Received: 21 June 2020 | Revised: 16 September 2020 | Accepted: 16 September 2020
DOI: 10.1002/nme.6549

RESEARCH ARTICLE

WILEY

A thread-block-wise computational framework for large-scale hierarchical continuum-discrete modeling of granular media

Shiwei Zhao | **Jidong Zhao** | **Weijian Liang**

Department of Civil and Environmental Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong

Correspondence
Shiwei Zhao, Department of Civil and Environmental Engineering, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong.
Email: ceswzhao@ust.hk

Funding information
Hong Kong Scholars Program, Grant/Award Number: XJ2018049; National Natural Science Foundation of China, Grant/Award Numbers: 11972030, 51679207, 51909095; Research Grants Council of Hong Kong, Grant/Award Numbers: CRF Project No. C6012-15G, GRF Project No. 16207319, TBRS Project No. T22-603/15N

Abstract
This article presents a novel, scalable parallel computing framework for large-scale and multiscale simulations of granular media. Key to the new framework is an innovative thread-block-wise representative volume element (RVE) parallelism, inspired by the resemblance between a typical multiscale computational hierarchy and the hierarchical thread structure of graphics processing units (GPUs). To solve a hierarchical multiscale problem, all computation in an RVE is assigned a single block of threads so that the RVE runs entirely on a GPU to avoid frequent data exchange with the host CPU. The thread blocks can meanwhile run in an asynchronization mode, which implicitly guarantees the independence of inter-RVE computation as featured by the hierarchical multiscale structure. The parallel computing algorithms are formulated and implemented in an in-house code, *GoDEM*, involving the GPU-specific techniques such as coalesced access, shared memory utilization, and unified memory implementation. Benchmark and performance tests are conducted against an open-source CPU-based DEM code under three typical loading conditions. The performance of *GoDEM* is examined with varying thread-block size and register pressure of the GPU, and RVE number. It reveals that increasing GPU occupancy by decreasing register pressure results in a significant degradation rather than improvement in performance. We further demonstrate that the proposed GPU parallelism framework may achieve a saturated speedup of approximately 350 compared with the single-CPU-core code. As a demonstration on its application for multiscale modeling of granular media, the material point method is coupled with the new framework powered DEM to simulate a typical engineering-scale problem involving tens of millions of total particles having to be handled. It demonstrates that a speedup of approximately 91 can be achieved by using the proposed framework, compared with the performance of a similar CPU program running on a cluster node of 44 parallel threads. The study offers a viable future solution to large-scale and multiscale modeling of granular media.

KEYWORDS
continuum-discrete coupling, DEM, granular media, MPM, multiscale modeling, parallel computing

[Correction added on 05 November 2020, after first online publication: In Funding information, the GRF Project No. “16205418” has been corrected to “16207319”.]

1 INTRODUCTION

Granular materials are widely encountered not only in nature but also in the practice of a wide range of industry and engineering operations, such as grain handling and storing in the agricultural industry, the construction of geostructures in civil engineering, and the processing of powders in chemical engineering and pharmaceutical industry. The mechanical behavior of granular media underpins the design, operation, and risk management during the course of practice for all these processes. Complex and intriguing phenomena arising from granular media under external loadings, such as anisotropy and liquefaction,[1] strain localization and failure,[2] and noncoaxiality[3] and the rich transitional behavior between fluid and solid, have long been captivating for researchers across many disciplines of science and engineering.[4-8] Yet puzzles remain and challenges persist for the past century of granular media research, as Science Magazine (2005)[9] rated a general theory for granular media among 125 big unsolved questions facing scientific inquiry for the coming century.

These pertinent issues to granular media pose tremendous challenges for the community of computational mechanics, which has long been used to tackle granular media based on either continuum[10] or discrete[11] theories and methodologies. Continuum-based approaches typically employ phenomenological constitutive relations to describe the mechanical behaviors of a granular material. By contrast, discrete-based methods, exemplified by the discrete element method (DEM),[12] enable more physical considerations at lower scales (e.g., grain-scale) such that the inherent discontinuum of granular media can be captured. In DEM, the motion of each individual particle is tracked by Newton's laws of motion in conjunction with contact force models. Despite the rather simple contact models used at grain scales, DEM has been demonstrated to be capable of capturing a rich spectrum of characteristics of a granular material and offering physically sound interpretations on macroscopic observations.[13-16] More recent progresses in both theoretical and methodological aspects further enable us to consider particle shape[17] and particle roughness[18] with improved confidence, and to tackle multiphase, multiphysics granular problems based on coupling with other computational methods including the computational fluid dynamics (CFD)[19] and the lattice Boltzmann method (LBM).[20]

Notwithstanding the various merits, DEM has its unresolved issues. The high computational cost has been an outstanding one when it has to deal with large-scale problems. A direct and practical approach to speeding up a large-scale DEM simulation is parallelizing the program, that is, by parallel computing. Conventional parallel computing has commonly been implemented on the CPU-based computing system with two prevailing parallelizing standards: (1) OpenMP for single multiprocessor machine with shared memory (e.g., a node of a cluster)[21] and (2) message passing interface (MPI) for clusters with distributed memory.[22] In addition to the CPU-based parallel computing, the general-purpose graphics processing unit (GPU) computing has emerged to be a frontrunner in parallel computing for its outstanding computational performance and high memory bandwidth. That is benefited from the architecture of modern GPUs that equip significantly more transistors for arithmetic logic units (ALUs) but less for flow control than CPU architecture.[23] In a nutshell, the modern GPU is specifically designed for parallel computing, more powerful but less expensive than CPU-based computing for large-scale simulations. It has hence drawn increasing interest in accelerating DEM simulations.[24-26] For example, a recent work reported a solution approach to dealing with billion-degree-of-freedom dynamics problems on a workstation with one GPU,[27] and the approach has been implemented in an open-source code Chrono.[28] However, these accelerating schemes, either on CPUs or GPUs, focus largely on problems with an entire domain (intuitively a huge packing) composed of discrete particles. They frequently encounter tremendous challenges to simulate real engineering-scale problems, for example, a typical foundation failure in geotechnical engineering, if natural grain sizes are to be respected.

More recently, a hierarchical framework for multiscale modeling of granular materials has been attracting particular attention,[29-35] which pushes a successful marriage between DEM and the continuum-based method such as the finite element method (FEM) and the material point method (MPM). The hierarchical approach differs from the other coupling schemes such as the concurrent approach[36,37] and the two-scale FEM approach.[38,39] In the hierarchical framework, continuum boundary value problems (BVPs) can be solved by the continuum-based method at the macroscopic scale in conjunction with the homogeneous responses of representative volume elements (RVEs) that are captured by DEM instead of a conventional phenomenological constitutive relation. Specifically, DEM-simulated RVEs serve as Gaussian quadrature points or material points in the hierarchical coupling of FEM×DEM[29] or MPM×DEM,[34] respectively, bridging the micro- (discrete-grain) and macro- (continuum) scales of granular media. Notably, the hierarchical framework brings two noteworthy aspects of improvements: (1) the characteristics of a granular material can be readily modeled by the continuum-based methods with DEM-simulated RVEs taking the place of phenomenological constitutive models and (2) the domain occupied only by RVEs needs to be simulated by DEM instead of the entire domain of a granular material, thereby considerably reducing the computational cost of DEM and improving the efficiency. Moreover, the hierarchical framework has such a parallel nature that all RVEs can be independently simulated in parallel. Indeed, Guo and Zhao[31] proposed a parallel hierarchical coupling of FEM×DEM at the RVE level with MPI on the CPU-computing system.

The DEM simulation of RVEs holds the majority of computational complexity in the aforementioned hierarchical framework of multiscale modeling. This article proposes a novel, efficient, robust, and scalable parallelism framework of discrete element modeling of RVEs on a GPU, coined as thread-block-wise RVE modeling, motivated by the analogous hierarchy of the multiscale hierarchical framework and the hierarchical organizational structure of GPU threads. In the proposed framework, each RVE corresponds to a block of GPU threads where each thread undertakes the computation involving single or several particles and/or contacts, as will be explained in Section 3. The proposed thread-block-wise RVE parallelism framework differs completely from the various GPU-accelerated DEM studies reported in the literature, for example, References 24-27, and among others. Three significant novelties of the proposed framework are highlighted as follows: (1) Conventional GPU-accelerated DEMs are often developed to simulate problems with an entire domain composed of DEM particles, while the present framework is specifically proposed for hierarchical multiscale modeling by DEM based on coupled continuum-discrete methods. The present framework empowers us to simulate much larger scale engineering boundary value problems than the conventional parallel DEMs can do. (2) In conventional GPU-accelerated DEM, only critical processes of DEM computation, such as contact detection, contact force summation, and integration of particle motion, are implemented and run on GPUs, while the main routine running on CPUs sequentially invoke these critical processes during each DEM iteration; therefore, the GPU needs to communicate with the host CPU frequently during each DEM iteration, which necessarily forfeits the full capability of what GPU can offer. By contrast, the proposed framework runs RVEs entirely on GPUs, which helps avoid the frequent communication between the GPU and the host CPU, and benefits for the performance of the Unified Memory technique employed in this work. (3) In analogy to the RVEs in the proposed framework, there are also subdomains in the conventional GPU-accelerated DEMs, which can be solved by one or more blocks of GPU threads. However, one subdomain shares boundary particles (often called ghost particles) with its adjacent subdomains so that the computation of subdomains is not entirely independent of each other. By contrast, in the proposed framework, the computation of all RVEs is completely independent of each other, which facilitates the parallel computing in nature. Moreover, the independence of computation for difference RVEs is implicitly guaranteed according to the asynchronization of thread blocks on a GPU.

The rest of this article is organized as follows. Section 2 introduces a brief theoretical background of modeling RVEs using DEM with periodic boundary conditions, which offers convenience to depict the parallel algorithms of thread-block-wise RVEs on a GPU in Section 3. For the completeness of the presentation, we also present the pseudo-codes of the proposed GPU algorithms along with the description of the parallelism framework. Benchmark and performance tests on the proposed parallelism framework and algorithms are carried out and are further compared against CPU codes in Section 4. As a demonstration of applying the proposed framework to speeding up hierarchical multiscale modeling of granular media, the material point method (MPM) is coupled with DEM to solve an engineering-scale problem in Section 5. Section 6 presents the concluding remarks of this study. Tensorial indicial notations and Einstein summation convention are followed in the study unless otherwise stated.

2 | DISCRETE ELEMENT MODELING OF RVES

2.1 | Discrete element method

2.1.1 | Governing equations

DEM considers the motion of a particle is governed by the Newton-Euler equation as

$$F_i = m\dot{v}_i, \tag{1a}$$

$$T_i = I_{ij}\dot{\omega}_j - \epsilon_{ijk}I_{kl}\omega_j\omega_l, \tag{1b}$$

where ϵ_{ijk} is the permutation tensor; F_i and T_i are the resultant force and torque acting on the particle center of mass, respectively; v_i and ω_i are the translational and angular velocities, respectively, and the over dot denotes derivation with respect to time; m is the particle mass; I_{ij} is the principal moment tensor of inertia around the mass center ($I_{ij} = 0$ for $i \neq j$).

For spherical particles, Equation (1b) is reduced as $T_i = I_{ij}\dot{\omega}_j$ due to $I_{11} = I_{22} = I_{33}$; the resultant force F_i and the resultant torque T_i are given as

$$F_i = F_i^b + \sum_{c \in N_c} f_i^c, \tag{2a}$$

$$T_i = \sum_{c \in N_c} \epsilon_{ijk} r_i^c f_j^c, \tag{2b}$$

where F_i^b is the body force; f_i^c is the contact force at contact c; N_c is the number of particles contacting with the given particle; r_i^c is the position vector of contact point at contact c with respect to the particle center of mass.

2.1.2 | Integration of motion

The motion of a particle (translation and rotation) is solved explicitly by using a central difference scheme with time step of Δt. The prevailing leapfrog algorithm (Verlet scheme)[40] is employed to integrate particle translation such that the velocity and position are given by

$$v_i^{t+\frac{\Delta t}{2}} = v_i^{t-\frac{\Delta t}{2}} + \dot{v}_i^t \Delta t, \tag{3a}$$

$$x_i^{t+\Delta t} = x_i^t + v_i^{t+\frac{\Delta t}{2}} \Delta t, \tag{3b}$$

where Δt is the time step, and the superscript denotes the variable at the corresponding time; the acceleration $\dot{v}_i^t$ is calculated in terms of the Newton equation in Equation (1a) with artificial damping applied as follows

$$\Delta \dot{v}_i^t = -\alpha_d \mathrm{Sign}(\dot{v}_i^t, v_i^t) \dot{v}_i^t, \tag{4a}$$

$$v_i^t = v_i^{t-\frac{\Delta t}{2}} + \frac{1}{2} \dot{v}_i^t \Delta t, \tag{4b}$$

where α_d denotes a damping coefficient; $\mathrm{Sign}(x, y)$ is the sign-function which returns a value of 1 if x and y have the same sign, otherwise, -1. Hence, the damped acceleration reads

$$\dot{v}_i^t = \frac{F_i}{m} + \Delta \dot{v}_i^t. \tag{5}$$

With respect to particle rotation, a quaternion $q(q_w, q_x, q_y, q_z) = \cos\frac{\hat{\theta}}{2} + (\hat{e}_x \boldsymbol{i} + \hat{e}_y \boldsymbol{j} + \hat{e}_z \boldsymbol{k}) \sin\frac{\hat{\theta}}{2}$ is usually employed to track particle orientation and rotation, where $\hat{\theta}$ is the angle of the particle rotating around a unit axis $\hat{\boldsymbol{e}}(\hat{e}_x, \hat{e}_y, \hat{e}_z)$.[17] Particle rotation can be solved with a similar scheme to particle translation mentioned above for spherical particles. The rotational velocity and orientation are given by

$$\omega_i^{t+\frac{\Delta t}{2}} = \omega_i^{t-\frac{\Delta t}{2}} + \dot{\omega}_i^t \Delta t, \tag{6a}$$

$$q^{t+\Delta t} = \Delta q^{t+\Delta t} q^t, \tag{6b}$$

where $\Delta q^{t+\Delta t}$ is the rotational increment (i.e., $\omega_i^{t+\frac{\Delta t}{2}} \Delta t$) in quaternion. However, for nonspherical particles (strictly with nonequal principal moments of inertia), it is worth noting that the integration of rotation is complicated (see Reference 41), which is beyond the scope of this study. Interested readers are referred to the literature for discrete element modeling of nonspherical particles.[17,42]

