Comp. Part. Mech.
DOI 10.1007/s40571-016-0116-5

Parallel peridynamics–SPH simulation of explosion induced soil fragmentation by using OpenMP

Houfu Fan[1] · **Shaofan Li**[1]

Received: 4 September 2015 / Revised: 3 May 2016 / Accepted: 23 May 2016

Abstract In this work, we use the OpenMP-based shared-memory parallel programming to implement the recently developed coupling method of state-based peridynamics and smoothed particle hydrodynamics (PD-SPH), and we then employ the program to simulate dynamic soil fragmentation induced by the explosion of the buried explosives. The paper offers detailed technical description and discussion on the PD-SHP coupling algorithm and how to use the OpenMP shared-memory programming to implement such large-scale computation in a desktop environment, with an example to illustrate the basic computing principle and the parallel algorithm structure. In specific, the paper provides a complete OpenMP parallel algorithm for the PD-SPH scheme with the programming and parallelization details. Numerical examples of soil fragmentation caused by the buried explosives are also presented. Results show that the simulation carried out by the OpenMP parallel code is much faster than that by the corresponding serial computer code.

Keywords Buried explosive · OpenMP · Parallel computation · Peridynamics · SPH · Soil fragmentation

1 Introduction

Simulations of the soil fragmentation induced by the buried explosives are of great challenge in computational geomechanics, owing to the complexities in representing the massive fragments generated during the explosion process by using conventional mesh based methods. To correctly characterize the interaction between the soil and the explosive, the particle method may be a better approach. Unfortunately, some of the weak form-based particle methods are computationally expensive, and some of the strong form-based particle methods, e.g., SPH, have issues with accuracy and convergence [1].

The state-based peridynamics (PD) and the smoothed particle hydrodynamics (SPH) are two popular particle methods extensively used in the current researches and developments. PD is a computational non-local continuum theory and formulation proposed by Silling [2]. In addition to the fact that the PD method is efficient, convergent, and accurate, there is one unique feature of the PD method—it can seamlessly incorporate any macroscale constitutive relations in classical continuum mechanics framework. The PD method has been successfully employed in simulations of many fracture or damage processes. For instance, Foster et al. (2010) investigated a viscoplastic bar under impact [3]; Tuniki (2012) employed state-based peridynamics to predict the fracture processes of concrete [4], and Lai et al. (2015) simulated the fragmentation of geomaterial induced by impluse loads using the state-based peridynamics [5]. The SPH method is a particle method that is designed to solve hydrodynamics problems, which can be utilized to simulate explosion [6]. However, the SPH method has some deficiencies, such as low accuracy [7,8], tensile instability [9], and difficulties in enforcing the essential boundary conditions [10]. In [11], the present authors developed the PD-SPH coupling scheme, which incorporates the advantages of both the PD and SPH methods, such that the theoretical and numerical issues in soil explosions are carefully addressed.

Parallel computing is a type of computations that makes use of multiple processing units simultaneously to solve a large-scale problem, which is usually divided into many smaller parts, and each part is calculated by a computing

✉ Shaofan Li
shaofan@berkeley.edu

[1] Department of Civil and Environmental Engineering, University of California, Berkeley, CA 94720, USA

Published online: 09 June 2016

processing unit. Parallel computers can be classified with two independent dimensions of *Instruction Stream* and *Data Stream*, each of which can be either *Single* or *Multiple* [12]. For a typically MIMD (multiple instruction, multiple data) multicore processor, several parallel programming models are in common use, such as shared memory (without threads), threads, distributed memory (message passing), data parallel, hybrid, single program multiple data and multiple program multiple data. In this work, we are focusing on the a specific implementation of the thread model—the OpenMP, where a single program can have more than one command flow, all of which operate together in the shared-memory system. The efficiency of a computer program can be significantly increased on a multithreaded processor if it is designed to make the different threads cooperate well with each other.

Traditionally, parallel programming is accomplished by the message-passing interface (MPI) library [13], which is a language-independent communications protocol used for programming parallel computers. It is a portable, widely available and accepted standard for parallel programming. However, writing parallel code using MPI can be very challenging, as it requires the data structures be properly partitioned to balance the load and communications between different processors. There is no incremental path in parallelizing a program by MPI—one cannot write a serial code and parallelizes the sections as needed. For the dynamic soil fragmentation problem with moderate size, we usually have relatively small datasets that only require moderate computer memory but fast computation. Thus, it may be advantageous to employ OpenMP technique to speed up the computation, because the OpenMP programming model is designed and suitable for multithreaded programming environments, in which the data structures are shared among threads, and thus there is no need to copy data between execution contexts [14]. The OpenMP programming model can help developers make the parallel applications much easier while retaining the overall look of serial programming. In particular, the OpenMP parallel technique discussed here may even be used in some desktop computers and laptops in daily academic and research development environment to avoid the cumbersome in developing a full parallel MPI program.

Moreover, with most distributed memory platforms nowadays consisting of symmetric multi-processor (SMP) or non-uniform momery access (NUMA) nodes, it makes a lot of sense to use OpenMP with MPI. SMP means a platform of shared-memory hardware architecture where multiple processors share a single address space and have equal access to all resources. NUMA is often made by the linking of two or more SMP machines, which do not have equal access to all memories. In fact, OpenMP and MPI can perfectly work together. OpenMP feeds the cores on each node and MPI communicates between the nodes. This is what is called hybrid programming.

In this work, the OpenMP parallel programming is employed to implement a PD-SPH coupling method. The detailed PD-SPH coupling scheme, the PD constitutive modeling of the soil medium, and validation of the PR soil model have been discussed previously in [11]. In this paper, we are focusing on providing the programming details of the OpenMP parallel computation of the PD-SPH coupling scheme and possible improvements regarding the performance of the parallelism. The paper is organized as follows. In Sect. 2, PD, SPH methods, and their coupling scheme are reviewed. The technical details of OpenMP programming and implementation of the PD-SPH coupling scheme are discussed in Sect. 3. In Sect. 4, we present the numerical simulation results that are obtained from the numerical tests to evaluate the performance of the parallel PD-SPH coupling computer code. The presentation is ended with discussions in Sect. 5.

2 Overview of the PD-SPH coupling scheme

In this section, an overview of the fundamentals of the state-based peridynamics, the smoothed particle hydrodynamics (SPH), and their coupling scheme are discussed.

2.1 Peridynamics theory and formulations

The state-based peridynamics theory was first introduced by Silling [2], which is a non-local continuum theory of solid mechanics. As shown in Fig. 1, we are considering a continuum domain Ω_0 that is discretized into a set of material particles $\mathbf{X}_A$ with associated volume V_A^0 and mass density ρ_A^0, where $A = 1, 2, ..., \infty$ is the particle index. Note that in this paper, we choose the initial configuration as the material configuration.

A material particle $\mathbf{X}_A$ only interacts with any particle $\mathbf{X}_B$ within a local region, called the horizon $\delta_{\mathbf{X}_A}$. The particles within this local region form a family $\mathcal{H}_{\mathbf{X}_A}$. The relative position vector pointing from particle $\mathbf{X}_A$ to $\mathbf{X}_B$ in the reference configuration is called a bond, which shall be denoted as

$$\mathbf{X}_{A\to B} := \mathbf{X}_B - \mathbf{X}_A \ . \tag{1}$$

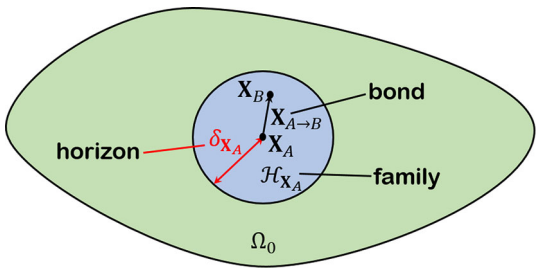

Fig. 1 The reference configuration of a continuum domain Ω_0 in the theory of state-based peridynamics

Subjected to certain motion or deformation $\boldsymbol{\chi}$, the continuum body deforms and the bond $\mathbf{X}_{A\to B}$ in the reference configuration becomes

$$\mathbf{x}_{A\to B} := \mathbf{x}_B - \mathbf{x}_A = \underline{\mathbf{Y}}(\mathbf{X}_{A\to B}) \tag{2}$$

where $\underline{\mathbf{Y}}$ is a non-linear quantity called the deformation state, similar to a tensor in classical continuum mechanics. Here, the deformation state $\underline{\mathbf{Y}}$ maps an undeformed bond $\mathbf{X}_{A\to B}$ to a deformed bond $\mathbf{x}_{A\to B}$.

