Multiple-GPU parallelisation of three dimensional material point method based on single-root complex

Youkou Dong, Lan Cui, and Xue Zhang

Manuscript submitted to International Journal for Numerical Methods in Engineering on 14/07/2021

1st Revised manuscript resubmitted on 12/10/2021

2nd Revised manuscript resubmitted on 30/11/2021

Youkou Dong

Associate Professor

College of Marine Science and Technology, China University of Geosciences, 388 Lumo Road, Wuhan 430074, China

State Key Laboratory of Coastal and Offshore Engineering, Dalian University of Technology, 2 Linggong Road, Dalian, 116024, China

Tel: +86 132 1271 4650

Email: dongyk@cug.edu.cn

Lan Cui (corresponding author)

Assistant Professor

State Key Laboratory of Geomechanics and Geotechnical Engineering, Institute of Rock and Soil Mechanics, Chinese Academy of Sciences, Wuhan 430071, China

Email: lcui@whrsm.ac.cn

Xue Zhang

Lecturer

Department of Civil Engineering and Industrial Design, University of Liverpool, Liverpool, UK

College of Marine Science and Technology, China University of Geosciences, China
xue.zhang2@liverpool.ac.uk

Abstract

As one of the arbitrary Lagrangian-Eulerian methods, the material point method (MPM) owns intrinsic advantages in simulation of large deformation problems by combining the merits of the Lagrangian and Eulerian approaches. Significant computational intensity is involved in the calculations of the MPM due to its very fine mesh needed to achieve a sufficiently high accuracy. A new multiple-GPU parallel strategy is developed based on a single-root complex architecture of the computer purely within a CUDA environment. Peer-to-Peer (P2P) communication between the GPUs is performed to exchange the information of the crossing particles and ghost element nodes, which is faster than the heavy send/receive operations between different computers through the infiniBand network. Domain decomposition is performed to split the whole computational task over the GPUs with a number of subdomains. The computations within each subdomain are allocated on a corresponding GPU using an enhanced 'Particle-List' scheme to tackle the data race during the interpolation from associated particles to common nodes. The acceleration effect of the parallelisation is evaluated with two benchmarks cases, mini-slump test after a dam break and cone penetration test in clay, where the maximum speedups with 1 and 8 GPUs are 88 and 604, respectively.

Keywords: material point method, parallel computation, cone penetration test, mini slump test

1. Introduction

The material point method (MPM), one of the arbitrary Lagrangian Eulerian methods, owns intrinsic advantages in simulation of large deformation problems by combining the merits of the Lagrangian and Eulerian methods[1-4]. The Lagrangian particles, inheriting all the history-dependent information of material, are allowed to move through the background Eulerian mesh, while the mesh is always fixed in space to avoid the potential mesh distortion. The MPM, introduced to solid mechanics[5] from computational fluid dynamics[6], was used to simulate high explosive explosions[7], propagation of wood cracks[8], impact between solid bodies[9-12], fluid-structure interactions[13] and computer animations[14-16]. In the recent decade the MPM was applied to geotechnical engineering to investigate runout of submarine landslides[17-20], penetration and pull-out of structures[21-23] and flow of granular materials[24-26]. Coupling analysis of pore or free water and soil, mainly used in the analysis of slope stability[27-30], is a new trend of the MPM simulations.

One of the main obstacles to the widespread application of the MPM is its low computational efficiency, especially for large-scale and long-period problems. As the particles mostly are not at the optimum locations for integration in the elements[31-33], the mesh adopted in the MPM should be much finer than that in large deformation finite element analysis to obtain sufficient accuracies[3, 34-35]. Structured elements, used as often as the unstructured elements[36-37], bring extra computational loads with identical mesh size from the concerning domain to the far field. Elements in singularity zone around structures need to be further refined for soil-structure interaction problems[38-39]. Although an initial assignment of four particles in per element is often sufficient to obtain a smooth stress/strain field in many cases of MPM simulations[40], the configuration of 16 particles in each element sometimes is necessary for high-speed impacting problems[14, 41]. Therefore, most existing MPM analyses were limited to small-scale problems or two-dimensional plane-strain scenarios[14, 21, 41].

Parallel computation on the central processing units (CPU) or graphic processing units (GPU) is the most viable option to promote the efficiency of the MPM, which often requires special treatments to make the algorithm more parallelisable. Acceleration effect of the parallelisation can be significantly influenced by different parallel techniques and hardware platforms. Reference [9] and [42] proposed a single-CPU parallelisation scheme of the MPM with the loop-based parallel library OpenMP, achieving a five-fold speedup over a sequential calculation by mobilising eight CPU cores. The OpenMP-based parallelisation is quite simple by invoking an executable directive before each loop operation; however, its limitation is also obvious as most commercially available CPUs have less than 32 cores. Reference [43] developed a multiple-CPU parallelisation strategy using the message passing interface (MPI), accelerating the computation for up to 2,500 times with 16,384 CPU cores on a supercomputer. In comparison with the CPU parallelisation, the state-of-the-art GPU parallelisation is more cost effective as each GPU hosts thousands of GPU cores[44-45], but its parallel techniques are more complex. Reference [46] proposed a specialised parallelisation scheme with single GPU by using the compute unified device architecture (CUDA), obtaining speedups of around 25 times given double precision numbers were used. Limited by the memory size dedicated on the GPU, the maximum number of particles allowed in the MPM model was around six million. Reference [47] adopted a similar technique to parallelise an implicit MPM algorithm and applied it to computer animations. Reference [48] then extended the framework to orchestrate multiple GPUs on a multiple-computer cluster based on a hybrid MPI-CUDA environment, which was sensitive to the data exchange between the computers through a private network. Given 16 GPUs were used on four tandem computers, up to 900 times speedup was then obtained with the maximum number of particles as 96 million. Recently, Reference [49] further optimised the massively parallel framework of the MPM and achieved over 100 times speedup on a single GPU; however, its acceleration effect on multiple-GPU platform is heavily dependent on the hardware performance for the stream event synchronisation on the specific GPU device; as a result, speedup of the framework is not always so high (~ 100) as presented in Reference [49], especially for medium scale problems with less than 500,000 particles; and much complexities were caused in the parallel scheme (such as the Particle-Grid offset technique) and the data transport between the GPUs (such as the AoSoA data structure), which may undermine the reliability and maintainability of the programme. Therefore, a reliable and efficient MPM program based on a simple multiple-GPU parallel framework is still needed.

In this paper, a parallelisation strategy with multiple GPUs is developed within the CUDA environment. Different to that in Reference [48], the mobilised GPUs are hosted in an identical computer platform with a shared random access memory (RAM). Peer-to-Peer (P2P) communication between the GPUs is performed to exchange the information of the crossing particles and ghost element nodes, which is faster than the heavy send/receive operations between different computers through the infiniBand network in Reference [48]. Domain decomposition is performed to split the whole computational task over the GPUs with a number of subdomains. The computations within each subdomain are allocated on a corresponding GPU and the MPM algorithm on each GPU is parallelised with the technique proposed in Reference [46] with specific improvements, which further enhance the speedup and reliability of the computation. The calculation results are assembled on the shared RAM of the computer through the connection with the GPU devices. In comparison to the parallel framework in Reference [49], the parallel strategy in this study is more reliable and friendly to the new developers of the MPM, which also presents satisfying acceleration effects. Specifically, this paper includes the following contributions: (1) an efficient parallel technique is proposed to invoke multiple GPUs on an identical computer using P2P communication with each other; (2) a hybrid memory IO framework is developed based on the shared RAM and distributed GPU memory hierarchy; (3) an enhanced 'Particle-List' scheme to parallelise the interpolation from particles to nodes, which is also parallelised on GPUs and hence avoids the frequent data exchange between the CPU and GPUs; (4) the parallelised MPM algorithm is extended from two to three dimensional, which is more computationally intensive and requires larger memory space.

2. Material point method

2.1 MPM program

The parallelisation strategy was developed based on an in-house program, MPM-GeoFluidFlow, which stems from an open-source package, Uintah (http://uintah.utah.edu/), and features a novel contact algorithm 'Geo-contact'[50], as well as a particle reseeding technique[51]. Geo-contact, specialised for soil-structure interactions, was developed from the conventional contact algorithm with enhancement of a penalty function[5, 13, 50, 52-53]. The explicit updated Lagrangian calculation in each incremental step was based on the uGIMP method[40, 54]. Meshes with identical sizes of square elements were used[18, 41], and unstructured elements can be found in Reference [22, 23]. The definition of the stresses and strains followed finite strain theory taking account of the incremental rotation of the configurations between time steps for objectivity: the stresses were measured with the Cauchy stress and updated with the Jaumann rate, and the strains were calculated with the deformation gradient. Applications of the programme are mainly focused on penetrometer penetration[51], submarine landslide[55], and impact dynamics[18, 41]. In this paper we only describe the framework utilised to solve the mass and momentum equations, but it can be applied straightforwardly to other boundary-value problems, such as heat flux in an energy equation[7].

2.2 Governing equations

The formulation was derived from the conservation of mass and linear momentum balance. The conservation of mass requires that the time derivative of the mass entering or leaving a specific domain is zero, which can be written in mathematical form as

$$\frac{\partial \rho}{\partial t} + \rho \nabla v = 0 \tag{1}$$

in which ρ is the material density, v is the velocity and t is the time. In the MPM, Equation (1) is satisfied naturally by discretising the objects into a cloud of Lagragian particles with consistent masses and volumes[33].

The linear momentum balance means that the time-variation of the linear momentum of a material is equal to the resultant of the internal and external forces, i.e. Newton's second law of motion:

$$\rho \frac{\partial v}{\partial t} = \nabla \sigma + \rho b \tag{2}$$

in which σ is the Cauchy stress, and b is the body force. Equation (2) is the strong form of the conservation of linear momentum, which is usually difficult to achieve as a closed-form solution due to mathematical difficulties. Therefore, the weak form is derived instead, expressed as

$$\int_V \rho u \frac{\partial v}{\partial t} dV = -\int_V \sigma \nabla u dV + \int_V \rho u b dV + \int_V u T dS \tag{3}$$

in which u is the virtual velocity, V and S are the volume and surface area, and T is the prescribed surface traction. Numerical integration is adopted with the simplification of lumped mass, producing a concise form

$$ma = F^{\text{ext}} + F^{\text{int}} \tag{4}$$

where m is the lumped mass, a is the acceleration, F^{ext} and F^{int} are the external and internal forces, respectively.

2.3 Numerical procedures

The explicit integration scheme was adopted to solve the governing equations. The history-dependent information carried by particle p are: position X_p, mass m_p, volume V_p, density ρ, velocity v_p, deformation rate D_p, vorticity W_p, stress σ_p, and external force f_p^{ext}. The governing equations (3) and (4) are solved on element nodes in terms of variables interpolated from the particles, i.e. mass m_i, velocity v_i, momentum M_i, acceleration a_i, internal force F_i^{int}, external force F_i^{ext}, normal direction ω_i^{norm} and tangential direction ω_i^{tang}, where the subscript i represents the node number. For the soil-structure interaction problems, the structure is simplified as a rigid body. The main functions within each incremental step are:

(i) Initialisation of nodal variables. The time step always starts with the initialisation of the nodal variables of the structure and soil, which will be automatically abandoned at the end of the step.