2.1.3 | Contact force model

For two moving spherical particles (denoted by Particle 1 and Particle 2), as shown in Figure 1(A), contact occurs if and only if the distance between the two centers of particles is less than the sum of their radii, that is,

$$\|\boldsymbol{b}\| < R^{(1)} + R^{(2)}, \tag{7}$$

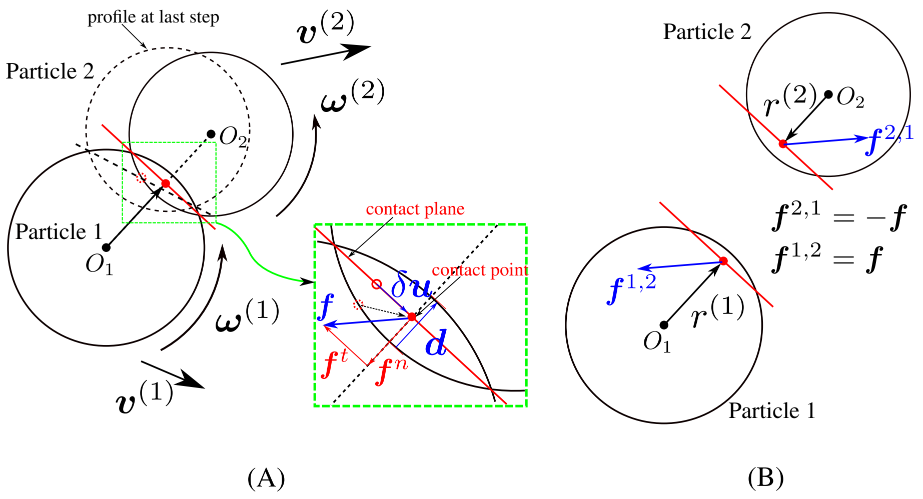

FIGURE 1 Two contacting particles: (A) the configuration of motion; (B) the resultant contact forces [Colour figure can be viewed at wileyonlinelibrary.com]

where R is particle radius with the superscript (1) or (2) denoting Particle 1 or Particle 2 hereafter; $\boldsymbol{b}$ is the branch vector joining the centers of the two particles, given as

$$b_i = x_i^{(2)} - x_i^{(1)}. \tag{8}$$

The intersection plane of the surfaces of the particles is taken as the contact plane. Its direction, that is, contact normal, is defined as the unit vector of $\boldsymbol{b}$

$$n_i = b_i/\|\boldsymbol{b}\|. \tag{9}$$

The penetration of the two particles is given as

$$d_i = (R^{(1)} + R^{(2)} - \|\boldsymbol{b}\|)n_i. \tag{10}$$

With the assumption that the contact force acts at the intersection point (i.e., contact point) of the contact plane and the branch line (from O_1 to O_2), the relative velocity of Particle 1 to Particle 2 at the contact point is given as

$$v_i^{1,2} = v_i^{(2)} - v_i^{(1)} + \epsilon_{ijk}\omega_j^{(2)}r_k^{(2)} - \epsilon_{ijk}\omega_j^{(1)}r_k^{(1)}. \tag{11}$$

The incremental tangential displacement at the current time step, that is, the incremental displacement of the contact point along the contact plane, is given as

$$\delta u_i = (v_i^{1,2} - n_k v_k^{1,2} n_i)\Delta t. \tag{12}$$

For the convenience of implementation, contact force f_i is split into two orthogonal components: normal contact force f_i^n and tangential contact force f_i^t (see Figure 1). The force–displacement law in conjunction with the Coulomb friction model is employed as the contact force model at the microscopic scale,[12] given as

$$f_i^n = -k_n d_i, \tag{13a}$$

$$\Delta f_i^t = -k_t \delta u_i, \tag{13b}$$

and

$$f_i^t = \begin{cases} (\|\boldsymbol{f}'^t\| + \|\Delta\boldsymbol{f}^t\|)\frac{\delta u_i}{\|\delta u\|}, & \text{if } \|\boldsymbol{f}'^t\| + \|\Delta\boldsymbol{f}^t\| \leq \mu\|\boldsymbol{f}^n\|, \\ \mu\|\boldsymbol{f}^n\|\frac{\delta u_i}{\|\delta u\|}, & \text{otherwise,} \end{cases} \tag{14}$$

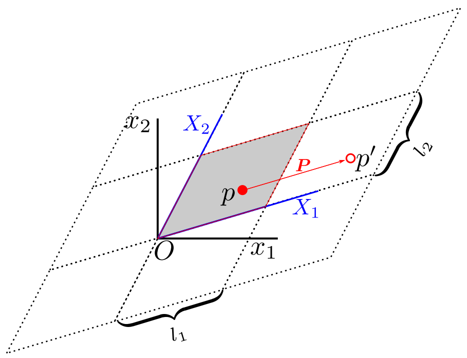

FIGURE 2 Periodic cell ("solid") and its neighbor images ("open-dashed") aligned in a lattice form [Colour figure can be viewed at wileyonlinelibrary.com]

where Δf_i^t is the incremental tangential contact force; $\boldsymbol{f}'^t$ is the tangential contact force at the previous time step; μ is the coefficient of friction. As shown in Figure 1(B), the contact forces acting on Particle 1 and Particle 2, respectively, denoted by $\boldsymbol{f}^{1,2}$ and $\boldsymbol{f}^{2,1}$ follow a relation with the contact force $\boldsymbol{f}$

$$f_i^{1,2} = -f_i^{2,1} = f_i. \tag{15}$$

2.2 | Periodic boundary conditions

2.2.1 | Periodic cell

Periodic boundary conditions are, in general, introduced to reduce the boundary effect from rigid boundaries such as rigid confining walls in DEM simulations.[43-45] For simplicity but without losing generality, a parallelepiped-shaped cell is adopted as the simulation domain of an RVE. Figure 2 shows a two-dimensional (2D) illustration of an RVE cell as a parallelogram and its neighbor images periodically repeated in a lattice form. For the convenience of presentation and implementation in the following algorithms, two coordinates systems are introduced: one is the fixed global Cartesian coordinate system; the other is the local oblique Cartesian coordinate system with basis vectors along the boundaries of the RVE cell. These two coordinate systems are also known as Eulerian (spatial) and Lagrangian (material) coordinates in continuum mechanics, respectively, but intuitively denoted as global and local coordinate systems for short hereafter.

The global and local coordinates follow a relation of transformation as

$$x_i = H_{ij} X_j, \tag{16a}$$

$$X_j = H_{jk}^{-1} x_k, \tag{16b}$$

where H_{ij} is the deformation (gradient) tensor with columns as the basis vectors of the cell, while H_{ij}^{-1} is for the inverse transformation. Points at the RVE cell also repeat periodically in the cell images in a similar fashion as the cell. Specifically, given a point p in the RVE cell, its image p' in the other cells can be periodically shifted in the local coordinate system by

$$X_i(p') = X_i(p) + P_i \tag{17}$$

with

$$P_i = p_{ij} l_j, \tag{18a}$$

$$p_{ij} = \begin{cases} \left\lfloor \dfrac{X_i(p')}{l_j} \right\rfloor, & \text{if } i = j, \\ 0, & \text{otherwise,} \end{cases} \tag{18b}$$

where P_i is the periodic (shifted) vector; p_{ij} is coined as a period tensor with zero off-diagonal, and its main diagonal is period number for the corresponding axis; l_j is the base length of the cell along X_j, as shown in Figure 2; $\lfloor * \rfloor$ denotes rounding down to the nearest integer.

2.2.2 | Homogeneous deformation

Applying derivative with respect to time at the both sides of Equation (16a), we have

$$\dot{x}_i = \underbrace{\dot{H}_{ij}X_j}_{v_{hi}} + \underbrace{H_{ij}\dot{X}_j}_{v_{fi}}, \tag{19}$$

where v_{hi} is the affine mean-field velocity, which is attributed to the macroscopic homogeneous deformation of the RVE cell; v_{fi} is the fluctuating velocity, that is, particle velocity driven by the resultant force on the particle, which is, however, nonaffine. With Equation (17), the mean-field velocity $v_{hi}(p')$ and the fluctuating velocity $v_{fi}(p')$ at the image p' of a point p can be written as

$$v_{hi}(p') = v_{hi}(p) + \dot{H}_{ij}P_j, \tag{20a}$$

$$v_{fi}(p') = v_{fi}(p). \tag{20b}$$

Therefore, in the presence of periodic boundary conditions, the mean-field velocity v_{hi} is nonperiodic, while the fluctuating velocity v_{fi} is periodic. Accordingly, the period needs to be considered for the computation of relative velocity in Equation (11) due to v_{hi}.

Recalling the integration of particle motion in Section 2.1.2, particle translation is solved explicitly with a central-difference scheme in the global coordinate system. Due to the homogeneous deformation of the RVE cell, the additional velocity is applied to each individual particle, which is, thus, deduced in the global coordinate system with Equations (16a) and (19), given by

$$v_{hi} = L_{ij}x_j, \tag{21a}$$

$$L_{ij} = \dot{H}_{ik}H^{-1}_{kj}, \tag{21b}$$

where L_{ij} is the velocity gradient tensor of the cell deformation. Accordingly, the induced acceleration due to the cell deformation is given by

$$\dot{v}_{hi} = \dot{L}_{ik}x_k + L_{ik}\dot{x}_k. \tag{22}$$

Therefore, the additional incremental velocity $\Delta\boldsymbol{v}_h^{t+\frac{\Delta t}{2}}$ at time $t + \frac{\Delta t}{2}$ reads

$$\Delta\boldsymbol{v}_h^{t+\frac{\Delta t}{2}} = \Delta\boldsymbol{L}^t\boldsymbol{x}^t + \underbrace{\boldsymbol{L}^t\boldsymbol{v}^{t-\frac{\Delta t}{2}}}_{\Delta\dot{\boldsymbol{v}}^t}\Delta t, \tag{23}$$

which is further summed to the right side of Equation (3a) to integrate particle translation when the macroscopic deformation of the RVE cell is applicable. It is worth noting that the macroscopic homogeneous deformation of the RVE cell does not yield additional angular velocities for individual particles.

2.3 | Homogenized stress and strain

The homogenized stress tensor σ_{ij} within an RVE assembly is given by Love formula as[46]

$$\sigma_{ij} = \frac{1}{V}\sum_{c\in V} f_i^c b_j^c, \tag{24}$$

where V is the volume of the assembly; f_i^c and b_j^c are the contact force and the branch vector, respectively. The mean stress p and deviatoric stress q are given by

$$p = \frac{1}{n}\sigma_{ii}, \tag{25a}$$

$$q = \sqrt{\frac{3^{n-2}}{2}\sigma'_{ij}\sigma'_{ij}} \tag{25b}$$

in which σ'_{ij} is the deviatoric stress tensor, $\sigma'_{ij} = \sigma_{ij} - p\delta_{ij}$, where δ_{ij} is the Kronecker delta (substitution tensor); $n = 2$ or 3 for 2D or 3D, respectively.

In the hierarchical multiscale modeling framework by coupling either FEM×DEM or MPM×DEM, all RVEs are subjected to small deformation increments during each DEM step. That is to say, loading RVEs is strain-controlled. As for the strain measure, the nonrotational deformation of the periodic cell (RVE) is homogenized by using the periodic boundaries in terms of the infinitesimal strain tensor ϵ_{ij}, that is,

$$\epsilon_{ij} = \frac{1}{2}(H'_{ij} + H'_{ji}) - \delta_{ij}, \tag{26}$$

where H'_{ij} is the deformation gradient tensor with respect to the reference configuration. The volumetric strain ϵ_v and the deviatoric strain ϵ_q are given by

$$\epsilon_v = \epsilon_{ii}, \tag{27a}$$

$$\epsilon_q = \sqrt{\frac{2}{3^{n-2}}\epsilon'_{ij}\epsilon'_{ij}} \tag{27b}$$

in which ϵ'_{ij} is the deviatoric strain tensor, $\epsilon'_{ij} = \epsilon_{ij} - \frac{1}{n}\epsilon_v\delta_{ij}$; $n = 2$ or 3 for 2D or 3D, respectively. Note that for the strain-controlled loading at each DEM step, the velocity gradient is performed.

3 | PARALLEL ALGORITHMS OF THREAD-BLOCK-WISE RVES

3.1 | RVEs parallelized at the thread-block level

Nvidia's CUDA (Compute Unified Device Architecture) platform provides a scalable programming model for GPU computation, where tens of thousands of concurrent threads offered by a modern GPU are organized in a hierarchy of thread groups as illustrated in Figure 3. The top-level is called Grid, which is composed of many equal-sized (i.e., the same number of threads) Blocks of threads. Both Grid and Block can be up to three-dimensional (3D), making the hierarchy like a multidimensional array so that each thread in the Grid can be located like accessing an element of an array by index.

In analogy to the hierarchy of GPU threads, the intrinsic multiscale characteristic yields a similar hierarchy of grains for granular media, as shown in Figure 3. In detail, discrete grains are grouped into RVEs, which correspond to material points at a higher level of scale, and the material points or RVEs are further grouped together to a continuum at the macroscopic scale. Motivated by the similarity between these two hierarchies, we propose three mappings at different hierarchical levels as follows: a single GPU thread corresponds to a single or multiple grains; a block of threads is for a single RVE, and the entire grid of threads is for the macro continuum. That is to say, the computation task of each RVE is handled by a block of threads, that is, thread-block-wise discrete element modeling of RVE. In addition, threads from different blocks run independently and asynchronously, but the threads from the same block will be synchronized automatically after the blockwise task is over. As a result, tens of thousands of blockwise RVEs can be simulated concurrently and asynchronously. However, it is worth pointing out that the total number of threads concurrently running is limited due to the hardware resource. For example, a GeForce RTX 2080 Ti GPU card has 68 so-called streaming multiprocessors (SMs) with 1024 physically concurrent lanes for each, so that the total number of physically concurrent threads is $68 \times 1024 = 69,632$. Nevertheless, a grid can have a maximum dimension of $(2^{31} - 1, 2^{16} - 1, 2^{16} - 1)$ and up to 1024 threads per block, that is, a massive number of threads in total, while the overloaded threads (i.e., the rest threads excluding the 69,632 concurrent threads) have to be queued for running.