In the peridynamics theory, the balance of the linear momentum is expressed in the form,

$$\rho_A^0 \ddot{\mathbf{u}} = \int_{\mathcal{H}_{\mathbf{X}_A}} \left[\underline{\mathbf{T}}(\mathbf{X}_A, \mathbf{X}_{A\to B}) - \underline{\mathbf{T}}(\mathbf{X}_B, \mathbf{X}_{B\to A})\right] \mathrm{d}V_{\mathbf{X}_B} + \rho_A^0 \mathbf{b}, \tag{3}$$

where $\underline{\mathbf{T}}$ is the force state; $\mathbf{u}$ is the displacement, and $\mathbf{b}$ is the external body force.

The force state can be expressed in terms of the first Piola–Kirchhoff (PK1) stress as

$$\underline{\mathbf{T}}(\mathbf{X}_A, \mathbf{X}_{A\to B}) = \omega(|\mathbf{X}_{A\to B}|)\mathbf{P}_{\mathbf{X}_A}\mathbf{K}_{\mathbf{X}_A}^{-1} \cdot \mathbf{X}_{A\to B}, \tag{4}$$

where $\omega(|\mathbf{X}_{A\to B}|)$ is a positive scalar influence function, $|\mathbf{X}_{A\to B}|$, and $\mathbf{K}_{\mathbf{X}_A}$ is the reference shape tensor, defined as

$$\mathbf{K}_{\mathbf{X}_A} = \int_{\mathcal{H}_{\mathbf{X}_A}} \omega(|\mathbf{X}_{A\to B}|)\mathbf{X}_{A\to B} \otimes \mathbf{X}_{A\to B} dV_{\mathbf{X}_B} . \tag{5}$$

There is a corresponding deformed shape tensor $\mathbf{N}_{\mathbf{X}_A}$, defined as

$$\mathbf{N}_{\mathbf{X}_A} := \int_{\mathcal{H}_{\mathbf{X}_A}} \omega(|\mathbf{X}_{A\to B}|)\mathbf{x}_{A\to B} \otimes \mathbf{X}_{A\to B} dV_{\mathbf{X}_B} . \tag{6}$$

By assuming that

$$\mathbf{x}_{A\to B} = \underline{\mathbf{Y}}(\mathbf{X}_{A\to B}) = \mathbf{F}_{\mathbf{X}_A} \cdot \mathbf{X}_{A\to B}, \tag{7}$$

where $\mathbf{F}_{\mathbf{X}_A}$ is a second-order tensor, which can be viewed as the approximated deformation gradient at particle A, one can find that

$$\begin{aligned}\mathbf{N}_{\mathbf{X}_A} &= \mathbf{F}_{\mathbf{X}_A} \cdot \left[\sum_{\mathbf{X}_B \in \mathcal{H}_{\mathbf{X}_A}} \omega(|\mathbf{X}_{A\to B}|)\mathbf{X}_{A\to B} \otimes \mathbf{X}_{A\to B} V_B^0\right] \\ &= \mathbf{F}_{\mathbf{X}_A} \cdot \mathbf{K}_{\mathbf{X}_A} .\end{aligned} \tag{8}$$

That is

$$\mathbf{F}_{\mathbf{X}_A} = \mathbf{N}_{\mathbf{X}_A} \cdot \mathbf{K}_{\mathbf{X}_A}^{-1} . \tag{9}$$

In the Peridynamics and SPH coupling region, in order to couple them together, we sometimes need to treat an SPH particle $\mathbf{X}_B \in \mathcal{H}_{\mathbf{X}_A}$ as a peridynamics particle. In fact, one can directly treat $\mathbf{X}_B$ as a peridynamics particle by constructing the same influence function $\omega(|\mathbf{X}_{A\to B}|)$, and the shape tensor $\mathbf{K}_{X_B}, \mathbf{N}_{X_B}$. By combining Eqs. (3)–(9), we can express the force acting on a peridynamics particle as

$$\begin{aligned}\mathbf{f}_A = \sum_{\mathbf{X}_B \in \mathcal{H}_{\mathbf{X}_A}} \Big[&\omega(|\mathbf{X}_{A\to B}|)\mathbf{P}_{\mathbf{X}_A}\mathbf{K}_{\mathbf{X}_A}^{-1} \cdot \mathbf{X}_{A\to B} \\ &-\omega(|\mathbf{X}_{B\to A}|)\mathbf{P}_{\mathbf{X}_B}\mathbf{K}_{\mathbf{X}_B}^{-1} \cdot \mathbf{X}_{B\to A}\Big] V_B^0 .\end{aligned} \tag{10}$$

For the SPH particle, the only quantity unknown is the first Piola-Kirchhoff stress tensor (PK1) $\mathbf{P}_{\mathbf{X}_A}$, which can be obtained by

$$\mathbf{P}_{\mathbf{X}_B} = -J_B p_B \mathbf{F}_{\mathbf{X}_B}^{-T}, \tag{11}$$

where p_B is the pressure, and $\mathbf{F}_{\mathbf{X}_B}$ is the deformation gradient at the particle $\mathbf{X}_B$ and $J_B = \det[\mathbf{F}_{\mathbf{X}_B}]$.

2.2 Basic formulation of SPH

The SPH method was first developed for fluid hydrodynamics problems. There are two key ingredients in SPH methodology: Kernel approximation and Particle approximation [16].

In the SPH method, a continuous field $f(\mathbf{x})$ is approximated as

$$\langle f(\mathbf{x})\rangle = \int_\Omega f(\mathbf{y}) W(|\mathbf{x}-\mathbf{y}|)\mathrm{d}V_{\mathbf{y}}, \tag{12}$$

where $\langle\rangle$ is the kernel approximation operator [17], and W is the so-called kernel function. Using the smoothing function, the gradient of $f(\mathbf{x})$, i.e., $\nabla f(\mathbf{x})$, can be written as

$$\langle \nabla f(\mathbf{x})\rangle = -\int_\Omega f(\mathbf{y}) \nabla_{\mathbf{x}} W(|\mathbf{x}-\mathbf{y}|)\mathrm{d}V_{\mathbf{y}}, \tag{13}$$

where the minus sign results from the integration by parts, which is standard in SPH formulation [18]. The discretized form of Eqs. (12) and (13) are

$$\langle f(\mathbf{x}_A)\rangle = \sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} f(\mathbf{x}_B) W(|\mathbf{x}_A - \mathbf{x}_B|) V_B, \tag{14}$$

and

$$\langle \nabla f(\mathbf{x}_A)\rangle = -\sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} f(\mathbf{x}_B) \nabla_{\mathbf{x}_A} W(|\mathbf{x}_A - \mathbf{x}_B|) V_B, \tag{15}$$

where $\mathcal{S}_{\mathbf{x}_A}$ represents the supporting (influence) domain of particle $\mathbf{x}_A$, similar to the definition of horizon $\mathcal{H}_{\mathbf{X}_A}$ in the state-based peridynamics.