(ii) Interpolation from particles to nodes. The masses and momenta of the associated particles (inherited from the previous incremental step) are interpolated to the nodes

$$m_i = \sum_p S_{ip} m_p \tag{5}$$

$$M_i = \sum_p S_{ip} m_p v_p \tag{6}$$

$$\omega_i^{\mathrm{norm}} = \frac{\sum_p \nabla S_{ip} m_p}{\left\| \sum_p \nabla S_{ip} m_p \right\|} \tag{7}$$

where S_{ip} and ∇S_{ip} are the shape function and its gradient at node i evaluated at particle p, respectively[40]; $\sum_p$ represents the summation over all related particles. The derivation of the normal direction ω_i^{norm} in Eq. (7) can be referred to in Reference [52-53]. For the soil, the internal force is obtained

$$F_i^{\mathrm{int}} = -\sum_p \nabla S_{ip} \sigma_p V_p \tag{8}$$

The tractions on the Neumann boundary is calculated[56-57]

$$F_i^{\mathrm{ext}} = \sum_p S_{ip} f_p^{\mathrm{ext}} V_p \tag{9}$$

(iii) Calculate nodal velocities and accelerations. The velocities and accelerations on the background mesh can be obtained. At the commencement of the incremental step, the velocity of the node is

$$v_i = \frac{M_i}{m_i} \tag{10}$$

The acceleration for the soil node from the internal and external forces can be calculated from the governing equation as

$$a_i = \frac{F_i^{\text{int}} + F_i^{\text{ext}}}{m_i} \tag{11}$$

Then the nodal velocity is updated as

$$v_i' = v_i + a_i \Delta t \tag{12}$$

where Δt is the time increment and determined through the Courant–Friedrichs–Lewy stability condition

$$\Delta t = \frac{\varphi h}{\sqrt{(\lambda + 2G)/\rho}} \tag{13}$$

where φ is the Courant number, h is the size of the square element, and G and λ are the Lamé's parameters.

For the soil node in contact with a structure moving with a prescribed velocity v_0, v_i' is further adjusted depending on the adopted contact algorithm 'Geo-contact'[50]. The soil may be in contact with the structure if the soil mass projections are non-zero within the predefined area of the structure. For a specific node i of the soil in contact, its normal relative velocity to the structure is $\Delta v_i^{\text{norm}} = (v_i' - v_0)\,\omega_i^{\text{norm}}$, with v_0 as the velocity of the structure. Node i of the soil can be distinguished as approaching or departing from the structure with the relative normal velocity

$$\begin{aligned} &\Delta v_i^{\text{norm}} > 0, \text{ approach} \\ &\Delta v_i^{\text{norm}} < 0, \text{ depart} \end{aligned} \tag{14}$$

The normal contact strategy between the soil and the structure is realised by adjusting the normal relative velocity by $\Delta v_i^{\text{norm},*}$: (i) for soil node i approaching the structure, the normal relative velocity is eliminated; and (ii) for soil node i departing from the structure, the normal relative velocity is eliminated only if no separation between the structure and the soil is considered (otherwise, the normal relative velocity is maintained).

The relative tangential velocity of the soil node i to the structure is

$$\Delta v_i^{\text{tang}} = \left(v_i' - v_0\right)\omega_i^{\text{tang}}$$
$$\omega_i^{\text{tang}} = \omega_i^{\text{norm}} \times \frac{\left(v_i' - v_0\right)\times\omega_i^{\text{norm}}}{\left|\left(v_i' - v_0\right)\times\omega_i^{\text{norm}}\right|} \tag{15}$$

where function '×' represents the cross product. The shear along the interface is governed by the Coulomb friction law, i.e. the adjusted tangential relative velocity $\Delta v_i^{\text{tang,*}}$ is bounded by $\mu_{\text{c}}\Delta v_i^{\text{norm,*}}$, in which μ_{c} is the Coulomb friction coefficient. In geotechnical applications involving soils with low permeability, a threshold value of the friction stress is usually applied for total stress analyses under undrained conditions

$$\tau = \alpha s_{\text{u}} \tag{16}$$

where τ is the maximum shear stress along the interface and α is the limiting shear stress ratio, ranging from 0 to 1. So the tangential relative velocity will be adjusted by

$$\Delta v_i^{\text{tang,*}} = \min\left(\Delta v_i^{\text{tang}}, \mu_{\text{c}}\Delta v_i^{\text{norm,*}}, \frac{\alpha s_{\text{u}} A_i \Delta t}{m_i}\right) \tag{17}$$

where A_i is the interface area represented by node i.

A penalty factor β_i is then introduced to the overall adjustment of the relative velocity Δv_i^{adju} to obtain a smooth reaction force

$$\Delta v_i^{\text{adju}} = \beta_i\left(\Delta v_i^{\text{norm,*}} + \Delta v_i^{\text{tang,*}}\right)$$
$$\beta_i = 1 - \left(\frac{\min\left(s_i, h\right)}{h}\right)^k \tag{18}$$

where s_i is the distance from node i to the surface of the structure and k is the penalty power. The total contact force on the structure is

$$P = \sum_i \frac{m_i \Delta v_i^{\text{adju}}}{\Delta t} \tag{19}$$

The new velocity of node i is $v_i^{\text{new}} = v_i' + v_i^{\text{adju}}$. Roller (Neumann) boundary condition can be imposed by removing the new nodal velocity normal to the boundary. Then, the overall acceleration for the current time step at soil node i is

$$a_i^{\text{new}} = \frac{v_i^{\text{new}} - v_i}{\Delta t} \tag{20}$$

(iv) Update particle state. The strains of the soil particles are calculated with the deformation gradient using an updated formulation

$$F_p^{\text{new}} = f_p F_p \tag{21}$$

where f_p is the relative deformation gradient

$$f_p = I + \sum_i \nabla S_{ip} v_i^{\text{new}} \tag{22}$$

with I indicating the identity matrix. The stresses and material properties of the soil particles are calculated using an elastic-perfectly plastic constitutive model with the deformation rate D_p and vorticity W_p

$$D_p = \frac{1}{2}\left[\sum_i \nabla S_{ip} v_i^{\text{new}} + \left(\sum_i \nabla S_{ip} v_i^{\text{new}}\right)^{\text{T}}\right] \tag{23}$$

$$W_p = \frac{1}{2}\left[\sum_i \nabla S_{ip} v_i^{\text{new}} - \left(\sum_i \nabla S_{ip} v_i^{\text{new}}\right)^{\text{T}}\right] \tag{24}$$

where the superscript T means the transposition of a tensor. The definition of the stresses follows finite strain theory taking account of the incremental rotation of the configurations between time steps for objectivity, the trial stresses being measured with the Cauchy stress and updated with the Jaumann rate according to

$$\sigma_p^{\text{trial}} = \sigma_p + \Delta t\left[\left(\sigma_p W_p - W_p \sigma_p\right) + C D_p\right] \tag{25}$$

where $\boldsymbol{C}$ is the fourth-order stiffness tensor. The trial Cauchy stresses should satisfy the von Mises criterion

$$f = \sqrt{2J_2} - \sqrt{\frac{2}{3}} s_{\text{u}} \le 0 \tag{26}$$

where J_2 is the second deviatoric stress invariant. Otherwise, the trial Cauchy stresses will be updated with radial return mapping as the Mises yield surface is circular in the π plane.

In addition, the velocities and positions are updated by mapping the nodal accelerations and velocities

$$v_p^{\text{new}} = v_p + \sum_i S_{ip} a_i^{\text{new}} \Delta t \tag{27}$$

$$X_p^{\text{new}} = X_p + \sum_i S_{ip} v_i^{\text{new}} \Delta t \tag{28}$$

For the structure moving with a prescribed velocity v_0, its velocity is unchanged and the new position is updated by addition with $v_0 \Delta t$.

3. Multiple-GPU platform

Expansion from a single-GPU into multiple-GPU parallelisation can be categorised into two directions: within a single computer and across multiple computers[48]. This paper focuses on the former (Figure 1). A GPU has different memory hierarchies, such as the register, texture, constant, shared, local and global memories. The global memory, the largest memory on each GPU, is the main space to save the variables in the calculations. Access to the global memory from the multiprocessors needs to be coalesced on aligned contiguous memory addresses to achieve a high bandwidth. The GPUs are plugged on the PCI-E slots on the motherboard of the host computer and connected with the shared RAM via quick-path interconnect (QPI), shaping a shared-distributed hybrid memory hierarchy. Computational tasks shared between the GPU devices necessitate data manipulations between the RAM and the GPU memories, which is through the PCI-Express bus and controlled by the PCI-E controller element on the CPU.

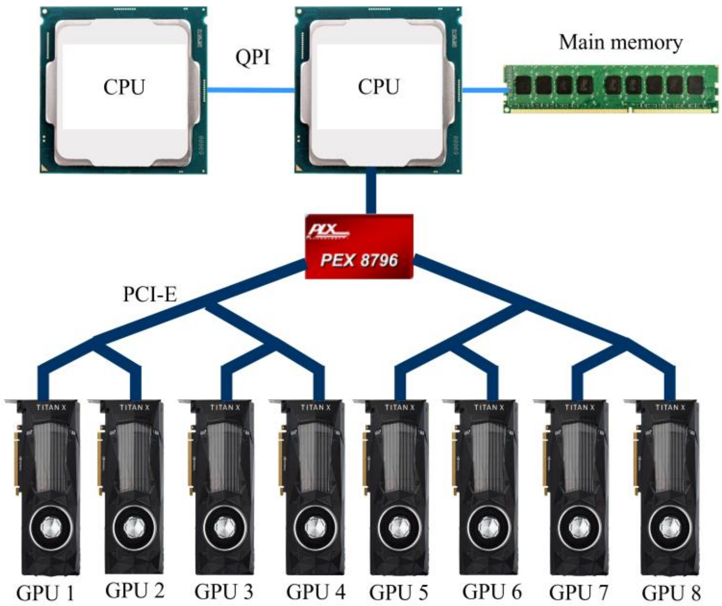


Figure 1 Configuration of a multiple-GPU system with a single root complex (Reference: https://www.servethehome.com/single-root-or-dual-root-for-deep-learning-gpu-to-gpu-systems/)

Conventional communication between the GPUs was traditionally in procedures with MPI and CUDA commands[48]: (i) read the information on the global memory of the local GPU; (ii) copy to the RAM on the computer via PCI-E (also implicitly via CPU and QPI); (iii) write to the global memory of the remote GPU from the shared RAM. Overhead on the communication is often the bottleneck of parallelisation since different components are involved, especially for frequent synchronisation between the GPUs in each incremental step. Also, the hybrid MPI-CUDA environment causes extra complexities to the coding and increases the risk of failure in the execution. In modern PCI-E architectures, GPUs connected to the same PCI-E root are allowed to access the global memory of each other with the P2P communication technique without using the RAM as a transit storage. Therefore, MPI send/receive manipulations as in Reference [48] can be avoided with more conveniences and higher performance.

4. Parallelisation of MPM

Parallelisation of the in-house program MPM-GeoFluidFlow was specially tuned for a single-GPU[46] and a multiple-computer multiple-GPU frameworks[48]. In this work, the program was parallelised on the platform of a one-computer multiple-GPU system with a single-root PCI-E complex (Figure 1). The parallelisation was performed purely within the CUDA environment without the MPI interface required in the conventional multi-GPU parallelisation[46, 48]. Improvements were made on the data transport scheme between the shared RAM and the member GPUs, which was boosted by taking advantage of the P2P technique supported on the single-root complex PCI-E architecture. Parallelisation of the function ‘Interpolation from particles to nodes’ was optimised with an enhanced ‘Particle-List’ scheme, which is parallelised on the GPUs. As a result, the MPM computation was further accelerated and the complexity of the code was reduced. The original two dimensional framework was extended to three dimensional.