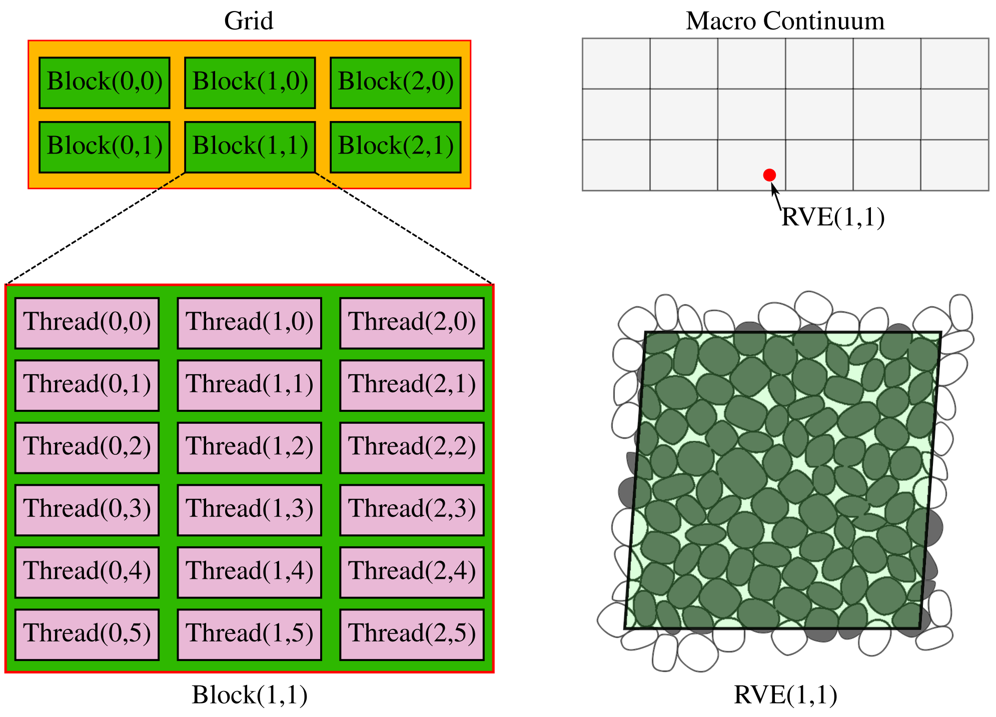
Grid
Block(0,0)
Block(1,0)
Block(2,0)
Block(0,1)
Block(1,1)
Block(2,1)
Thread(0,0) Thread(1,0) Thread(2,0)
Thread(0,1) Thread(1,1) Thread(2,1)
Thread(0,2) Thread(1,2) Thread(2,2)
Thread(0,3) Thread(1,3) Thread(2,3)
Thread(0,4) Thread(1,4) Thread(2,4)
Thread(0,5) Thread(1,5) Thread(2,5)
Block(1,1)
Macro Continuum
RVE(1,1)
RVE(1,1)

FIGURE 3 A hierarchy of thread groups (left column) versus a hierarchy of material points or RVEs (right column) [Colour figure can be viewed at wileyonlinelibrary.com]

Algorithm 1. Pseudocode of the workflow kernel

```
   Input: The set of RVEs, $RVE_n$; the total number of RVEs in the simulation, $N_{\mathrm{RVE}}$;
1  for each block i in the Grid of Blocks of threads do
2      if $i > N_{RVE}$ then
3          //run the RVE on Block i;
4          for each step of $RVE_i$ do
5              update the geometric info of $RVE_i$;
6              if updateNL then
7                  updating reference positions of particles;
8                  updating neighbor list;
9              Computing contact forces for all contacts;
10             Integrating motion of all particles;
```

Algorithm 1 summarizes the pseudocode of running thread-block-wise RVEs in parallel on a GPU. The total number of running steps of each RVE can be different, which offers the flexibility of performing different loading strains on different RVEs in the possible hierarchical multiscale modeling, for example, FEM×DEM[29] or MPM×DEM.[34] For a single DEM step (or iteration), three main procedures are sequentially executed as in a general implementation of DEM: (a) updating the neighbor list, (b) computing contact forces for all contacts, and (c) integrating motion of all particles. However, in this article, these three procedures are definitely parallelized in thread blocks for running on a GPU, and their corresponding algorithms are introduced in the following sections. Note that updating the neighbor list is more time-consuming than the other two procedures, but it can be executed less frequently and only be triggered by the switch *updateNL* that will be flagged in the integration procedure of particle motion, referring to Section 3.4 for details.

3.2 | Neighbor list with periodic boundary conditions

As a critical ingredient of DEM, contact detection among particles takes most of the running time during the course of a single DEM step. A naive approach to searching all contacts in an RVE is the so-called brute-force search, which, however, has a time complexity of $O(N_p^2)$ (N_p is particle number). For a better performance, the neighbors of each

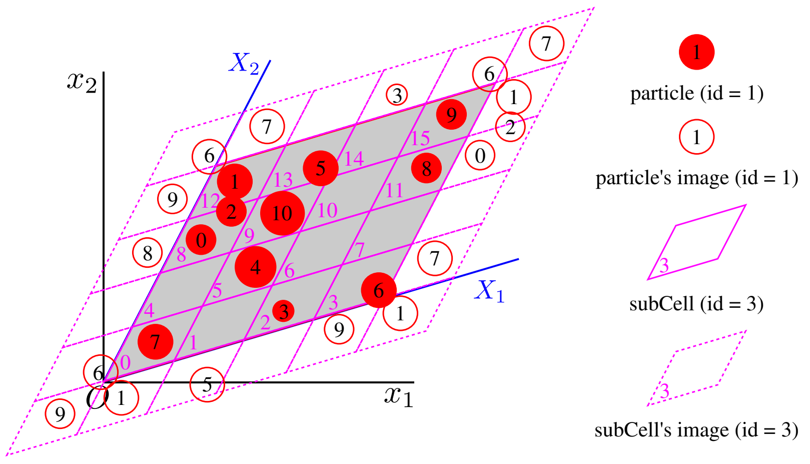

FIGURE 4 An RVE cell ("shadowed") partitioned by equal-sized subCells with periodic boundary conditions [Colour figure can be viewed at wileyonlinelibrary.com]

particle are cached so that the possible contacts of a given particle are searched within its neighbors. Moreover, the neighbors of a particle remain unchanged within certain DEM steps. Hence, a neighbor list is established for storing neighbors of particles in the entire RVE. Motivated by the algorithm proposed by Nishiura and Sakaguchi,[47] we proposed a new algorithm to create the neighbor list with respect to periodic boundary conditions for thread-block-wise RVEs on a GPU.

Figure 4 exemplifies a snapshot of an RVE configuration in 2D (with a few particles only for the sake of presentation), where the entire domain of the RVE is partitioned by a series of equal-sized subCells with the same shape as the RVE cell. Both particles and subCells are labeled by two consecutive integer sequences starting from zero, respectively. The order of labeling particles is arbitrary, and the sort-and-relabel introduced by Nishiura and Sakaguchi[47] is not necessary for our proposed algorithms. Instead, the subCells maintain a prescribed order implicitly to facilitate locating themselves within the RVE cell.

For the convenience of implementation, an integer coordinate system is introduced for subCells, as shown in Figure 5, which is attached to the local coordinate system. The *id* of a given subCell is defined in terms of the coordinates (X'_1, X'_2) for 2D by

$$id = D_x X'_2 + X'_1, \tag{28a}$$

$$D_x = \left\lceil \frac{l_1}{l_x^s} \right\rceil \tag{28b}$$

so that the coordinates (X'_1, X'_2) can be decoded readily as

$$X'_2 = \left\lfloor \frac{id}{D_x} \right\rfloor, \tag{29a}$$

$$X'_1 = id - D_x X'_2, \tag{29b}$$

where D_x is the number of subCells spanning over the RVE cell along X_1; l_x^s and l_1 are the lengths of the subCell and the RVE cell along X_1, respectively; $\lfloor * \rfloor$ and $\lceil * \rceil$ denote rounding down and up to the nearest integers, respectively.

Note that the minimum size of a subCell is controlled by how many particles to be covered by the subCell, which affects the size of memory allocation of lists or arrays in the algorithms introduced in this work. Moreover, both sizes and labels of the subCells are updated according to the deformation of the RVE cell. The detail of the algorithm of neighbor list is depicted as follows.

3.2.1 | Particle-subCell list: Mapping subCell id to particle id

Given a set of particle positions X_{N_p} (N_p is particle number) in the global coordinate system, loop all particles and transform each X_i into the local coordinate system $\tilde{X}_i$ by Equation (16b). Then, the local position $\tilde{X}_i$ is reduced by Equation (17) so that the wrapped position $\tilde{X}_i$ stays within the RVE cell. The corresponding period is also recorded in a list P_{N_p}. Note

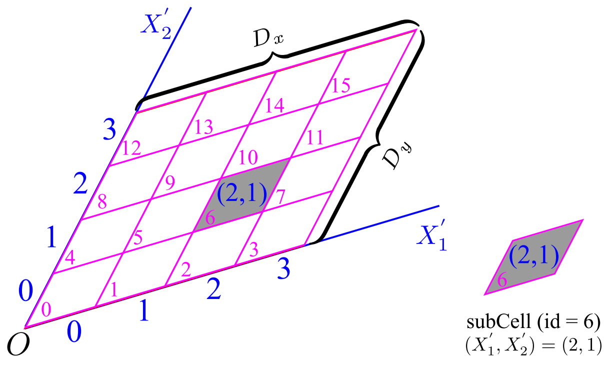

FIGURE 5 Integer coordinate system attached at the local coordinate system for subCell locating [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Exemplified particle-subCell list PC_{N_p} for mapping subCell id to particle id

Particle id (i)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
SubCell id (PC_i)	8	12	8	2	5	13	3	0	11	15	9

that the subscript in the variables refers to an index for accessing a list or array rather than an indicial notation hereafter unless otherwise stated. For example, N_p in X_{N_p} denotes the dimension of a set or list X, while i in X_i denotes the ith item in X, bewaring of that i starts from 0 for a C-like language in programming.

Next, the id PC_i of the subCell that particle i belongs in can be identified by Equation (28). The pseudocode of creating a particle-subCell list is shown in Algorithm 2. The particle-subCell list for the exemplified RVE in Figure 4 is in Table 1.

Algorithm 2. Creating particle-subCell list PC_{N_p}

```
    Input: The set of particle positions at the global coordinate system, $X_n$; The particle number in the RVE, $N_p$;
    Output: The set of particle position at the RVE local coordinate system, $\tilde{X}_n$; The set of periods of particles, $P_n$;
            The particle-cell list, $PC_n$;
1   for each thread i in the Block of threads do
2       while $i > N_p$ do
3           $\tilde{X}_i$ = local position from $X_i$ by Eq. (16b);
4           $\tilde{X}_i, P_i$ = reduced coordinates and period by Eqs. (17) and (18);
5           $PC_i$ = id of the subCell by Eq. (28);
6           $i = i + BlockDim$;
```

3.2.2 | SubCell-particle list: Mapping particle id to subCell id

With a particle-subCell list PC_{N_p}, it is necessary to create a subCell-particle list CP_{N_s} (N_s is subCell number) so that all particles for a given subCell i are accessible immediately by a sublist CP_i. To this end, a direct algorithm is traversing all particles, that is, all items in PC_{N_p}, and pushing back the particle id into the sublist CP_i for a given subCell id i. The time complexity of pushing back particle ids into the list is $O(N_p)$, which cannot be less for a serial running. Parallelly traversing all particles seems to promote the performance, but meanwhile, special attention should be paid to the possible race conditions that may yield a worse performance even wrong results. In detail, a sublist CP_i will be read and/or written by multithreads at the same time, that is, a race condition. Thus, these multithreads have to be serialized manually (e.g., using atomic operation or mutex lock) to make sure correct results, which, however, definitely slows down the parallelism. As a workaround, the parallelism is conducted on subCells instead of particles to avoid race conditions. Listed in Algorithm 3 is the corresponding pseudocode, in which all particles are traversed for each subCell id in a brute-force fashion, but the test condition at Line 4 is executed so fast that the total time complexity remains $O(N_p)$. Table 2 lists the subCell-particle list CP_{N_s} for the exemplified RVE configuration in Figure 4.

SubCell id (i)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Particle id (CP_i)	7	–	3	6	–	4	–	–	[0, 2]	10	–	8	1	5	–	9

TABLE 2 Exemplified subCell-particle list CP_{N_s} for mapping particle id to subCell id

Algorithm 3. Creating subCell-particle list CP_{N_s}

```
Input: The particle-subCell list, $PC_{N_p}$; the subCell number, $N_s$; the particle number, $N_p$;
Output: The list of particle ids indexed by cell ids, $CP_{N_s}$;
1 for each thread i in the Block of threads do
2     while $i<N_s$ do
3         for each particle id j do
4             if $PC_j = i$ then
5                 push back particle id j into $CP_i$;
6                 j++;
7         $i = i + BlockDim$;
```

Algorithm 4. Creating neighbor list for particle id and period of N_p particles

```
Input: The set of wrapped global particle positions, $\tilde{X}_n$; The set of particle radii, $R_n$; The set of periods of
       particles, $P_n$; The amplified factor, $\delta$;
Output: The neighbor list of particle ids, NL and particle periods, $NLP;N_{N_p}^{jgi};N_{N_p}^{jli}$;
1  for each thread i in the Block of threads do
2      while $i<N_p$ do
3          $X_i$ = wrapped global position from $\tilde{X}_i$;
4          $i = i + BlockDim$;
5  for each thread i in the Block of threads do
6      while $i<N_p${loop all particles} do
7          for each subCell id k in {$CP_i$ and its adjacent} do
8              for each particle id j in subCell k and {$j \neq i$} do
9                  if distance is less than the threshold, that is, Eq. (30b) then
10                     if $j>i$ then
11                         push j and $p^g$ into $NL_i$ and $NLP_i$ respectively from left;
12                     else
13                         push j and $p^g$ into $NL_i$ and $NLP_i$ respectively from right;
14         record $N_i^{jgi}$ and $N_i^{jli}$;
15         $i = i + BlockDim$;
```

3.2.3 | Neighbor list for particle id and period

Algorithm 4 lists the pseudocode of creating the neighbor list for particle id and period. Prior to creating the neighbor list, the wrapped local particle positions $\tilde{X}_{N_p}$ need transforming back to the global coordinate system by Equation (16a), yielding the wrapped global particle positions X_{N_p}. The neighbors of a given particle i can be searched with the following three steps:

(1) Locating the subCell that the particle belongs in, that is, the sunCell id given by PC_i.

(2) Finding the nearest adjacent subCells in which the particles are the potential neighbors of the given particle i. There are $3^n - 1$ ($n = 2$ or 3 for 2D or 3D, respectively) adjacent subCells for a given subCell. At the subCell's integer coordinate system, referring to Figure 5, the coordinates of adjacent subCells are shifted by $\Delta X'$ ($\Delta X' \in \{-1, 0, 1\}$) for each axis. Note that the one with all $\Delta X'$ equal to 0 along all axes is the subCell of interest itself. In the presence of the periodic boundary conditions, the adjacency of subCell situating at the boundary may have a coordinate X' negative or beyond the number of subCell at the corresponding axis. In this case, the subCell at the opposite boundary is shifted by a certain period vector as the adjacent subCell. In a vivid 2D illustration in Figure 6, attention is paid to the subCell of interest with id equal to 12, where part of its adjacent subCells is shifted from the opposite boundary (see the arrows in Figure 6(A) for shifting action). Similar to locating a particle, we also attach a period p_k^s (but only in $\{-1, 0, 1\}$) to a subCell k. For example, subCell 15 is shifted to the left with a period of -1 along X_1', while subCell 0 is shifted to the top with a period of 1 along X_2'. Figure 6(B) shows shifting periods of subCell 12 and its adjacent subCells. The shifted adjacent subCells with at least a shifting period nonzero can be regarded as images of the corresponding subCells. Hence, the particles within these images of subCells are also images as shown in Figure 6(C). However, we still regard these particle images as real particles (wrapped into the RVE cell) staying within the RVE cell since the shifting periods are already recorded.

(3) Traversing all particles belong in the subCell of interest and its adjacent subCells. A particle j is taken as a neighbor if the distance d between particle j and particle i is less than a threshold, that is,

$$\|X_i - X_j - \boldsymbol{d}_{shift}\| < \delta(R_i + R_j), \tag{30a}$$

$$\boldsymbol{d}_{shift} = \boldsymbol{p}_k^s \boldsymbol{l}, \tag{30b}$$

where R_{N_p} is a set of particle radii; $\boldsymbol{d}_{shift}$ and $\boldsymbol{p}_k^s$ are the shifting vector and period of the adjacent subCell k; $\boldsymbol{l}$ is the base vector of the RVE cell boundary; δ is an amplified factor to scale up the searching radius. Then, the neighbor lists NL_i and NLP_i are filled with the neighbor particle id j and the relative shifting period p^g of the neighbor particle j at the global coordinate system. To facilitate the implementation, both j and p^g are pushed into the sublists from left if particle id j is greater than particle id i, referring to the columns with $i = 0, 1$ in Table 3, otherwise from right, referring to the columns with $i = 2, 6$ in this table. The relative shifting period p^g is given by

$$p^g = \begin{cases} P_i - P_j + p^s, & \text{if } j > i, \\ P_j - P_i - p^s, & \text{otherwise}, \end{cases} \tag{31a}$$

where P_{N_p} is the set of particle periods; p^s is the shifting period of the subCell k that the neighbor particle j belongs in.