In order to represent constant and linear fields correctly, the following corrected gradient operator is usually employed:

$$\overline{\nabla}_{\mathbf{x}_A} W(|\mathbf{x}_A - \mathbf{x}_B|) = -\mathbf{M}_{\mathbf{x}_A}^{-1} \nabla_{\mathbf{x}_A} W(|\mathbf{x}_A - \mathbf{x}_B|), \tag{16}$$

where the matrix $\mathbf{M}_{\mathbf{x}_A}$ is given as

$$\mathbf{M}_{\mathbf{x}_A} = \sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} (\mathbf{x}_B - \mathbf{x}_A) \otimes \nabla_{\mathbf{x}_A} W(|\mathbf{x}_A - \mathbf{x}_B|) V_B \, . \tag{17}$$

With these formulas, we can directly use the discretized SPH equations of motion to simulate the explosive as

$$\frac{D\rho_A}{Dt} = \sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} m_B (\mathbf{v}_B - \mathbf{v}_A) \cdot \overline{\nabla}_{\mathbf{x}_B} W(|\mathbf{x}_B - \mathbf{x}_A|), \tag{18}$$

$$\frac{D\mathbf{v}_A}{Dt} = -\sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} m_B \left(\frac{p_A}{\rho_A^2} + \frac{p_B}{\rho_B^2} + \Pi_{AB}\right) \overline{\nabla}_{\mathbf{x}_B} W(|\mathbf{x}_B - \mathbf{x}_A|), \tag{19}$$

$$\frac{De_A}{Dt} = \frac{1}{2} \sum_{\mathbf{x}_B \in \mathcal{S}_{\mathbf{x}_A}} m_B \left(\frac{p_A}{\rho_A^2} + \frac{p_B}{\rho_B^2} + \Pi_{AB}\right) (\mathbf{v}_A - \mathbf{v}_B) \cdot \overline{\nabla}_{\mathbf{x}_B} W(|\mathbf{x}_B - \mathbf{x}_A|), \tag{20}$$

$$\frac{D\mathbf{x}_A}{Dt} = \mathbf{v}_A, \tag{21}$$

where ρ, $\mathbf{v}$, e, p are the density, velocity, internal energy, and pressure at the corresponding particle. $\frac{D()}{Dt}$ represents the time derivative of the quantity in the bracket. Π_{AB} is the standard Monaghan viscosity [19]. For the detailed derivation of these equations, the readers may consult [1,6]. The pressure of the high explosive (TNT) is obtained from the following equation of state of the explosive gas model for TNT:

$$p = (\gamma - 1)\rho e, \tag{22}$$

where the factor $\gamma = 1.4$ and initial energy $e_0 = 4.69 \times 10^6 J$ are taken from [16].

In SPH–Peridynamics coupling region, we may need to "treat" a peridynamics particle as a SPH particle in order to couple it with the rest of true SPH particles. Thus when we correspond a peridynamics particle to a SPH particle, the corresponding SPH quantities of the peridynamics particle, such as the density ρ_B, velocity $\mathbf{v}_B$, and pressure p_B are needed, and they have to be identified in order to employ the SPH governing Eqs. (18)–(21)). In the coupling procedure, the particle's SPH velocity $\mathbf{v}_B$ is always available. Moreover, other related variables may also be calculated, i.e, the density can be calculated as

$$\rho_B = \frac{1}{J_B} \rho_B^0, \tag{23}$$

and the pressure can be obtained as

$$p_B = -\frac{1}{3} \mathrm{tr}(\boldsymbol{\sigma}_B) = -\frac{1}{3J_B} \mathrm{tr}(\mathbf{P}_{\mathbf{X}_B} \mathbf{F}_{\mathbf{X}_B}^T), \tag{24}$$

where ρ_B^0 is the initial density at particle $\mathbf{X}_B$.

3 OpenMP Parallel programming of the PD-SPH coupling scheme

3.1 Introduction to the OpenMP

OpenMP is an application programming interface (API) for explicit and portable shared-memory multithreaded parallel programming in C, C++, and Fortran [14,20,21]. In multithreaded programming environment, an application is broken into subtasks or threads, which run in parallel or sequentially. OpenMP is not a new programming language, but the version that is used in this work contains a set of compiler directives and callable runtime library routines that "decorate" (extend) C, C++, or Fortran, to express shared-memory parallelization. OpenMP simplifies the complex task of code parallelization, allowing even novices to move directly from serial programming styles to parallel programming. A developer who is familiar with a language (such as C, C++, or Fortran) only needs to learn the set of compiler directives. These directives, which tells the compiler which part of code to parallelize and how to do it, are added to the code without altering the logical behavior of the serial program. The whole multithreaded task is then handled by the compiler.

As shown in Fig. 2, OpenMP makes use of a parallel design pattern called the fork-join model. An OpenMP program starts with the master thread running sequentially, which branches into a specific number of threads in response to the OpenMP directives that carry out the corresponding work concurrently. After finishing the work in the parallel region, these threads merge together with the only master thread left, running sequentially again. By using the OpenMP directives, a serial program can be gradually changed into a parallel one. The OpenMP directives are in the form of comments in C, C++, or Fortran, such that without turning on the OpenMP parallelizing feature of a compiler, the resulting OpenMP-based parallel program is completely equivalent to the original serial one.

For most OpenMP directives, a structured block of code is marked for parallelization with an entry point at the beginning and an exit point at the end. The number of threads involved in the parallel region can be left default as the environment variable or specified by a runtime function that is called within the program. This enables the change of the number of threads created within different parallel regions. In addition, nested parallel regions are allowed, such that each

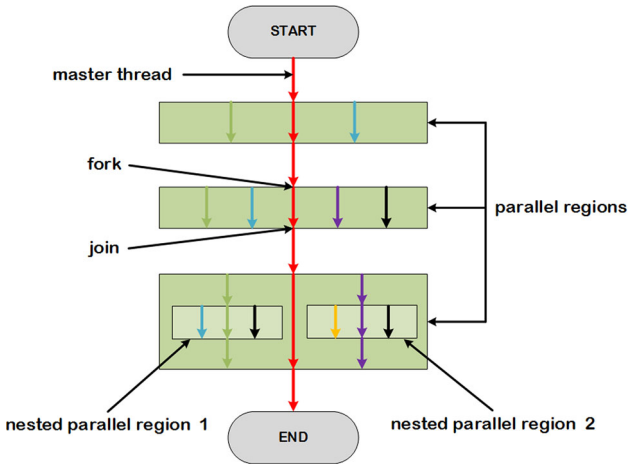

Fig. 2 Fork-join model: a way of setting up and executing parallel programs such that the sequential execution branches off ("fork") in parallel at a specific point in the program, to merge together ("join") at another subsequent point and resume sequential execution. Parallel regions can be nested

of the created thread in the original parallel region can fork a team of threads in the parallel region of their own, with the specific thread itself being the corresponding master thread.

3.2 How to parallel the PD-SPH coupling computation code

In this part, we shall discuss the detailed implementation of the PD-SPH coupling scheme by using OpenMP parallel computation. To do so, we first implement a serial program of the PD-SPH coupling scheme, and then we identify the most time-consuming parts of the serial program. Subsequently, OpenMP directives are gradually applied to, wherever possible, the corresponding source codes of these parts in the serial program, turning it to a parallel one. The general computational structure of the PD-SPH coupling scheme is shown in Fig. 3, based on which the serial code is first developed. In short, the computational structure can be categorized into three parts: Preprocess, Initialize, and Solve.

In the first step (Preprocess), reading the input file cannot be parallelized, because the file handle can only be assigned to one thread at any time. Allocating the calculation matrices can be parallelized by using the *section* directive [14], but it will not make much differences compared to a serial code, because there are only a few number of matrices to be allocated. In our framework of the computational particle mechanics, the particle positions, volumes, and horizon sizes are all obtained from that of a typical FEM mesh—node coordinates and element connectivity [23]. Computing the particle volume and horizon size can be parallelized by using the OpenMP directives *parallel* and *do*. However, again, it will not help much in reducing the total computational time, given that this part will only be executed once and there is only a single loop over all the elements involved in the execution. Constructing the particle linklists requires a double loop over the total number particles in the system, which can be and should be parallelized. The second step (Initialize) mainly aims to set the initial values of some of the allocated matrices, which can be parallelized but will make little contribution to the overall performance of the code.