4.1 Task distribution and assembly

In the pre-processing stage, the whole domain of the task is assigned over the shared RAM on the computer, with the material discretised into a cloud of particles and a structured mesh constructed. The history-dependent information carried on the particles and the temporary variables for the element nodes are declared. Then the computational domain is evenly decomposed into a number of subdomains to distribute the entire task onto the individual GPUs (Figure 2). The number of elements in each subdomain is around M/n, where M is the total number of elements in the whole domain and n the total number of subdomains. The discretised particles are associated with the background subdomain by their locations. Variables of the particles and the element nodes in each subdomain are copied from the computer RAM to the global memory of each GPU through the PCI-E and QPI. Therefore, the space of the shared RAM should be larger than the total size of the global memories of the hosted GPUs. Two additional layers of ghost elements in each direction are generated out of the computational subdomain, which is to maintain the continuity of the information at the border of the subdomain. The roller (Neumann) boundary condition is implemented on the outer boundaries of the outer subdomains. Calculations of the MPM algorithm within each subdomain is parallelised on the corresponding GPU within the CUDA environment. The interpolated information on the overlapping ghost element nodes from the neighbouring subdomains will be added and saved on both sides into the GPU global memories. Therefore, the neighbouring subdomains are essentially boundary conditions to each other. After the computation, the information of the particles in the subdomains is re-assembled into the RAM and will be used for post-processing.

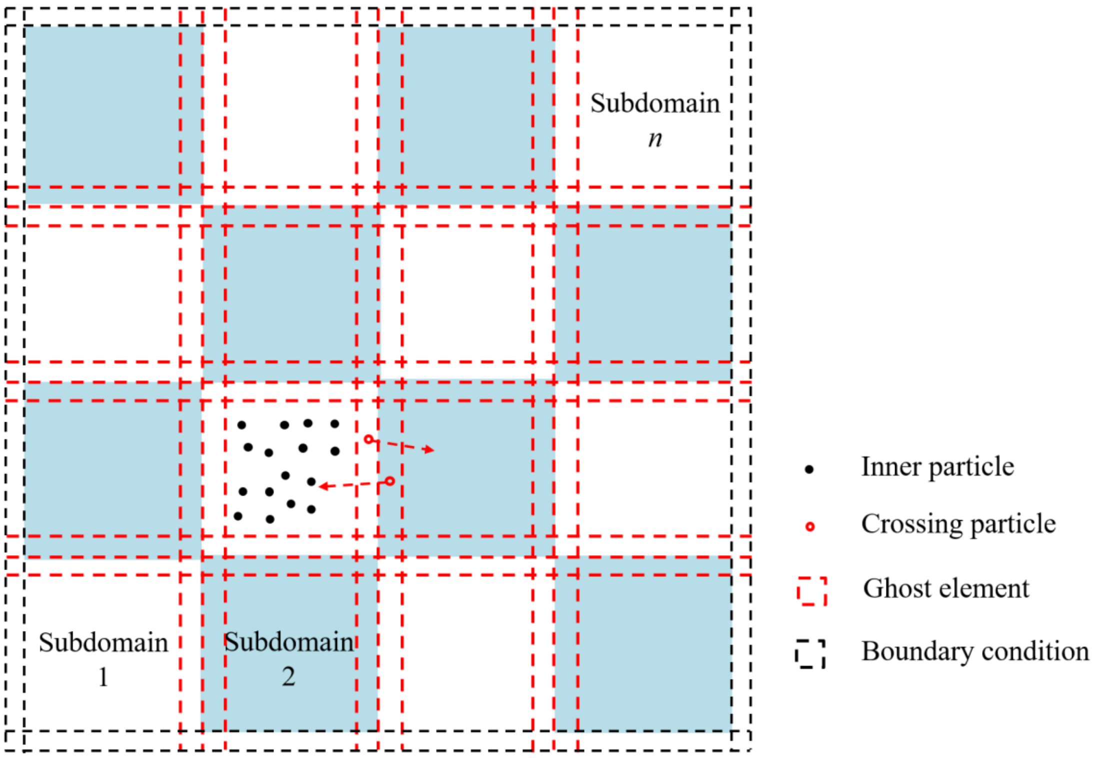


Figure 2 Domain decomposition in multiple-GPU parallelisation

4.2 Parallelisation on each GPU

In order to control the overhead of the frequent data transfer between the RAM and the GPUs, all the variables of the particles and nodes required in the essential calculations of Eqs. (5) – (28) are reserved on the global memory of the GPUs, and are accessible from the active multiprocessors. The functions 'Initialisation of nodal variables' and 'Calculate nodal velocities and accelerations' (Eqs. (10) – (20)) are parallelised over the nodes, i.e. updating the information of one node on one GPU core (Figure 3). In the thread for node *i*, only the information of the node is involved in the calculations as in Eqs. (10) – (20), which means the parallel computations in the GPU cores are independent to each other. Therefore, the two functions present relatively high parallelisability and are straightforward to parallelise over the nodes. Particularly for the soil-structure interactions by Eqs. (14) – (18), the velocity adjustment on each node, controlled by its kinematic state relative to the structure, is independent to each other and can be parallelised over the nodes. As a whole system, the total reaction force on the structure (Eq. (19)) will be collected from the contacting nodes on the GPUs and superimposed on the CPU. For the function 'Update particle state' (Eqs. (21) – (28)), the workload is decomposed over the particles, for each of which the interpolated information are superimposed from the surrounding nodes sequentially. Different to the writing operations, which may induce data race in the memory and will be discussed later, reading data from the identical memory address by different threads is allowed in the parallelisation. Furthermore, a high bandwidth can be achieved when the multiprocessors reading the consecutive memory addresses. Hence, the function 'Update particle state' is expected to have a high acceleration effect by the parallelisation.

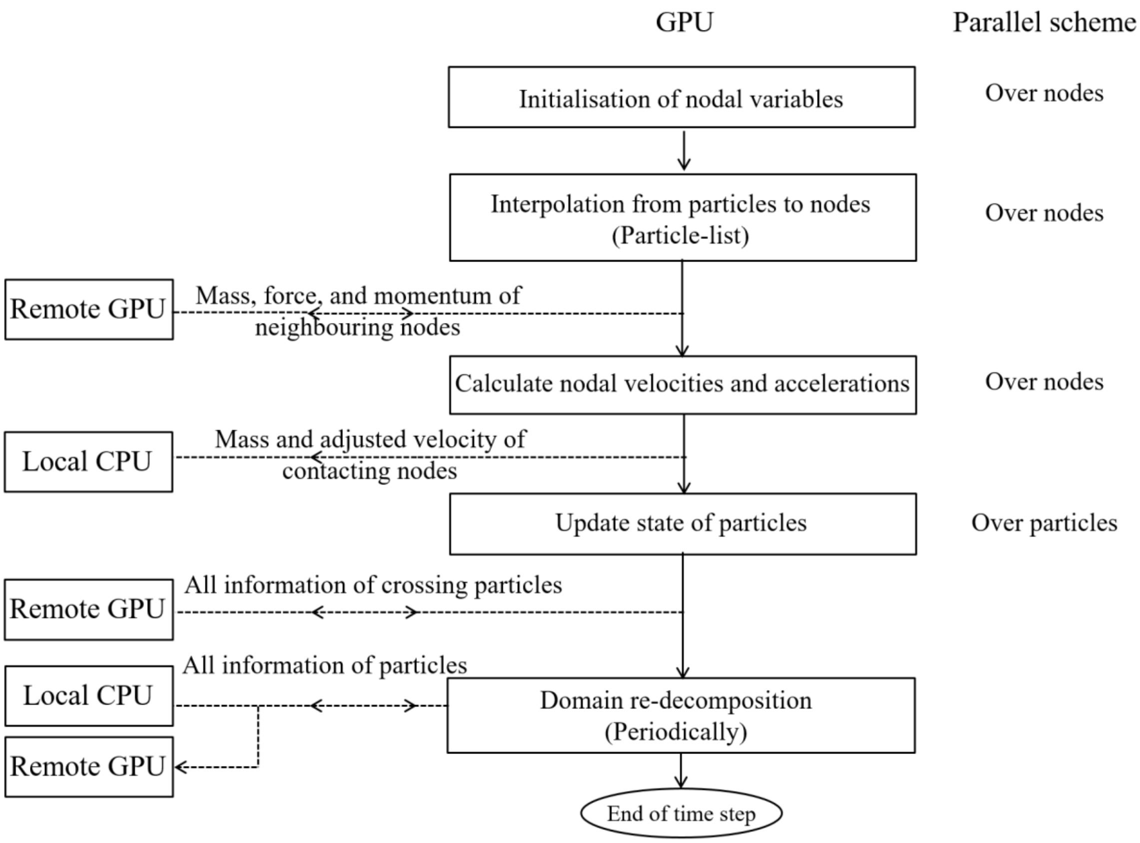
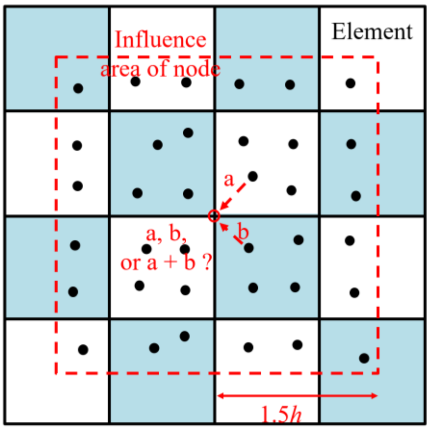
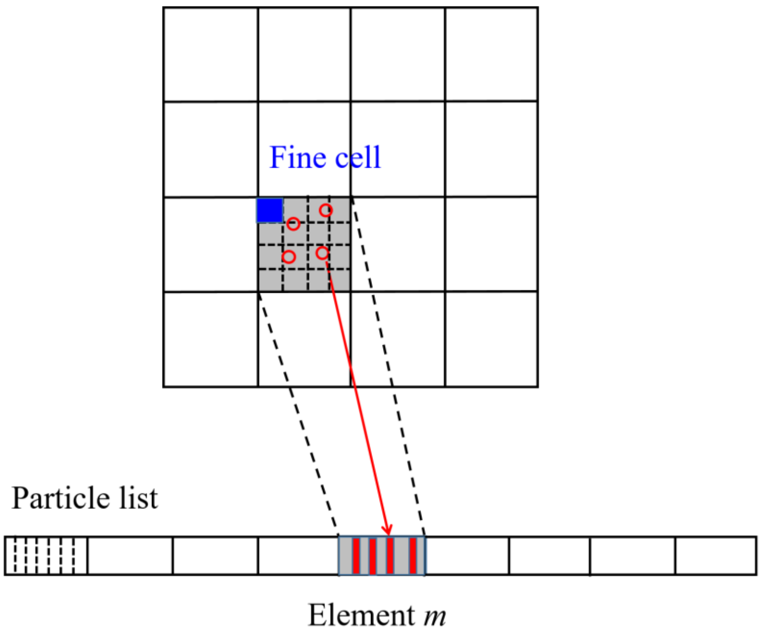


Figure 3 Essential operations in multiple-GPU parallelisation of MPM

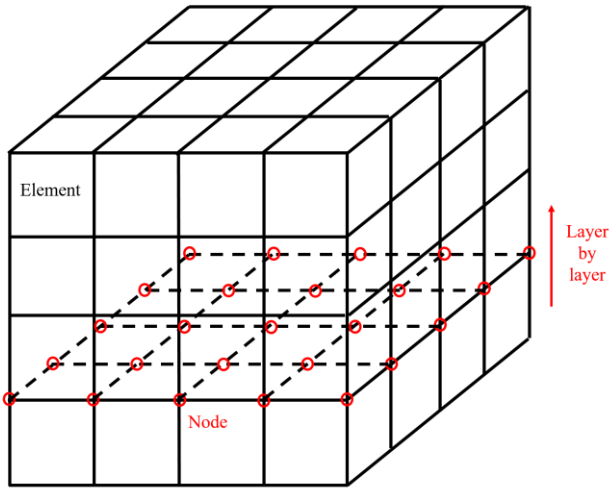
In contrast, the function 'Interpolation from particles to nodes' is more difficult to parallelise. If the function is simply parallelised over the particles, i.e. the interpolation from each individual particle to its related nodes is configured on a thread, data race can be induced by concurrently writing the interpolation outcomes from different GPU cores into identical addresses of the global memory (Figure 4a). The data race makes the final writings to the memory unpredictable, which means the interpolations would be erroneous. Reference [47] avoided the data race by using the atomic operations, which is often supported in the modern GPUs. However, the atomic operation is not recommended as it is essentially a sequential writing action to the memory, which undermines the overall acceleration effect. In Reference [49], a special technique 'Particle-Grid offset' was developed to tackle the data race; however, a series of complicated operations are involved to make the algorithm parallelisable. In Reference [46] and [48], the problem was solved by parallelising the function over the nodes, for each of which the interpolation outcomes from associated particles were superimposed sequentially. Before the parallel computation, a particle list for each node needs to be constructed, which was saved on the RAM and was transported to the GPU periodically. Comparing with the ‘Particle-Grid offset’ technique, the ‘Particle-List’ scheme seems to be more accessible as only two steps (generation of particle list and interpolation operation) are required. However, the updating frequency of the particle list, often determined by experience and through trial calculations, depends on the mesh size and the intrinsic characteristics of the problem to be analysed and may be very high (such as once for five incremental steps) for impact problems. In three-dimensional analysis, each node may be associated with around 216 particles (6 × 6 × 6; Figure 4a) as regulated by the uGIMP shape function[40, 54]; therefore, a particle list relating the surrounding particles for all the nodes would exhaust the memory space on the GPU, which is typically upper bounded by 24 GB.



