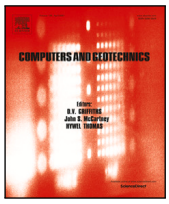
Computers and Geotechnics 180 (2025) 107113

Contents lists available at ScienceDirect

Computers and Geotechnics

journal homepage: www.elsevier.com/locate/compgeo



Research paper

A sparse-memory-encoding GPU-MPM framework for large-scale simulations of granular flows

Hao Chen, Shiwei Zhao *, Jidong Zhao *

Department of Civil and Environmental Engineering, The Hong Kong University of Science and Technology, Hong Kong Special Administrative Region

ARTICLE INFO

Keywords:
MPM
GPU
Sparse memory
Large deformation
Granular flow

ABSTRACT

The Material Point Method (MPM) is increasingly recognized as an effective tool for simulating complex granular flows. While GPU computing has been widely used in MPM applications for large-scale problems, its heavy reliance on contiguous memory distribution can significantly hinder efficiency and limit simulation capabilities due to memory capacity constraints. This study presents a sparse-memory-encoding framework that incorporates advanced algorithms to address these limitations in large-scale simulations. We introduce a novel algorithm for atomic-free dual mapping between material points and nodes, in conjunction with warp-wise particle-to-grid mappings organized within a block-cell-material point hierarchy. Moreover, the framework features an efficient memory shift algorithm that optimizes memory usage for material properties. This optimization enables the seamless integration of commonly used material constitutive models, including elastic, elastoplastic, and hyper-plastic models, as well as various iteration schemes such as "update stress first", "update stress last", and "modified update stress last" within a cohesive framework. Furthermore, the framework accommodates incorporating diverse boundary conditions, such as Dirichlet, Neumann, and arbitrary-shaped rigid body contact, thus broadening its applicability to real-world engineering challenges, including landslides. The framework can effectively and efficiently handle large-scale, high-fidelity simulations of granular flows.

1. Introduction

The Material Point Method (MPM) has emerged as a highly effective numerical tool for simulating granular materials, particularly in the context of extreme deformation scenarios, as demonstrated in various recent studies (Gaume et al., 2018; Iverson, 2012; Pudasaini and Krautblatter, 2021; Zheng et al., 2023; Coombs et al., 2018; Jin et al., 2021). MPM distinguishes itself apart from traditional approaches, such as the Finite Element Method (FEM) (Hrennikoff, 1941) and the Discrete Element Method (DEM) (Cundall and Strack, 1979), through its innovative dual formulation that combines Lagrangian and Eulerian perspectives. This hybrid methodology enables MPM to proficiently model granular materials that exhibiting both solid and fluid-like characteristics, and facilitates accurate representations of history-dependent material responses while mitigating severe mesh distortion issues commonly encountered in other techniques. Furthermore, the reliance on point sampling, as opposed to complex mesh tessellation, enhances the versatility and applicability of MPM (Zhang et al., 2024).

In recent decades, MPM has gained surging popularity, supported by numerous research initiatives that have produced well-established open-source codes, including NairnMPM (Hammerquist and Nairn, 2017), CB-Geo (Kumar et al., 2019), Anura3D (Anura3D MPM Research Community, 2024), and Karamelo (de Vaucorbeil et al., 2021). These frameworks utilize low-level backends in C++ and Fortran, along with parallel computing libraries such as Open Multi-Processing (OpenMP) (Dagum and Menon, 1998) and Message-Passing Interface (MPI) (Clarke et al., 1994), enabling the distribution of computational workloads across multiple CPU cores or nodes. Furthermore, they incorporate advanced high-order interpolation techniques, such as the Generalized Interpolation Material Point (GIMP) method (Bardenhagen et al., 2004), B-Splines (Steffen et al., 2008; Yamaguchi et al., 2021), and Moving Least Squares (MLS) approaches (Hu et al., 2018), thereby enhancing the efficacy of MPM. The integration of both structured and unstructured meshes (Cao et al., 2024; Bardenhagen et al., 2004) further extends the applicability of MPM to complex realistic problems characterized by intricate boundary conditions (Wang et al., 2021; Bird et al., 2024; Remmerswaal, 2023). However, challenges remain regarding the inefficiencies associated with large-scale CPU-based MPM implementations. Buckland et al. (2024) noted that as the number of material points (MPs) increases to millions or even billions,

* Corresponding authors.
E-mail addresses: ceswzhao@ust.hk (S. Zhao), jzhao@ust.hk (J. Zhao).

https://doi.org/10.1016/j.compgeo.2025.107113
Received 27 November 2024; Received in revised form 13 January 2025; Accepted 21 January 2025
Available online 4 February 2025
0266-352X/© 2025 Elsevier Ltd. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

CPU-parallel models relying on multiple threads or processes may experience diminished acceleration ratios due to excessive communication overhead among cores or nodes.

More recently, GPU-driven parallel techniques, particularly those utilizing NVIDIA Compute Unified Device Architecture (CUDA) (NVIDIA et al., 2020), have emerged as a robust alternative in the realm of parallel computing. These techniques leverage lightweight scheduling and inherent communication parallelism typical of many-core architectures, making them particularly advantageous for particle-based methods, such as DEM (Zhao et al., 2023) and MPM (Dong et al., 2015, 2022; Zhang et al., 2023). Notably, various optimization techniques have been proposed for GPU-based MPM. For example, atomic operations (Feng and Xu, 2021; Wyser et al., 2021) have been introduced for particle-to-grid (P2G) mapping to alleviate race conditions, especially when the number of MPs surpasses the number of lattice nodes. Moreover, recently proposed GPU-friendly implicit solvers have shown significant improvements in computational performance (Wang et al., 2024; Zhou et al., 2024).

The computer graphics community has made significant strides in advancing GPU-based MPM techniques. A prominent example is the Taichi programming language (Hu et al., 2019), which incorporates sparse memory structures into GPU MPM (Museth, 2013; Setaluri et al., 2014), effectively addressing memory capacity constraints. The Taichi language has facilitated the development of several MPM frameworks based on its principles (Hu et al., 2018; Li et al., 2023; Shi et al., 2024), successfully balancing ease of code development with computational efficiency through a Pythonic programming style. However, it is important to note that Taichi MPM primarily employs the Array of Structures (AOS) memory format, which may not optimize coalesced memory access and high cache-hit rates on GPUs. Subsequent investigations by Gao et al. (2018) and Wang et al. (2020b) have explored the Structure of Arrays (SOA) and Array of Structures of Arrays (AOSOA) memory formats for GPU MPM, aligning with the Single Instruction Multiple Threads (SIMT) design paradigm.

However, these methodologies primarily address immediate needs in real-time animations in computer graphics, limiting their direct applicability for modeling of challenging engineering problems (Feng and Xu, 2021), such as granular flows involving complex materials and boundary conditions (BCs). For instance, GPU-based MPM techniques in computer graphics generally implement a hyper-plastic model (Chen and Baladi, 1985), where the plastic component and stress are directly correlated with the strain energy of MPs. In contrast, granular flow systems often exhibit rate-dependent mechanical behavior, necessitating a broader range of material models, including hyper-plastic, incremental elastoplastic, and non-local viscoplastic models (Kamrin and Koval, 2012). Previous research related to computer graphics has not extensively explored the diverse state variables associated with different material types and the appropriate application of iterative schemes, such as Update Stress First (USF), Update Stress Last (USL), or Modified Update Stress Last (MUSL). Moreover, there has been limited consideration of material-related boundary aspects, including BC time series (e.g., sinusoidal versus constant models), targeted applications for BCs (whether at nodes, MPs, or both), BC imposition involving transformations (e.g., rotation or affine transformations) (Bing et al., 2019; Liang et al., 2023), and the geometry of BCs. For example, addressing debris flow hazards in mountainous regions may require convoluted digital elevation models that are often analytically undefined or difficult to represent numerically.

This study presents a novel GPU-based MPM framework specifically designed to address large-scale engineering challenges involving complex materials and BCs. The framework leverages sparse memory and incorporates a block-cell-MP parallelism topology, enhanced by fine-grained, warp-intrinsic memory access. It implements a unified memory shift of state variables of material constitutive models across kernel executions and timesteps. This approach decouples GPU MPM optimization from material type dependencies, and making the framework adaptable to various material models, including elastoplastic and hyper-plastic types. Furthermore, beyond conventional Dirichlet and Neumann boundary conditions, this study introduces an efficient DEM-enriched contact algorithm, enabling precise modeling of interactions between MPs and arbitrary boundary geometries, such as those represented by Standard Triangle Language (STL) meshes.

The remainder of this paper is organized as follows: Section 2 presents a concise theoretical and technical overview of the MPM. Section 3 explores the GPU-based algorithms of our MPM framework, while Sections 4 and 5 examine the validation process and evaluate the efficiency of the GPU-based MPM. Concluding remarks are presented in Section 6.

2. Prevailing solution schemes

2.1. Governing equations

The MPM, first introduced by Sulsky et al. (1994), is designed for modeling large deformations using a hybrid framework that integrates both Lagrangian and Eulerian approaches. In MPM, the solution domain is discretized into Lagrangian points, while the kinematics and dynamics of these points are updated utilizing a background mesh consistent with the Eulerian framework. The governing equations, which encapsulate mass and momentum conservations within the MPM formulation, can be expressed in strong form as follows:

$$\frac{D\rho}{Dt} = 0 \tag{1a}$$

$$\rho \boldsymbol{a} = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{b} \tag{1b}$$

where $\boldsymbol{a}$ represents acceleration, the $\boldsymbol{\sigma}$ denotes the Cauchy stress, ρ indicates density, and $\boldsymbol{b}$ signifies body-force acceleration. By multiplying Eq. (1b) with the weighting term $\delta \boldsymbol{u}$, we obtain the weak form of the momentum equation:

$$\int_{\Omega} \rho \boldsymbol{a} \cdot \delta \boldsymbol{u} dV = \int_{\Omega} \rho \boldsymbol{b} \cdot \delta \boldsymbol{u} dV + \int_{\partial\Omega} \boldsymbol{\tau} \cdot \delta \boldsymbol{u} dS - \int_{\Omega} \boldsymbol{\sigma} \cdot \nabla \delta \boldsymbol{u} dV \tag{2}$$

where Ω denotes the solution domain, and $\partial\Omega$ represents the domain boundary. The term $\boldsymbol{\tau} = \boldsymbol{\sigma} \cdot \boldsymbol{n}$ signifies the surface traction acting on the domain boundary, with $\boldsymbol{n}$ being the outward unit normal vector at the boundary in the current configuration.