Here, the main attention should be paid to those routines in the third step (Solve), which has to be run multiple times depending on the total simulation steps. In fact, all loops in this step are to be parallelized to the largest extent. Without going to the details of every single loop, the most time-consuming parts of the third step are highlighted in Fig. 3 (thicker boxes), as documented in Table 1 in Sect. 4. These parts all contains multiple loops over the total number of particles in the system. Updating the SPH particle linklist has to be done at each time step, because the SPH kernel depends on the Eulerian kernel. Computing both the SPH particle velocities and PD particle accelerations requires a loop over all the particles (SPH or PD), with a nested loop over neighboring particles in the corresponding supporting domain or horizon. As an illustration, we shall show the parallelizing details for the computing of the PD acceleration, with which the parallelization for the SPH particle velocity can be easily replicated. Before proceeding, the formulation for the PD acceleration are provided below. By combining Eqs. (3) and (4), one can obtain

$$\ddot{\mathbf{u}}_A = \frac{1}{\rho_A^0} \sum_{B=1, B\neq A}^{N_A} \Big\{ \omega(|\mathbf{X}_{A\to B}|)\mathbf{P}_{\mathbf{X}_A}\mathbf{K}_{\mathbf{X}_A}^{-1}\mathbf{X}_{A\to B} - \omega(|\mathbf{X}_{B\to A}|)\mathbf{P}_{\mathbf{X}_B}\mathbf{K}_{\mathbf{X}_B}^{-1}\mathbf{X}_{B\to A} \Big\} V_B^0 + \mathbf{b}_A. \tag{25}$$

As shown in Fig. 3, the Velocity-Verlet time integration algorithm is employed for the integration of the rate equations in the system [24]. To obtain the accelerations of the peridynamics particles, one needs to have the updated PK1 stress and PD artificial viscosity ready (see Fig. 3), which is converted to a PK1-type stress quantity in the actual implementation. The corresponding parallelism of the stresses is omitted here, to limit the length of presentation. With the PK1 stress and PD artificial viscosity ready, the parallelization of the computing the PD acceleration is shown in Fig. 4.

The initialization of the acceleration $acc(ndim, totmp)$ is done in the parallel region (line 15–22 in Fig. 4). $totmp$ is the total number of particles in the system, and $ndim$ is the space dimension of the problem. There are a team of *num of threads* threads that work concurrently in this region. $iptype(totmp)$ is an array that identifies the type of a particle; $iptype(i) = 1$ and $iptype(i) = 2$ indicate that particle i is an SPH and PD one, respectively. The main part of the code is essentially a

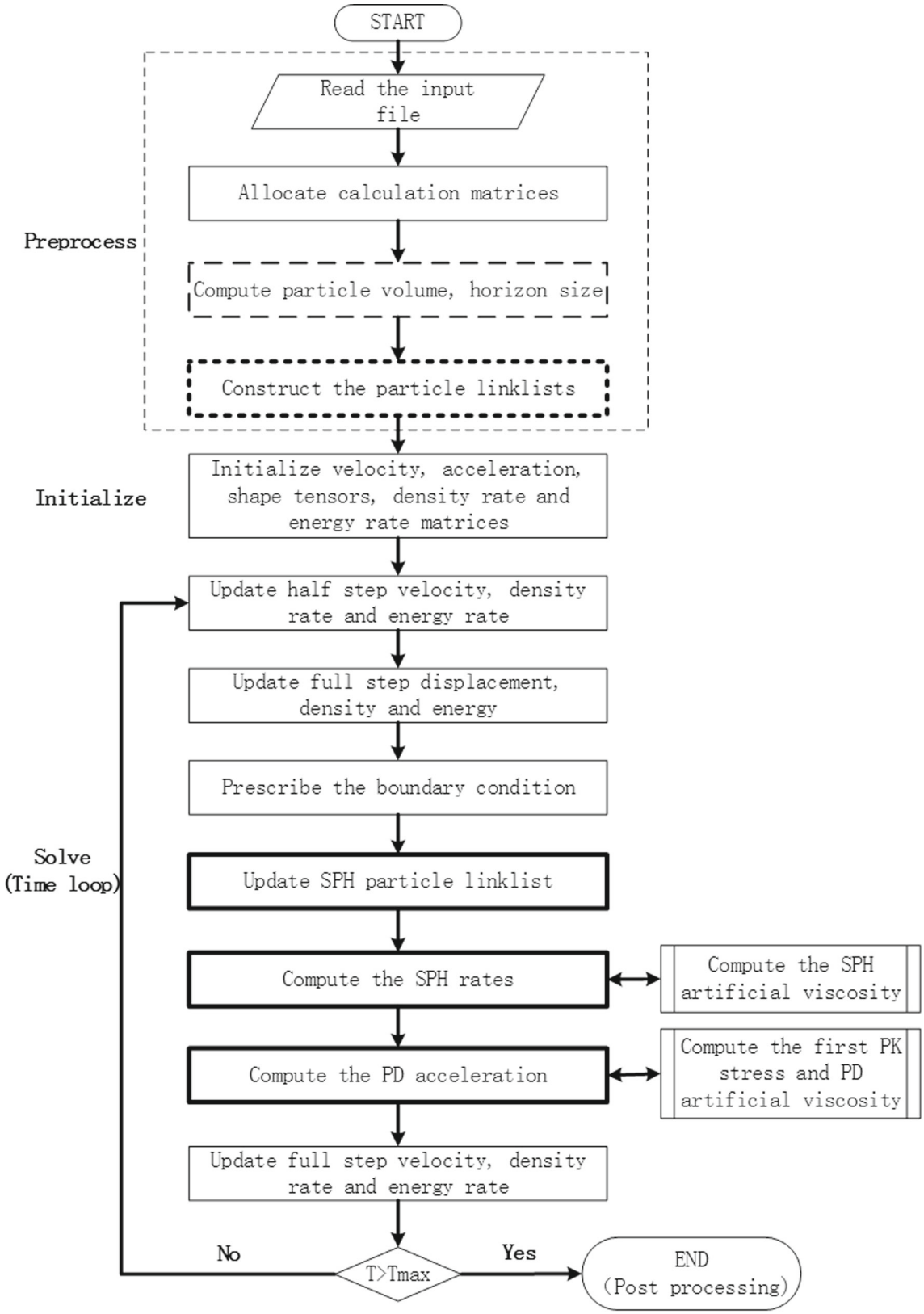

Fig. 3 Flow chart for the PD-SPH coupling scheme

```
14  !! initialize
15  !$omp parallel private(i) num_threads(numofthreads)
16  !$omp do
17  do i = 1, totmp
18    if(iptype(i) == 2) cycle
19    acc(:,i) = 0.d0
20  end do
21  !$omp end do
22  !$omp end parallel
23
24  !!! for each material point
25  !$omp parallel private(i,pk1i,invKi,forcei,piv1i) num_threads(numofthreads)
26  !$omp do
27  do i = 1, totmp
28    if(iptype(i) == 2) cycle
29    !! first PK stress at i
30    pk1i = PKI(:,:,i)
31    !! Kinv at i
32    invKi = Kinv(:,:,i)
33    !! initialize the force vector acting on particle i
34    forcei = 0.d0
35    !! artificial PKI at particle i
36    piv1i = detF(i)*pat(i)*transpose(inverse(F_mat(:,:,i)))
37    !! loop over its family members
38    !$omp parallel private(ii,j,xi,eta,pk1j,invKj,piv1j) num_threads(numofthreads)
39    !$omp do reduction(+:forcei)
40    do ii = 1,numfam(i)
41      j = linkfam(ii,i) !! particle index of the ii-th family member
42      xi = xr(:,j) - xr(:,i) !! bond vector xi_{ij}
43      eta = xr(:,j)+disp(:,j) - xr(:,i)-disp(:,i)
44      pk1j = PKI(:,:,j) !! first PK stress at j
45      invKj = Kinv(:,:,j) !! Kinv at j
46      !! artificial PKI at particle j
47      piv1j = detF(j)*pat(j)*transpose(inverse(F_mat(:,:,j)))
48      forcei = forcei + bweight(ii,i)*(matmul(pk1i+piv1i,matmul(invKi,xi)) &
49        + matmul(pk1j+piv1j,matmul(invKj,xi)))*dvol(j)
50    end do
51    !$omp end do
52    !$omp end parallel
53    acc(:,i) = acc(:,i) + forcei/rhos
54  end do
55  !$omp end do
56  !$omp end parallel
```