(a) Data race



(b) Particle list



(c) Parallelisation by layers

Figure 4 Parallelisation of the function 'Interpolation from particles to nodes'

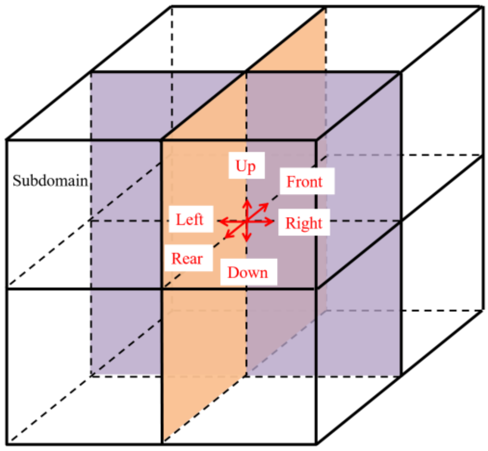
In this work, the 'Particle-List' scheme was improved by generating the particle list for each element rather than the node, which means the total memory requirement by the particle list is $O(N_p)$ and much smaller than that by the original scheme $O(216N_i)$, where N_p and N_i are the total number of the particles and nodes, respectively. The generation of the particle list is parallelised with the GPUs instead of the CPU sequential operations in the original scheme. A fully engaged element often accommodates 4 or 16 particles[40, 41]. To avoid the potential data race, an expanded particle list of each element is adopted, which was developed from a similar method used in the creation of contact pair list in the discrete element method[58]. The original elements are evenly divided into a number of finer cells (Figure 4b). Each fine cell corresponds to a specific memory address of the particle list, and accommodates only one particle or less[59]. Then, the particle sorting operations can be parallelised across the GPU cores over the particles, in which each memory address is written with only one or less particle ID without the risk of data race. The number of the fine cells in each element, dependent on the smallest distance between the particles controlled by the strains, can be determined through trial calculations and $8 \times N_{PPC}$ is often acceptable, where N_{PPC} represents the particle number in per element. For specific problems with extreme strains of material, the number of the fine cells can be increased and a novel technique in Reference [60] reseeding the particles in the elements is also suggested. The particle list is sparse due to the large number of empty fine cells out of the engaged ones (Figure 4b), which still means a heavy memory requirement. An additional compression step is then performed to obtain a dense particle list. The enhanced 'Particle-List' scheme is fully performed on the GPUs and need no data transport between the RAM and the GPUs. Therefore, improvement of the speedup of the parallelisation is expected with the enhanced scheme.

The interpolations from the particles to nodes are then parallelised over the nodes. The particles on the particle list of the surrounding elements within the influence range of each node are involved in the interpolation on one GPU thread. In consideration of the heavy requirement of memory space for the particle list of large-scale problems, the generation of particle list and the interpolations are performed in layers of nodes (Figure 4c).

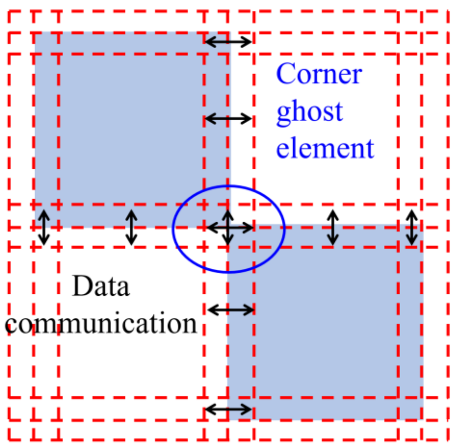
4.3 GPU communications

In order to keep the continuity of the stress and strain field between the connected subdomains, the interpolated variables on the border nodes within the neighbouring subdomains, including the mass, momentum and internal force, by the function ‘Interpolation from particles to nodes’ is superimposed at each incremental step (Figure 3). The operations are implemented with the direct P2P operations between the GPU memories for the single-root complex PCI-E framework, which owns much higher bandwidths (5 GB/s) than that used in Reference [48] through the RAM and the infiniBand network (1.25 GB/s) within MPI environment (Figure 1). The send and receive operations can be performed in bi-direction with a modern PCI-E, which doubles the intrinsic bandwidth of the data migration to 10 GB/s. Furthermore, the transfer of data between the GPUs can be hidden by the essential calculations on each GPU, which means overhead on the synchronisation process is virtually nil and a perfect scaling may be possible. In comparison, the data transfers within MPI environment accounts for about 5% of the total computational effort[48]. The exchanged information of the ghost element nodes from the neighbouring GPUs is added to the counterparts on the local GPU (Figure 2).

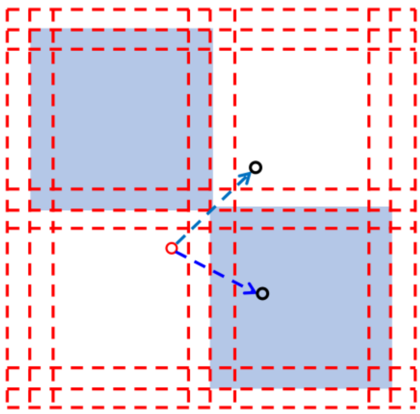
The particles may move across the subdomains and hence migrate from a GPU to its neighbour, for which all the information of the migration particles should be transferred to the new subdomain with P2P operations similar to that exchange the neighbouring nodal information. The subdomains communicate with their neighbours in six directions (left-right; front-rear; up-down) (Figure 5a). For most ghost elements (Figure 5b) and particle migrations (Figure 5c), send/receive operations are performed between two subdomains in two directions; there are also some corner ghost elements are shared by four or eight subdomains and some particles moves to unconnected subdomains (such as upper-right subdomain), which need more than one synchronisation step (Figure 5). The particle migration can be time-consuming if performed in every incremental step. If a particle moves from the current subdomain to the neighbouring one between the particle migration operations, the interpolations from the particle are allocated to the ghost element nodes of the current subdomain, which will be synchronised between the neighbouring subdomains as described previously. Therefore, the interpolation results are accurate if only the particles do not move across the outer layer of ghost elements of the current subdomain. The one layer of ghost elements essentially functions as a buffering zone of the particle migration between the subdomains. Therefore, the particle migration is conducted for once in a number of steps in this study, which is controlled by the time of particles move across the outer ghost element and depends on the mesh and particle discretization. In the calculations, the number of step to migrate the particles can be determined by an experimental computation, which is often selected as 5 in this study.



(a) GPU communications between subdomains



(b) Corner ghost elements



(c) Particle migration

Figure 5 GPU communications

In the calculations, void areas tend to be formed in the subdomains due to the large deformation of materials, e.g. in the mini-slump test in Section 5.1, which idle the computational resource. The migration of the particles between the subdomains may also undermine the workload balance among the GPUs. A simple procedure of domain re-decomposition is then performed to updates the subdomain dimensions periodically at intervals of a large number of incremental steps, such as 50,000. The communication overhead for the neighbouring nodes and the migration particles between the subdomains may be non-ignorable but acceptable in many cases, which can be compensated by the substantial calculations in each GPU. The information of all the particles in the subdomains are gathered from the GPUs' global memories to the shared RAM of the computer. The upper and lower boundaries of the material are derived from the particle coordinates, based on which the boundaries of the computational domain in the following steps are updated. Then the task distribution operations in Section 4.1 is re-performed by evenly decompose the computational domain. The present domain de-composition procedure is mainly to solve the problem of void areas, and somehow mitigate the workload imbalance between the GPUs. More advanced technique with adaptive subdomain sizes to balance the workloads between the GPUs will be developed in the future work.

5. Performance assessment

The accuracy and convergence for the standard MPM algorithm with explicit calculations have been assessed in Reference [32, 35], which may be inherited here due to the trivial modifications. The benchmark cases, mini-slump test and cone penetration test, were simulated to assess the acceleration effect of the multiple-GPU parallelisation scheme. The parallel computations were performed on a single-computer server, which hosts 8 NVIDIA Titan Xp GPUs based on a single-root complex (Figure 1) and 2 Intel Xeon E5-2687WV4 CPUs. Each CPU has 12 cores with frequency of 3.0 GHz; on each GPU, a total number of 3840 cores are accessible and the dedicated global memory is 12 GB; The RAM space of the server is 256 GB. The operating system was Ubuntu 18.04, the C++ compiler was gcc 5.3.0 and the GPU compiler was CUDA v8.0.44. All the computations were based on double-precision numbers to guarantee the accuracy.

The soil was considered to be an elastic-perfectly plastic material and regulated with the von Mises yield criterion. The Poisson's ratio of the soil was selected as 0.49. The time step was calculated by

$$\Delta t \leq \frac{\alpha d}{\sqrt{(\lambda + 2G)/\rho}} \tag{29}$$

where G is the shear modulus of soil, λ is the Lamé constant, d represents the mesh size, and α is the Courant number. The 'speedup' factor was to characterise the acceleration effect of the parallel computations: Speedup = $T_{Sequential}$ / $T_{Parallel}$, where $T_{Sequential}$ and $T_{Parallel}$ are the runtimes of the CPU sequential and GPU parallel calculations over a number of incremental steps (Appendixes A and B).

