. C., Li, T., Liu, F. et al. (2020). Three-dimensional discrete element method simulation system of the interaction between irregular structure wheel and lunar soil simulant. *Advances in Engineering Software, 148(1),* 102873. DOI 10.1016/j.advengsoft.2020.102873.
7. Wang, S. Q., Zhang, Q., Ji, S. Y. (2021). GPU-Based parallel algorithm for super-quadric discrete element method and its applications for non-spherical granular flows. *Advances in Engineering Software, 151(1),* 102931. DOI 10.1016/j.advengsoft.2020.102931.
8. Fu, T. F., Xu, T., Heap, M. J., Meredith, P. G., Mitchell, T. M. (2020). Mesoscopic time-dependent behavior of rocks based on three-dimensional discrete element grain-based model. *Computers and Geotechnics, 121(1),* 103472. DOI 10.1016/j.compgeo.2020.103472.
9. Xu, T., Fu, T. F., Heap, M. J., Meredith, P. G., Mitchell, T. M. et al. (2020). Mesoscopic damage and fracturing of heterogeneous brittle rocks based on three-dimensional polycrystalline discrete element method. *Rock Mechanics and Rock Engineering, 53(12),* 5389–5409. DOI 10.1007/s00603-020-02223-y.
10. Yu, Y., Paulino, G., Luo, Y. Z. (2010). Finite particle method for progressive failure simulation of truss structures. *Journal of Structural Engineering, 137(10),* 1168–1181. DOI 10.1061/(ASCE)ST.1943-541X.0000321.
11. Ting, E. C., Shih, C., Wang, Y. K. (2004). Fundamentals of a vector form intrinsic finite element: Part I. Basic procedure and a plane frame element. *Journal of Mechanics, 20(2),* 113–122. DOI 10.1017/S1727719100003336.
12. Ting, E. C., Shih, C., Wang, Y. K. (2004). Fundamentals of a vector form intrinsic finite element: Part II. Plane solid elements. *Journal of Mechanics, 20(2),* 123–132. DOI 10.1017/S1727719100003348.
13. Shih, C., Wang, Y. K., Ting, E. C. (2004). Fundamentals of a vector form intrinsic finite element: Part III. Convected material frame and examples. *Journal of Mechanics, 20(2),* 133–143. DOI 10.1017/S172771910000335X.
14. Wu, T. Y. (2013). Dynamic nonlinear analysis of shell structures using a vector form intrinsic finite element. *Engineering Structures, 56(Supplement C),* 2028–2040. DOI 10.1016/j.engstruct.2013.08.009.
15. Zheng, Y. F., Yang, C., Wan, H. P., Luo, Y. Z., Li, Y. et al. (2020). Dynamics analysis of spatial mechanisms with dry spherical joints with clearance using finite particle method. *International Journal of Structural Stability and Dynamics, 20(3),* 2050035. DOI 10.1142/S0219455420500352.
16. Zheng, Y. F., Wan, H. P., Zhang, J. Y., Yang, C., Luo, Y. Z. et al. (2021). Local-coordinate representation for spatial revolute clearance joints based on a vector-form particle-element method. *International Journal of Structural Stability and Dynamics, 21(7),* 2150093. DOI 10.1142/S0219455421500930.
17. Yu, Y., Zhu, X. Y. (2016). Nonlinear dynamic collapse analysis of semi-rigid steel frames based on the finite particle method. *Engineering Structures, 118(1),* 383–393. DOI 10.1016/j.engstruct.2016.03.063.
18. Yang, C., Shen, Y. B., Luo, Y. Z. (2014). An efficient numerical shape analysis for light weight membrane structures. *Journal of Zhejiang University Science A, 15(4),* 255–271. DOI 10.1631/jzus.A1300245.
19. Duan, Y. F., Wang, S. M., Wang, R. Z., Wang, C. Y., Shih, J. Y. et al. (2018). Vector form intrinsic finite-element analysis for train and bridge dynamic interaction. *Journal of Bridge Engineering, 23(1),* 04017126. DOI 10.1061/(ASCE)BE.1943-5592.0001171.
20. Liu, F. H., Yu, Y., Wang, Q. H., Luo, Y. Z. (2020). A coupled smoothed particle hydrodynamic and finite particle method: An efficient approach for fluid-solid interaction problems involving free-surface flow and solid failure. *Engineering Analysis with Boundary Elements, 118(1),* 143–155. DOI 10.1016/j.enganabound.2020.03.006.
21. Hallquist, J., Goudreau, G., Benson, D. (1985). Sliding interfaces with contact-impact in large-scale lagrangian computations. *Computer Methods in Applied Mechanics and Engineering, 51(1–3),* 107–137. DOI 10.1016/0045-7825(85)90030-1.

22. Yu, P. C., Peng, X. Y., Chen, G. Q., Guo, L. X. (2020). Openmp-based parallel two-dimensional discontinuous deformation analysis for large-scale simulation. *International Journal of Geomechanics, 20(7),* 04020083. DOI 10.1061/(ASCE)GM.1943-5622.0001705.
23. Peng, X. Y., Chen, G. Q., Yu, P. C., Zhang, Y. B., Zhang, H. et al. (2020). A full-stage parallel architecture of three-dimensional discontinuous deformation analysis using OpenMP. *Computers and Geotechnics, 118(1),* 103346. DOI 10.1016/j.compgeo.2019.103346.