Fig. 4 OpenMP program for computation of the PD acceleration

parallel region (line 25–56 in Fig. 4) that loops over all the PD particles in the system, with a nested parallel region (line 38–52 in Fig. 4) that loops over the particles (both SPH and PD) within the corresponding horizon. Moreover, we would like to emphasize one particular point regarding the arguments of the clause *private* in each parallel region. Take the *private* clause at line 25 for instance, it is easy to see that the variable i has to be private, given that it is the iterative index for the corresponding *do* loop (line 27–54 in Fig. 4) in the parallel region. The rule to decide whether any other variables are private or not is a variable that depends on the iterative index i has to be private. Variables not declared by the *private* clause are public variables by default (except the iterative indices of loops). For better illustration, the key variables used in the source code are explained here: $pk1i$ ($pk1j$) is the PK1 stress at particle i (j); $invKi$ ($invKi$) is the inverse of the shape tensor $\mathbf{K}_{\mathbf{X}_i}^{-1}$ ($\mathbf{K}_{\mathbf{X}_j}^{-1}$) at the particle i (j); $pat(i)$ ($pat(j)$) is the artificial viscosity-induced

pressure at particle $i(j)$; $piv1i$ ($piv1j$) is the contribution to the PK1 stress due to the PD artificial viscosity at particle $i(j)$; $numfam$ and $linkfam$ provide the information of the linklist; $forcei$ is the total force per unit volume applied at particle i; $xi(eta)$ is the bond vector pointing from i to j in the reference (current) configuration, respectively; $bweight$ is the influence function; and $rhos$ is the density in the reference configuration.

4 Numerical examples

In this section, we first briefly discuss the constitutive equations that are used to describe the soil medium and the corresponding time integration scheme, and then we present the results of numerical simulations that are conducted to test the performance of the parallel PD-SPH program.

4.1 Constitutive equations of the soil and the time integration

In this work, the Drucker–Prager (DP) plasticity model is employed to characterize the behavior of soil medium. The constitutive updates for a non-linear finite deformation are first presented, which are then expressed in the corresponding peridynamics formulations.

Formally, the yield function of the DP model may be expressed as

$$\begin{cases} f = \|\mathbf{s}\| - (A(\phi')c' - B(\phi')p') \le 0 \\ A(\phi') = \frac{2\sqrt{6}\cos\phi'}{3+\beta\sin\phi'} \\ B(\phi') = \frac{2\sqrt{6}\sin\phi'}{3+\beta\sin\phi'}\,, \ -1 \le \beta \le 1 \end{cases}, \tag{26}$$

where $\mathbf{s}$ is the deviatoric stress tensor; ϕ' represents the effective friction angle; c' denotes the effective cohesion (Pa); and p' is the mean effective stress. In Eq. (26), when $\beta = 1$ and $\beta = -1$, f represents either the triaxial extension (TE) corners or the triaxial compression (TC) corners of the Mohr–Coulomb yield surface, respectively [22]. Considering a non-associative flow, we may write the Drucker–Prager (DP) plastic potential function as

$$\begin{cases} g = \|\mathbf{s}\| - (A(\psi')c' - B(\psi')p') \\ A(\psi') = \frac{2\sqrt{6}\cos\psi'}{3+\beta\sin\psi'} \\ B(\psi') = \frac{2\sqrt{6}\sin\psi'}{3+\beta\sin\psi'}\,, \ -1 \le \beta \le 1 \end{cases}, \tag{27}$$

where ψ' represents the effective dilation angle.

The Helmholtz free energy function $\rho_s\Psi$ per unit soil volume can be decomposed into elastic part and inelastic parts as follows:

$$\rho_s\Psi(\boldsymbol{\epsilon}^e, \boldsymbol{\zeta}) = \frac{1}{2}\boldsymbol{\epsilon}^e : \mathbf{D}^e : \boldsymbol{\epsilon}^e + \frac{1}{2}\boldsymbol{\zeta}\cdot\mathbf{H}\cdot\boldsymbol{\zeta}, \tag{28}$$

where $\boldsymbol{\epsilon}^e$ is the elastic strain tensor, $\mathbf{D}^e$ the elastic modulus tensor, $\mathbf{H}$ the hardening/softening modulus matrix, and $\boldsymbol{\zeta}$ a set of strain-like internal state variables (ISVs) associated with plastic hardening.

From the Helmholtz free energy in Eq. (28), one can derive the rate equations of stress ($\boldsymbol{\sigma}'$) and stress-like ISVs ($\mathbf{q}^\zeta$) as

$$\dot{\boldsymbol{\sigma}}' = \frac{\partial(\rho_s\Psi)}{\partial\dot{\boldsymbol{\epsilon}}^e} = \mathbf{D}^e : \dot{\boldsymbol{\epsilon}}^e = \mathbf{D}^e : (\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^p) \tag{29}$$

$$\dot{\mathbf{q}}^\zeta = \frac{\partial(\rho_s\Psi)}{\partial\dot{\boldsymbol{\zeta}}} = \mathbf{H}\cdot\dot{\boldsymbol{\zeta}}, \tag{30}$$

where $\boldsymbol{\epsilon}^p$ and $\mathbf{q}^\zeta = \{c', \phi', \psi'\}^T$ are the plastic strain tensor and the specific internal states variables, respectively. It is assumed that the ISVs are to evolve under an independent, linear hardening model, i.e.,

$$\mathbf{H} = \begin{bmatrix} H^c & 0 & 0 \\ 0 & H^\phi & 0 \\ 0 & 0 & H^\psi \end{bmatrix}, \tag{31}$$

where H^c, H^ϕ, H^ψ are the linear hardening/softening modulus for c', ϕ', ψ', respectively. Based on the plastic potential function (see Eq. (27)), the evolution of plastic flow can be described by the following equation,

$$\begin{aligned} \dot{\boldsymbol{\epsilon}}^p &= \dot{\gamma}\frac{\partial g}{\partial\boldsymbol{\sigma}'} = \dot{\gamma}\left(\frac{\partial\|\mathbf{s}\|}{\partial\boldsymbol{\sigma}'} + B(\psi')\frac{\partial p'}{\partial\boldsymbol{\sigma}'}\right) \\ &= \dot{\gamma}\left(\frac{\mathbf{s}}{\|\mathbf{s}\|} + \frac{1}{3}B(\psi')\mathbf{I}\right), \end{aligned} \tag{32}$$

where $\mathbf{I}$ is the second-order identity tensor. The evolution equations of the ISVs are given as

$$\dot{\mathbf{q}}^\zeta = \dot{\gamma}\mathbf{H}\cdot\mathbf{h}(\boldsymbol{\sigma}, \mathbf{q}^\zeta) \tag{33}$$

$$\mathbf{h} = -\frac{\partial f}{\partial\mathbf{q}^\zeta}\,, \tag{34}$$

where $\mathbf{h}$ is a hardening function. Using the principle of the maximum plastic dissipation, one can obtain

$$\mathbf{h} = \begin{pmatrix} A(\phi') \\ \frac{\partial A(\phi')}{\partial\phi'}c' - \frac{\partial B(\phi')}{\partial\phi'}p' \\ \frac{\partial A(\psi')}{\partial\psi'}c' - \frac{\partial B(\psi')}{\partial\psi'}p' \end{pmatrix}. \tag{35}$$

Using the consistency condition $\dot{f} = 0$, the plastic multiplier $\dot{\gamma}$ can be derived as

$$\begin{cases} \dot{\gamma} = \frac{1}{\chi}\frac{\partial f}{\partial\boldsymbol{\sigma}'} : \mathbf{D}^e : \dot{\boldsymbol{\epsilon}} \\ \chi = \frac{\partial f}{\partial\boldsymbol{\sigma}'} : \mathbf{D}^e : \frac{\partial g}{\partial\boldsymbol{\sigma}'} - \frac{\partial f}{\partial\mathbf{q}^\zeta} : \mathbf{H}\cdot\mathbf{h}. \end{cases} \tag{36}$$