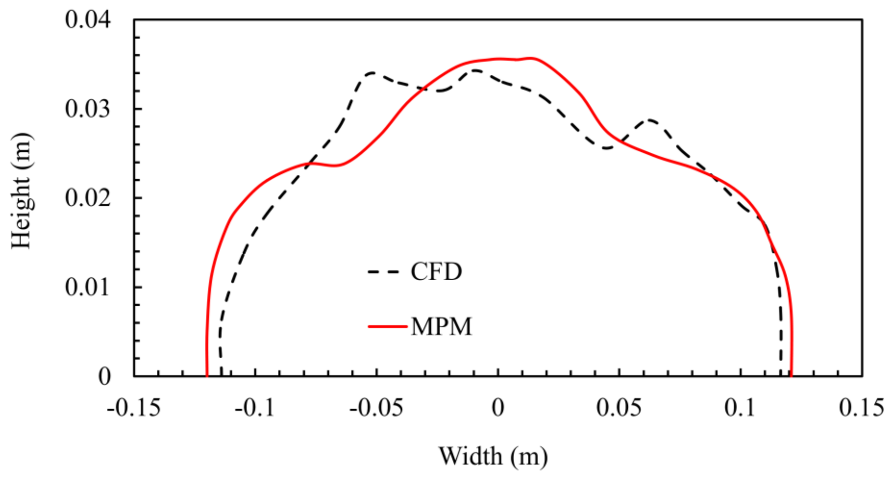
5.1 Mini-slump test

Submarine landslides are known as one of the most hazardous threat to the submarine structures, featuring enormous volumes of sediments running at very high velocities and reaching very far runout distances before final deposition[61]. The sliding behaviour of the soil can be studied in laboratory through the mini-slump test, in which the soil is released from a cylinder and then runout along a flat base. The test performed by Reference [62] with the remoulded soil from Heimdal, Norway was simulated using the MPM (Figure 6a). The cylinder had a height H of 120 mm and a diameter of 100 mm. The mechanical behaviour of the soil was considered by a Herschel-Bulkley (H-B) model[63]

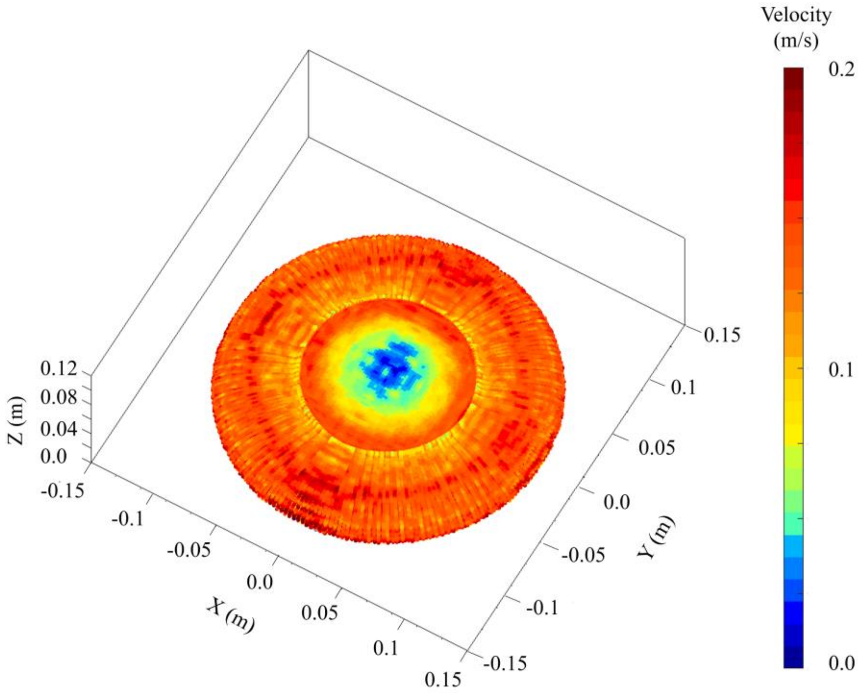
$$s_{\mathrm{u}} = s_{\mathrm{u0}} + K\gamma^{n} \tag{30}$$

where s_u is the undrained shear strength of the soil, s_{u0} is the threshold shear strength s_{u0} = 200 Pa, γ the shear strain rate, K the consistency coefficient and n the shear-thinning index. In the experimental test, the parameters in Eq. 2 were determined as s_{u0} = 200 Pa, K = 15 Pa·s^n, and n = 0.35. In the MPM analysis, the mesh size d was selected as H/60, which was validated to be sufficiently fine in viscous sliding problems of soil as it presents similar results with a finer mesh H/120[18, 41]. The Young's modulus was 500 times the undrained shear strength s_u. In total, there were 393,216 slurry particles. The flat base was assumed as a no-slip boundary. The time step Δt was determined with a Courant number of 0.3. Another numerical simulation using the computational fluid dynamics (CFD) method was also performed by Reference [64]. Figure 6b shows the final morphologies of the soil predicted by the CFD and MPM analyses, which matches well with each other. The slumped width of the soil was 0.12 m in the experiment and the final height was 0.04 m, which are close to that of the numerical predictions. The profile of the soil at 0.3 s in the MPM simulation was shown in Figure 6c.

(a) Laboratory test[65]



(b)



(c) 0.3 s

Figure 6 Mini-slump test and runout morphologies

To investigate the acceleration effect of the GPU parallel framework, the particle number N_p was increased from 393,216 to 1,572,864, 2,102,203, 4,343,424, 19,232,055, 34,744,320, 50,035,200 and 121,065,216. The total memory space engaged in the simulation is proportional to the particle number, which increases from 0.08 GB (393,216 particles) to 23.5 GB (121,065,216 particles). In Reference [46], the maximum particle number for a 2D model with a memory size of 4 GB was 50,035,200, and will be much less for a 3D model than the counterpart in this study. The reason is that the size of the particle list for the element nodes in Reference [46] was nearly 25 times of the total particle number, which is avoided in this study by establishing a particle list for the elements in each layer. Therefore, the maximum particle number accommodated in 8 GPUs can be up to 400,035,200 as the concern of many large-scale geotechnical problems. A total runtime within 100 incremental steps was recorded for each case with the CPU sequential and GPU parallel simulations, in which the acceleration effect by the GPU parallel strategy is clearly demonstrated (Table 1). Within the CPU sequential calculations, the runtime is linearly proportional to the scale of the case in terms of particle number. The speedup linearly increases for less than 4,000,000 particles (Figure 7); if the computational scale is enlarged further with more particles, the GPU seems to be fully loaded, and the acceleration effect presents a good scaling behaviour and converges to an average speedup of about 80. The average speedup of ~ 80 in this study is much higher than that in Reference [35] of around 20 for two main reasons: the GPU Titan Xp in this study outperforms that GTX 780M in Reference [46]; the parallelisation schemes are optimised in this study in terms of particle list and memory access. Among all the functions of the MPM, the function 'Interpolation from particles to nodes' consumes the most computational efforts for around 70%, which is due to the non-coalesced memory access when writing the interpolations to the nodal addresses. The overhead on the establishment of the particle list takes less than 2% of the total runtime, much less than that in Reference [46] and [48] of around 15%, as the operations are fully moved and parallelised onto the GPU. The remaining 28% of the computations is mainly on the function 'Update particle state', while that on the functions 'Initialisation of nodal variables' and 'Calculate nodal velocities and accelerations' are ignorable. Specifically, the average speedup for the function 'Interpolation from particles to nodes' is around 65, while it increases to about 170 for the functions 'Initialisation of nodal variables' and 'Calculate nodal velocities and accelerations'. Therefore, the function 'Interpolation from particles to nodes' is the bottleneck of the parallelisation of the MPM. Additional calculations were performed for the case with 4,343,424 particles by using naïve atomic operations to parallelise the function 'Interpolation from particles to nodes', which presents very low speedup of less than 5 due to the heavy writing conflicts between the neighbouring particles. Further experiments show that the speedup with the naïve atomic operations is not stable, which varies with data structure of the particles. Due to the very low efficiency, atomic operations are not recommended in many parallel algorithms.

The acceleration effect of the multiple-GPU parallelisation over the CPU sequential computations is shown in Table 1 and Figure 8. The total runtime for 100 incremental steps rather than one step was recorded for each case to avoid the random errors during recording. For example, in the case with 393,216 particles, the average runtime for the cases with GPU is smaller than 0.2 s, which will be significantly influenced by a small random error. The average runtime for an incremental step can be easily obtained from the total value for 100 steps. The performance of the GPU is maximised when it is fully loaded with workload. Given one or two GPUs are invoked, the GPU seems to be fully loaded with 4 million particles: for the cases with < 4 million particles on each GPU, the speedup increases with the particle number when using an identical number of GPUs. The maximum speedup with less than two GPUs is around $80N_{gpu}$ with N_{gpu} as the number of GPUs, which is similar to the average speedup predicted previously with one GPU. When four or eight GPUs are mobilised, the GPU is not fully loaded even with 30 million particles on each GPU. The reason is not very clear but is inferred to be related to the scheduling elements in the CPU. The overall speedup with less than 8 GPUs is fitted as $71.5N_{gpu}$. Although it is not meaningful to compare the speedups of the multiple-GPU parallel schemes based on different hardware and software, the speedups of about $59N_{gpu}$ and $110N_{gpu}$ in Reference [48] and [49] were obtained. In consideration of the convenient implementation and high reliability of the present parallel framework, the speedup of $71.5N_{gpu}$ is quite satisfactory. Due to the optimisation of communication between the neighbour domains as described in Section 3.4, overhead on the synchronisation of the ghost nodal information, the particle migration, and the domain re-decomposition is ignorable with less than 2% when comparing to that on the essential computations. Therefore, the parallel framework of the MPM with multiple-GPU in this study presents good behaviour in terms of hardware communications, which has been the common problem of many numerical methods. Also, the advantage of the P2P technique based on the single-root complex structure of the PCI-E is well presented.

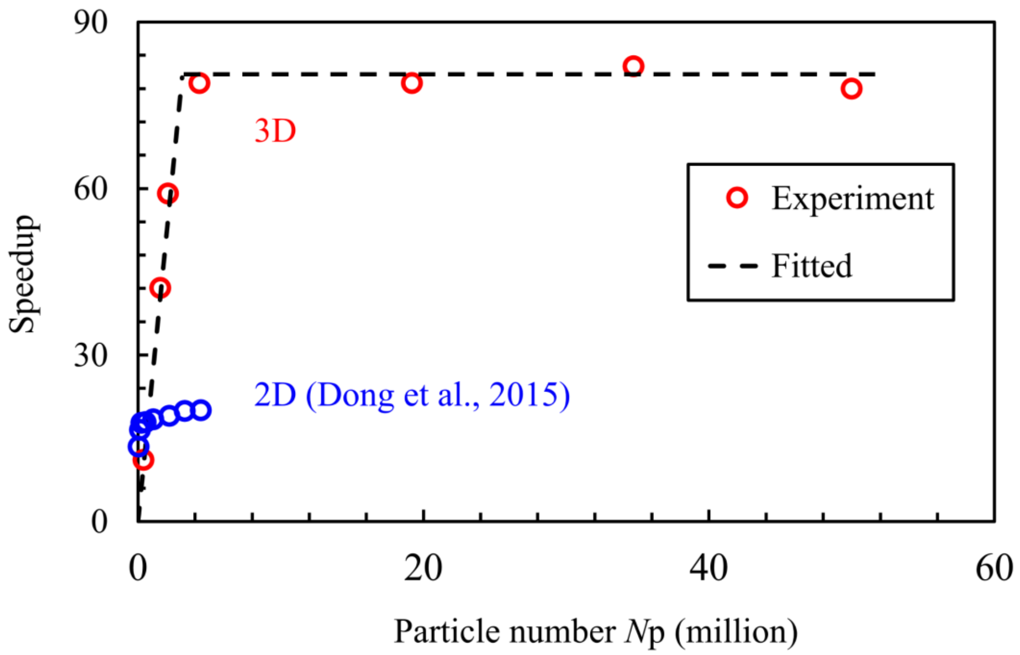


Figure 7 Speedups of GPU parallelisation with one GPU

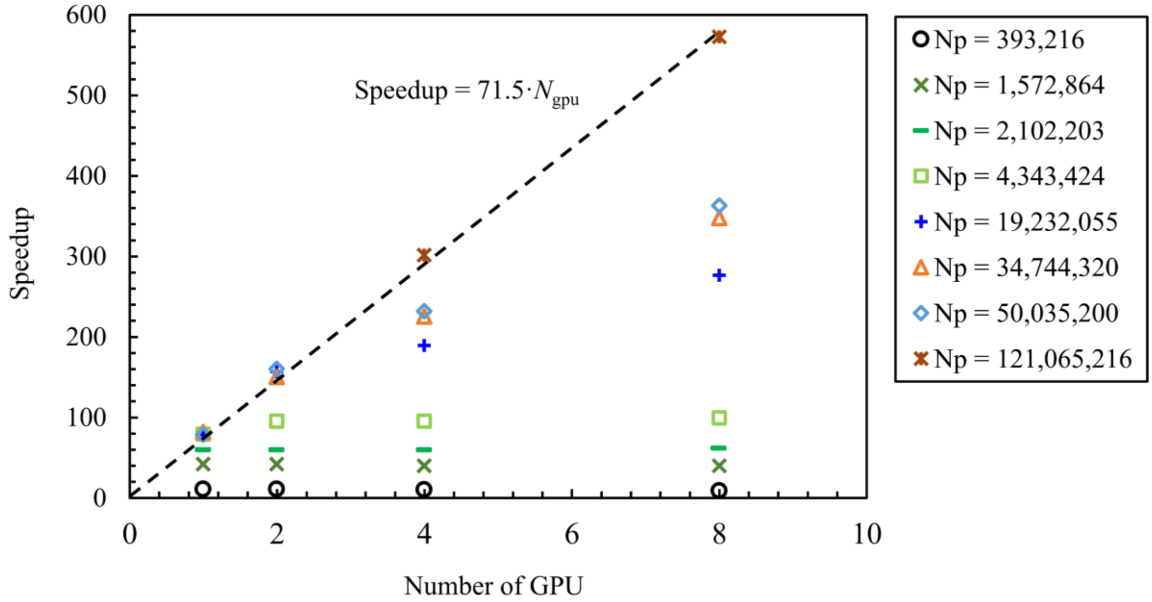


Figure 8 Speedups of GPU parallelisation with multiple-GPU

Table 1 Speedups of GPU parallel simulations in 100 incremental steps for mini-slump test cases