2.2. Temporal and spatial discretization

Eq. (2) is discretized and integrated over the background mesh, following an iterative procedure comprising four primary stages, as illustrated in Fig. 1. In an initial P2G stage, the kinematics of the MPs, including mass, momentum and stress, are extrapolated to the nodes, to establish the discrete momentum conservation equations. The next grid-to-grid mapping (G2G) stage, features a Newtonian integration at the nodes to update dynamic variables such as acceleration and velocity. These updated values are then interpolated back to the MPs in the third grid-to-particle mapping (G2P) stage. In the final particle-to-particle mapping (P2P) stage, the MPs are updated in terms of their position and state variables. The necessary discrete calculations for each stage within a single timestep of the MPM iteration are detailed in Eqs. (3)(a-f):

$$m_i^{(n)} = \sum_p m_p^{(n)} S_{ip}^{(n)} \tag{3a}$$

$$\boldsymbol{p}_i^{(n)} = \sum_p \boldsymbol{p}_p^{(n)} S_{ip}^{(n)} \tag{3b}$$

$$\boldsymbol{f}_{i,ext}^{(n)} = \sum_p m_p^{(n)} \boldsymbol{b}_p^{(n)} S_{ip}^{(n)} + \int_{\partial\Omega_\tau} \boldsymbol{\tau} S_{ip}^{(n)} dA \tag{3c}$$

$$\boldsymbol{f}_{i,int}^{(n)} = -\sum_p V_p^{(n)} \boldsymbol{\sigma}_p^{(n)} \cdot \nabla S_{ip}^{(n)} \tag{3d}$$

$$\boldsymbol{f}_i^{(n)} = \boldsymbol{f}_{i,ext}^{(n)} + \boldsymbol{f}_{i,int}^{(n)} \tag{3e}$$

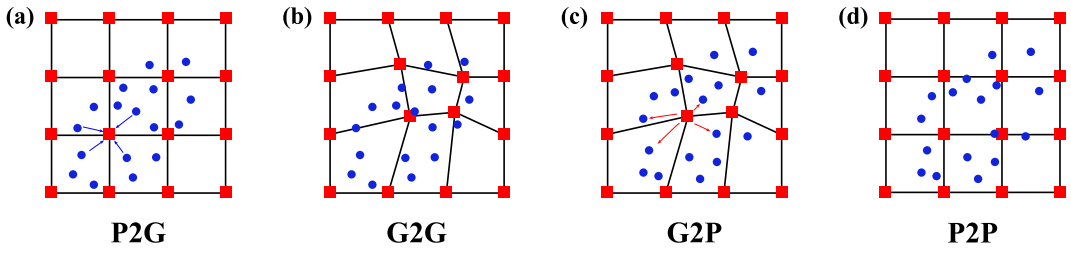


Fig. 1. The four primary stages of iteration in the general MPM process.

$$\boldsymbol{p}_i^{(n+1)} = \boldsymbol{p}_i^{(n)} + \boldsymbol{f}_i^{(n)}\Delta t \tag{3f}$$

where S_{ip} and ∇S_{ip} represent the shape function and its gradient, respectively, while V_p denotes the volume of the MPs. Mass (m_p), external force ($\boldsymbol{f}_{i,\text{ext}}$), internal force ($\boldsymbol{f}_{i,\text{int}}$), and momentum ($\boldsymbol{p}$) are extrapolated from the MPs to the background grid, as detailed in Eqs. (3a), (3e), and (3b). Eq. (3f) presents the Newtonian integration of the mapped momentum.

The increments in strain, denoted as $d\boldsymbol{\epsilon}_p^{t+\Delta t}$, and the increments in spin, represented by $d\boldsymbol{\omega}_p^{t+\Delta t}$, associated with the velocity gradient $\boldsymbol{L}_p^{t+\Delta t}$, are quantified as follows Liang et al. (2024):

$$\boldsymbol{L}_p^{n+1} = \sum_i \nabla S_{ip}\boldsymbol{v}_i^{n+1} \tag{4a}$$

$$d\boldsymbol{\epsilon}_p^{n+1} = \frac{1}{2}(\boldsymbol{L}_p^{n+1} + (\boldsymbol{L}_p^{n+1})^T)\Delta t \tag{4b}$$

$$d\boldsymbol{\omega}_p^{n+1} = \frac{1}{2}(\boldsymbol{L}_p^{n+1} - (\boldsymbol{L}_p^{n+1})^T)\Delta t \tag{4c}$$

For the incremental scheme of stress update, the relationships are given by:

$$\boldsymbol{F}_p^{n+1} = (\Delta t\boldsymbol{L}_p^n + \boldsymbol{I})\boldsymbol{F}_p^n \tag{5a}$$

$$\boldsymbol{\sigma}_p^{n+1} = \boldsymbol{\sigma}_p^n + \boldsymbol{D}^{ep}d\boldsymbol{\epsilon}_p^{n+1} + d\boldsymbol{\omega}_p^{n+1}\cdot\boldsymbol{\sigma}_p^n - \boldsymbol{\sigma}_p^n\cdot d\boldsymbol{\omega}_p^{n+1} \tag{5b}$$

where $\boldsymbol{F}_p$ represents the deformation gradient of the material point, $\boldsymbol{\sigma}_p$ denotes the stress tensor, and $\boldsymbol{D}^{ep}$ is the elastoplastic Jacobian.

2.3. Solution schemes

Several options are available for enhancing MPM iterations for a solution. A crucial factor is the selection of the shape function (S_{ip}) and its gradient (∇S_{ip}) used during the P2G and G2P stages. One efficient and fundamental choice is the linear shape function, introduced by Sulsky et al. (1994), which simplifies the treatment of MPs as entities without an associated geometrical volume. In the context of mass interpolation, the linear shape function is expressed as follows:

$$m(\mathbf{x},t) = \sum_p m_p N_{ip}(\mathbf{x},\mathbf{x}_p) \tag{6a}$$

$$N_{ip}(\mathbf{x},\mathbf{x}_p) = \begin{cases} 0 & \mathbf{x}-\mathbf{x}_p \leqslant -L \\ 1+(\mathbf{x}-\mathbf{x}_p)/L & -L < \mathbf{x}-\mathbf{x}_p \leqslant 0 \\ 1-(\mathbf{x}-\mathbf{x}_p)/L & 0 < \mathbf{x}-\mathbf{x}_p \leqslant L \\ 0 & L < \mathbf{x}-\mathbf{x}_p \end{cases} \tag{6b}$$

where m_p represents the mass of the MP, N_{ip} is the linear shape function, L denotes the node space, and $\mathbf{x}_p$ is the position vector of the MP.

However, employing the linear shape function can lead to significant numerical instability, particularly when the material particles cross the background cells, i.e., cell-crossing issues. Apart from this, lower-order interpolation methods may result in abrupt loss of gradients. Conversely, higher-order interpolation techniques, such as the GIMP, help to alleviate cell-crossing issues. In GIMP, the shape functions are defined as follows Bardenhagen et al. (2004):

$$S_{ip} = \frac{1}{V_p}\int_{\Omega_p\cap\Omega}\chi_p(\mathbf{x})N_{ip}(\mathbf{x})dV \tag{7a}$$

$$\nabla S_{ip} = \frac{1}{V_p}\int_{\Omega_p\cap\Omega}\chi_p(\mathbf{x})\nabla N_{ip}(\mathbf{x})dV \tag{7b}$$

where χ_p represents the characteristic function within a virtual MP domain that is typically defined to be orthogonal and aligned with its background cell. The GIMP approach employs an integration technique that incorporates the multiplication of characteristic function and linear shape functions within the MP domain, facilitating a smooth transition of the gradient shape function between adjacent background cells. This study adopts the GIMP method as a showcase. It is worth mentioning that other higher-order counterparts, such as B-Splines (Steffen et al., 2008) or MLS (Hu et al., 2018), can also be considered conveniently in our proposed framework.

The second alternative arises in the G2P/P2P stage, which varies based on whether the updated velocity of nodes, updated acceleration of nodes, or both influence the kinematics of the MP. The first method is defined by the Particle-In-Cell (PIC) approach (Harlow, 1964), whereas the second is characterized by the FLuid-Implicit-Particle (FLIP) method (Brackbill and Ruppel, 1986). These methods exhibit different dissipation and stability characteristics: PIC tends to provide greater numerical stability, whereas FLIP demonstrates lower artificial dissipation. This study adopts a hybrid approach that combines both PIC and FLIP (Hammerquist and Nairn, 2017), weighted by a fraction (β):

$$\boldsymbol{V}^{n+1} = (1-\beta)\boldsymbol{V}^n + S_{ip}\boldsymbol{a}\Delta t + \beta S_{ip}\boldsymbol{v}^n \tag{8a}$$

$$\mathbf{x}^{n+1} = \mathbf{x}^n + S_{ip}\boldsymbol{v}^{n+1}\Delta t - 0.5[S_{ip}\boldsymbol{a} + \frac{\beta}{\Delta t}(\boldsymbol{V}^n - S_{ip}\boldsymbol{v}^n)]\Delta t^2 \tag{8b}$$

where $\boldsymbol{v}^n$ represents the velocity on the background grid at step n, $\boldsymbol{a}^n$ is the acceleration of nodes at step n, $\boldsymbol{V}^n$ signifies the velocity of the material points at step n, and $\boldsymbol{V}^{n+1}$ denotes the velocity of the material points at step $n+1$. The term S_{ip} indicates the matrix of interpolation shape functions that correspond to the degrees of freedom between nodes and material points, and $\mathbf{x}^n$ denotes the material point position at step n. The PIC fraction (β) acts as a weighting parameter that regulates the contribution of PIC within the hybrid scheme. A value of $\beta = 1$ corresponds to the full PIC scheme, whereas $\beta = 0$ signifies exclusive reliance on the FLIP method.