Table 3 shows both neighbor lists NL_{N_p} and NLP_{N_p} for the exemplified RVE configuration in Figure 4. It is worth noting that each sublist has an equal-length segment of memory preallocated on a GPU, which should be sufficiently large to cover all possible neighbors for each particle but sufficiently small to avoid a significant waste of hardware resources (i.e., GPU memory). The reader may wonder that the neighbor lists presented here doubly store the neighbor information,

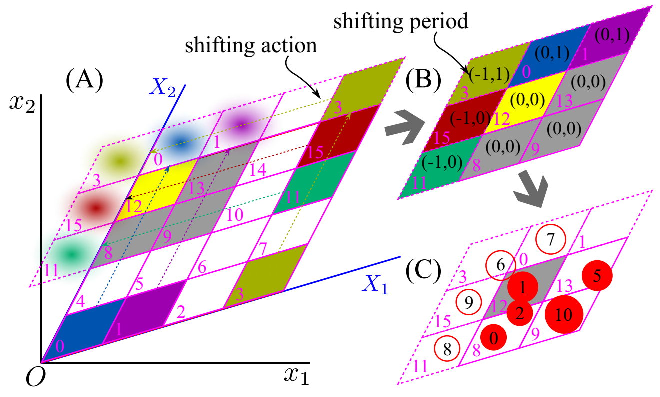

FIGURE 6 A subCell (id = 12) surrounded by adjacent subCells with periodic boundary conditions [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 3 Exemplified neighbor lists NL_{N_p} and NLP_{N_p} for particle id i and period within an N_p-particle RVE

i	0	1	2	3 4 5	6	7 8 9 10
NL_i	$[2,*]$	$[2,6,*]$	$[*,0,1]$	$[*][*][*]$	$[*,1]$	$[*][*][*][*]$
NLP_i	$[(0,0),*]$	$[(0,0),(-1,1),*]$	$[*,(0,0),(0,0)]$	$[*][*][*]$	$[*,(1,-1)]$	$[*][*][*][*]$
N_i^{jgi}	1	2	0	0 0 0	0	0 0 0 0
N_i^{jli}	0	0	2	0 0 0	1	0 0 0 0
S_i^{jgi}	0	1	3	3 3 3	3	3 3 3 3

Note: The star symbol * denotes the preserved GPU memory not updated yet for the list.

TABLE 4 Exemplified contact list

Contact id (i)	0	1	2
Particle id1 (C_i^{id1})	0	1	1
Particle id2 (C_i^{id2})	2	2	6
Period (C_i^{period})	(0,0)	(0,0)	(−1,1)

resulting in data redundancy; for example, Particle 2 is stored in the neighbor list of Particle 0, while Particle 0 is restored in the neighbor list of Particle 2. However, the presented data structure is designed due to the following facts: (1) the equal-sized sublist offers better efficiency in data accessing on a GPU, for example, coalesced access from global memory; (2) the particle-contact list introduced in the next subsection benefits from such a structure. In addition, we also note that it is not necessary to erase or initialize the memory area prior to updating NL_{N_p} and NLP_{N_p} (see the nonupdated memory denoted by a star * in Table 3). Another two accompanying lists $N_{N_p}^{jgi}$ and $N_{N_p}^{jli}$ are introduced to record the numbers of neighbors with id j greater or less than particle id i, respectively, so that the neighbors for a given particle i can be accessed readily.

3.2.4 | Contact list and particle-contact list

As aforementioned, the neighbor list doubly stores the information of neighbors (contact pairs). Hence, the contact pairs are accessible by traversing only half of the neighbor list, for example, neighbors with id j greater than i. Indeed, the list of contact ids can be established by

$$cid = S_i^{jgi} + j, \quad j \in \{0, \dots, N_i^{jgi} - 1\}, \tag{32a}$$

$$S_i^{jgi} = \begin{cases} 0, & \text{if } i = 0, \\ \sum_{k=0}^{i-1} N_k^{jgi}, & \text{otherwise}, \end{cases} \tag{32b}$$

where $S_{N_p}^{jgi}$ is the prefix sum of $N_{N_p}^{jgi}$, referring to Table 3 as an example. Prior to creating the contact list, $S_{N_p}^{jgi}$ is computed first by using the parallel algorithm[48] with one block of threads, and the total number N_c of contacts is then given by

$$N_c = S_{N_p-1}^{jgi} + N_{N_p-1}^{jgi}. \tag{33}$$

For a given contact i, we introduce three ingredients, namely particle $id1$, particle $id2$ for the two contacting particles, respectively, and contact *period* by which particle $id2$ is shifted to particle $id1$ for contact computation at the global coordinate system. For the convenience of implementation, $id1$ is specified less than $id2$ by default. Then, three lists $C_{N_c}^{id1}$, $C_{N_c}^{id2}$ and $C_{N_c}^{period}$ are allocated for the lists of particle $id1$, particle $id2$, and contact *period* for all contacts, respectively. The contact list for the exemplified RVE configuration is listed in Table 4. Note that the contact periods listed in this table are calculated assuming zero-periods of particles, that is, $P_i = 0$ in Equation (31).

Algorithm 5. Creating contact list and particle-contact list *NLC*

```
Input: $N^{jgi}$; $N^{jli}$; $NL$; $NLP$; $S^{jgi}$;
Output: $C^{id1}$; $C^{id2}$; $C^{period}$; $NLC$;
1  for each thread i in the Block of threads do
2      while $i<N_p$ do
3          for each j in $\{1,\dots,N_i^{jgi}\}$ do
4              particle $id2$ = the j-th item in $NL_i$;
5              $cid = S_i^{jgi} + id2$;
6              $C_{cid}^{id1} = i$;$C_{cid}^{id2} = id2$;$C_{cid}^{period}$ = the j-th item in $NLP_i$;
7              the j-th from the left in $NLC_i = cid$ ;
8              for each k in $\{1,\dots,N_j^{jgi}\}$ do
9                  if the k-th from the right in $NL_j$ is equal to i then
10                     the k-th from the right in $NLC_j = cid$ ;
11         $i = i + BlockDim$;
```

Algorithm 5 lists the pseudocode of creating both contact list and particle-contact list. For a given particle i, a particle-contact list NLC_i is designed to group all its contact ids. The particle-contact list NLC_{N_p} has the same size as the neighbor list NL_{N_p} so that it can be accessed in a similar fashion as NLC_{N_p}, which significantly facilitates the implementation on a GPU. Hence, we first traverse the neighbor list NL_i with particle id j greater than i (i.e., the left N_i^{jgi} particles) to obtain a sublist contact ids by Equation (32) and store in the corresponding position in NLC_i. Then, the contact id for particles i and j is pushed into the corresponding position (where NL_j is equal to i) in NLC_j. Table 5 lists the particle-contact list NLC_{N_p} for the exemplified RVE configuration in Figure 4.

3.3 | Contact force

For a given contact i with two particles C^{id1} and C^{id2}, the position of particle $id2$ is first shifted by $\boldsymbol{d}_{shift}$, that is,

$$\boldsymbol{d}_{shift} = C_i^{period}\boldsymbol{l}, \tag{34}$$

where $\boldsymbol{l}$ is the base vectors of the RVE cell. Then, the contact geometric quantities (such as branch vector $\boldsymbol{b}$, contact normal $\boldsymbol{n}$, and penetration $\boldsymbol{d}$) can be obtained by Equations (8), (9), and (10), respectively, in terms of particle positions ($X_{C_i^{id1}}$ and $X_{C_i^{id2}}$), particle radii ($R_{C_i^{id1}}$ and $R_{C_i^{id2}}$). For a contact i where the penetration depth is greater than zero (i.e., real contact), the normal contact force FN_i is obtained by Equation (13a); as for the tangential contact force, the tangential displacement increment $\delta\boldsymbol{u}$ is computed by Equation (12) with the relative velocity in Equation (11) in consideration of the mean-field velocity in Equation (20a) due to the deformation of RVE cell, then substituted into Equation (13b) for the incremental tangential contact force $\Delta\boldsymbol{f}^t$.

The Coulomb condition is performed on the accumulated tangential contact force in terms of the present normal contact force FN_i by Equation (14), yielding the tangential contact force FT_i. With the tangential contact force, the torques for both particle $id1$ and particle $id2$ are obtained by

$$C_i^{t1} = \boldsymbol{r}_1 \times FT_i, \tag{35a}$$

$$C_i^{t2} = \boldsymbol{r}_2 \times FT_i, \tag{35b}$$

TABLE 5 Exemplified particle-contact list NLC_{N_p} within an N_p-particle RVE

Particle id *i*	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
NLC_i	[0, *]	[1, 2, *]	[… , 0, 1]	[*]	[*]	[*]	[*, 2]	[*]	[*]	[*]	[*]

Note: The star symbol * denotes the preserved GPU memory not updated yet for the list.

where $C^{t1}_{N_c}$ and $C^{t2}_{N_c}$ are the lists of torques on particle $id1$ and particle $id2$, respectively; $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the position vectors of the contact point with respect to the two particle centers, respectively; $\times$ denotes cross product. Algorithm 6 lists the pseudocode of computing contact forces in parallel on a GPU.

Algorithm 6. Contact force computation at the global coordinate system

```
Input: $X_{N_p}$; $R_{N_c}$; $V_{N_c}$; $C^{id1}_{N_c}$; $C^{id2}_{N_c}$;$C^{period}_{N_c}$; $C^{id2pre}_{N_c}$; $S^{jgipre}_{N_p}$; $FT^{pre}_{N_c}$;
Output: $FN_{N_c}$; $FT_{N_c}$; $C^{t1}_{N_c}$; $C^{t2}_{N_c}$; $CST_{N_c}$;
1  for each thread i in the Block of threads do
2      while $i<N_c$ do
3          $id1 = C^{id1}_i$; $id2 = C^{id2}_i$;
4          $X_{id2}$ shifted by $d_{shift}$ in Eq. (34);
5          $\boldsymbol{b}$ by Eq. (8), $\boldsymbol{n}$ by Eq. (9), $\boldsymbol{d}$ by Eq. (10) with $X_{C^{id1}_i}, X_{C^{id2}_i}, R_{C^{id1}_i}, R_{C^{id2}_i}$;
6          if it is a real contact then
7              $FN_i$ by Eq. (13a);
8              $\boldsymbol{v}^{1,2}$ by Eq. (11) and Eq. (20a), $\delta\boldsymbol{u}$ by Eq. (12), $\Delta\boldsymbol{f}^t$ by Eq. (13b);
9              finding previous contact id $cid^{pre}$; $ft^{pre} = FT^{pre}_{cid^{pre}}$;
10             $FT_i$ subjected to the Coulomb condition in Eq. (14) with $FN_i$;
11             $CP^{t1}_i$, $CP^{t2}_i$ by Eq. (35);
12             $CST_i = (FN_i + FT_i) \otimes \boldsymbol{b}$;
13             //$\otimes$ denotes dyadic product
14         else
15             $FN_i, FT_i, C^{torque1}_i, C^{torque2}_i, CST_i$ are set to zeros;
16         $i = i + BlockDim$;
```

3.4 | Integration of particle motion

Given that contact forces and torques for all contacting particle pairs are stored independently in the lists (FN_{N_c}, FT_{N_c}, $C^{t1}_{N_c}$, and $C^{t2}_{N_c}$), the resultant force $\boldsymbol{f}$ and torque $\boldsymbol{T}$ for each particle can be obtained in parallel by summing over all contacts that belong to the particle as

$$\boldsymbol{f} = \sum_{c\in C^{jgi}_i} (FN_c + FT_c) - \sum_{c\in C^{jli}_i} (FN_c + FT_c), \tag{36a}$$

$$\boldsymbol{T} = \sum_{c\in C^{jgi}_i} C^{t1}_c + \sum_{c\in C^{jli}_i} C^{t2}_c \tag{36b}$$

with

$$C^{jgi}_i = \{c|\ \text{the } k\text{th item from the left in } NLC_i, k = 1, \dots, N^{jgi}_i\}, \tag{37a}$$

$$C^{jli}_i = \{c|\ \text{the } k\text{th item from the right in } NLC_i, k = 1, \dots, N^{jli}_i\}, \tag{37b}$$

where C^{jgi}_i and C^{jli}_i are the contact subsets of particle i with its neighbor j greater than or less than i, respectively. As for Equation (36), it is clear that each item in both $C^{t1}_{N_c}$ and $C^{t2}_{N_c}$ is accessed only once without any race conditions for multithreads. By contrast, each item in both FN_{N_c} and FT_{N_c} is accessed twice, since each contact force is shared by two particles and decoded with Equation (15) for saving GPU memory, which may result in race conditions. However, such a worse case can be successfully avoided by partitioning the twice access into two subloops so that each item in both FN_{N_c} and FT_{N_c} is accessed only once at each loop, as listed at Line 3 and Line 5 in Algorithm 7.

Algorithm 7. Integrating particle motion at the global coordinate system

```
Input: $X_{N_p}$;$R_{N_p}$;$X^{ref}_{N_p}$;$V_{N_p}$;$W_{N_p}$;$FN_{N_c}$;$FT_{N_c}$;$C^{t1}_{N_c}$;$C^{t2}_{N_c}$;$MI_{N_p}$;$NLC_{N_p}$;
Output: updated $X_{N_p}$; updated $V_{N_p}$; updated $W_{N_p}$;
1  for each thread i in the Block of threads do
2      while $i<N_p$ do
3          for each contact $c \in C^{jgi}$ in Eq. (37a) do
4              $\boldsymbol{f} += FN_c + FT_c$; $\boldsymbol{T} += C^{t1}_c$;
5          for each contact $c \in C^{jli}$ in Eq. (37b) do
6              $\boldsymbol{f} -= FN_c + FT_c$; $\boldsymbol{T} += C^{t2}_c$;
7          $\dot{\boldsymbol{v}} = \boldsymbol{f}/M_i$; $\dot{\boldsymbol{\omega}} = \boldsymbol{T}/I_i$;
8          $\Delta\boldsymbol{L} = \boldsymbol{L} - \boldsymbol{L}'$;
9          damping $\dot{\boldsymbol{v}}$ and $\dot{\boldsymbol{\omega}}$ with Eqs. (4) and (38);
10         updating $V_i$ and $W_i$ with Eqs. (39a) and (39b);
11         $X_i += V_i\Delta t$;
12         $\boldsymbol{L}' = \boldsymbol{L}$;
13         if ($\|X_i - X^{ref}_i\| < threshold$) set updateNL true;
14         $i = i + BlockDim$
```

With the resultant force and torque of particle i, the corresponding linear acceleration $\dot{\boldsymbol{v}}$ and angular acceleration $\dot{\boldsymbol{\omega}}$ are obtained by Equations (1a) and (1b), respectively, which are then damped with Equation (4) ($\dot{\boldsymbol{\omega}}$ is damped by a similar equation). Note that in the presence of periodic boundary conditions, the linear acceleration $\dot{\boldsymbol{v}}$ is damped with respect to the fluctuating velocity $\boldsymbol{v}_f$ of the particle rather than the total one V_i (V_{N_p} is the list of particle linear velocities). In other words, the linear velocity $\boldsymbol{v}$ in Equation (4b) should exclude the mean-field velocity, that is,

$$\boldsymbol{v}_f = V_i - \boldsymbol{L}'X_i, \tag{38}$$

where $\boldsymbol{L}'$ is the velocity gradient at the last time step. The increments for both linear and angular velocities are given by

$$\Delta\boldsymbol{v} = \Delta\boldsymbol{L}X_i + (\dot{\boldsymbol{v}}_d + \boldsymbol{L}V_i)\Delta t, \tag{39a}$$

$$\Delta\boldsymbol{\omega} = \dot{\boldsymbol{\omega}}_d\Delta t, \tag{39b}$$

where $\dot{\boldsymbol{v}}_d$ and $\dot{\boldsymbol{\omega}}_d$ are damped linear and angular accelerations, respectively. The linear velocity V_i and angular velocity W_i of particle i are then updated in terms of Equations (39a) and (39b), respectively, followed by updating the position X_i. Note that it may be not necessary to save the orientation for spherical particles. Furthermore, the flag *updateNL* introduced in Section 3.1 (see Line 6 in Algorithm 1) is updated once the accumulative displacement (i.e., $\|X_i - X^{ref}_i\|$) of a particle with respect to its reference position X^{ref}_i exceeds the prescribed *threshold* (e.g., one subCell size). The list of reference positions $X^{ref}_{N_p}$ for all particles needs updating with the current positions prior to updating the neighbor list. The entire pseudocode of integrating particle motion in parallel is listed in Algorithm 7.