N_p	Memory size (GB)	CPU sequential	1 GPU		2 GPUs		4 GPUs		8 GPUs	
		Runtime (s)	Runtime (s)	Speedup	Runtime (s)	Speedup	Runtime (s)	Speedup	Runtime (s)	Speedup
393,216	0.08	180	16	11	17	11	18	10	19	9
1,572,864	0.3	760	18	42	18	42	19	40	19	40
2,102,203	0.5	921	16	59	16	59	16	59	15	61
4,343,424	0.9	1,890	24	79	20	95	20	95	19	99
19,232,055	3.2	9,552	121	79	61	157	51	189	35	276
34,744,320	6.4	19,100	232	82	127	150	85	225	55	347
50,035,200	10.2	27,400	350	78	171	160	118	232	75	363
121,065,216	23.5	78,700	—	—	—	—	261	301	137	572

5.2 Cone penetration test

The cone penetrometer has been considered as the most widely used in-situ geotechnical instrument to obtain the sequence and the physical and mechanical properties of the subsurface strata. For the cone penetrated in pure clays, the penetration resistance is related with the undrained shear strength of the soil s_u through the calibration of a bearing factor N_k, which was often investigated with theoretical, experimental and numerical analyses[65-67]. The numerical model of a cone penetration test used in Reference [68] with a large deformation finite element (LDFE) method was duplicated in this study. The standard cone had a diameter of D = 35.7 mm and its tip had an angle of 60°, as shown in Figure 9. Quarter of the model was simulated by taking advantage of its symmetry to save the runtime. In Reference [69], a smaller model with a wedge of 20° in the rotational direction was simulated, which was also proven to be accurate to represent the full model of the test. Herein, the parallel efficiency is the main concern, therefore, the quarter model was used and the wedge model will be adopted in the future applications. The chamber extensions on the horizontal and vertical directions were 2.8*D* and 8*D*, respectively. The mesh size $d = D/36$, which is satisfactorily fine to achieve a convergent prediction of N_k. In each element fully occupied by the soil, 2 × 2 particles were configured prior to the calculation. In total, 24,000,000 soil particles were discretised. The cone was assumed to be rigid and smooth. The penetration speed of the cone was taken as 2.8*D* /s (0.1 m/s), which was verified as sufficiently low to use the dynamic formulation to simulate the quasi-static process[69]. The submerged density of the soil was 1500 kg/m^3. The geostatic stresses induced by the self-weight of soil were not considered. The clay had a uniform shear strength of s_u = 10 kPa and a soil rigidity index G/s_u = 100.

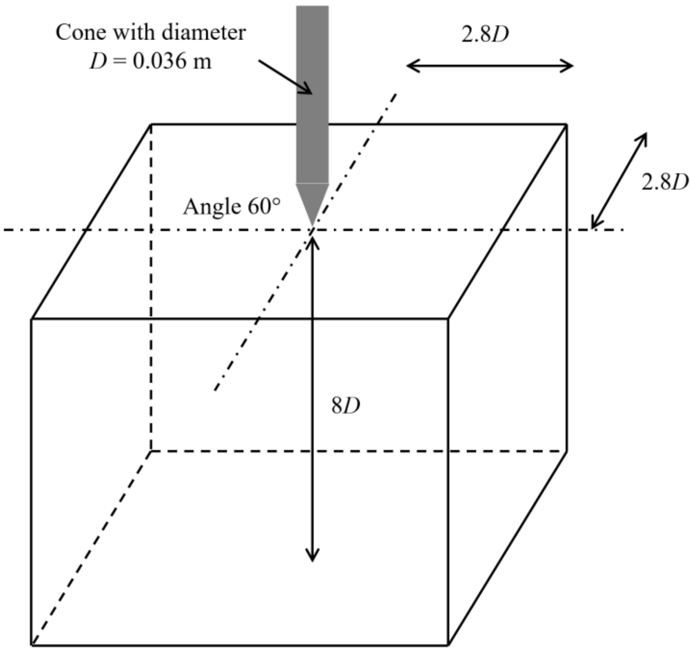


Figure 9 Setup of the cone penetration test

Profile of the bearing factor N_k is plotted versus the normalised penetration depth w/D as shown in Figure 10, in which *w* is the penetration depth of the pipeline. The bearing capacity increases with the penetration of the cone, which stabilises at about 8.97 with w/D = 3. The profile obtained from the MPM analyses has some high-frequency fluctuations of around 1.5% due to the particles below the cone crossing the element boundaries in the penetration process, which can be mitigated with finer meshes without affecting the steady values[51]. The prediction by the MPM (8.97) is slightly lower than those by LDFE (9.65) and arbitrary Lagrangian Eulerian (ALE; 9.47) methods with discrepancies within 7.5%. Predictions of the bearing capacity factor of 11.1 and 9.7 are also available with the coupled Eulerian-Lagrangian method[68] and the strain path method[65], respectively.

Therefore, the bearing capacity factor obtained with the MPM is reliable. The velocity magnitude induced by the cone penetration at $w = 3.47D$ is shown in Figure 11.

In the calculations, the total runtimes for 100 incremental step were recorded for the CPU sequential and multiple-GPU parallel computations (Table 2). The speedup with one-GPU parallelisation is about 88, which increases to 382 with 8 GPUs invoked. To investigate the acceleration effect of the GPU parallel framework, the particle number N_p was modified from 24,000,000 (6 GB) to 5,000,000, 10,000,000, 50,000,000 (12 GB), and 100,000,000. The maximum speedup of 89 is achieved for all the cases with one GPU as the GPU is fully loaded with more than 4 million particles (Figure 12a). For all the cases, the maximum speedups with different number of GPUs are fitted as $85N_{gpu}$ (Figure 12b), which is higher than that for the mini-slump test cases. That is due to the larger void areas, with none essential computations, in the mini-slump test cases when the slurry expands on the base.

To summarise from the two benchmarks, very high speedups (> $71.5N_{gpu}$) are expected with the multiple-GPU parallelisation over the conventional CPU sequential calculations. However, it is also noteworthy that the real-time speedup for specific dynamic problems (such as slurry slump) with very large deformation of the material may be undermined by many factors, such as the percentage of void elements and the bandwidth of the P2P channels in the computer. The particle migrations between the GPUs bring extra complexity and may affect the robustness of the program. That means optimisation of the program itself can be very time-consuming. Also, the multiple-GPU parallelised program suffers from a lower portability since its software framework is dedicated to the hardware platform. Therefore, the topic of multiple-GPU parallelisation of the MPM remains to be open in a near future.

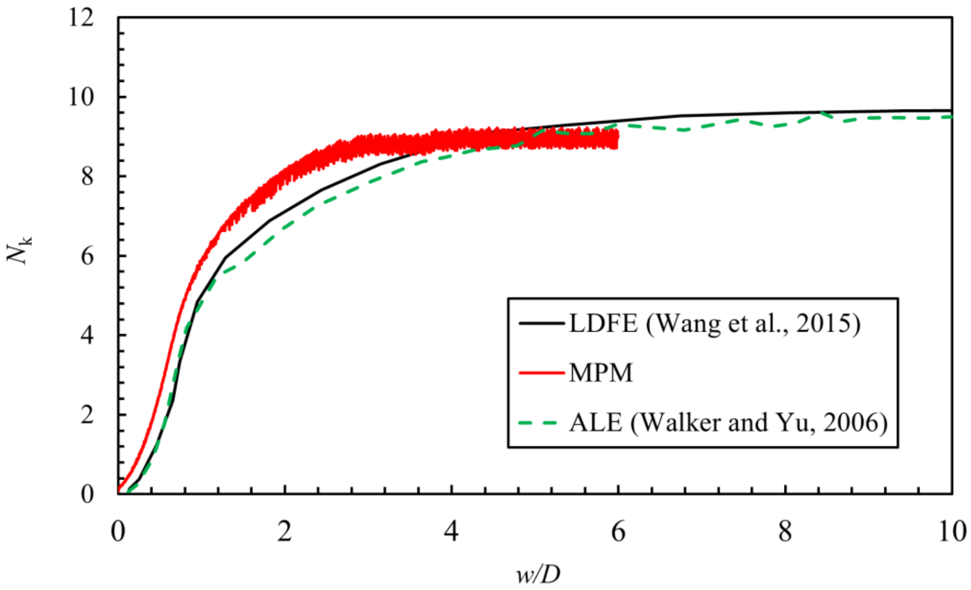


Figure 10 Profile of resistance to the cone penetration

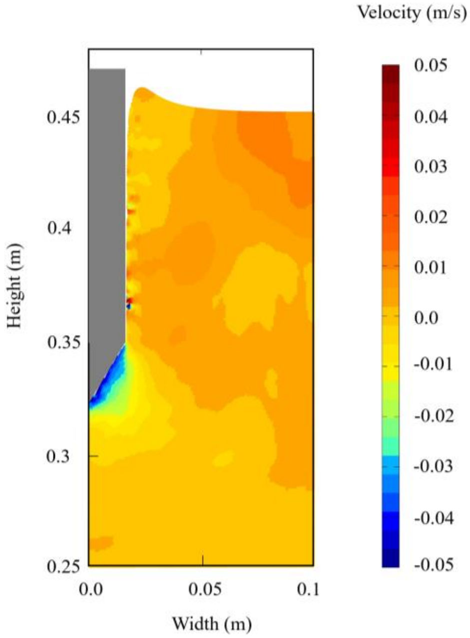


Figure 11 Velocity distribution in soil at w/D = 3.47

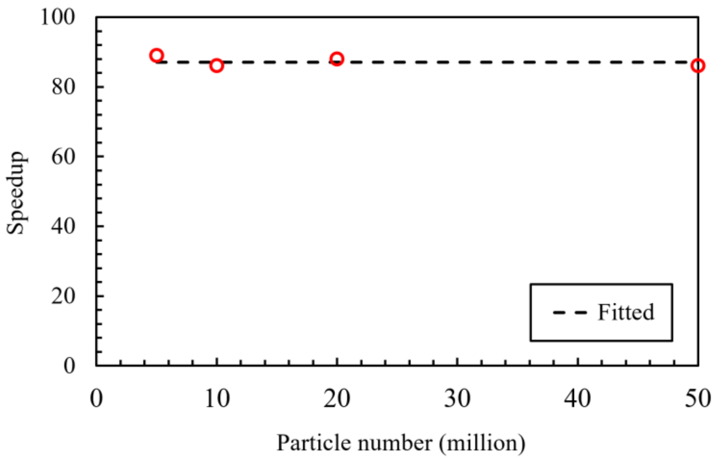
Table 2 Speedups of multiple-GPU parallel simulations in 100 incremental step for cone penetration