As illustrated in Fig. 1, the order of iteration stages is critical, determined by whether the stress update of MPs occurs before or after the G2G stage. In the USF scheme, the stress increment relies on the previous step's node velocities, while the USL scheme uses the current, updated dynamics for the stress update. A modified variant, termed MUSL, introduces an additional P2G step between G2G and the stress update, balancing both methodologies. The USF approach is adopted for this study, and a conventional CPU version workflow of USF is introduced for comparison (Algorithm 1).

Algorithm 1: MPM iteration within a single timestep.

[h]
foreach *p in points* **do**
    Initialization: Reset the information for all nodes.
    P2G 1: Transfer mass and momentum from material points to the grid.
    Node: Convection of the velocities at the nodes.
    Stress: Compute the strain and stress for the material points.
    P2G 2: Transfer forces from material points to the grid.
    G2G: Update the velocity and acceleration of the nodes.
    G2P and P2P: Transfer the updated acceleration and velocity from the grid back to the material points, while updating the necessary kinematics of the material points.

2.4. Porting from CPU to GPU: General ways on implementation

Transitioning the MPM workflow from CPU to GPU entails substantial modifications, especially in the initialization phase, the P2G and G2P mapping processes and the execution of material constitutive models. Nevertheless, certain aspects, such as the establishment of shape functions and the G2G stage, share some similarities.

(1) **Initialization**. A primary challenge in GPU-based MPM is initialization, where constraints on memory usage occur. Engineering simulations frequently necessitate a variety of geometries, including spheres, cubes, polygons, and other complex shapes defined analytically or numerically. These geometries require minimal MP resolutions to achieve accurate boundary conditions; however, in large-scale problems, high resolutions can frequently surpass the capacity of GPU memory which is often limited to around 100 GB, in contrast to the hundreds or thousands of GB available on CPUs. To mitigate this issue, we employ a sparse memory management strategy, which is elaborated in Section 3.1.
(2) **P2G and G2P**. The adaptations for P2G and G2P in a GPU context largely involve consideration of thread scheduling and memory access. Unlike CPU implementations that utilize loops over MPs or nodes in P2G/G2P, GPU implementations may encounter challenges such as non-coalesced memory access and race conditions. In P2G, where the number of MPs typically exceeds the number of nodes, many-to-one mapping is commonplace. Traditional GPU techniques for scattering attributes often depend on atomic operations, allowing simultaneous read/write requests to the same address, but only one operation is allowed to execute, controlled by hardware locks. In scenarios where MPs vastly outnumber nodes, this reliance on atomic operations can hinder performance, resembling serial CPU operations. To counteract this, we apply a warp-intrinsic method in combination with block-cell parallelism for P2G. For G2P, we adopt a SOA data layout to promote coalesced memory access and further implement multi-level memory deployment (register, shared, and global) to enhance cache efficiency, as detailed in Sections 3.3.1 and 3.3.2.
(3) **Material Constitutive Models**. The great variety and complexity of material models, such as hyper-plastic and elastoplastic models, poses substantial challenges on optimization for the transition from CPU to GPU. Complex models with extensive datasets may easily exceed the GPU memory page size limits (commonly 4 KB), leading to cache evictions and page faults that escalate memory latency and diminish performance. Moreover, differing material models introduce data dependencies; for instance, elastoplastic models may require incremental deformation gradients, which are not necessary for hyper-plastic models. To manage this variability, we implement a data shift algorithm that provides flexibility across material types, as discussed in Section 3.5.

In this study, all GPU MPM kernels are developed using CUDA/C++ to achieve optimal performance. Interfaces are encapsulated and organized into Python packages, facilitating a user-friendly simulation setup. Users can configure essential MPM parameters, such as background dimensions, damping factors, gravity, timestep, and iteration schemes, using only a few lines of Python code. Moreover, advanced functionalities, including MP sampling within complex solution domains, multi-material assignment, time-dependent boundary conditions, and periodic result archiving, are supported through user-defined Python functions. These auxiliary functions are translated into C++ objects by our Python interface for GPU MPM, which are then executed on the GPU kernel according to the user-specified configuration.

3. Sparse memory supported algorithms on GPU

A suitable data structure can reduce memory footprint, thereby improving the dimension of simulations, especially those with limited hardware configurations. A GPU-Sparse-Grid structure, adapted from Setaluri et al. (2014), is employed. Details will be provided in Sections 3.1 and 3.2. Moreover, another challenge regarding GPU-MPM performance is the data transfer between MPs and nodes, as MPM iterations involve frequent data resets and data interpolation or extrapolation. To address this challenge, a block-cell-MP hierarchical strategy is implemented for efficient P2G and G2P transfers, as discussed in Sections 3.3.1 and 3.3.2.

3.1. Sparse grid

The Lagrangian and Eulerian configurations of a typical MPM simulation are illustrated in Fig. 2, where a general multi-material scenario is simplified into a two-material condition for clearer demonstration. Material A (green particles) and Material B (red particles) occupy the bottom-left and top-right corners, respectively, to clearly showcase the different memory formats. According to the discretized form of MPM, data mapping between the MPs and nodes is guided by weights or gradient weights that MPs fetch from the supporting domain (brown regions) of the activated nodes (magenta squares). Thus, the access mode to the memory of either MPs or nodes is critical to the overall simulation efficiency. This section focuses on the memory of the nodes; for the MPs counterpart, please refer to the next section.

Fig. 2 depicts the activated nodes, represented by magenta squares, with their respective indices displayed along the left and bottom edges. The memory index, for example, is denoted as $0/1$, where 0 indicates the sparse memory index and 1 represents the continuous memory index. The distinction between these memory indices arises from the different allocation methods used for continuous and sparse memory. Continuous memory accesses the dynamics of all nodes within a grid consisting of $6\times4 = 24$ nodes, as illustrated in Fig. 2. Assuming that each node requires storage for N_d bytes, the overall memory requirement is $24N_d$. In contrast, the sparse memory method focuses solely on the activated nodes, resulting in a reduced memory requirement of $15N_d$.

It is essential to recognize that the oscillations resulting from particle transitions across cells require higher-order interpolation methods. Thus, at least one extra padding layer is necessary for the shape function domain. For example, Nairn and Hammerquist (2021) observed that in the GIMP framework, the resolution is typically set so that each direction includes two MPs, with each MP generally tracking nodes up to three cells away. By incorporating an additional padding layer, MPs in GIMP interpolation can efficiently map to a total of four nodes in each dimension.

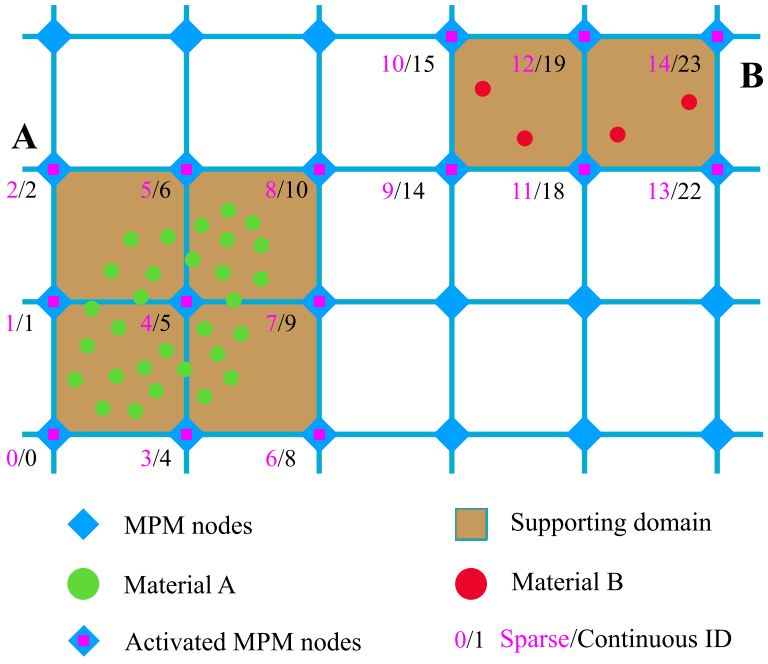


Fig. 2. Offset differences between continuous and sparse memory distributions in a two-material (A and B) MPM simulation.

3.2. Material points reorder

Fig. 3 demonstrates the organization of blocks, each consisting of four cells in both dimensions, for Materials A and B, illustrating the sparse memory mechanism used for compressing node indices. A key distinction between sparse and dense memory lies in the filtration-splice strategy. In dense memory mode, MPs consider all nodes as potential interpolation candidates, as evidenced by the brown indices (A: $\in$ $[0, 15]$ and B: $\in$ $[48, 63]$) shown in Fig. 3. Conversely, sparse mode operates differently by omitting any nodes within N/A blocks and incrementing node indices solely for non-N/A blocks, as indicated by the black strokes. This distinction underscores the inefficiency of dense memory, where the executor unnecessarily accesses nodes in two N/A blocks, effectively doubling memory access and reducing cache-hit rates.

In sparse memory mode, the filtration stage is followed by a splice stage that produces a finalized node index. The filtration process generates blocks irrespective of their physical adjacency, while consistently ensuring a memory adjacency feature. As illustrated in Fig. 3, blocks A and B are mapped to adjacent memory locations within a unit of 16 cells; specifically, block A occupies the first range of cells 0–15, while block B occupies the second segment of 0–15. The terms first and second refer to the global offset, while the ranges 0–15 denote the local offset. Consequently, the sparse memory updates the memory range of block B from $[48, 63]$ to $[16, 31]$.

It is important to note that both global and local offsets increase in a Z-Curve shape, arranged in column-major order. The Z-Curve facilitates a unique encoding and decoding process in dense memory mode, linking the decimal and binary representations of a given index (Setaluri et al., 2014). For instance, considering a cell with an index of 60, its decimal coordinates are (7, 5), which correspond to the binary values 0111 and 0101, respectively. By interleaving the bits of X and Y starting from the least significant bit and organizing them into 64-bit slots, as illustrated in Fig. 3, we obtain the encoded binary index (00111100) indicated below the brown index marker, following a color-to-color matching scheme. The final 4 bits, 1100, are derived from the last two bits of X (11) and the last two bits of Y (00), while the initial 4 bits, 0011, result from the interleaving of the higher two bits of X (01) and Y (01). It is evident that cells within the same block consistently share the same high 4 bits, with the low 4 bits varying from 0 to 15. These high bits signify a global offset in dense mode; for example, block B, represented as $0011_{(2)} = 3_{(10)}$, corresponds to a global offset of $3 \times 16 = 48$, whereas block A, $0000_{(2)} = 0_{(10)}$, denotes a zero global offset.