4 | GODEM TESTS

4.1 | Test setup

The open-source CPU-based DEM code, *SudoDEM*,[17,42,49] developed by the authors, is employed as a baseline to benchmark the proposed thread-block-wise algorithms on GPU. *SudoDEM* is available at its project page online, and the Python snippets and data sets involving in this section are available on the Github repository (https://github.com/SwaySZ/ExamplesSudoDEM) for interested readers. The proposed GPU algorithms are implemented using CUDA C++ in our in-house code, *GoDEM* (GPU-supported Object-oriented Discrete Element Modeling), which will be publicly shared in an open-source manner in the future. The GPU-specific techniques such as coalesced reading/writing, shared memory utilization and unified memory implementation are utilized in *GoDEM* for better computational efficiency.

SudoDEM runs in double-precision on a desktop with an Intel Core I7-6700 CPU (3.4 GHz, four physical cores, and eight logical cores) and 16 GB RAM, while *GoDEM* runs in single-precision on an Nvidia GeForce RTX 2080 Ti GPU card (68 streaming multiprocessors with 4352 CUDA cores and 11 GB GDDR6 memory). Note that the GPU computing prefers to use single-precision for performance, and the possible difference in results will be examined in the following section. More details on the specifications of the Intel CPU and the Nvidia GPU are available online. The operating system is Ubuntu 18.04, and the program compilers are GCC 7.4 and NVCC 10.1. It is worth noting that *GoDEM* runs entirely on the GPU without communicating with the host CPU during the course of RVE simulating, benefiting from our novel parallelism framework. Indeed, the performance of *GoDEM* is independent of the CPU performance.

4.2 | Validation

4.2.1 | Simulation setup

We prepare a dense RVE packing of 400 disks with radii uniformly distributed between 2.5 and 5.0 mm following the well-established protocol in the literature.[1,50] Using a dense specimen meets the following two highlighted considerations: (1) reducing the possible discrepancy in contact force (especially the tangential part) between the two initial packings for *SudoDEM* and *GoDEM* and (2) making the computation sufficiently intensive with increasing contact number for a better estimation of performance at the worst case. Moreover, the sample size of an RVE (i.e., 400 particles) ensures sufficiently isotropic fabric when subjected to isotropic compression (similar to consolidation in geomechanics) as reported in our previous study.[29] The simulation parameters are selected as follows: both normal and tangential contact stiffnesses k_n and k_t are set to 1×10^6 N/m; the friction of coefficient $\mu = 0.5$, the mass density of particle $\rho = 2650$ kg/m^3, and the artificial damping $\alpha_d = 0.3$. Figure 7 shows the initial configuration of the RVE packing with an isotropic confining stress of 100 kPa.

Uniaxial compression, simple shear, and biaxial compression tests are performed on the confined RVE packing with a loading strain rate of 0.05/s. As shown in Figure 8, the loading strain rate is applied to ϵ_{11}, ϵ_{01}, and ϵ_{11} by moving the corresponding periodic boundaries for uniaxial compression, simple shear, and biaxial compression tests, respectively. In addition, the side boundaries are fixed for the uniaxial compression test, while for the biaxial compression test a confining stress of σ_0 is maintained constant (i.e., 100 kPa) with a numerical stress-controlled servo mechanism. As for the simple shear test, the periodic cell experiences a homogeneous deformation with a constant velocity gradient ($L_{01} = 0.05$/s).

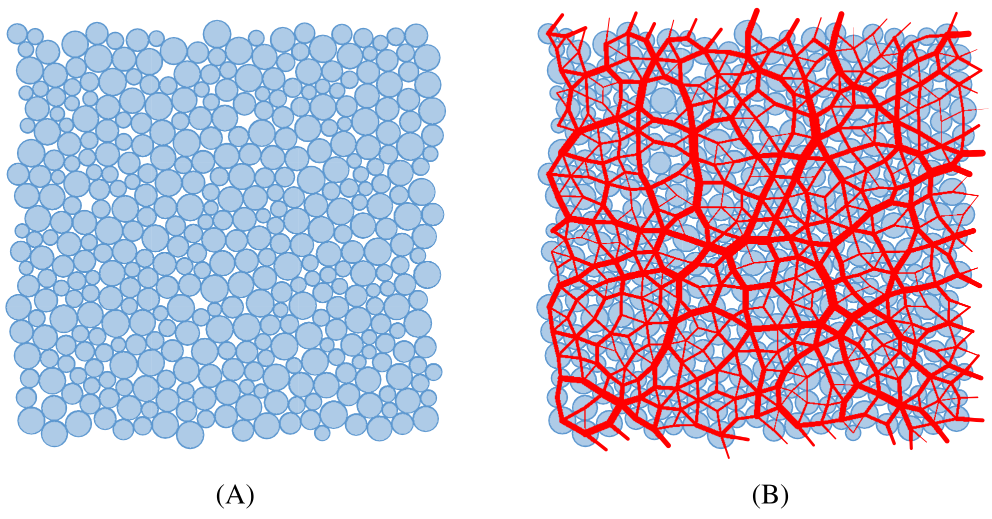

FIGURE 7 (A) Initial configuration of an RVE packing with an isotropic confining stress of 100 kPa and (B) the corresponding superimposed normal contact force chains [Colour figure can be viewed at wileyonlinelibrary.com]

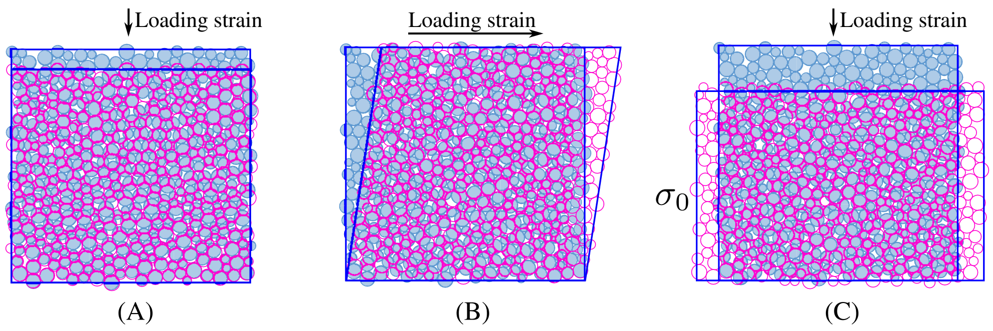

FIGURE 8 Loading conditions for the three tests: (A) uniaxial compression; (B) simple shear; (C) biaxial compression with a confining stress of σ_0. *Note:* solid and open disks correspond to initial and compressed/sheared configurations, respectively; blue wireframes indicate the periodic boundaries [Colour figure can be viewed at wileyonlinelibrary.com]

4.2.2 | Effect of thread-block size

Given that *GoDEM* runs completely on a GPU with multithreads for every single RVE, the results may vary once any race conditions occur among threads. To validate the implementation of the proposed algorithms, different numbers of threads (i.e., different thread-block sizes of 1, 32, 128, and 256 threads) are used for the three simulation tests. It is worth noting that the threads are handled group by group, and each group is composed of 32 threads, that is, a warp in the terminology of CUDA. All 32 threads in a warp execute the same instruction, that is, the single instruction multiple threads (SIMT) mechanism, meaning that the number of active threads is recommended to be integer multiple of 32 in practice for maximizing the utilization of hardware (only four warp schedulers per streaming multiprocessor). However, since a single thread is race-condition free, the block with a single thread is adopted to benchmark the multithread results hereby.

Figure 9 shows the deviatoric stress ratio q/p and volumetric strain for the three tests with different numbers of threads using *GoDEM*. It can be seen that there is no significant discrepancy at all between results from different numbers of threads for the uniaxial compression test and the simple shear test, shown in Figure 9(A,B), respectively. Interestingly, as for the biaxial compression test, both deviatoric stress ratio and volumetric strain are not significantly influenced by thread number until reaching some level (approximately 4% here) of axial strain in Figure 9(C,D), respectively. Nevertheless, it is not surprising to see the accumulative discrepancy in results causing by thread number for the biaxial compression test, and the reason is analyzed as follows. Compared with the uniaxial compression and simple shear tests, one more module has been installed in the biaxial compression test to offer a constant confining stress of σ_0, that is, stress-controlled servo, in which the stress is obtained by summing over all contacts in parallel. Since floating-point arithmetic is nonassociative, that is, $(a+b)+c \neq a+(b+c)$, summing up the stress in a different order (access order varies with thread number) can yield different results. The discrepancy in stress is then propagated back into the system due to the stress-controlled servo for the biaxial compression test, thereby varying results after some level of loading strain. Nevertheless, the discrepancy in results is not significant, as can be seen in Figure 9(C,D). Moreover, in multiscale modeling by using either FEM×DEM[29] or MPM×DEM[34] coupling approaches, the strain-controlled deformation (i.e., applying incremental strain to an RVE assembly) is applied rather than the stress-controlled servo, thereby no issue as mentioned earlier at all.

4.2.3 | Comparison with CPU results

Prior to comparing the performance of between *GoDEM* and *SudoDEM*, the results from *GoDEM* need validating against those from *SudoDEM*. To this end, the results from *GoDEM* using 128 threads (without losing generality) are plotted against those from single-CPU-core *SudoDEM* in Figure 10 for the three simulation tests. It is clear that the GPU results are well consistent with the CPU results, validating the implementation of the proposed algorithms accordingly. However, one may see a small discrepancy in results between these of the two codes for the biaxial compression test, which is caused by the stress-controlled servo due to the nonassociative property of float point arithmetic as analyzed in the last subsection.

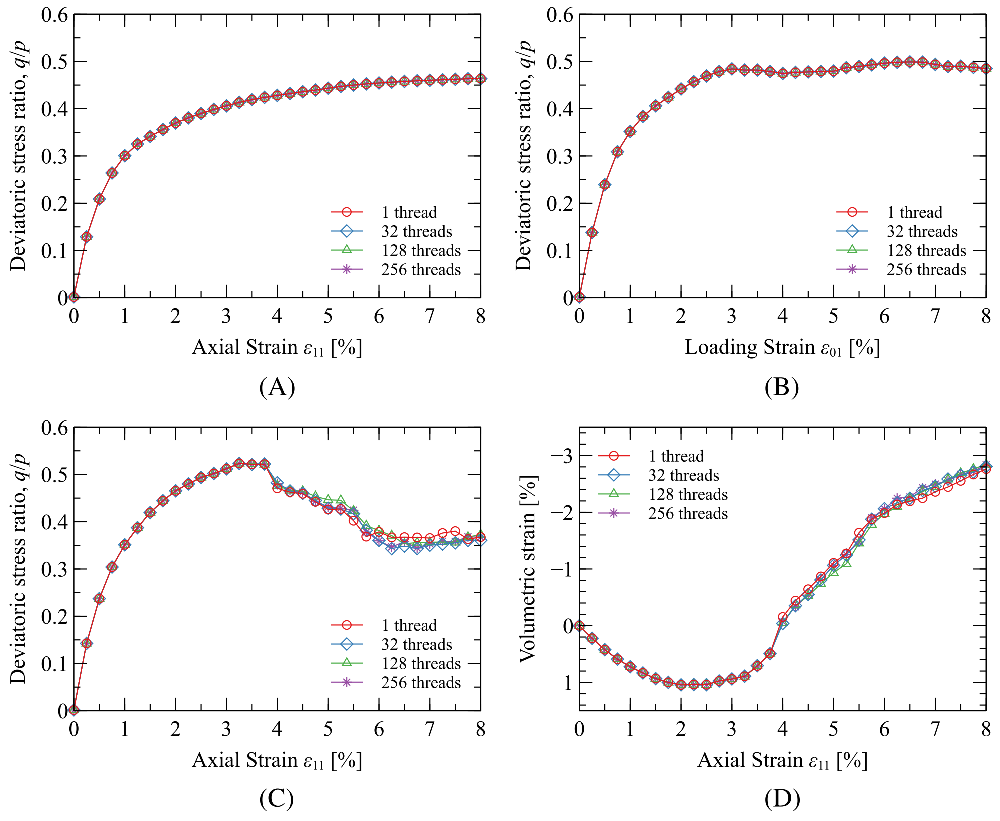

FIGURE 9 Comparison of results from different threads using *GoDEM*: deviatoric stress ratio q/p in (A) uniaxial compression, (B) simple shear and (C) biaxial compression tests; and (D) volumetric strain in the biaxial compression test [Colour figure can be viewed at wileyonlinelibrary.com]

TABLE 6 Four groups of *GoDEM* configurations

			Max. blocks per SM		
Group tag	**Threads (T) per block**	**Registers (R) per thread**	**Block limit**	**Register limit**	**Occupancy**
G128T-130R	128	130	8	3	37.5%
G256T-130R	256	130	4	1	25.0%
G128T-64R	128	64	8	8	100%
G256T-64R	256	64	4	4	100%

4.3 | Performance

4.3.1 | Test setup and remarks

The performance of a GPU code is sensitive to the run-time configuration, such as thread-block size, shared memory usage, and register pressure due to the limited hardware resources. For example, the GPU card employed in this study, Nvidia GeForce RTX 2080 Ti, has one TU102 GPU with the Turing architecture: the maximum number of resident threads per block is 1024; the maximum number of resident blocks per Streaming Multiprocessor (SM) is 16; there are 64 KB shared memory per SM, and 48 KB shared memory per block by default, and 64 KB 32-bit registers per SM. In the implementation of *GoDEM*, shared memory is dynamically allocated for each block/RVE with $2N_p$ bytes for particle positions and 4*BlockDim* bytes for cache so that the GPU occupancy is not limited by shared memory for the following test setting (see, e.g., Table 6). Note that shared memory has much lower latency than global memory, and it can considerably promote the performance. However, it is the responsibility of the designer to make sure a correct access pattern when using shared memory; otherwise, the performance becomes even worse when there are bank conflicts in shared memory. Detailed introduction to shared memory is beyond the scope of this work, and interested readers are referred to the literature.[48]

The register-usage has nothing to do with computational results but may influence the computational efficiency to some extent. Indeed, the register operation is hidden inside the processing of a high-level language compiler (e.g., GCC