N_p	Memory size (GB)	CPU sequential	1 GPU		2 GPUs		4 GPUs		8 GPUs	
		Runtime (s)	Runtime (s)	Speedup	Runtime (s)	Speedup	Runtime (s)	Speedup	Runtime (s)	Speedup
5,000,000	1.2	2,882	32	89	22	128	15	188	10	282
10,000,000	2.6	5,670	66	86	40	140	29	195	18	320
24,000,000	6	13,640	155	88	82	166	59	232	36	382
50,000,000	12	27,333	318	86	165	166	102	267	54	505
100,000,000	24	55,612	—	—	—	—	158	352	92	604

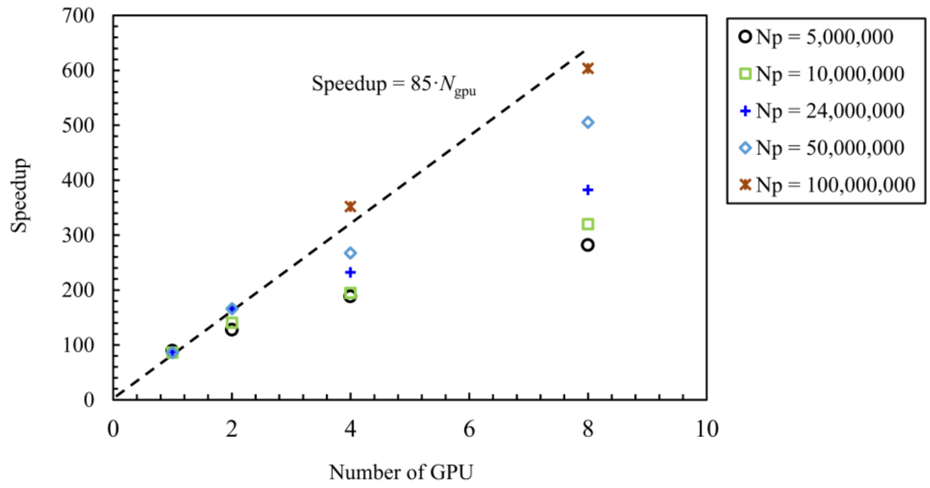
6. Conclusions

As one of the arbitrary Lagrangian-Eulerian methods, the material point method (MPM) owns intrinsic advantages in simulation of large deformation problems by combining the merits of the Lagrangian and Eulerian approaches. Significant computational intensity is involved in the calculations of the MPM due to its very fine mesh needed to achieve a high accuracy. Considering the limitations with the CPU and single-GPU performance, multiple-GPU parallelisation provides a promising means to boost the computational efficiency of the MPM. In this study, a new multiple-GPU parallel strategy was developed based on a single-root complex architecture of the computer within a CUDA environment. Peer-to-Peer (P2P) communication between the GPUs was performed to exchange the information of the crossing particles and ghost element nodes, which is faster than the heavy send/receive operations between different computers through the infiniBand network. Domain decomposition is performed to split the whole computational task over the GPUs with a number of subdomains. Within each GPU, a particle list was constructed for each node to avoid the data race when parallelising the 'Interpolation from particles to nodes'.

The acceleration effect of the parallelisation was evaluated with two benchmarks cases, mini-slump test and cone penetration test. The maximum speedups with 1 GPU was 88, and increased to 604 using 8 GPUs. Among all the functions of the MPM, the function 'Interpolation from particles to nodes' consumes the most computational efforts for around 70%, which is due to the non-coalesced memory access when writing the interpolations to the nodal addresses. The overhead on the establishment of the particle list takes less than 2% of the total runtime, much less than that in Dong et al. (2015) and Dong and Grabe (2018) of around 15%, as the operations are fully moved and parallelised onto the GPU. The remaining 28% of the computations is mainly on the function 'Update particle state', while that on the functions 'Initialisation of nodal variables' and 'Calculate nodal velocities and accelerations' are ignorable.



(a) One GPU (Np = 24,000,000)



(b) Multiple GPUs

Figure 12 Speedups of GPU parallelisation with multiple-GPU

Acknowledgements

This paper was supported by the National Natural Science Foundations of China (Grant No. 51909248) and the Open Research Fund of State Key Laboratory of Coastal and Offshore Engineering, Dalian University of Technology (Grant No. LP2012).

This work was also supported by the NVIDIA Corporation with the donation of the GPUs Geforce Titan Xp and GeForce Titan V.

The authors would also like to acknowledge the valuable input of Dr. Xinlei Wang at Shenzhen Zenus software technology LtD. (formerly PhD student at Zhejiang University) through personal communications.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Di Y, Yang J, Sato T. An operator-split ALE model for large deformation analysis of geomaterials. *International Journal for Numerical and Analytical Methods in Geomechanics*. 2007; 31: 1375-1399.

2. Nazem M, Sheng D, Carter JP, Sloan SW. Arbitrary Lagrangian–Eulerian method for large-strain consolidation problems. *International Journal for Numerical and Analytical Methods in Geomechanics*. 2008; 32: 1023-1050.

3. Wang D, White DJ, Randolph MF. Large-deformation finite element analysis of pipe penetration and large-amplitude lateral displacement. *Canadian Geotechnical Journal*. 2010; 47(8): 842–856.

4. Zhang Z, Pan Y, Wang J, Zhang H, Chen Z, Zheng Y, Ye H. A total-Lagrangian material point method for coupled growth and massive deformation of incompressible soft materials. *International Journal for Numerical Methods in Engineering*. 2021; DOI: https://doi.org/10.1002/nme.6787.

5. Sulsky D, Zhou SJ, Schreyer HL. Application of a particle-in-cell method to solid mechanics. *Comput Phys Commun*. 1995; 87: 236–52.

6. Harlow FH. The particle-in-cell computing method for fluid dynamics. *Methods Comput Phys*. 1964; 3: 319–43.

7. Ma S, Zhang X, Lian Y, Zhou X. Simulation of high explosive explosion using adaptive material point method. *Computer Modeling in Engineering & Sciences*. 2009; 39(2):101-123.

8. Nairn JA. Material point method calculations with explicit cracks. *Computer Modeling in Engineering & Sciences*. 2003; 4 (6): 649–664.

9. Huang P, Zhang X, Ma S, Huang X. Contact algorithms for the material point method in impact and penetration simulation. *International Journal for Numerical Methods in Engineering*. 2011; 85(4): 498-517.

10. de Vaucorbeil A, Nguyen VP, Hutchinson C. A total-Lagrangian material point method for solid mechanics problems involving large deformations. *Computer Methods in Applied Mechanics and Engineering*. 2019; 360: 112783.

11. Nguyen VP, de Vaucorbeil A, Nguyen CT, Mandal TK. A generalized particle in cell method for explicit solid

dynamics. *Computer Methods in Applied Mechanics and Engineering*. 2020; 371: 113308.

12. de Vaucorbeil A, Nguyen VP. Modelling contacts with a total Lagrangian material point method. Computer Methods in Applied Mechanics and Engineering. 2020; 373: 113503.
13. York AR, Sulsky D, Schreyer HL. Fluid–membrane interaction based on the material point method. *International Journal for Numerical Methods in Engineering*. 2000; 48(6): 901–924.
14. Stomakhin A, Schroeder C, Chai L, Teran J, Selle A. A material point method for snow simulation. *ACM Trans Graph (TOG)*. 2013; 32(4): 102.
15. Jiang C, Gast T, Teran J. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Transactions on Graphics*. 2017; 36(4): 152.
16. Fei Y, Guo Q, Wu R, Huang L, Gao M. Revisiting integration in the material point method: a scheme for easier separation and less dissipation. *ACM Trans. Graph*. 2021; 40(4): 109.
17. Soga K, Alonso E, Yerro A, Kumar K, Bandara S. Trends in large-deformation analysis of landslide mass movements with particular emphasis on the material point method. *Géotechnique*. 2015; 66(3): 248–273.
18. Dong Y, Wang D, Randolph MF. Investigating of impact forces on pipeline by submarine landslide using material point method. *Ocean Engineering*. 2017; 146: 21–28.
19. Pinyol NM, Alvarado M, Alonso EE, Zabala F. Thermal effects in landslide mobility. *Géotechnique*. 2018; 68(6): 528-545.
20. Zhao E, Dong Y, Tang Y, Sun J. Numerical Investigation of Hydrodynamics and Local Scour around Submarine Pipeline under Joint Effect of Solitary Wave and Current. *Ocean Engineering*. 2021; 222: 108553.
21. Coetzee CJ, Vermeer PA, Basson AH. The modelling of anchors using the material point method. *International Journal for Numerical and Analytical Methods in Geomechanics*. 2005; 29(9): 879–895.
22. Ceccato F, Bisson A, Cola S. Large displacement numerical study of 3D plate anchors. *European Journal of Environmental and Civil Engineering*. 2017; 1–19.
23. Wang L, Coombs WM, Augarde CE, Cortis M, Brown MJ, Brennan AJ, Knappett JA, Davidson C, Richards D, White DJ, Blake AP. An efficient and locking-free material point method for three-dimensional analysis with simplex elements. *International Journal for Numerical Methods in Engineering*. 2021; 122(15): 3876-3899.
24. Mast CM, Arduino P, Mackenzie-Helnwein P, Gregory RM. Simulating granular column collapse using the Material Point Method. *Acta Geotechnica*. 2015; 10: 101–116.
25. Yuan W, Wang H, Zhang W, Dai B, Liu K, Wang Y. Particle finite element method implementation for large deformation analysis using Abaqus. *Acta Geotechnica*. 2021; 12: 1-14.
26. Zhang W, Zhong Z, Peng C, Yuan W, Wu W. GPU-accelerated smoothed particle finite element method for large deformation analysis in geomechanics. *Computers and Geotechnics*. 2021; 129: 103856.
27. Abe K, Kenichi S, Samila B. Material point method for coupled hydromechanical problems. *Journal of Geotechnical and Geoenvironmental Engineering*. 2014; 140(3): 04013033.
28. Yerro A, Alonso EE, Pinyol NM. The material point method for unsaturated soils. *Géotechnique*. 2015; 65(3): 201-217.
29. Bandara S, Ferrari A, Laloui L. Modelling landslides in unsaturated slopes subjected to rainfall infiltration using material point method. *International Journal for Numerical and Analytical Methods in Geomechanics*. 2016; 40(9): 1358–1380.
30. Troncone A, Conte E, Pugliese L. Analysis of the slope response to an increase in pore water pressure using the material point method. *Water*. 2019; 11(7): 1446.

31. Wallstedt PC, Guilkey JE. Improved Velocity Projection for the Material Point Method. *CMES-Computer Modeling in Engineering & Sciences*. 2007; 19(3): 223–232.

32. Wallstedt PC, Guilkey JE. An evaluation of explicit time integration schemes for use with the generalized interpolation material point method. *Journal of Computational Physics*. 2008; 227: 9628-9642.

33. Nair J, Hammerquist C. Material point method simulations using an approximate full mass matrix inverse. *Computer Methods in Applied Mechanics and Engineering*. 2021; 377(2): 113667.

34. Buzzi O, Pedroso DM, Giacomini A. Caveats on the implementation of the generalized material point method. *Computer Model Eng Sci*. 2008; 31(2): 85–106.

35. Steffen M, Kirby RM, Berzins M. Analysis and reduction of quadrature errors in the material point method (MPM). *Int J Numer Meth Eng*. 2008; 76(6): 922–48.

36. Yerro A, Alonso EE, Pinyol NM. Run-out of landslides in brittle soils. *Computers and Geotechnics*. 2016; 80: 427-439.

37. Charlton TJ, Coombs WM, Augarde CE. iGIMP: An implicit generalised interpolation material point method for large deformations. *Computers and Structures*. 2017; 190: 108-125.

38. Gan Y, Sun Z, Chen Z, Zhang X, Liu Y. Enhancement of the material point method using B-spline basis functions. *International Journal for Numerical Methods in Engineering*. 2018; 113: 411-431.