The int64 index, encoded from the X and Y coordinates of any cell, implies that blocks can be sorted based on their leading cell index and then forwarded to assist in a consistent splice process. Since block B holds a global offset of 48, which is larger than the global offset of 0 of block A, block B is joined to the tail of block A and assigned a sparse index of 1. Assuming there are more than two blocks perhaps hundreds of thousands of blocks registered as non-N/A, we can adopt the same encoding method and sort the global offsets of the blocks. The sorted indices will replace their global offsets and contribute to a memory offset by adding the local offset decoded from the lowest 4 bits. Note that if the simulation is carried out in three dimensions, the int64 index changes slightly; the high bits consist of 58 digits, and the low bits consist of 6 digits.

3.2.1. Int64 index of material points

Since each MP within the grid domain can receive a unique cell index which is encoded in int64 format, a reordering of the MPs is expected to increase the memory cache hit rate. Under two-dimensional conditions, two supporting mask indices are defined to encode the decimal X or Y coordinate into binary as $mask_X = 0xaaaaaaaaaaaaaaac$ and $mask_Y = 0 \times 5555555555555553$, and the encoding process is shown in Algorithm 2.

For all the MPs, the int64 indices are determined from their positions and the spacing of nodes, as shown in Algorithm 3.

3.2.2. Hash the int64 index to blocks

Fig. 4 demonstrates a sparse sampling of MPs. Three types of blocks are considered according to different material resolutions: **Blk**, **Blk ghost**, and **N/A**. The **Blk** type plays a dominant role in the P2G and G2P stages. **Blk ghost**, which contains no MPs, supports padding of attributes in the high-order interpolation method. **N/A** blocks are assigned no operations and serve as virtual placeholders in sparse

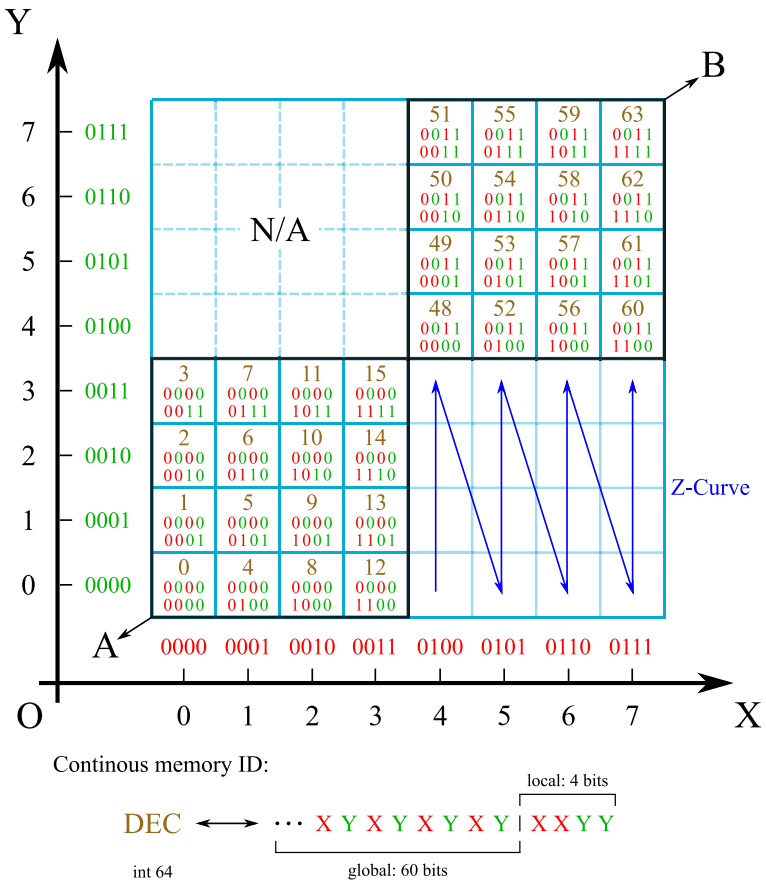


Fig. 3. Memory indices transfer from continuous to sparse.

Algorithm 2: To convert the material point index from decimal to binary.

Input: Point binary row/column index *id* and *maskid*
Output: Encoded binary index *eid*

```
Function maskPos(id, maskid):
    eid ← 0
    masks ← {0x aaaaaaaaaaaaaaac, 0x 5555555555555553}
    mask ← masks[maskid]
    while id ≠ 0 do
        for i ← 0 to 64 do
            if (mask & (1 ≪ i)) ≠ 0 then
                eid ← eid | ((id &1) ≪ i)
                id ← id ≫ 1
    return eid
```

Algorithm 3: Calculation of int64 indices for material points.

Input: point number n, point position p, node spacing d
Output: point index *pid*

```
for i ← 0 to n − 1 do
    x_index ← int(p[i][0]/d[0]);
    y_index ← int(p[i][1]/d[1]);
    // ◁ Algorithm 2
    x_e,index ← maskPos(x_index, 0) ;
    y_e,index ← maskPos(y_index, 1) ;
    pid[i] = x_e,index | y_e,index;
```

memory mode.

Recalling the filtration-splice procedure necessary for sparse memory, it is crucial to determine the number of **Blk** and sort them prior to any subsequent MPM iterations. After obtaining the int64 indices of MPs via Algorithm 3, the next step is to create a hash for block filtration. Each MP decodes the high bits of its int64 index to calculate a block index, which is then hashed into a large hash table according to its magnitude. This hashing process operates in parallel across blocks. If the first MP in a block successfully completes the int64 index calculation and hashes the block index into the hash table, a boolean flag is set to True. However, concerns regarding the size of the hash table must be addressed: a table that is too small increases the likelihood of hash collisions, while an excessively large table leads to unnecessary memory overhead. To mitigate these issues, we propose a hash table size that is double the predicted number of blocks, although this remains an open question.

Algorithm 4: Create hash maps of the parent block of material points.

Input: point index *eid*, hashsize *hsize*
Output: hash values *pval*

```
for id ∈ eid do
    bid = dec(id >> 4);
    hashid = hashfunction(bid, hsize);
    if atomicRead(pval[hashid]) == 0 then
        atomicWrite(pval[hashid], 1);
    else
        return ;
```

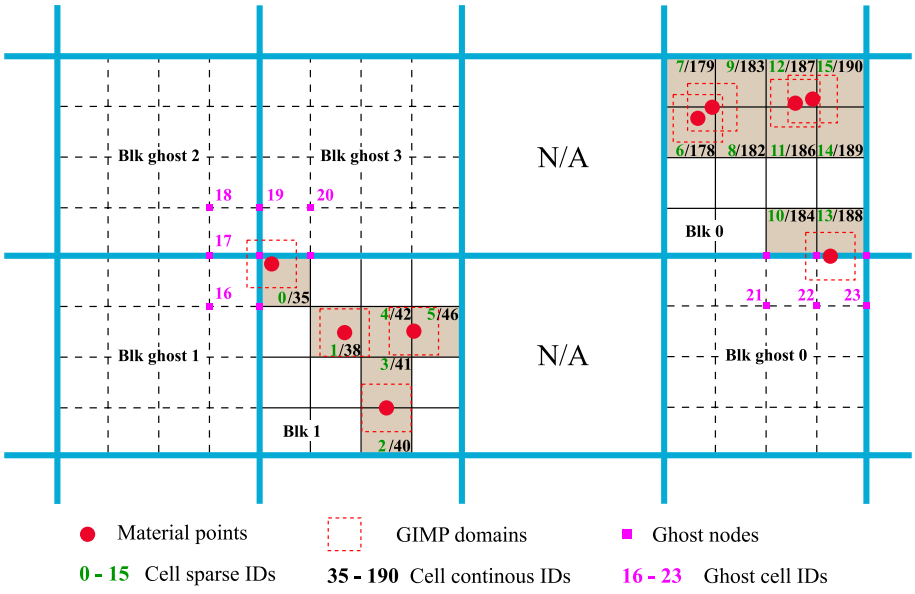


Fig. 4. Reorder of the material points according to a hierarchical block-cell topology.

Table 1
The hash map of sparse distributed blocks.

Hash id	0	1	2	...	4997	4998	4999	5000	5001	...	N
Hash state	0	1	0	...	0	1	0	0	1	...	1
Prefix-sum	0	1	0	...	0	10	0	0	213	...	754

Table 2
Mapping between unsorted indices of MPs and cells.

Material point ID	0	1	2	3	4	5	6	7	8 ($N_p - 1$)
Cell continuous ID	187	178	46	35	187	40	38	178	188

Table 3
Mapping between sorted indices of MPs and cells.

Parent block ID	32	32	32	32	176	176	176	176	176
Cell continuous ID	35	38	40	46	178	178	187	187	188
Material point ID	3	6	5	2	7	1	0	4	8
New block ID	0	0	0	0	1	1	1	1	1
New cell local ID	3	6	8	14	2	2	11	11	12
New material point ID	0	1	2	3	4	5	6	7	8

Table 1 pairs a hash table of size N with hash states corresponding to Algorithm 4. Any occupied hash table slots are marked as True and attached to a global block index. The splice stage will execute a prefix-sum and use the results to replace the original global block index with a compressed sparse block index.

3.2.3. Hash the int64 index to cells

Our previous study on GPU-accelerated DEM introduced a dual mapping technique that integrates particles and cells (Zhao et al., 2021). This innovative method is designed to efficiently sort and organize particles within each cell. Building on this concept, we have developed a MP-cell dual mapping framework utilizing a key–value bound sort based on the int64 index of the MPs and the sparse indices of the cells. For instance, consider the randomly distributed MPs illustrated in Fig. 4 and detailed in Table 2. The first row of Table 2 presents an unsorted index sequence ranging from 0 to $N_p - 1$, where N_p represents the total number of MPs. Subsequently, the results of applying a key–value bound sort to the data in Table 2 are summarized in Table 3.