Eqs. (29), (30) and (36) are the equations for constitutive update of a non-linear plastic constitutive model of the soil medium. In this work, the following explicit update algorithm is adopted:

$$\boldsymbol{\sigma}'^{,tr}_{n+1} = \boldsymbol{\sigma}'_n + \mathbf{D}^e : \Delta\boldsymbol{\epsilon} \tag{37}$$

$$f^{tr}_{n+1} = \|\mathbf{s}^{tr}\| - \left(A(\phi'_n)c'_n - B(\phi'_n)(p')^{\mathrm{tr}}_{n+1}\right), \tag{38}$$

where the superscript tr stands for the corresponding variables at the trial phase. The trial phase calculation may be summarized as follows:

if $f^{\mathrm{tr}}_{n+1} < 0$, elastic phase: update $\boldsymbol{\sigma}'_{n+1} = \boldsymbol{\sigma}'^{,\mathrm{tr}}_{n+1}$ and $\mathbf{q}^{\zeta}_{n+1} = \mathbf{q}^{\zeta}_n$.

if $f^{\mathrm{tr}}_{n+1} \geq 0$, plastic phase:

$$\Delta\gamma = \frac{f^{tr}_{n+1}}{2\mu + KB(\phi')B(\psi') + H^c(A(\phi'))^2} \tag{39}$$

$$\boldsymbol{\sigma}'_{n+1} = \boldsymbol{\sigma}'^{,\mathrm{tr}}_{n+1} - \Delta\gamma(KB(\psi')\mathbf{I} + 2\mu\hat{\mathbf{n}}_{n+1}) \tag{40}$$

$$\mathbf{q}^{\zeta}_{n+1} = \mathbf{q}^{\zeta}_n + \Delta\gamma\mathbf{H}\cdot\mathbf{h}(\boldsymbol{\sigma}, \mathbf{q}^{\zeta}) \,. \tag{41}$$

The trial stress is calculated based on Eq. (37), under the assumption of small deformation. However, generally speaking, the soil medium under blast loading shall undergo finite deformation with large rotations. Thus, it should be replaced by a formulation accounting for finite deformation. In this work, the time integration suitable for non-linear finite deformation is adopted, which is based on the Hughes–Winget incrementally objective algorithm [15].

In the Hughes–Winget algorithm, an intermediate configuration at time step $n+\alpha$ is defined

$$\mathbf{x}^{n+\alpha} = (1-\alpha)\mathbf{x}^n + \alpha\Delta\mathbf{u}, \tag{42}$$

where the scalar parameter $\alpha = 0.5$. Following the derivation of the approximated deformation gradient in Eq. (9), the deformation gradient at the configuration $\mathbf{x}_{n+\alpha}$ can be calculated as

$$\mathbf{F}^{n+\alpha} = \frac{\partial\mathbf{x}^{n+\alpha}}{\partial\mathbf{X}} = \left(\sum_{\mathbf{X}_B\in\mathcal{H}_{\mathbf{X}_A}} \omega(|\boldsymbol{\xi}|)(\mathbf{x}^{n+\alpha}_B - \mathbf{x}^{n+\alpha}_A)\otimes\mathbf{X}_{A\to B}\right)\cdot\mathbf{K}^{-1}_{\mathbf{X}_A}. \tag{43}$$

Meanwhile, the gradient of $\Delta\mathbf{u}$ with respect to the reference configuration can be written as

$$\nabla_{\mathbf{x}}\Delta\mathbf{u} = \frac{\partial(\Delta\mathbf{u})}{\partial\mathbf{X}} = \left(\sum_{\mathbf{X}_B\in\mathcal{H}_{\mathbf{X}_A}} \omega(|\boldsymbol{\xi}|)(\Delta\mathbf{u}_B - \Delta\mathbf{u}_A)\otimes\mathbf{X}_{A\to B}\right)\cdot\mathbf{K}^{-1}_{\mathbf{X}_A}. \tag{44}$$

By using the chain rule, one can then obtain the increment of the deformation gradient

$$\mathbf{G}_{n+\alpha} = \frac{\partial(\Delta\mathbf{u})}{\partial\mathbf{x}_{n+\alpha}} = \nabla_{\mathbf{x}}\Delta\mathbf{u}\cdot\mathbf{F}^{-1}_{n+\alpha}, \tag{45}$$

which can be split into the following two parts:

$$\boldsymbol{\gamma}_{n+\alpha} = (\mathbf{G}_{n+\alpha} + \mathbf{G}^T_{n+\alpha})/2 \tag{46}$$

$$\boldsymbol{\omega}_{n+\alpha} = (\mathbf{G}_{n+\alpha} - \mathbf{G}^T_{n+\alpha})/2. \tag{47}$$

The objective effective stress increment can then be obtained as

$$\Delta\boldsymbol{\sigma}' = \mathbf{D}^e : \boldsymbol{\gamma}. \tag{48}$$

Finally, Eq. (37) can be replaced by the following equations:

$$\boldsymbol{\sigma}'_{n+1} = \hat{\boldsymbol{\sigma}}'_n + \Delta\boldsymbol{\sigma}' \tag{49}$$

$$\hat{\boldsymbol{\sigma}}'_n = \boldsymbol{\Omega}^T_{n+\alpha}\cdot\boldsymbol{\sigma}'_n\cdot\boldsymbol{\Omega}_{n+\alpha} \tag{50}$$

$$\boldsymbol{\Omega}_{n+\alpha} = \mathbf{I} + (\mathbf{I} - \alpha\boldsymbol{\omega}_{n+\alpha})^{-1}\cdot\boldsymbol{\omega}_{n+\alpha} \,. \tag{51}$$

4.2 Soil fragmentation induced by blast load

Numerical simulations are conducted to test the performance of parallel PD-SPH program. The simulation model is chosen to be a cubic soil block with a cubic TNT explosive buried inside. Figure 5 shows the schematic of the setup. The TNT explosive is located at very center of the soil block, with all the surfaces parallel to those of the soil block. The side lengths of cubic soil block and the dimension of the TNT explosive are $L = 0.03$ m and a, respectively. The whole system is discretized into 29,791 particles. The particle positions (nodes), associated volumes, and their corresponding horizon sizes $\delta_{\mathbf{X}}$ or smoothing lengths $h_{\mathbf{x}}$ are generated from an FEM mesh of 29,791 nodes and 27,000 hexahedron elements. The SPH particles represent the computational domain of the explosive, and the peridynamics particles represent the soil medium (see Fig. 5). Given that the explosion force produced by the explosive is so dominant, the gravitational force, and frictions force between the peridynamics particles and the surrounding environment may be neglected. That is, the whole system is subjected to the PD-SPH particle interactions only. The constitutive relations, material properties of the soil medium are the same as that in [11]. The simulation time step size is set to be $\mathrm{d}t = 5.0\times10^{-8}$ s.

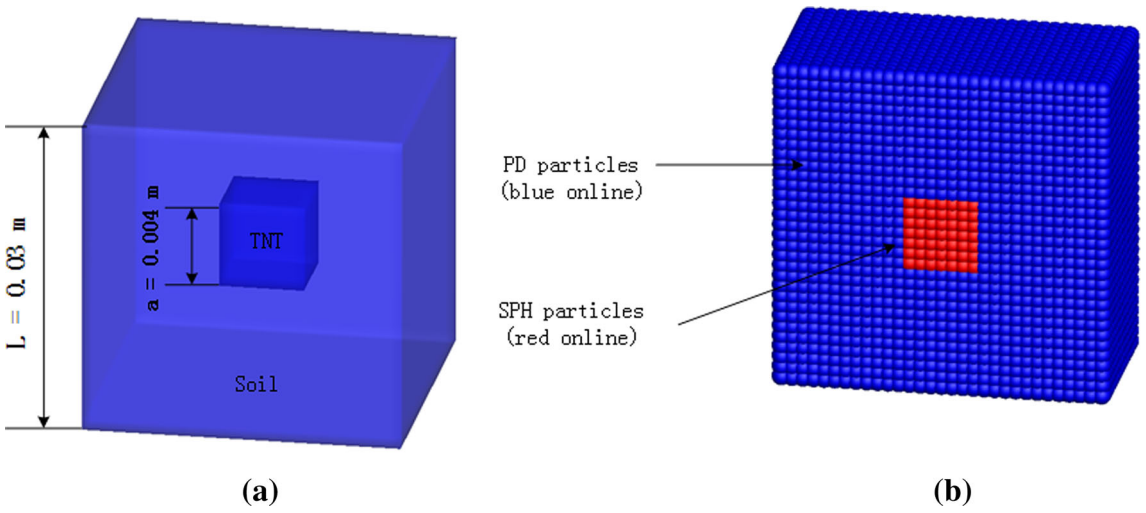

Fig. 5 A cubic soil block with a cubic TNT explosive embedded inside (located at the very center, with all the surfaces parallel to the corresponding ones of the soil block): **a** geometrical configuration. **b** Discretization of the domain with 29,791 particles