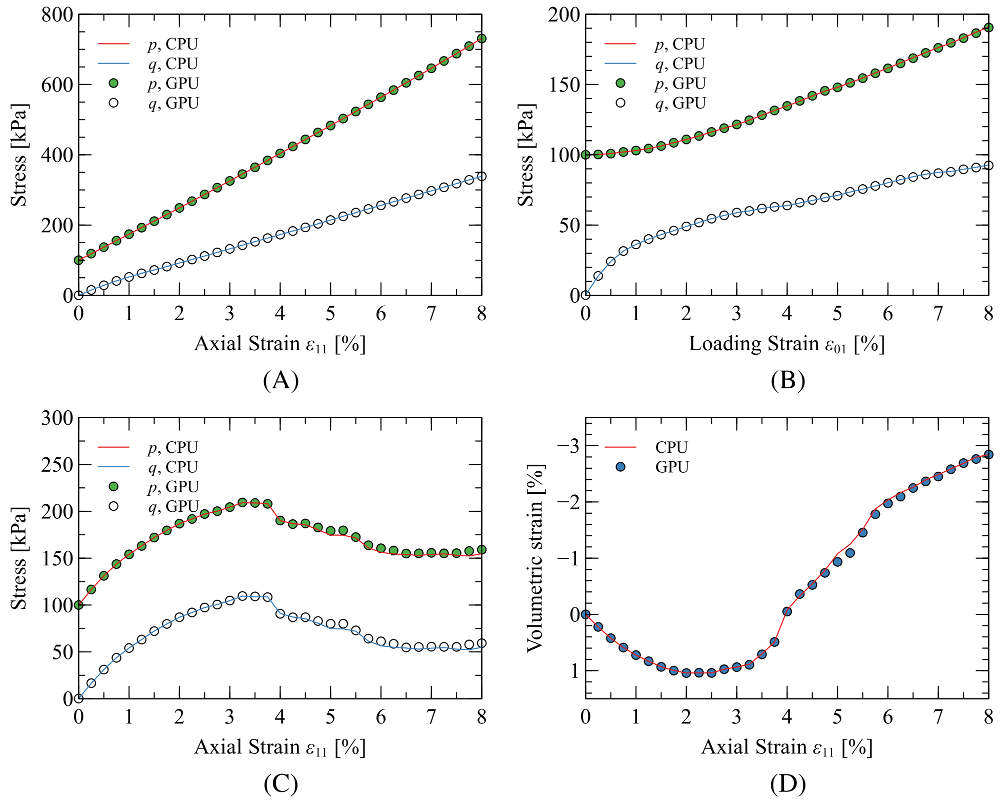

FIGURE 10 Comparison of results from *SudoDEM* (single CPU-core) and *GoDEM* (128 GPU-threads): mean stress p and deviatoric stress q in (A) uniaxial compression, (B) simple shear and (C) biaxial compression tests; and (D) volumetric strain in the biaxial compression test [Colour figure can be viewed at wileyonlinelibrary.com]

and NVCC for compiling C/C++ source files). Registers can be regarded as a block of on-chip memory with the lowest latency in reading and writing, also known as L0 Cache. As documented in Nvidia's guide,[23] registers are much faster than global memory (the off-chip RAM), so that a program runs faster with higher usage of registers in general. However, the register file size is extremely limited, for example, only 65,536 (64 KB) 32-bit registers per SM on Nvidia's Turing architecture. Therefore, a high register-usage may cause significant register pressure on multithreads running concurrently, thereby resulting in low occupancy for an SM. With the default compiler's optimization provided by NVCC 10.1, *GoDEM* has a relatively high register-usage of 130 registers per thread. After a quick calculation, an SM can run at most $65,536/130 \approx 504$ threads concurrently, which almost halves the designed capability (1024 threads for the Turing architecture) of an SM, that is, a low SM occupancy. Moreover, the number of active threads is set to an integer multiple of 32 for maximizing the utilization of warp schedulers. Hence, the theoretical maximum thread number that can be issued is 480 (15 warps) instead of the aforementioned 504 for a register pressure of 130. To improve the occupancy, the register pressure is reduced as a trade-off. Hence, the second version of *GoDEM* binary is compiled with a moderate register-usage of 64 registers per thread by forcing the compiler to rearrange the register-usage with the option "maxrregcount," thereby no register pressure on the SM with 1024 threads.

To sum up, we compile two binaries of *GoDEM* with 130- and 64-register pressure per thread, respectively. Following the same simulation setups as introduced in Section 4.2.1, the performance tests on both single and many RVEs are carried out with the three typical tests, including uniaxial compression, simple shear, and biaxial compression. For the single-RVE test, only one block of threads is launched, and the performance is monitored with different block sizes. For the many-RVE test, sequential blocks of threads are launched for each RVE with the same simulation, and the performance is recorded for different RVE numbers.

4.3.2 | On a single RVE

The computational performance of *GoDEM* is examined first for a single RVE packing, which can be quantified by the computational speed, that is, simulation steps (iterations) per wall-clock second. We run each simulation test 10 times with different thread numbers (1, 32, 64, 128, and 256), and each simulation runs 30,000 steps (iterations) in total. The average computational speeds of the two versions of (130- and 64-register-per-thread) *GoDEM* are recorded for the three

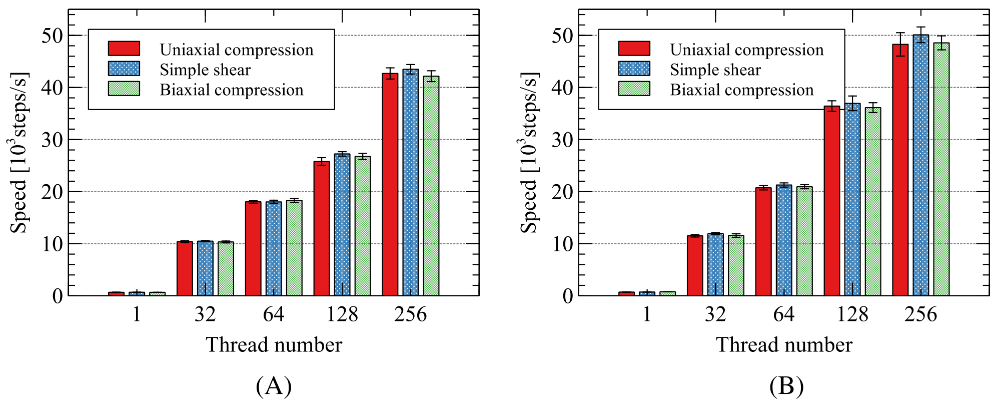

FIGURE 11 Computational speed (steps per wall-clock second) of a single RVE simulation for the three tests varying with GPU thread number using *GoDEM*: (A) 64 registers per thread and (B) 130 registers per thread. Error bars represent the standard deviation for 10 repetitions [Colour figure can be viewed at wileyonlinelibrary.com]

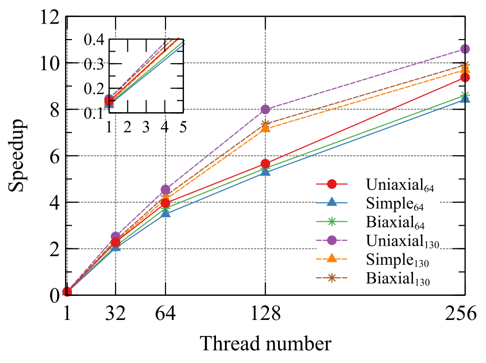

FIGURE 12 Speedup of *GoDEM* with respect to single-CPU-core *SudoDEM* on a single RVE varying with thread number. "Uniaxial$_*$," "Simple$_*$," and "Biaxial$_*$" are short for uniaxial compression, simple shear, and biaxial compression tests with * registers per thread, respectively [Colour figure can be viewed at wileyonlinelibrary.com]

simulation tests in Figure 11. It can be seen that there is no significant difference among the average speeds of the three simulation tests, indicating that the computational efficiency is relatively stable for different loading paths performed on the RVE packing. Moreover, the computational speed increases with thread number increasing as expected.

The average computational speeds of *GoDEM* is then normalized by that of single-CPU-core *SudoDEM* to obtain speedup ratios, as shown in Figure 12. The speedup is approximately 0.15 for a single GPU thread, indicating that the performance of a single GPU thread is much lower than that of a CPU thread. However, the performance of the GPU is promising with increasing thread-usage. For example, the performance of a single warp (32 threads) doubles with respect to single-CPU-core performance. Increasing thread number definitely promotes the computational efficiency of a single RVE simulating, but the incremental speedup per thread is likely to decrease with further increasing thread number, implying an increasing waste of hardware resources meanwhile. A reasonable explanation is that part of threads in the thread-block is likely to become idle. Taking the 256-thread setting as an example, there are $400 - 256 = 144$ particles left after the first-round loop for particlewise parallelism (e.g., particle motion integration in Algorithm 7), so that $256 - 144 = 112$ threads are idle for the second-round loop. Furthermore, special attention is paid to the thread-block size of 128 threads, where the register pressure has a significant effect on the performance of *GoDEM*. It suggests that increasing GPU occupancy by decreasing register-usage may not necessarily yield a better performance, which is verified in the following subsection.

4.3.3 | On many RVEs

The performance analysis of *GoDEM* on simulating a single RVE ends up with the inference that the thread-block sizes of 128 or 256 threads in conjugation with either 130- or 64-register pressure can achieve a relatively high speedup ratio for many RVEs simulating in parallel. Hence, we increase the RVE number (up to 10,000 RVEs composed of four million particles in total) and conduct another four groups of tests with four combinations of setting listed in Table 6, respectively.

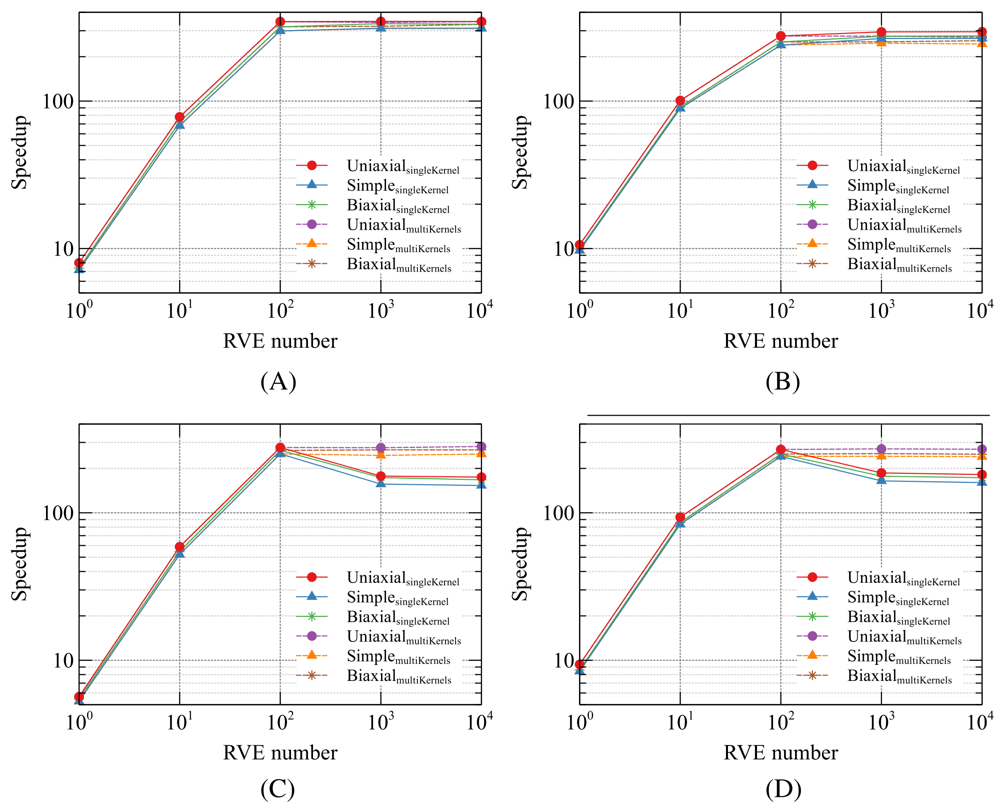

FIGURE 13 Speedup of *GoDEM* with respect to single-CPU-core *SudoDEM* varying with RVE number for different thread-block-wise configurations (optimization schemes): (A) G128T-130R, (B) G256T-130R, (C) G128T-64R, and (D) G256T-64R. "Uniaxial," "Simple," and "Biaxial" are short for uniaxial compression, simple shear, and biaxial compression tests, respectively [Colour figure can be viewed at wileyonlinelibrary.com]

Due to the limited hardware resources, the maximum concurrent block number per SM is limited by both block size and register pressure, respectively, listed in the fourth and fifth columns of Table 6. Accordingly, the occupancy of an SM is theoretically calculated as the ratio of the maximum active warps to the device-supported maximum warps ($1024/32 = 32$ for the Turing architecture), referring to the last column of Table 6.

As for the performance of single-CPU-core *SudoDEM*, given that sequentially simulating a number of RVEs (e.g., 10,000 RVEs) is time-consuming, the computational speed for a single RVE is taken as the baseline to evaluate the speedup of *GoDEM*. It is worth noting that the performance of *SudoDEM* is likely to degrade to some extent due to heavier memory cache with RVE number increasing, thereby that the speedup ratio is somewhat underestimated.

The computation of a set of RVEs is encapsulated into a so-called Kernel function that is invoked by the host CPU but runs on GPUs. Simulations run with a single Kernel function by which all RVEs are attached automatically to thread blocks at once. The speedups of *GoDEM* for the four configurations are plotted against RVE number in Figure 13 (see the solid lines). For the configurations, G128T-130R and G256T-130R shown in Figure 13(A,B), respectively, the speedup ratio of *GoDEM* increases and then reaches a plateau (up to approximately 350) with RVE number increasing, while the speedup ratio increases first to a peak (approximately 280), then decreases to a plateau (approximately 170) for the configurations G128T-64R and G256T-64R shown in Figure 13(C,D), respectively. Note that the plateau corresponds to the maximum occupancy that all SMs have already reached. Thus, it is evident that increasing occupancy can either increase or decrease performance. For example, G128T-130R has a higher occupancy and a better performance than G256T-130R; however, G128T-64R has a higher occupancy than G128T-130R but a significant degradation in performance. Therefore, it can be concluded that increasing occupancy does not always increase performance. Furthermore, it is of interest to see that the peak and/or plateau of speedup first appears at an RVE number of approximately 100. A reasonable explanation is given as follows: the GPU card, GeForce RTX 2080 Ti, is equipped with 68 SMs so that there are no resource limits for SM occupancy at a small RVE number (e.g., 100 RVEs) because the GPU driver assigns thread-blocks/RVEs to SMs in such a manner that the workloads on all SMs are as balanced as possible. With this fact, as can be seen in the figure, it is not surprising that the performance for the configuration with a larger block size is better than that with a smaller block size for RVE number smaller than 100.