Cell indices, a crucial component of the key–value sort, are presented in the fifth row of Table 3 and represent an increasing order distribution. Unique cell indices indicate that a single MP is associated with a given cell, while duplicate indices signify multiple MPs sharing the same cell. Correspondingly, duplicated block indices in the fourth row illustrate that multiple cells are activated within the same blocks. The arrangement of these two sorted rows implies that any duplicate cells or blocks are likely to be closely connected in memory, resulting in favorable cache hit rates. Building on this premise, compressed and zero-based new indices for blocks and MPs are derived in the fourth and sixth rows of Table 3, thereby completing the sparse memory mapping for blocks and MPs. It is important to note that the new cell indices presented in the fifth row do not follow a zero-based or sequential numbering system; rather, they are defined by their local offset within the parent block to enhance the efficiency of sparse memory index calculations. Additional technical details are provided in Algorithm 5.

Two auxiliary data structures, the boundary of cells and the boundary of blocks, are introduced in Algorithm 5 to facilitate index calculation for blocks and cells in the filtration and splice sparse memory operations. These two arrays are initialized to False. During the reorder process in the splice stage, any MPs or cells residing at a border between regions of duplicates and non-duplicates will set their parent cell's or block's boundary arrays to True. The two arrays, with sampled True and False states, will be subjected to a prefix-sum, and their discontinuous dense mode indices will be converted to continuous indices in sparse memory mode. Each block will be assigned a range of MP indices denoted as pid_{rb}, and each cell will similarly be assigned data denoted as pid_{rc}. Accordingly, each block and cell can calculate the number of MPs they hold by subtracting the range of MPs of their previous neighbor from their own. When Algorithm 5 is done, a dual mapping between MPs-blocks and MPs-cells is fully established, and these data will be used in the P2G and G2P stages. Algorithm 5 not only reports the relationship between MPs and blocks/cells but also clarifies how the MPs' order changes between any two neighboring timesteps. In other words, a bi-directional query table is obtained to help any MP with index id_n at step n to find the index at the last step id_{n-1} or at the next step id_{n+1}. In this way, kinematics or dynamics of MPs can be shifted easily between any two time series. For special cases, such as material types that always need a mapping between step 0 and step n instead of a common mapping between step $n-1$ and n, a direct query table can also be maintained by a reverse ordering of the id_n, as shown in Algorithm 6.

Algorithm 5: Create hash maps of the parent cells of material points.

Input: sorted int64 IDs ids, sorted material point sequence pid_0
Output: local continuous IDs of cell cid, particle range of blocks pid_{rb}, particle range of cells pid_{rc}, new block IDs bid

$bc_{cell} \leftarrow \{0\} * N_p$; $bc_{block} \leftarrow \{0\} * N_p$; $pid = pid_0$;
for $i \leftarrow 0$ **to** N_p **do**
  $parid = ids[i]$; $globit = parid >> 4$; $locbit = parid \,\&\, 15$;
  if $i < N_p - 1$ **then**
    if $globit! = ids[i+1] >> 4$ **then**
      $bc_{block}[N_p - 1] = 1$;
      $bc_{cell}[N_p - 1] = 1$;
    else
      if $locbit! = ids[i+1]\&15$ **then**
        $bc_{cell}[N_p - 1] = 1$;
  else
    $bc_{cell}[N_p - 1] = 1$;
    $bc_{block}[N_p - 1] = 1$;
$cid \leftarrow newbc_{cell} \leftarrow prefixsum(bc_{cell})$;
$bid \leftarrow newbc_{block} \leftarrow prefixsum(bc_{block})$;
for $i \leftarrow 1$ **to** N_p **do**
  if $newbc_{block}[i] != newbc_{block}[i-1]$ **then**
    $pid_{rb}[newbc_{block}[i]].x = i$; $pid_{rb}[newbc_{block}[i-1]].y = i$;
    $pid_{rc}[cid[i]][newbc_{block}[i]].x = i$;
    $pid_{rc}[cid[i]][newbc_{block}[i-1]].y = i + 1$;
  else
    if $newbc_{cell}[i] != newbc_{cell}[i-1]$ **then**
      $pid_{rc}[cid[i]][newbc_{block}[i]].x = i$;
      $pid_{rc}[cid[i]][newbc_{block}[i]].y = i + 1$;

Algorithm 6: Dual-mapping for material point IDs.

Input: sorted material point sequence pid_0, material point number N_p
Output: reverse material point IDs map pid_{rmap}

for $i \leftarrow 0$ **to** N_p **do**
  $pid_{rmap}[pid_0[i]] = i$;

3.3. GPU cell-wise mapping between particles and grid nodes

3.3.1. Mapping from particles to grid nodes (P2G)

Up to now, all the facilities for P2G are ready, including the establishment of sparse indices for MPs, cells, and blocks. Auxiliary metadata that help to explain the relationships among the indices of MPs, cells, and blocks are also listed. Fig. 5 presents the two subtypes of the P2G stage: block-block P2G, represented by **Blk 0-Blk 1**, and block-ghost block P2G, represented by **Blk 0** to **Blk ghost 0-3**. Under the application of the GIMP interpolation method, the shape function domain of each MP is 3 × 3 in two dimensions, and with one layer of padding forms a final domain of 4 × 4. This means that P2G requires both intra-block and inter-block memory access.

The magenta MPs in Fig. 5 represent intra block-block P2G, while those colored red, blue, and green and located at the margins of the block denote P2G between block and ghost block. Accordingly, two different strategies are proposed for them: gathering P2G and scattering P2G. The gathering P2G is executed from the node side, where no race conditions are implied. The scattering is called from the MP side, where an appropriate race condition remedy is required.

Algorithm 7: P2G in gathering mode.

Input: space of nodes $space$, attributes of material points $data_p$, position of material points pos_p, particle range of block pid_{rb}, status of block $state_b$, cell status of block $state_c$, particle int64 index pid, hash table of blocks $hash_b$
Output: attributes of internal nodes $data_{ndi}$

$thread = 64$;
$shmem = \{\}$;
for $i \leftarrow 0$ **to** N_b **do**
  $block\_int64\_id = pid[pid_{rb}[i].x]$;
  for $thid \leftarrow 0$ **to** $thread$ **do**
    $nd\_id = th\%16$;
    $cell\_id = th/16$;
    $cell\_x = cell\_id/4 - 2$; $cell\_y = cell\_id\%4 - 2$;
    $block\_offx = cell\_x < 0\,?\,-1 : 0$; $block\_offy = cell\_y < 0\,?\,-1 : 0$;
    $new\_bid = hash_b(block\_int64\_id + int2(block\_offx, block\_offy))$;
    $cell\_x\% = 4$; $cell\_y\% = 4$;
    $cell\_id = cell\_x * 4 + cell\_y$;
    $cell\_status = state_b[cell\_id][new\_bid]$;
    if $cell\_status$ **then**
      $pid\_begin = pid_{rc}[cell\_id][new\_bid].x$;
      $pid\_end = pid_{rc}[cell\_id][new\_bid].y$;
      $local\_pdata = \{\}$;
      for $iterp \leftarrow pid\_begin$ **to** pid_end **do**
        $GIMP(local\_pdata, space, cell\_id, pos_p[iterp], data_p[iterp])$;
      $sharedMemoryWrite(shmem, thid)$;
  $\_\_syncthreads()$;
  $blockReduce()$;
  $globalMemoryWrite(shmem, data_{ndi})$;

Refer to Algorithm 7 for the gathering mode of nodes in P2G. Under GIMP interpolation, each node may search for MPs in attached blocks, up to a number of 4 in two dimensions. With 16 cells per block, there may be up to 64 candidate cells to search for MPs. Each thread needs a prerequisite check to determine whether the target cell exists or not, either due to a N/A block or being outside of the shape function domain. For any existing cell, the thread proceeds to perform a weighted interpolation from any child MPs in this cell. Shared memory is allocated at the block level to collect the P2G gathered attributes from MPs. Considering the large ratio of MPs to nodes, at least one warp will be assigned for each node to reduce and gather the attributes of the MPs. After a warp-wise reduction operation accelerated by CUDA intrinsics, the shared memory is written to the global memory of nodes, and the gathering P2G is completed. We follow the rule that bordering nodes only bound to the block in the top-right direction if any BCs need a P2G stage in attributes as well. This regulation aims to remove any duplicate P2G in the BC attributes interpolation.

Block-ghost block P2G adopts a different scatter behavior (Algorithm 8) than block-block P2G because a single ghost block may link to multiple non-ghost blocks, causing the gather loop to experience significant memory address jumps when switching among blocks. As a result, each non-N/A block will independently perform a scatter process and map attributes to the target ghost block. The scatter of non-N/A blocks uses a lazy interpolation strategy, as the attributes in the shape function domain are first reduced into a virtual cache and then the combined results are transferred to the destination memory. This approach helps to mitigate the write latency caused by memory jumps.