Table 1 Simulation time (in seconds) cost for each part of the code. For all the three cases, the total number of particles in the system is 29,791, and the number of SPH particles is different

No. of SPH particles	Preprocess		Initialize	Solve				Total
	Linklist	Others		SPHlink	SPH rates	PD acceleration	Others	
343	11.24	0.52	0.19	15.87	39.60	36.29	8.55	111.55
1331	10.78	0.53	0.60	58.85	38.16	33.18	8.36	149.33
3375	10.79	0.47	1.57	153.41	37.32	31.39	7.93	240.84

The strictly non-parallel portion of the code is $<0.5\,\%$

First, the program is executed with one single thread for 100 time steps, and the time spent in each section of the code is recorded, as shown in Table 1. Three different cases of different explosive sizes are considered, with the side length $a = 0.004, 0.006,$ or 0.008 m. The corresponding number of SPH particles in the three cases are 343, 1331, and 3375. One can see that the most time-consuming parts of the program is the compiling of the Linklist in the Preprocess and the SPH linklist, SPH rates, and PD accelerations in the Solve. One may also notice that as the number of SPH particles increases, the construction of the SPH linklist gradually dominates the consumption of the total simulation time. We would like to mention that the non-parallel portion of the code only includes the "Preprocess–Others" and "Initialize." The "Solve–Others" mainly serves as an update to the velocity and displacement arrays based on the accelerations, which are all easily parallelized.

For a fixed number of processors N performing the parallel fraction of the work, the potential speed-up is

$$\text{Speed-up} = \frac{1}{1 - P + P/N}\,. \tag{52}$$

For instance, if 75 % of the code is parallelized, the maximum speed-up is 4 (assuming N goes to infinity), meaning that the code can run four times as faster as than that of the code without parallelization, from which one can see that a high percentage of the code that can be parallelized is a crucial factor to the possible overall performance of the parallel code. With the data provided in Table 1, one may easily find that for the present case, the strictly non-parallel portion of the code is $<0.5\,\%$. In parallel programming, the speed-up ratio of a code is defined as

$$\text{Sp} := \frac{\text{Time for the original code}}{\text{Time for the improved code}} \tag{53}$$

and the parallel efficiency is defined as

$$\text{Ep} := \frac{\text{Sp}}{N}, \tag{54}$$

where N is the number of threads used in the corresponding simulation.

To test the parallel performance of the code, a series of simulations are conducted with number of threads (N) ranging from 1 to 32 in a CPU model of Intel(R) Xeon(R) E5-2698 of main frequency 2.3 GHz. The OpenMP fortran90 code is compiled by using the Intel Parallel Studio XE 2015. Figure 6 shows the simulation times versus the number of threads par-

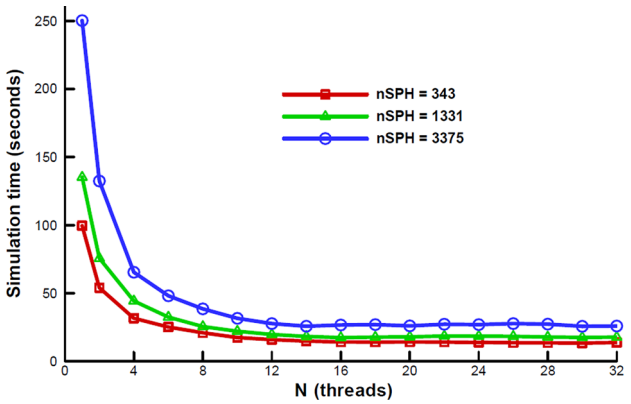

Fig. 6 Simulation times cost versus the number of threads participated

ticipated in the computations. For all the three different cases, it can be easily seen that the simulation times decrease for the first few number of threads (1–10) and gradually slow down until almost reaching a plateau (16–32).

The speed-up ratio (Sp) and parallel efficiency (Ep) of the code are also plotted with their respective ideal cases of a strictly non-parallel portion 0.5 % in Fig. 7. Note that the parallel efficiency for the case with more SPH particles tends to be higher than the one with smaller number of SPH particles. This can be ascribed to the fact that the case with lower number of SPH particles limits the number of particles assigned for each thread, which to certain extent increases the overhead in managing the threads.

For the three different cases, as the number of threads N goes from 1 to 14, the actual speed-up ratios of the code increase from 1.0 to approximately 7.01, 7.73, and 9.52 and the parallel efficiency deceases from 1.0 to 0.48, 0.53, and 0.69, respectively. As the number of threads continuous increases, there speed-up ratios become stagnant and the parallel efficiencies simply decrease linearly. There are several reasons that can be accounted for this phenomenon. First, N threads may have N times the computation power, but the memory bandwidth is shared by N processors and performance degradation shall be observed when they compete for the shared-memory bandwidth. One would expect, for a fixed number of particles, the existence of a critical number of threads (here 14), after which the increasing number of threads participated would not help in reducing the simulation time. Second, data synchronization requires that an upcoming process must wait until all the data it depends on are finished computing from a previous process. In a particular computation block, some threads may have finished their computation much earlier, but has to wait until all the threads are done before moving on. For instance, all the SPH-linklists have to be ready before the SPH-related rates are updated; all the information on the PK1 stress and artificial viscosity have to be in place, before the computing of the acceleration of peridynamics particles started (see Fig. 3). In addition, many other common problems in parallel computing may also affect the overall speed-up of the code, like memory hierarchy of the processor family, data localization, and reutilization.

To improve the parallel performance of the current PD-SPH program, one can computer problems of large size (granularity) such that the fraction of code that can be parallelized is larger; use a better computer to reduce the compete for resources of the shared-memory system; and improve the load balancing such that each thread would take approximately the same amount of time to finish their work in a parallel region. In fact, by using the PD-SPH program, simulations of a system of total 4,891,414 particles are carried out by using 16 and 32 threads, respectively, in the same Intel(R) Xeon(R) E5-2698 CPU. The total simulation time recorded for the 16 threads case is around 77.5 hours and that for the 32

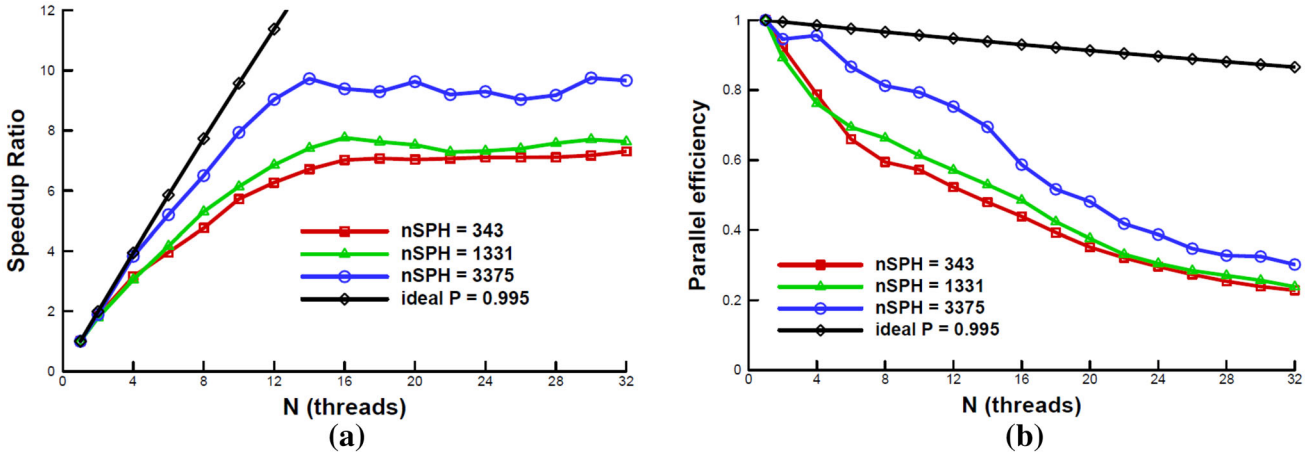

Fig. 7 Performance of the OpenMP parallel code of PD-SPH coupling **a** speed-up ratio **b** parallel efficiency