24. Tang, J. Z., Zheng, Y. F., Yang, C., Wang, W., Luo, Y. Z. (2020). Parallelized implementation of the finite particle method for explicit dynamics in GPU. *Computer Modeling in Engineering & Sciences, 122(1),* 5–31. DOI 10.32604/cmes.2020.08104.
25. Bartezzaghi, A., Cremonesi, M., Parolini, N., Perego, U. (2015). An explicit dynamics GPU structural solver for thin shell finite elements. *Computers & Structures, 154(1),* 29–40. DOI 10.1016/j.compstruc.2015.03.005.
26. Cao, X. G., Cai, Y., Cui, X. Y. (2020). A parallel numerical acoustic simulation on a GPU using an edge-based smoothed finite element method. *Advances in Engineering Software, 148(1),* 102835. DOI 10.1016/j.advengsoft.2020.102835.
27. NVIDIA (2021). *CUDA C++ programming guide*. Santa Clara, CA: NVIDIA.
28. Cai, Y., Wang, G. P., Li, G. Y., Wang, H. (2015). A high performance crashworthiness simulation system based on GPU. *Advances in Engineering Software, 86(1),* 29–38. DOI 10.1016/j.advengsoft.2015.04.003.
29. Chung, J. T., Lee, J. M. (1994). A new family of explicit time integration methods for linear and non-linear structural dynamics. *International Journal for Numerical Methods in Engineering, 37(23),* 3961–3976. DOI 10.1002/nme.1620372303.
30. Hulbert, G. M., Chung, J. T. (1996). Explicit time integration algorithms for structural dynamics with optimal numerical dissipation. *Computer Methods in Applied Mechanics and Engineering, 137(2),* 175–188. DOI 10.1016/S0045-7825(96)01036-5.
31. Chen, S. Y., Wang, G. H., Li, X. M., Zhang, Q. Y., Shi, Z. P. et al. (2020). Formalization of camera pose estimation algorithm based on rodrigues formula. *Formal Aspects of Computing, 32(4),* 417–437. DOI 10.1007/s00165-020-00520-5.
32. Liu, G. R., Quek, S. S. (2013). *The finite element method: A practical course.* Oxford: Butterworth-Heinemann.
33. Simo, J. C., Hughes, T. J. (1998). *Computational inelasticity.* New York: Springer-Verlag.
34. Borja, R. I. (2013). *Plasticity: Modeling & computation.* Berlin, Heidelberg: Springer-Verlag.
35. Hughes, T. J. R. (1980). Generalization of selective integration procedures to anisotropic and non-linear media. *International Journal for Numerical Methods in Engineering, 15(9),* 1413–1418. DOI 10.1002/nme.1620150914.
36. Flanagan, D. P., Belytschko, T. (1981). A uniform strain hexahedron and quadrilateral with orthogonal hourglass control. *International Journal for Numerical Methods in Engineering, 17(5),* 679–706. DOI 10.1002/nme.1620170504.
37. Munjiza, A., Andrews, K. R. F. (1998). NBS contact detection algorithm for bodies of similar size. *International Journal for Numerical Methods in Engineering, 43(1),* 131–149. DOI 10.1002/(SICI)1097-0207 (19980915)43:1<131::AID-NME447>3.0.CO;2-S.
38. Chen, H., Lei, Z., Zang, M. Y. (2014). LC-Grid: A linear global contact search algorithm for finite element analysis. *Computational Mechanics, 54(5),* 1285–1301. DOI 10.1007/s00466-014-1058-5.
39. Zang, M. Y., Gao, W., Lei, Z. (2011). A contact algorithm for 3D discrete and finite element contact problems based on penalty function method. *Computational Mechanics, 48(5),* 541–550. DOI 10.1007/s00466-011-0606-5.
40. Zavarise, G., de Lorenzis, L. (2009). The node-to-segment algorithm for 2D frictionless contact: Classical formulation and special cases. *Computer Methods in Applied Mechanics and Engineering, 198(41–44),* 3428–3451. DOI 10.1016/j.cma.2009.06.022.
41. Pennestrì, E., Rossi, V., Salvini, P., Valentini, P. P. (2016). Review and comparison of dry friction force models. *Nonlinear Dynamics, 83(4),* 1785–1801. DOI 10.1007/s11071-015-2485-3.

42. Lacasta, A., Morales-Hernández, M., Murillo, J., García-Navarro, P. (2014). An optimized GPU implementation of a 2D free surface simulation model on unstructured meshes. *Advances in Engineering Software, 78(1),* 1–15. DOI 10.1016/j.advengsoft.2014.08.007.
43. Hammer, M. E. (2013). Frictional mortar contact for finite deformation problems with synthetic contact kinematics. *Computational Mechanics, 51(6),* 975–998. DOI 10.1007/s00466-012-0780-0.
44. Yang, B., Laursen, T. A., Meng, X. (2005). Two dimensional mortar contact methods for large deformation frictional sliding. *International Journal for Numerical Methods in Engineering, 62(9),* 1183–1225. DOI 10.1002/nme.1222.
45. Neto, D. M., Oliveira, M. C., Menezes, L. F., Alves, J. L. (2016). A contact smoothing method for arbitrary surface meshes using nagata patches. *Computer Methods in Applied Mechanics and Engineering, 299(1),* 283–315. DOI 10.1016/j.cma.2015.11.011.
46. Areias, P., Rabczuk, T., de Melo, F. Q., de Sá, J. C. (2015). Coulomb frictional contact by explicit projection in the cone for finite displacement quasi-static problems. *Computational Mechanics, 55(1),* 57–72. DOI 10.1007/s00466-014-1082-5.