The parallel topology in terms of blocks and threads is derived from the distribution density of MPs. A direct block-to-thread logic

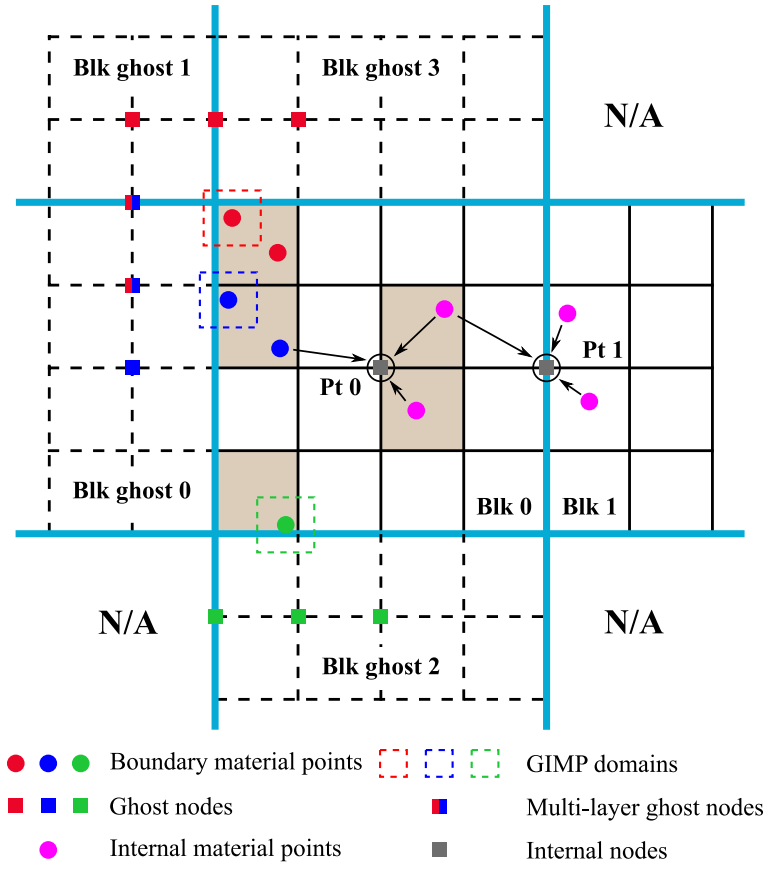


Fig. 5. Mapping the attributes of the material points onto the nodes.

is not straightforward but is modified at an block-warp-thread level. For example, if there are two blocks, A and B, consisting of different numbers of particles 500 and 100, respectively, and a minimum number of threads activated is 256, block A will be divided into two bins with 256 threads each, equivalent to 8 warps, with the latter bin padded by $256 - 244 = 12$. Block B will be assigned to one bin corresponding to 8 warps and padded by $256 - 100 = 156$. Therefore, the stream processors of the GPU card schedule a total of 24 warps for 3 blocks. In each warp, threads with sequential indices are guaranteed to belong to MPs that possess good locality, i.e., they either share the same cells or the same bordering nodes. Warp intrinsics can then be applied for better P2G reduction and to facilitate efficient coalesced reads and writes from shared memory to global memory.

3.3.2. *Mapping from grid nodes to particles (G2P)*

The interpolation of dynamics from nodes to MPs follows a point-side gather policy, which means only non-ghost blocks execute the G2P stage. As shown in Fig. 6, MPs collect the updated dynamics regardless of whether the nodes are inside blocks or ghost blocks (indicated by dashed padding cells). The algorithm for the G2P stage is similar to that of Algorithm 8, without another scatter component as in the P2G stage.

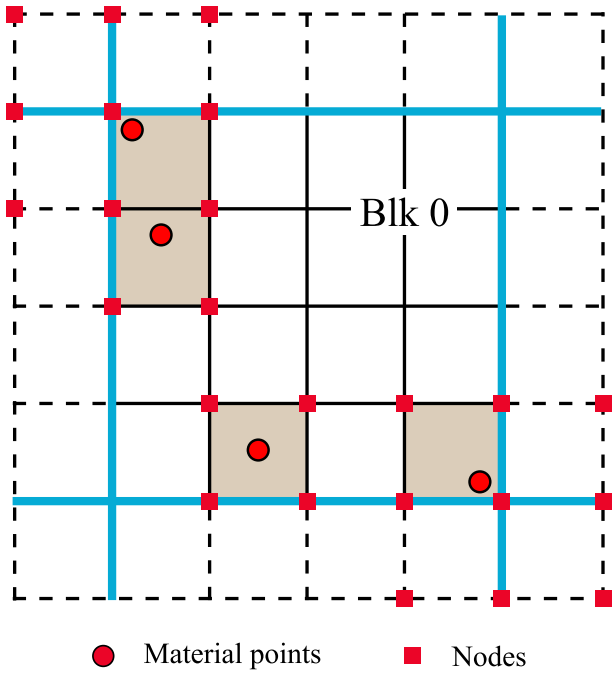


Fig. 6. Interpolation of dynamics from nodes to material points.

3.4. *Boundary condition*

Summarized by the De Vaucorbeil et al. (2020), types of BCs in MPM can classified into Dirichlet, Neumann and rigid bodies contact. From the implementation prospects, an synonym of the BC types of Dirichlet and Neumann can be also defined as: MP and node BC.

The imposition of MP BC primarily covers moving loads, traction loads, body force loads, and gradient loads. Except for body forces, which simply introduce user-defined forces with time dependencies on MPs, the other three types require extrapolation to all nodes surrounding the MPs; otherwise, artefacts might be anticipated. Moving loads

Algorithm 8: P2G in scattering mode.

Input: split material points offset pt_begin, split material points number $N_{p,b}$, attributes of material points $data_p$, position of material points pos_p, particle range of block pid_{rb}, status of block $state_b$, cell status of block $state_c$, particle int64 index pid, hash table of blocks $hash_b$

Output: attributes of internal nodes $data_{ndi}$

```
thread = 256;
shmem = {} * 16;
for i ← 0 to N_{b,split} do
    base_pt_id = pid_{rb}[i];
    for thid ← 0 to min(N_{p,b}, thread) do
        global_pt_id = thid + pt_begin + base_pt_id;
        local_cell_id = pid[global_pt_id]&15;
        cell_mask = warpReduce(local_cell_id != pid[global_pt_id + 1]&15);
        cell_status = {};
        for c ← 0 to 16 do
            cell_id = c/16;
            cell_x = cell_id/4 − 2; cell_y = cell_id%4 − 2;
            block_offx = cell_x < 0 ? −1 : 0; block_offy = cell_y < 0 ? −1 : 0;
            new_bid = hash_b(block_int64_id + int2(block_offx, block_offy));
            cell_x% = 4; cell_y% = 4;
            cell_id = cell_x * 4 + cell_y;
            cell_status[c] = state_b[cell_id][new_bid];
        local_data_nd = GIMP(local_cell_id, cell_status, data_p);
        warpReduce(cell_mask, shmem, local_data_nd);
    __syncwarps();
    for ndid ← 0 to 16 do
        atomicGlobalMemoryWrite(shmem, data_ndi);
```

require that the nodes within the shape function domain of MPs receive specified value constraints (such as velocity, pore pressure, or temperature) from the MPs. This reception can occur through direct inheritance or as a weighted value based on the shape function. Traction and gradient loads do not require all nodes around the MPs but only those around two vertices on one side of the edge of the uGIMP domain of MPs, if a direction is provided. BC values or gradients are first determined at the vertices of the MPs and then mapped to the supporting nodes of these vertices through weighted mapping. Referring to Algorithm 4, once the int64 index of MPs is obtained, the block index and local offset with one layer of padding also become apparent. The bounding cell and the surrounding nodes can then be identified for BC imposition. Since a cluster of MPs within one cell may share the same BC value in single-material mode and have limited BC values in multi-material mode, BC imposition is not performed in parallel on the MP side but rather on the cell side. Each cell selects the first MP in each material domain within its MP range (pid_{rc}) and performs weighted extrapolation of the BC to eliminate any heavy atomic operations locally.

Node BC applies any predefined constraints on nodes and keeps a fixed target throughout the entire simulation. This BC usually consists of symmetric and mirror velocity constraints or value settings for displacement, temperature, and surcharge under the assumption of small deformations. If any MPs move out of the shape function domain, they will lose connection to the BCs. Executing node BCs involves checking whether node memory is actually allocated in either blocks or N/A blocks. If not, the BC will be omitted. Otherwise, the X and Y coordinates of nodes will be encoded into an int64 index and transferred to the indexes of cells and blocks to affect the MPs. The BC information will be collected by MPs during any necessary stages, e.g., deformation gradient calculations or G2P stages, and will be reset at the end of each timestep. Compared to the MP BC, the node BC overhead is much lower, but its versatility is also reduced, as the nodes are fixed and fail to track any moving MPs to provide an attached BC.

Rigid body contact aims to provide a DEM-contact law-driven boundary condition that moves with its prescribed velocity. The influence of rigid body contact on MPs does not require a node to extrapolate any dynamics; instead, it corrects the MPs' response based on a DEM contact criterion. Specifically, if any contact penetration occurs between an MP, represented by a virtual radius, and a rigid body geometry, the normal and tangential forces are determined by contact laws, such as the Linear-Spring law, Hertz–Mindlin law, or advanced barrier law adopted in Jiang et al. (2022) and Liang et al. (2024). The shapes of the rigid bodies can be analytical, such as spheres, rectangles, or super-ellipsoids, or they can be numerical, formed by geometric primitives like lines and triangular meshes. In the context of granular flow, a ground with complex elevation features is often involved as a boundary condition. Thus, a contact correction for the rigid body boundary condition between MPs and triangular mesh terrain is used for demonstration, but it can also be extended to other shapes without incurring significant overhead.

Metadata describing the terrain typically require preprocessing to convert them into a more program-friendly data format, optimizing for parallel processing and computational efficiency. For instance, irregular digital elevation model mappings are often re-interpolated into an orthogonal mesh format. This results in a structured matrix of adjacent cells, where each cell encompasses four vertices of elevation data, and two triangles are derived from each cell to generate the corresponding face normal information. Our framework efficiently manages the interaction between material points and digital terrain structures in parallel by employing a division operation between the positions of the MPs and the terrain space to ascertain the cell index.

Given a material point denoted by its position vector $\boldsymbol{x}_p$ and a virtual radius r_p, the indices of potential contact cell candidates are determined as follows:

$$\boldsymbol{x}_c = \boldsymbol{x}_p \pm r_p \vec{\boldsymbol{e}} \tag{9a}$$

$$\boldsymbol{id}_c = int(\frac{\boldsymbol{x}_c}{\boldsymbol{\delta}}) \tag{9b}$$

where $\boldsymbol{x}_c$ represents the base vertex coordinates of the cell, $\vec{\boldsymbol{e}}$ signifies the normalized unit direction, $\boldsymbol{id}_c$ corresponds to the cell index vector, and $\boldsymbol{\delta}$ denotes the mesh spacing.

The one-dimensional flattened indices are constructed as follows:

$$\boldsymbol{id}_{all} = cell[\boldsymbol{id}_{c,xmin} : \boldsymbol{id}_{c,xmax}, \boldsymbol{id}_{c,ymin} : \boldsymbol{id}_{c,ymax}, \boldsymbol{id}_{c,zmin} : \boldsymbol{id}_{c,zmax}] \tag{10}$$

where $\boldsymbol{id}_{all}$ encapsulates all potential contact candidates within a flattened one-dimensional index. Notably, this approach can be optimized when dealing with fixed boundary digital terrain; since the cell information derived from the int64 index has already been processed in the P2G stage, only a linear offset from the MPM mesh to the digital terrain is required, thereby eliminating the need for further floating-point division and index flattening.