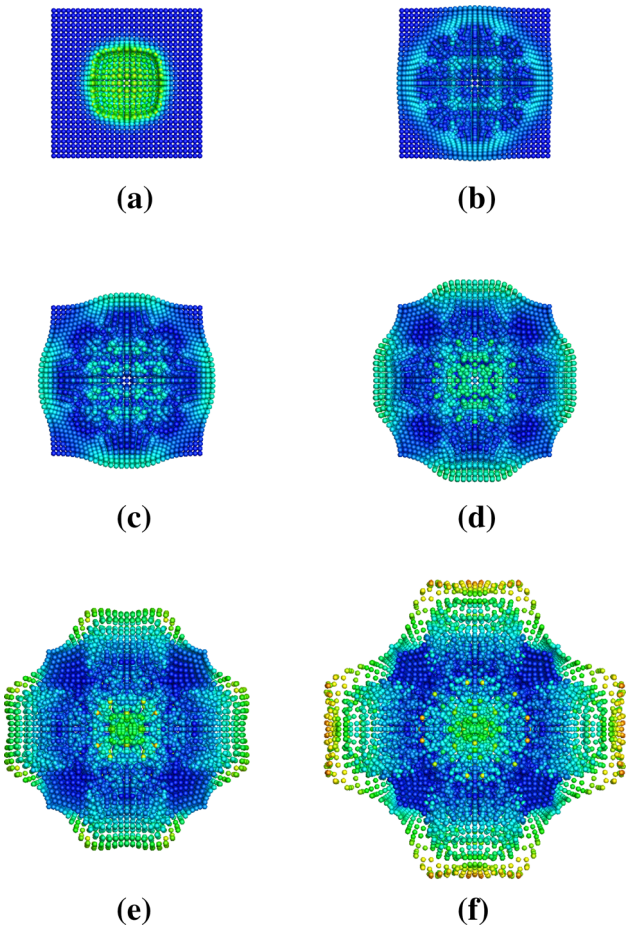

Fig. 8 Dynamic fragmentation process of a soil block due to the embedded explosive; view from the front of a vertical cross section. Contour color: velocity magnitude **a** $t = 5.0\,\mu s$. **b** $t = 20.0\,\mu s$. **c** $t = 30.0\,\mu s$. **d** $t = 40.0\,\mu s$. **e** $t = 50.0\,\mu s$. **f** $t = 60.0\,\mu s$.

threads case is around 42.0 hours. If we treat the 16 threads case as the original "serial" code, the parallel efficiency of the 32 threads case can be evaluated as $(76.5/42.0)/2 = 0.91$. This further indicates that the quick decay of the parallel efficiency in Fig. 7 is mainly due to the lack of computational load in each thread.

Given that our main purpose is to present the parallelization details and performance of the parallel PD-SPH coupling code, the validity of the modeling method in capturing the explosion is not discussed here. Interested readers may consult [11]. But for the sake of completeness, the dynamic fragmentation process of the soil block due to the buried TNT explosion is provided in Fig. 8. One can see that the soil particles around the TNT explosive gain very large momentums at the first few steps. After hitting by the first shock wave, the soil medium gradually expand in every directions. The results from simulations with different numbers of threads are the same, as what they should be.

5 Conclusions

In this work, the OpenMP-based parallel programming is adopted to implement the recently developed PD-SPH coupling scheme, and several numerical calculations are conducted to evaluate its performance. We have discussed computational or algorithmic structure of the OpenMP parallelization of PD-SPH coupling in details. A numerical example of soil fragmentation caused by the buried TNT explosive is performed by using the OpenMP parallel computing. By running the simulation with number of threads ranging from 1 to 32, the computational time, the speed-up ratio, and parallel efficiency of the code are analyzed. In this study, we have found that the OpenMP can improve the computation efficiency of particle method computation in some regular computers without high-performance capacities. The OpenMP can successfully speed up the computation, owing to the fact that it can always be employed to parallelize the time-consuming sections of the corresponding serial code.

Acknowledgments This work was supported by the ONR MURI Grant N00014-11-1-0691. This support is gratefully appreciated.

References

1. Li S, Liu WK (2004) Meshfree and particle methods. Springer, Berlin
2. Silling SA, Epton M, Weckner O, Xu J, Askari E (2007) Peridynamic states and constitutive modeling. J Elast 88:151–184
3. Foster JT, Silling SA, Chen WW (2010) Viscoplasticity using peridynamics. Int J Numer Methods Eng 81(August 2009):1242–1258
4. Tuniki BK (2012) Peridynamic constitutive model for concrete. PhD thesis, Osmania
5. Lai X, Ren B, Fan H, Li S, Wu CT, Richard AR, Liu L (2015) Peridynamics simulations of geomaterial fragmentation by impulse loads. Int J Numer Analy Methods Geomech 39:189–213
6. Liu MB, Liu GR, Lam KY, Zong Z (2003) Smoothed particle hydrodynamics for numerical simulation of underwater explosion. Comput Mech 30(2):106–118
7. Quinlan NJ, Basa M, Lastiwka M (2006) Truncation error in mesh-free particle methods. Int J Numer Methods Eng 66(August 2005):2064–2085
8. Liu MB, Liu GR (2006) Restoring particle consistency in smoothed particle hydrodynamics. Appl Numer Math 56:19–36
9. Swegle JW, Hicks DL, Attaway SW (1995) Smoothed particle hydrodynamics stability analysis. J Comput Phys 116:123–134
10. Campell PM (1989) Some new algorithms for boundary value problems in smooth particle hydrodynamics. Technical report, Technical Report DNA-TR-88-286
11. Fan H, Bergel GL, Li S (2015) A hybrid peridynamics-SPH simulation of soil fragmentation by blast loads of buried explosive. Int J Impact Eng 87:14–27
12. Chudik J, David G, Kotov VE, Mirenkov NV, Ondas J, Plander I, Valkovskii VA (2013) Algorithms, software and hardware of parallel computers. Springer, Berlin
13. Gropp W, Lusk E, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, 1st edn. MIT press, Cambridge

14. Chapman B, Gabriele G, Van Der Pas R (2008) Using OpenMP: portable shared memory parallel programming. MIT press, Cambridge
15. Hughes TJR, Winget J (1980) Finite rotation effects in numerical integration of rate constitutive equations arising in large-deformation analysis. Int J Numer Methods Eng 15:1862–1867
16. Liu GR, Liu MB (2003) Smoothed particle hydrodynamics: a meshfree particle method. World Scientific, Singapore
17. Fulk DA (1994) A numerical analysis of smoothed particle hydrodynamics. PhD thesis, Air Force Institute of Technology
18. Rabczuk T, Belytschko T, Xiao SP (2004) Stable particle methods based on Lagrangian kernels. Comput Methods Appl Mech Eng 193(12–14):1035–1063
19. Monagphan JJ (1994) Simulating free surface flows with SPH. J Comput Phys 110(2):339–406
20. Chandra R (2001) Parallel programming in OpenMp. Morgan Kaufmann, San Francisco
21. Sato M (2002) OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors. In: 15th International Symposium on System Synthesis 2002, pp 109–111
22. Zienkiewicz OC, Chan AHC, Pastor M, Schrefler BA, Shiomi T (1999) Computational geomechanics. Wiley Chichester, Chichester
23. Ren B, Fan H, Bergel GL, Regueiro RA, Lai X, Li S (2015) A peridynamics-SPH coupling approach to simulate soil fragmentation induced by shock waves. Comput Mech 55:287–302
24. Swope William C (1982) A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. J Chem Phys 76(1982):637
25. Bergeron D, Walker R, Coffey C (1998) Detonation of 100-gram anti-personnel mine surrogate charges in sand-a test case for computer code validation. Technical report, Defence Research Establishment Suffield, Ralston ALTA (CAN)