39. Sun Z, Gan Y, Huang Z, Zhou X. A local grid refinement scheme for B-spline material point method. *International Journal for Numerical Methods in Engineering*. 2020; 121(11): 2398-2417.

40. Bardenhagen SG, Kober EM. The generalized interpolation material point method. *Computer Model Engineering and Science*. 2004; 5(6): 477–96.

41. Dong Y, Wang D, Randolph MF. Quantification of impact forces on fixed mudmats from submarine landslides using the material point method. *Applied Ocean Research*. 2020; 146: 21-28.

42. Zhang Y, Zhang X, Liu Y. An alternated grid updating parallel algorithm for material point method using OpenMP. *Computer Modeling in Engineering & Sciences*. 2010; 69(2): 143–165.

43. Parker SG. A component-based architecture for parallel multi-physics PDE simulation. *Future Generation Computer Systems*. 2006; 22(1): 204–216.

44. Stantchev G, Dorland W, Gumerov N. Fast parallel particle-to-grid interpolation for plasma PIC simulations on the GPU. *Journal of Parallel and Distributed Computing*. 2008; 68(10):1339–1349.

45. Gibson MJ, Keedwell EC, Savić DA. An investigation of the efficient implementation of cellular automata on multi-core CPU and GPU hardware. *Journal of Parallel and Distributed Computing*. 2015; 77: 11-25.

46. Dong Y, Wang D, Randolph MF. A GPU parallel computing strategy for the material point method. *Computers and Geotechnics*. 2015; 66: 31–38.

47. Gao M, Wang X, Wu K, Pradhana A, Sifakis E, Yuksel C, Jiang C. GPU optimization of material point methods. *ACM Transactions on Graphics (TOG)*. 2018; 37(6), 1-12.

48. Dong Y, Grabe J. Large scale parallelisation of the material point method with multiple GPUs. *Computers and Geotechnics.* 2018; 101: 149-158.

49. Wang X, Qiu Y, Slattery SR, Fang Y, Li M, Zhu S, Zhu Y, Tang M, Manocha D, Jiang C. A massively parallel and scalable multi-GPU material point method. *ACM Transactions on Graphics (TOG)*. 2020; 39(4): 30.

50. Ma J, Wang D, Randolph MF. A new contact algorithm in the material point method for geotechnical simulations. *International Journal for Numerical and Analytical Methods in Geomechanics*. 2014; 38(11): 1197-1210.

51. Dong Y. Reseeding of particles in the material point method for soil-structure interactions. *Computers and Geotechnics*. 2020; 127: 103716.

52. Bardenhagen SG, Brackbill JU, Sulsky D. The material-point method for granular materials. *Computer Methods in Applied Mechanics and Engineering.* 2000; 187(3-4): 529-541.

53. Bardenhagen SG, Guilkey JE, Roessig KM, Brackbill JU, Witzel WM. An improved contact algorithm for the material point method and application to stress propagation in granular material. *Computer Modeling in Engineering & Sciences.* 2001; *2*(4): 509-522.

54. Zhang D, Ma X, Giguere PT. Material point method enhanced by modified gradient of shape function. *Journal of Computational Physics*. 2011; 230(16): 6379-6398.

55. Dong Y, Wang D, Randolph M. Runout of submarine landslide simulated with material point method. *Journal of Hydrodynamics*. 2017; 29(3): 438-444.

56. Bing Y, Cortis M, Charlton TJ, Coombs WM, Augarde CE. B-spline based boundary conditions in the material point method. *Computers & Structures*. 2019; 212: 257-274.

57. Cortis M, Coombs WM, Augarde CE, Brown M, Brennan A, Robinson S. Imposition of essential boundary conditions in the material point method. *International Journal for Numerical Methods in Engineering*. 2018; 113(1): 130-152.

58. Nishiura D, Matsuo MY, Sakaguchi H. ppohDEM: computational performance for open source code of the discrete element method. *Computer Physics Communications*. 2014; 185(5): 1486-1495.

59. Nishiura D, Sakaguchi H. Parallel-vector algorithms for particle simulations on shared-memory multiprocessors. *Journal of Computational Physics*. 2011; 230(5): 1923-1938.

60. Dong Y. Reseeding of particles in the material point method for soil-structure interactions. *Computers and Geotechnics*. 2020; 127: 103716.

61. Locat J, Lee HJ. Submarine landslides: advances and challenges. *Canadian Geotechnical Journal*. 2002; 39(1): 193-212.

62. Thakur V, Degago SA. Quickness of sensitive clays. *Géotechnique Letters*. 2012; 2(3): 87-95.

63. Boukpeti N, White DJ, Randolph MF, Low HE. Strength of fine-grained soils at the solid-fluid transition. *Géotechnique*. 2012; 62(3): 213-226.

64. Fornes P, Bihs H, Thakur VKS. Implementation of non-Newtonian rheology for Debris Flow simulation with REEF3D. IAHR World Congress, 2017.

65. Teh CI, Houlsby GT. An analytical study of the cone penetration test in clay. *Géotechnique*. 1991; 41(1): 17-34.

66. Lu Q, Randolph MF, Hu Y, Bugarski IC. A numerical study of cone penetration in clay. *Géotechnique*. 2004; 54(4): 257-267.

67. Walker J, Yu HS. Analysis of the cone penetration test in layered clay. *Géotechnique*. 2010; 60(12): 939-948.

68. Wang D, Bienen B, Nazem M, Tian Y, Zheng J, Pucker T, Randolph MF. Large deformation finite element analyses in geotechnical engineering. *Computers and Geotechnics*. 2015; 65: 104-114.

69. Ceccato F, Beuth L, Vermeer PA, Simonini P. Two-phase Material Point Method applied to the study of cone penetration. *Computers and Geotechnics*. 2016; 80: 440-452.

Appendix A Snippets of pseudo code for multi-GPU parallelisation

Main Functions:

for (int GPU_ID = 0; GPU_ID < N_Gpus; GPU_ID++)
//N_Gpus is the total number of GPUs available
{

cudaSetDevice(GPU_ID);

//Function (i): Initialisation of nodal variables
int block_size = 32;
int n_blocks = int(N_G[GPU_ID] / 32);
//N_G is the total number of element node in each subdomain

Initialisation_of_Nodal_Variables << <n_blocks, block_size> >> (GPU_ID, m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang});

//Function (ii): Interpolation from particles to nodes
for (int Index_Layer = 0; Index_Layer < N_G_Z[GPU_ID]; Index_Layer ++)
// N_G_Z is the total number of layers of nodes along the height of the model
{
block_size = 32;
n_blocks = int(N_P[GPU_ID] / 32);
//N_P is the total number of particles in each subdomain

Generate_Particle_List << <n_blocks, block_size> >> (GPU_ID, Index_Layer, Particle_List, X_p);

block_size = 32;
n_blocks = int(GGx[GPU_ID] * GGy[GPU_ID] / 32);
// GGx and GGy are the total number of nodes in X and Y directions, respectively

Interpolation_From_Particles_To_Nodes << <n_blocks, block_size> >> (GPU_ID, Index_Layer, Particle_List, X_p, m_p, V_p, ρ, v_p, σ_p, f_p^{ext}, m_i, M_i, F_i^{int}, F_i^{ext}, ω_i^{norm});
}

//Function (iii): Calculate nodal velocities and accelerations
int block_size = 32;
int n_blocks = int(N_G[GPU_ID] / 32);

Calculate_Nodal_Velocities_and_Accelerations << <n_blocks, block_size> >> (GPU_ID, m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang});

//Function (iv): Update particle state

block_size = 32;
n_blocks = int(N_P[GPU_ID] / 32);
//N_P is the total number of particles in each subdomain

Update_Particle_State << <n_blocks, block_size> >> (GPU_ID, v_i, a_i, X_p, m_p, V_p, ρ, v_{p}, D_{p}, W_{p}, σ_{p});

}

Individual function:

__global__ void Initialisation_of_Nodal_Variables (GPU_ID, m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang})

```
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if(i < N_G)
        {
                //initialisation operations
        }
}
```

```
__global__ void Generate_Particle_List (GPU_ID, Index_Layer, Particle_List, Xp)
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if(i < N_P)
        {
                if ((Xp_X[i] >= Lower_X) && (Xp_X[i] <= Upper_X) &&
                 (Xp_Y[i] >= Lower_Y) && (Xp_Y[i] <= Upper_Y) &&
                 (Xp_Z[i] >= Lower_Z) && (Xp_Z[i] <= Upper_Z) )
                // Lower and Upper are the lower and upper coordinates of the influence area of the node layer, respectively
                {
                        Index_X = int((Xp_X[i] - Lower_X) / h / h_cell);
                        Index_Y = int((Xp_Y[i] - Lower_Y) / h / h_cell);
                        Index_Z = int((Xp_Z[i] - Lower_Z) / h / h_cell);
                        // h and h_cell are the sizes of the element and fine cell, respectively

                        List_Position = Index_Z*FCy*FCx + Index_Y*FCx + Index_X;
                        // FCx and FCy are the total numbers of the fine cell in X and Y directions, respectively

                        Particle_List[List_Position] = i;
                }
        }
}
```

__global__ void Interpolation_From_Particles_To_Nodes (GPU_ID, Index_Layer, Particle_List, X_p, m_p, V_p, ρ, v_p, σ_p, f_p^{ext}, m_i, M_i, F_i^{int}, F_i^{ext}, ω_i^{norm})

```
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if(i < GGx*GGy)
        {
                int NODE = Index_Layer*GGy*GGx + i;

                for(/*cycle of the elements around node i*/)
                {
                for(/*cycle of the particle list of element*/)
                {
                        Execute: Eqs. (5) – (9)
                }
                }
        }
}
```

__global__ void Calculate_Nodal_Velocities_and_Accelerations (GPU_ID, m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang})

```
{
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if(i < N_G)
        {
                Execute: Eqs. (10) – (20)
        }
}
```

```
__global__ void Update_Particle_State (GPU_ID, v_i, a_i, X_p, m_p, V_p, ρ, v_p, D_p, W_p, σ_p)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    if(i < N_P)
    {
        Execute: Eqs. (21) – (28)
    }
}
```

Appendix B Snippets of pseudo code for CPU sequential computations

//Function (i): Initialisation of nodal variables

Initialisation_of_Nodal_Variables (m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang});

//Function (ii): Interpolation from particles to nodes

Interpolation_From_Particles_To_Nodes (X_p, m_p, V_p, ρ, v_{p}, σ_{p}, f_p^{ext}, m_i, M_i, F_i^{int}, F_i^{ext}, ω_i^{norm});

//Function (iii): Calculate nodal velocities and accelerations

Calculate_Nodal_Velocities_and_Accelerations (m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang});

//Function (iv): Update particle state

Update_Particle_State (v_i, a_i, X_p, m_p, V_p, ρ, v_{p}, D_{p}, W_{p}, σ_{p});

Individual function:

void Initialisation_of_Nodal_Variables (m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang})

```
{
    for(/*cycle of the nodes*/)
    {
        //initialisation operations
    }
}
```

void Interpolation_From_Particles_To_Nodes (X_p, m_p, V_p, ρ, v_p, σ_p, f_p^{ext}, m_i, M_i, F_i^{int}, F_i^{ext}, ω_i^{norm})

{

for(/*cycle of the particles*/)

{

for(/*cycle of related nodes*/)

{

Execute: Eqs. (5) – (9)

}

}

}

void Calculate_Nodal_Velocities_and_Accelerations (m_i, v_i, M_i, a_i, F_i^{int}, F_i^{ext}, ω_i^{norm}, ω_i^{tang})

{

for(/*cycle of the nodes*/)

{

Execute: Eqs. (10) – (20)

}

}

void Update_Particle_State (v_i, a_i, X_p, m_p, V_p, ρ, v_p, D_p, W_p, σ_p)

{

for(/*cycle of the particles*/)

{

Execute: Eqs. (21) – (28)

}

}