Although there can be up to eight possible contact candidate cells, accounting for variations in the z-direction coordinates limits the looping to a maximum of four cells and eight triangles during contact force calculations. Considering a triangle defined by its vertices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$; the projection procedure from material points onto the triangle (Guilkey et al., 2021) is derived as follows:

$$\boldsymbol{n} = (\boldsymbol{B} - \boldsymbol{A}) \times (\boldsymbol{C} - \boldsymbol{A}) \tag{11a}$$

$$d_p = (\boldsymbol{x}_p - \boldsymbol{A}) \cdot \boldsymbol{n} \tag{11b}$$

$$\boldsymbol{p} = \boldsymbol{x}_p - d_p \boldsymbol{n} \tag{11c}$$

$$r_{proj} = \sqrt{r_p^2 - d_p^2} \tag{11d}$$

where $\boldsymbol{n}$ is the normal vector of the triangular plane, d_p represents the distance from the material point to the triangular plane, $\boldsymbol{p}$ denotes the

projected position vector of the material point onto the plane, and r_{proj} corresponds to the projected radius derived from the virtual radius.

The combination of $\boldsymbol{p}$ and r_{proj} defines a disk that enables the evaluation of potential intersections between the disk and the triangle. The intersection area S_{proj} between the disk and the triangle leads to the definition of a penalty β given by $\beta = S_{proj}/\pi r_{proj}^2 \in (0, 1)$, which is applied to the contact forces to mitigate excessive contact behavior, as shown below:

$$\boldsymbol{f}_n = \beta K_n (r_p - d_p)\boldsymbol{n} \tag{12a}$$

$$\boldsymbol{f}_t \mathrel{+}= \beta K_t \Delta \boldsymbol{v} \Delta t \tag{12b}$$

where $\boldsymbol{f}_n$ and $\boldsymbol{f}_t$ represent the normal and tangential contact forces, respectively; K_n and K_t indicate the normal and tangential contact stiffness; $\Delta \boldsymbol{v}$ signifies the velocity difference between the material point and the triangular plane, and Δt is the timestep.

This contact method exhibits a time complexity of $O(1)$, effectively bypassing the need for intensive loops through numerous triangular meshes, and negates the necessity for advanced techniques such as Level-Set methods for reflecting complex terrains. Should either the material shape or the boundary mesh become non-spherical, more sophisticated contact algorithms can be incorporated, as demonstrated in Liang et al. (2024) and Chen et al. (2023).

3.5. Iteration loop

We begin by outlining the essential properties of a typical material constitutive model to illustrate the implementation of memory shifts in GPU-based MPM simulations. Key material properties include the deformation gradient, rate of deformation gradient, stress, and plastic index. The update process for these properties can be executed in either incremental or full deformation gradient forms. In the incremental deformation approach, the rate of deformation gradient is updated first, followed by the stress. Within this framework, it is unnecessary to maintain a full form of deformation gradient matrix; only a memory cache for storing stress and plastic index values is required. Consequently, even if the MPs are reordered between time steps, the memory position shift of the deformation gradient remains dead status. The central focus here is only on establishing the rate of deformation gradient within each time step, which we characterize as an in-place shift.

For deformation-dependent materials, a memory cache for the full form of the deformation gradient is essential, necessitating memory shift operations at each time step. Once the in-place generation of the rate of deformation gradient is completed for a given time step, updates to the deformation gradient and stress are performed. At the end of each time step, the deformation gradient, stress, and plastic index undergo a memory shift. In three-dimensional space, this entails allocating $3\times3 = 9$ slots for the deformation gradient, 6 slots for stress, and potentially 1 slot for the plastic index, totaling 16 memory slots. A pre-allocated memory shift workspace serves as a buffer for encoding these timestep-wise memory shifts, utilizing the query table pid_{map} obtained from Algorithm 6. Beyond the in-place and timestep-wise shifts, a special scenario arises when certain materials require more complex stress notations (e.g., Kirchhoff stress) instead of standard Cauchy stress. To facilitate efficient transitions among different stress definitions, a constant initial volume is preserved, necessitating a memory slot for reverse-mapping the current reordered index to its initial state, as enabled by the mapping identifier pid_{rmap} defined in Algorithm 6.

In multi-material domains, each material block maintains its own mapping tables to record the local indices of material points. The material kernels execute in a serialized manner, ensuring local parallelism within each block while synchronizing with global material point reordering to optimize memory caching. In parallel with managing material properties during memory shifts, common kinematics encountered in MPM, such as mass, velocity, and position, are also addressed. Notably, specific optimizations can be employed when all materials share a same mass or when uniform GIMP assumptions apply, allowing for the elimination of mass and interpolation domain reordering. However, reordering of velocity and position is still necessary. In the context of the USF and USL, velocity is exclusively read during the P2G phase and written during the G2P phase, warranting a timestep-wise shift. Conversely, in the MUSL case, where remapping occurs between MPs and nodes, velocity experiences an in-place shift during the G2P stages. On the other hand, the position attribute consistently undergoes an in-place memory shift during the G2P phase, while its timestep-wise shift is conducted lazily during the reordering stage.

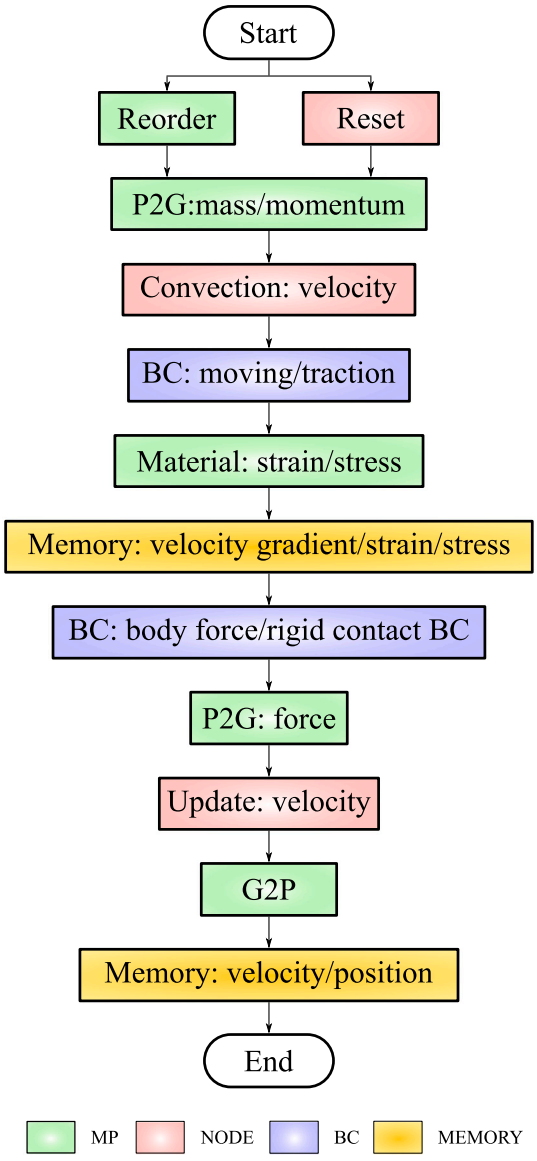


Fig. 7. The iteration flowchart of USF in GPU MPM.

The sparse MPM framework has the following iteration stages in one time step (using USF as an example) (see Fig. 7) :

3.6. Remarks

Frequently executing MP reordering may be unnecessary for quasi-static problems, as the advantages do not justify the costs associated with the reordering kernel. Instead, implementing periodic MP reordering can strike a reasonable balance. When allocating sparse memory for a block, the overall memory size is determined by the data channels

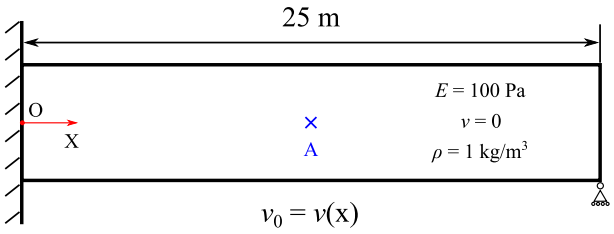


Fig. 8. Simulation setup for the one-dimensional wave propagation example.

present in the nodes. In a three-dimensional context, assuming a single phase of material, the total number of data channels for mass, acceleration, velocity, momentum, and force adds up to 13. This configuration remains manageable as long as the node channel size does not greatly exceed 4 KB, which is the standard memory unit size typically handled by CPUs and GPUs. However, in multi-phase or multi-physics problems, an increase in data channels can result in substantial performance degradation. To address this, our framework introduces a memory reuse technique that enables the sharing of workspace buffers between the material properties shift stage and the MP reordering stage, thereby reducing overall memory allocation. Another potential solution is to stagger the execution of the mechanical kernel and other kernels, which could facilitate latency hiding on modern GPUs by leveraging asynchronous computation and I/O kernels.

4. Benchmarks and scalability tests

4.1. Elastic solution: One-dimensional wave propagation

The one-dimensional wave propagation in an elastic beam is analyzed to validate the basic functionality of the proposed MPM framework, as illustrated in Fig. 8. The beam length is set to $L = 25$ m with all degrees of freedom constrained on the left side, while the right side remains only horizontal displacement. The material properties of the elastic beam are specified as follows: Young's modulus ($E = 100$ Pa), Poisson's ratio ($\nu = 0$), and density ($\rho = 1$ kg/m^3). An initial velocity distribution is defined as $v(0, x) = v_0 \sin(\frac{\pi}{2L}x)$, where v_0 has two options: 0.1 m/s and 0.75 m/s. These values are chosen to test for any potential cell-crossing issues. Since the Poisson's ratio is zero, the dimensions in the Y and Z directions do not influence the simulation, and their properties are ignored. The resolution of the cells in the X direction is set to 100, and 320,000 particles are generated. The simulation runs for 40 s with a timestep of 0.005 s, and no damping is applied.