Increasing occupancy by decreasing register pressure results in significant performance degradation when the maximum occupancy is reached. The deep reason may be that the register usage is simply spilled into L1 and L2 caches when the compiler reduces the register pressure below the imposed limit, that is, 64 registers per thread, making the memory

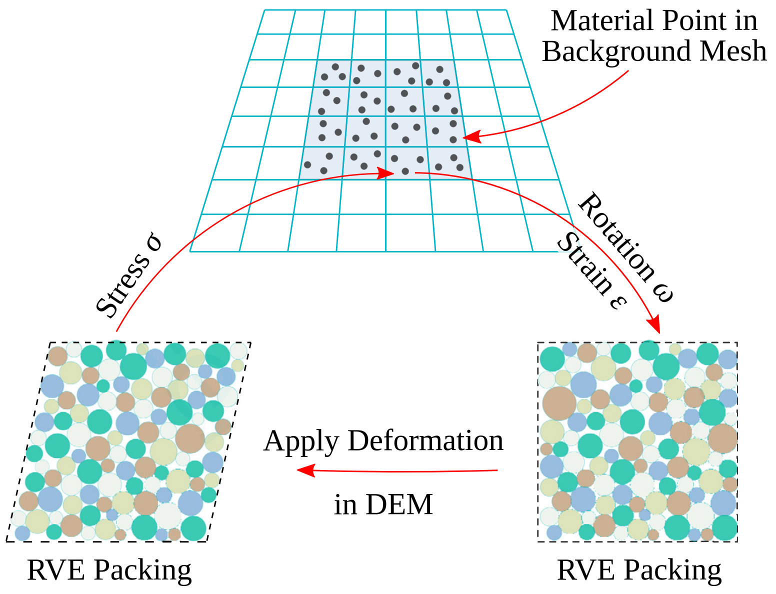

FIGURE 14 Coupling scheme of MPM and DEM, after[34] [Correction added on 05 November 2020, after first online publication: Figure 14 was published in black, which caused significant loss of information and has been replaced with the colored version.] [Colour figure can be viewed at wileyonlinelibrary.com]

cache overloaded. To verify this inference, we re-run all simulations with a sequence of Kernel functions by which each Kernel function handles only 100 RVEs. The corresponding speedup ratios are plotted in Figure 13 (see the dashed lines) together with that from a single Kernel function. It can be seen that the speedup ratio from multi-Kernels is almost identical with that from a single Kernel for 130-register pressure shown in Figure 13(A,B). By contrast, for 64-register pressure shown in Figure 13(C,D), it is clear that there is a significant promotion in performance when multi-Kernels are applied. With multi-Kernel functions, all SMs maintain a relatively low level of occupancy regardless of RVE number, which is almost the same as that of a single Kernel function with 100 RVEs, so that the performance reaches saturation with RVE number increasing beyond 100. Moreover, the memory cache (especially L1 cache) with multi-Kernel functions is indeed not as busy as that with single Kernel function, thereby yielding better performance.

5 | NEW PARALLEL COMPUTING POWERED HIERARCHICAL MULTISCALE MODELING: An MPM×DEM CASE

5.1 | Coupling scheme

The hierarchical multiscale coupling of MPM and DEM (MPM×DEM) has been well introduced in our previous work.[34] The critical techniques are depicted here for the completeness of the presentation. As illustrated in Figure 14, the macroscopic engineering domain is firstly discretized by a set of material points in MPM. The mechanical response of each material point is captured by an RVE (a DEM assembly) composed of discrete particles.

The MPM×DEM coupling cycle mainly comprises the following tasks:

(1) MPM derives the deformation of each material point in the continuum domain.
(2) The deformation information (incremental displacement gradient $d\boldsymbol{H}$, consisting of the incremental strain $\Delta\boldsymbol{\varepsilon}$ and incremental rotation $\Delta\boldsymbol{\omega}$) at each material point is passed to its corresponding RVE to serve as mesoscopic boundary conditions.
(3) DEM solves the motion of discrete particles inside every RVE.
(4) Cauchy stress is homogenized over the deformed RVE by Equation (24) and transferred back to its attached MPM material point for subsequent computation.

An open-source MPM solver *NairnMPM*[51] is employed to couple with a DEM solver, either *SudoDEM* or *GoDEM*, yielding two coupling MPM×DEMs, that is, *NairnMPM*/*SudoDEM*, and *NairnMPM*/*GoDEM*, respectively. In the proposed MPM×DEM coupling approach, DEM consumes the majority of computation time. Moreover, the computation of each RVE is independent with each other, which facilitates a highly parallel computing scheme. Hence, the computation of all RVEs are handled in parallel with *SudoDEM* or *GoDEM* running on CPU or GPU, respectively. Such a parallelism scheme helps substantially shorten the computational time and greatly enhance the performance of MPM×DEM for multiscale simulations.

In the following subsection, a practical engineering problem, rigid footing, is simulated by using the two coupling MPM×DEMs, where the simulated results and computational efficiency are examined.

5.2 | Simulation setup of rigid footing problem

As shown in Figure 15, two model setups are adopted for the rigid footing problem with two domains, the full domain and the half domain, respectively. Both domains are discretized into square elements with a dimension of 0.1 m by 0.1 m. The initial number of material points per MPM cell is set to 1. The lateral boundaries for the soil domain are constrained horizontally while the bottom is fixed in both directions. A constant, uniform surcharge $q_s = 20$ kPa is applied to the ground surface except the resting area of the footing. The surface of the footing is rough. Gravity is neglected here. Corresponding to the half and full domains, two simulation cases are denoted by Half and Full with 3,840,000 and 7,680,000 DEM particles. The problem sizes for the two cases are summarized in Table 7.

The RVE packing is prepared with a confining stress of 20 kPa, following the same protocol with the same material properties adopted in Section 4.2.1. The configuration of the initial RVE packing is similar to that shown in Figure 7(A) but with an isotropic stress of 20 kPa and a void ratio (2D) of 0.180 (a medium dense packing). It is worth noting that both RVE packings generated by *SudoDEM* and *GoDEM*, respectively, have almost the same configuration according to the preparation protocol depicted in Section 4.2.1, which ensures almost the same initial microstructure for two computing framework.

As for the hardware platform, the CPU-based MPM×DEM program runs on a cluster node with two Intel Xeon E7-2670 v3 (12 physical cores each, 2.3 GHz) and 128 GB DDR4-2133 RAM, while the GPU-based MPM×DEM program runs on an Nvidia RTX 2080 Ti GPU (11 GB GDDR6). Note that even though 44 logical cores are used in the simulation with the CPU-based MPM×DEM program, the simulation is still time-consuming. Hence, the CPU-based MPM×DEM only conducts the Half case simulation, while the GPU-based MPM×DEM simulates the two cases. In the simulation, the loading velocity for the footing is set to linearly increase up to 0.1 m/s and maintain constant thereafter in order to alleviate the stress oscillation caused by the potential dynamic effect. Note that the selected loading velocity is sufficiently small to loosely maintain a quasi-static condition but adequately large to ensure a feasible computational cost for the CPU-based MPM×DEM computation. The whole computation is terminated once the target settlement $d = 0.6$ m is reached.

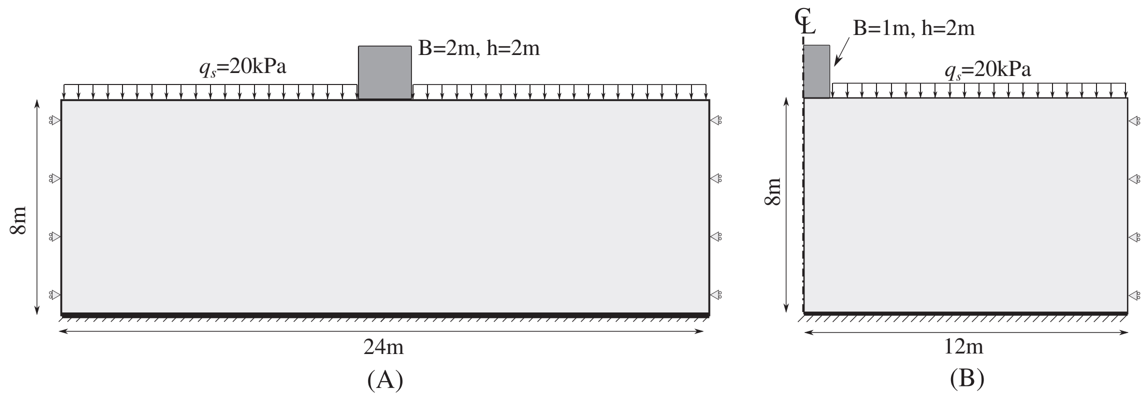

FIGURE 15 Geometric setup for rigid footing problem: (A) Full domain and (B) half domain

TABLE 7 Problem sizes for the two simulation cases

Case	Dimension (width*height)	Material point #	DEM particle #
Half	12*8	9600	3,840,000
Full	24*8	19,200	7,680,000

5.3 | Results and discussion

5.3.1 | Mechanical responses

The bearing capacity for the footing is calculated by dividing the total reaction force acting on the footing by its width. The variation of bearing capacity with the settlement of the footing is shown in Figure 16. As expected, CPU and GPU computing show almost identical results in the half domain case: the resistant pressures quickly build up, reach their peak $p_{peak} \approx 186$ kPa at $d = 0.12$ m before a mild drop. The resistant pressure for the full domain case is slightly smaller than the cases of half domain.

It is instructive to investigate the deformation pattern for these cases. As shown in Figures 17 and 18, which respectively depict the contour of displacement $\boldsymbol{u}$ and deviatoric strain ϵ_q, all three cases preserves a general shear failure pattern. For the Half domain cases, CPU and GPU based MPM×DEM once again offers almost identical results in terms of the failure pattern. As can be seen from Figure 18, three shear bands emerge within the soil domain. Two straight shear bands originate from the corner of the footing whereas the other curved one (main slip surface) arises from the intersection

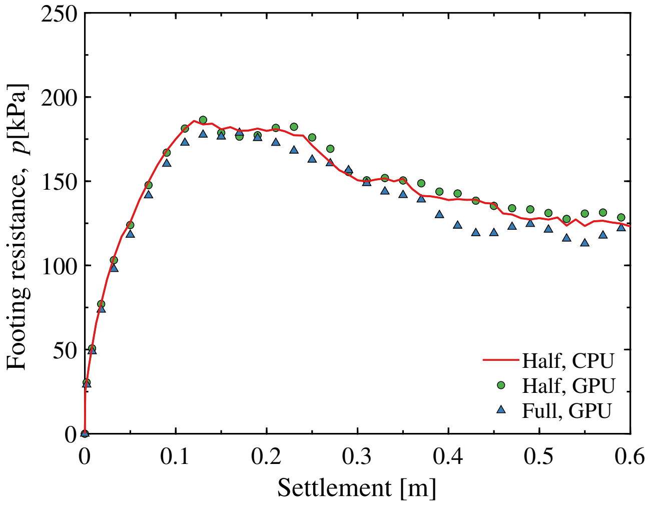

FIGURE 16 Resistant pressure acting on the footing [Colour figure can be viewed at wileyonlinelibrary.com]

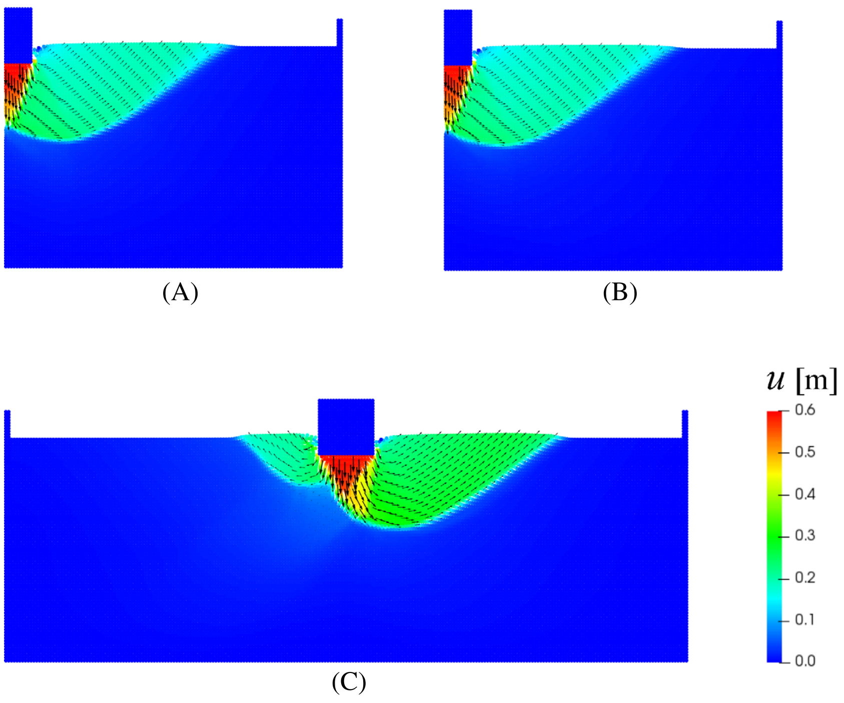

FIGURE 17 Displacement contours for footing problem: (A) Half domain, CPU; (B) half domain, GPU; and (C) full domain, GPU [Colour figure can be viewed at wileyonlinelibrary.com]

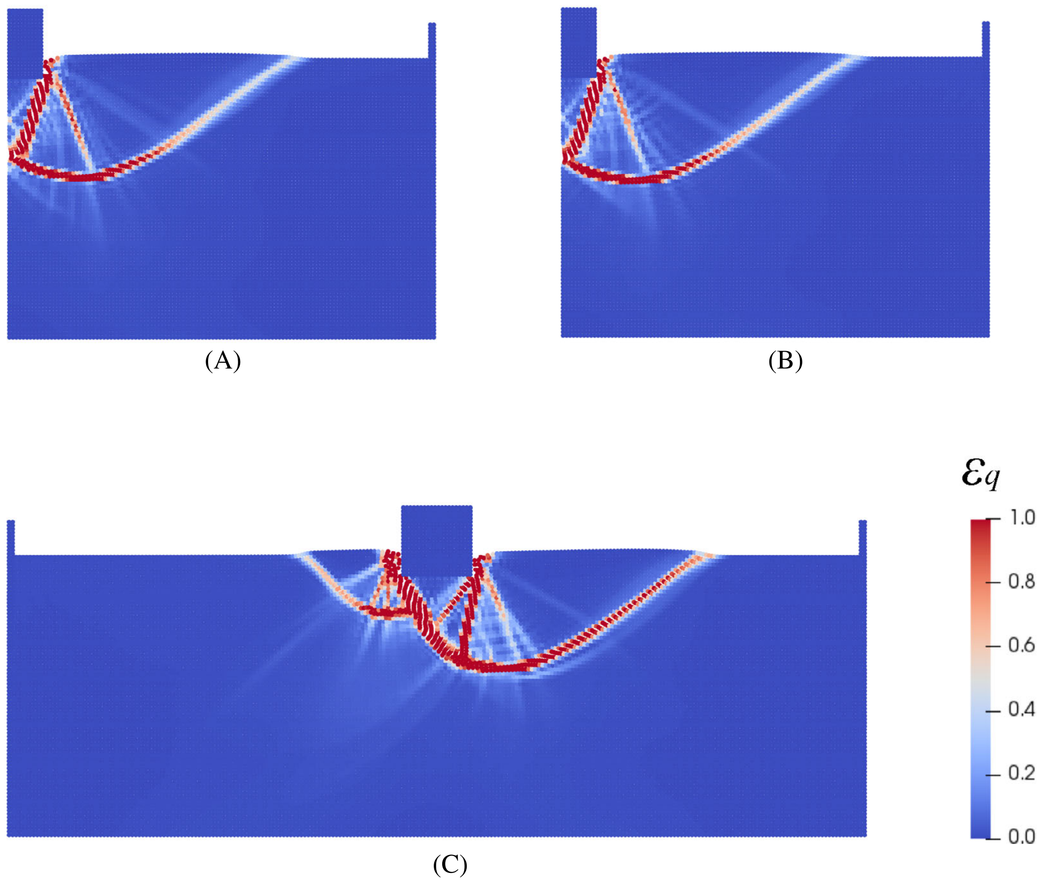

FIGURE 18 Deviatoric strain contours for footing problem: (A) Half domain, CPU; (B) half domain, GPU; and (C) full domain, GPU [Colour figure can be viewed at wileyonlinelibrary.com]

of one shear band and center line, and propagates toward the ground surface. The soil underneath the footing is mobilized downward and in turn pushes its surrounding soil aside along the main slip surface 17. As for the full domain case, it is interesting to observe that the final failure pattern is nonsymmetric. Such asymmetric pattern probably originates from the initial anisotropy of the RVE packing (though mild) that results in asymmetric configuration with respect to the central line at the particle scale.

5.3.2 | Computational efficiency

The elapsed (wall-clock) times for the three simulations are compared in Table 8 with both software and hardware configurations presented. For the half domain cases, the coupling program of NairnMPM and *GoDEM* takes only 16.6 min, which is significantly faster than the coupling program of NairnMPM and *SudoDEM*. The latter takes 25 h and 11.7 min to complete the simulation, even though it runs on a high compute end node (from HPC clusters at HKUST) with

TABLE 8 Computational time for running 6500 MPM steps

MPM×DEM	Hardware		Case	Elapsed time
NairnMPM	CPU	2x Intel Xeon E5-2670 v3 (2.3 GHz)[a]	Half	25 h 11.7 min
SudoDEM	Memory	128GB DDR4 RAM		
NairnMPM	GPU	1x Nvidia GeForce RTX 2080 Ti	Half	16.6 min
GoDEM	Memory	11GB GDDR6	Full	33.5 min

[a]24 physical cores (48 logical cores available) in total for two CPUs, but 44 logical cores are used in the simulation.

44 parallel threads. Notably, the proposed parallelism framework *GoDEM* elevates the performance of MPM×DEM coupling approximately 91 times faster, suggesting that the proposed framework can be successfully and efficiently applied to solving real engineering-scale problems. Moreover, for the two GPU simulations, the elapsed time of the full case is almost double that of the half domain case, implying that the proposed framework has an excellent scalability without showing appreciable degradation of computational efficiency. This feature can also be observed during the pure RVE running tests in Figure 13. Besides, it is worth noting that the simulation by the DEM solver dominates the running time (up to 90%) in the MPM×DEM coupling scheme.

6 | CONCLUDING REMARKS

We presented a novel and efficient GPU parallelism framework on discrete element modeling of representative volume elements (RVEs) of granular media, where all RVEs entirely run on a GPU without any interference from the host CPU during the course of a simulation, that is, thread-block-wise RVE modeling. Within the framework, the RVEs are parallelized at the thread-block level with implicit asynchronization for each other, thereby guaranteeing the inter-RVE independence that considerably promotes the parallel efficiency. Moreover, specific parallel algorithms of thread-block-wise RVEs are proposed to fully take the advantages of the parallel nature of the GPU, which are then implemented in the object-oriented program *GoDEM* using CUDA C++, and further benchmarked against the simulations by CPU code *SudoDEM* for different loading conditions, including uniaxial compression, simple shear, and biaxial compression tests. It is found that the single-precision *GoDEM* yields results well consistent with that from the double-precision *SudoDEM*, suggesting a sufficient accuracy of *GoDEM* in single-precision for RVE modeling. In the pure RVE parallelism test, the proposed implementation of *GoDEM* can achieve a saturated speedup of approximately 350 on an Nvidia GeForce RTX 2080 Ti GPU card with respect to the single-CPU-core *SudoDEM* on an Intel Core I7-6700 CPU, which shows a tremendous performance of a GPU with the proposed parallelism framework. Furthermore, a hierarchical coupling of MPM and DEM is proposed to simulate an engineering-scale problem. It demonstrates that a speedup of approximately 91 can be achieved by using the proposed framework, compared with the CPU program running on a cluster node (two Intel Xeon E5-2670 v3 CPUs) with 44 parallel threads. To sum up, the efficient GPU parallelism framework contributed in this work offers a novel pathway to considerably speed up the hierarchical multiscale modeling of granular media by coupling either FEM×DEM[29] or MPM×DEM.[34]

The present parallelism framework opens a number of exciting future opportunities. First, *GoDEM* can be readily extended to the three-dimensional without modification to the implementation; moreover, for simulations where particle shape does matter to the computational cost,[52,53] such as nonspherical particle shape[17] and particle crushing,[54] the presented framework can be readily adapted with only minor revisions of the CPU-based algorithms. For example, the authors developed an open-source code, *SudoDEM* (https://sudodem.github.io), for DEM modeling of nonspherical particles, which can be readily implemented in this framework. *GoDEM* can also be extended to accelerate multiphysics modeling, for example, modeling thermomehcanical responses of granular media.[55]

GoDEM is a generic thread-block-wise parallelism framework to accelerate hierarchical multiscale modeling, which is proposed to match the physical structure of thread-block computing units, for example, GPUs and TPUs. As for the efficiency, there are two major aspects of possible improvements on GPUs: (1) multi-GPU cards can be easily connected together with the help of unified memory to speed up the simulations for better efficiency; (2) warp divergence needs reducing for maximizing the utilization of warp schedulers, which is, however, nontrivial due to the conditional branches involved in the algorithm.

ACKNOWLEDGMENTS

This work was financially supported by the Hong Kong Scholars Program (2018), the National Natural Science Foundation of China (by Project No. 51679207, No. 51909095, and No. 11972030), Research Grants Council of Hong Kong (by GRF Project No. 16207319, TBRS Project No. T22-603/15N, and CRF Project No. C6012-15G). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the financial bodies.

ORCID

Shiwei Zhao https://orcid.org/0000-0002-3410-3935
Jidong Zhao https://orcid.org/0000-0002-6344-638X

REFERENCES

1. Guo N, Zhao J. The signature of shear-induced anisotropy in granular media. *Comput Geotech*. 2013;47:1-15.
2. Chen Q, Andrade JE, Samaniego E. AES for multiscale localization modeling in granular media. *Comput Methods Appl Mech Eng*. 2011;200(33-36):2473-2482.
3. Li X, Yu H-S. Particle-scale insight into deformation noncoaxiality of granular materials. *Int J Geomech*. 2015;15(4):04014061.
4. Ouadfel H, Rothenburg L. Stress–force–fabric'relationship for assemblies of ellipsoids. *Mech Mater*. 2001;33(4):201-221.
5. Nicot F, Darve F. The H-microdirectional model: accounting for a mesoscopic scale. *Mech Mater*. 2011;43(12):918-929.
6. Li XS, Dafalias YF. Anisotropic critical state theory: role of fabric. *J Eng Mech*. 2012;138(3):263-275.
7. Fonseca J, O'Sullivan C, Coop MR, Lee P. Quantifying the evolution of soil fabric during shearing using directional parameters. *Géotechnique*. 2013;63(6):487-499.
8. Herrmann HJ, Hovi J-P, Luding S. *Physics of Dry Granular Media*. Vol 350. Berlin, Germany: Springer Science & Business Media; 2013.
9. American Association for the Advancement of Science. So much more to know. *Science*. 2005;309(5731):78-102.
10. Gao Z, Zhao J, Li X-S, Dafalias YF. A critical state sand plasticity model accounting for fabric evolution. *Int J Numer Anal Methods Geomech*. 2014;38(4):370-390.
11. O'Sullivan C. Particle-based discrete element modeling: geomechanics perspective. *Int J Geomech*. 2011;11(6):449-464.
12. Cundall PA, Strack OD. A discrete numerical model for granular assemblies. *Géotechnique*. 1979;29(1):47-65.
13. Zhao S, Evans TM, Zhou X. Shear-induced anisotropy of granular materials with rolling resistance and particle shape effects. *Int J Solids Struct*. 2018;150:268-281.
14. Zhao S, Zhao J, Guo N. Universality of internal structure characteristics in granular media under shear. *Phys Rev E*. 2020;101(1):012906.
15. Lai Z, Chen Q, Huang L. Fourier series-based discrete element method for computational mechanics of irregular-shaped particles. *Comput Methods Appl Mech Eng*. 2020;362:112873.
16. Shi X, Nie J, Zhao J, Gao Y. A homogenization equation for the small strain stiffness of gap-graded granular materials. *Comput Geotech*. 2020;121:103440.
17. Zhao S, Zhao J. A poly-superellipsoid-based approach on particle morphology for DEM modeling of granular media. *Int J Numer Anal Methods Geomech*. 2019;43(13):2147-2169.
18. Feng Y, Zhao T, Kato J, Zhou W. Towards stochastic discrete element modelling of spherical particles with surface roughness: a normal interaction law. *Comput Methods Appl Mech Eng*. 2017;315:247-272.
19. Zhao J, Shan T. Coupled CFD–DEM simulation of fluid–particle interaction in geomechanics. *Powder Technol*. 2013;239:248-258.
20. Galindo-Torres S. A coupled discrete element lattice Boltzmann method for the simulation of fluid–solid interaction with particles of general shapes. *Comput Methods Appl Mech Eng*. 2013;265:107-119.
21. Kozicki J, Donze FV. A new open-source software developed for numerical simulations using discrete modeling methods. *Comput Methods Appl Mech Eng*. 2008;197(49-50):4429-4443.
22. Berger R, Kloss C, Kohlmeyer A, Pirker S. Hybrid parallelization of the liggghts open-source dem code. *Powder Technol*. 2015;278:234-247.
23. Nvidia Corporation CUDA *C++ Programming Guide, Version 10.1*, Santa Clara, CA: NVIDIA Corporation, (2019). https://docs.nvidia.com/.
24. Govender N, Wilke DN, Kok S, Els R. Development of a convex polyhedral discrete element simulation framework for NVIDIA Kepler based GPUs. *J Comput Appl Math*. 2014;270:386-400.
25. Gan J, Zhou Z, Yu A, GPU-based A. DEM approach for modelling of particulate systems. *Powder Technol*. 2016;301:1172-1182.
26. Spellings M, Marson RL, Anderson JA, Glotzer SC. GPU accelerated Discrete Element Method (DEM) molecular dynamics for conservative faceted particle simulations. *J Comput Phys*. 2017;334:460-467.
27. Kelly C, Olsen N, Negrut D. Billion degree of freedom granular dynamics simulation on commodity hardware via heterogeneous data-type representation. *Multibody Syst Dyn*. 2020;1-25. https://doi.org/10.1007/s11044-020-09749-7.
28. Chrono Project Chrono: an open source framework for the physics-based simulation of dynamic systems; (2020). http://projectchrono.org.
29. Guo N, Zhao J. A coupled FEM/DEM approach for hierarchical multiscale modelling of granular media. *Int J Numer Methods Eng*. 2014;99(11):789-818.
30. Liu Y, Sun W, Yuan Z, Fish J. A nonlocal multiscale discrete-continuum model for predicting mechanical behavior of granular materials. *Int J Numer Methods Eng*. 2016;106(2):129-160.
31. Guo N, Zhao J. Parallel hierarchical multiscale modelling of hydro-mechanical problems for saturated granular soils. *Comput Methods Appl Mech Eng*. 2016;305:37-61.
32. Guo N, Zhao J. 3D multiscale modeling of strain localization in granular media. *Comput Geotech*. 2016;80:360-372.
33. Argilaga A, Desrues J, Dal Pont S, Combe G, Caillerie D. FEM× DEM multiscale modeling: model performance enhancement from Newton strategy to element loop parallelization. *Int J Numer Methods Eng*. 2018;114(1):47-65.
34. Liang W, Zhao J. Multiscale modeling of large deformation in geomechanics. *Int J Numer Anal Methods Geomech*. 2019;43(5):1080-1114.
35. Wu H, Papazoglou A, Viggiani G, Dano C, Zhao J. Compaction bands in tuffeau de maastricht: insights from X-ray tomography and multiscale modeling. *Acta Geotech*. 2020;15(1):39-55.
36. Munjiza A, Lei Z, Divic V, Peros B. Fracture and fragmentation of thin shells using the combined finite–discrete element method. *Int J Numer Methods Eng*. 2013;95(6):478-498.
37. Fukuda D, Mohammadnejad M, Liu H, et al. Development of a gpgpu-parallelized hybrid finite-discrete element method for modeling rock fracture. *Int J Numer Anal Methods Geomech*. 2019;43(10):1797-1824.

38. Feyel F. A multilevel finite element method (fe2) to describe the response of highly non-linear structures using generalized continua. *Comput Methods Appl Mech Eng*. 2003;192(28-30):3233-3244.
39. Verhoosel CV, Remmers JJ, Gutiérrez MA, De Borst R. Computational homogenization for adhesive and cohesive failure in quasi-brittle solids. *Int J Numer Methods Eng*. 2010;83(8-9):1155-1179.
40. Rapaport DC, Rapaport DCR. *The Art of Molecular Dynamics Simulation*. Cambridge, MA: Cambridge University Press; 2004.
41. Fincham D. Leapfrog rotational algorithms. *Mol Simul*. 1992;8(3-5):165-178.
42. Zhao S, Zhang N, Zhou X, Zhang L. Particle shape effects on fabric of granular random packing. *Powder Technol*. 2017;310:175-186.
43. Thornton C. Numerical simulations of deviatoric shear deformation of granular media. *Géotechnique*. 2000;50(1):43-53.
44. Yang W, Zhou Z, Pinson D, Yu A. Periodic boundary conditions for discrete element method simulation of particle flow in cylindrical vessels. *Ind Eng Chem Res*. 2014;53(19):8245-8256.
45. Radjai F. Multi-periodic boundary conditions and the contact dynamics method. *Comptes Rendus Mécanique*. 2018;346(3):263-277.
46. Christoffersen J, Mehrabadi MM, Nemat-Nasser S. A micromechanical description of granular material behavior. *J Appl Mech*. 1981;48:339-344.
47. Nishiura D, Sakaguchi H. Parallel-vector algorithms for particle simulations on shared-memory multiprocessors. *J Comput Phys*. 2011;230(5):1923-1938.
48. Kirk DB, Wen-Mei WH. *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA: Morgan kaufmann; 2016.
49. Zhao S, Evans T, Zhou X. Effects of curvature-related DEM contact model on the macro-and micro-mechanical behaviours of granular soils. *Géotechnique*. 2018;68(12):1085-1098.
50. Zhao S, Zhou X. Effects of particle asphericity on the macro-and micro-mechanical behaviors of granular assemblies. *Granul Matter*. 2017;19(2):38.
51. Nairn JA. Material point method (NairnMPM) and finite element analysis (NairnFEA) open-source software; (2011). http://osupdocs.forestry.oregonstate.edu/index.php/NairnMPM.
52. Kawamoto R, Andò E, Viggiani G, Andrade JE. All you need is shape: predicting shear banding in sand with ls-dem. *J Mech Phys Solids*. 2018;111:375-392.
53. Xiao Y, Long L, Matthew Evans T, Zhou H, Liu H, Stuedlein AW. Effect of particle shape on stress-dilatancy responses of medium-dense sands. *J Geotech Geoenviron*. 2019;145(2):04018105.
54. Zhu F, Zhao J. A peridynamic investigation on crushing of sand particles. *Géotechnique*. 2019;69(6):526-540.
55. Zhao S, Zhao J, Lai Y. Multiscale modeling of thermo-mechanical responses of granular materials: a hierarchical continuum–discrete coupling approach. *Comput Methods Appl Mech Eng*. 2020;367:113100.

How to cite this article: Zhao S, Zhao J, Liang W. A thread-block-wise computational framework for large-scale hierarchical continuum-discrete modeling of granular media. *Int J Numer Methods Eng*. 2021;122:579–608. https://doi.org/10.1002/nme.6549