The evolution of the probed point (A) in terms of velocity and displacement is shown in Fig. 9. The analytical displacement and velocity are expected to be:

$$x = \frac{v_0}{\beta L \omega} \sin(\omega t) \tag{13a}$$

$$v = \frac{v_0}{\beta L} \cos(\omega t) \tag{13b}$$

$$\beta = \frac{\pi}{2L} \tag{13c}$$

$$\omega = \beta \sqrt{\frac{E}{\rho}} \tag{13d}$$

For consistent comparison, all variables are normalized against v_0. The results demonstrate a high level of agreement between the numerical solution (dots) and the analytical solution (lines), regardless of whether the velocity is low or high. This indicates the stability of the basic sparse memory MPM framework. The simulation operates at an approximate speed of 363 steps/s. However, due to the cubic sampling of the material points, the benefits of sparse encoding in terms of memory savings are negligible.

4.2. Plastic solution: Three-dimensional granular column collapse

A column collapse test based on Bui et al. (2008) is used to evaluate the effectiveness of the reordering technique when incorporating a state-variable plastic model, such as the Drucker–Prager (Drucker and Prager, 1952) model in this example.

The yield function and flow rule associated with the Drucker–Prager model can be expressed as follows:

$$F = \sqrt{J_2} + \eta \sigma_{\mathbf{m}} - \xi c \tag{14a}$$

$$G = \sqrt{J_2} + \bar{\eta} \sigma_{\mathbf{m}} \tag{14b}$$

where the J_2 is the second invariant of deviatoric stress tensor, $\sigma_{\mathbf{m}}$ is the spherical stress, c is the cohesion strength; η, ξ and $\bar{\eta}$ can be obtained by the friction angle (ϕ) and dilation angle (ψ),

$$\eta = \frac{3 \tan\phi}{\sqrt{9 + 12 \tan^2\phi}} \tag{15a}$$

$$\xi = \frac{3}{\sqrt{9 + 12 \tan^2\phi}} \tag{15b}$$

$$\bar{\eta} = \frac{3 \tan\psi}{\sqrt{9 + 12 \tan^2\psi}} \tag{15c}$$

The initial simulation configuration is shown in Fig. 10(a). A cube with dimensions of 0.2 m × 0.05 m × 0.1 m is positioned on the left side on a frictional base and is assigned 64,000 material points. The material properties following the Mohr–Coulomb model are as follows: Young's modulus ($E = 8.4 \times 10^5$ Pa), Poisson's ratio ($\nu = 0.3$), density ($\rho = 2650$ kg/m^3), friction angle ($\phi = 19.2°$), and zero cohesion. All side walls are used to confine the soil column, constraining only the normal displacement. After generating and consolidating the material points through a geostatic process, the column collapse is initiated by suddenly removing the outlet baffle. Damping is set to 1 during the geostatic process and adjusted to 0.1 during the collapse. The simulation duration is 0.5 s, with a timestep of 5×10^{-4} s.

Fig. 10(b) presents a comparative analysis of the experimental results (depicted by the black stroke) and the numerical results (illustrated by the solid contour). The observations reveal a significant deformation in the sand column, with the outline of the free surface aligning closely with the findings reported by Bui et al. (2008). Moreover, a critical failure line corresponds well with the experimental data, indicated by the pink triangles. A more detailed comparison of the profiles is illustrated in Fig. 11, where the data from this study (represented by markers) is shown to align closely with the lines derived from the experimental results (Bui et al., 2008). The simulation operates at a rate of approximately 4750 steps/s. However, we observe that the number of material points may be inadequate to accurately represent the framework's efficiency, as the warm-up cost of the kernel could be similar to the computation costs. Nonetheless, with respect to sparse memory configurations, a nodal memory savings of approximately 59% is achieved, as the upper right void space is not allocated for sparse memory.

4.3. Digital terrain contact: a block sliding on an inclined plane

A sliding block on an inclined digital terrain is used to test the MP-to-Tri-Mesh contact algorithm. As illustrated in Fig. 12(a), a cubic cluster of MPs is placed on a plane with an inclination angle of $\theta = 30°$. The cubic dimensions are 2 m × 1 m × 1 m, with a cell size of 0.0125 m. Notably, the checker pattern of green, red, and blue blocks attached to the cubic, represented by a 4 × 4 × 4 matrix, indicates how the sparse memory is organized. Each block has locally contiguous memory, enabling efficient intra-block P2G and G2P operations. Adjacent blocks with different colors engage in inter-block GIMP interpolation, mitigating the cell-crossing issue. The memory distribution of nodes is similar to that of the material points, with only one additional layer extending from the material point block, while other areas remain as

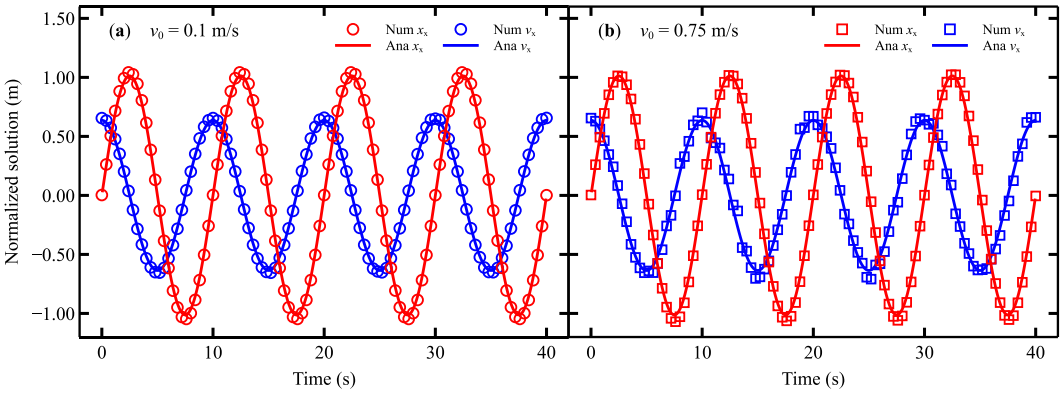


Fig. 9. Responses of velocity and displacement at the probed point (A) for initial velocities: (**a**) 0.1 m/s and (**b**) 0.75 m/s.

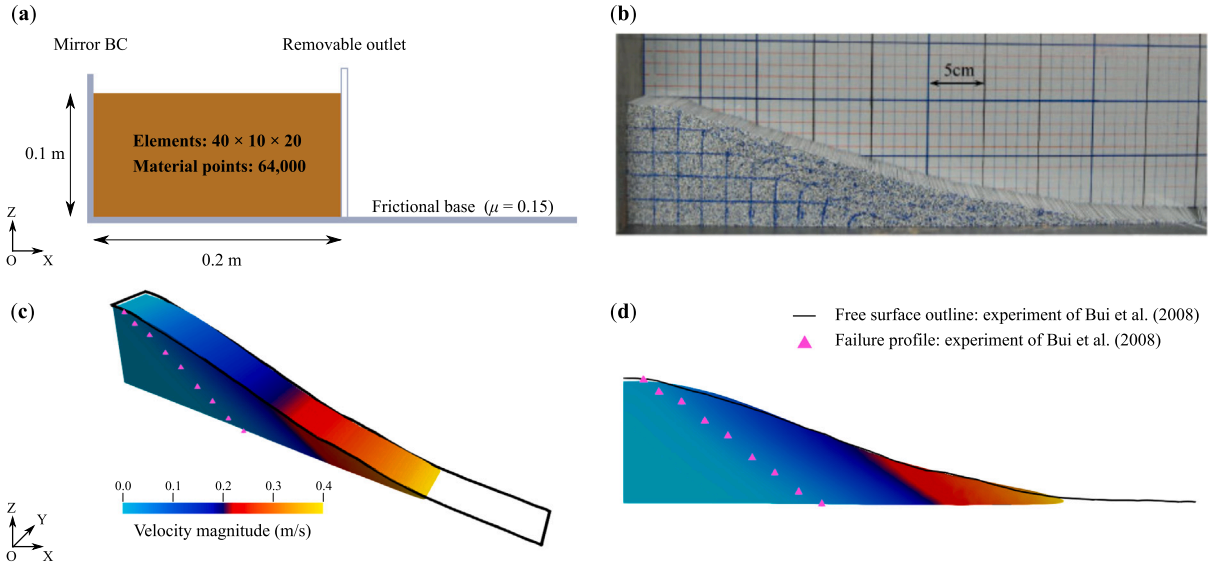


Fig. 10. Three-dimensional granular collapse: (**a**) initial settings, (**b**) experimental results sourced from Bui et al. (2008), (**c**) numerical results from this work, and (**d**) free surfaces and failure lines comparison in XOZ plane.

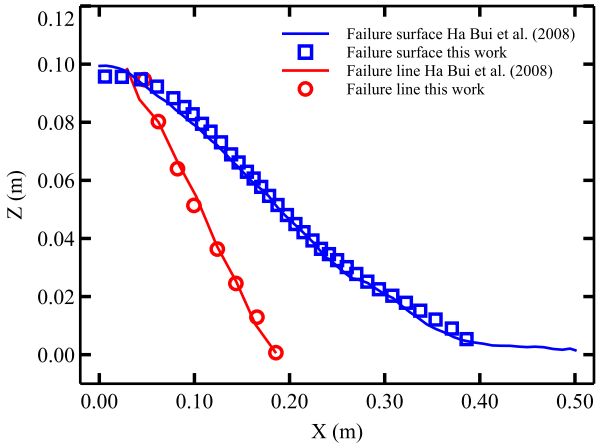


Fig. 11. Comparison of the profile of three-dimensional granular collapse: blue and red lines represent the failure surface and failure line from experimental results (Bui et al., 2008), while the markers denote results from this study.

ghost memory for abstraction.

The cubic is assigned with the following elastic properties: Young's modulus ($E = 1 \times 10^7$ Pa), Poisson's ratio ($\nu = 0.3$), and density ($\rho = 2650$ kg/m^3) to approximate a rigid material. The simulation targets a duration of 0.5 s under gravity. The friction coefficient between the cubic and the plane is set to $\mu = 0.5$. Fig. 12(b) compares the analytical and numerical results for velocity and displacement, using the equations: $x = \frac{1}{2}at^2$, $v = at$, and $a = g(\sin(\theta) - \mu\cos(\theta))$. The simulation operates at an approximate speed of 191 steps/s and is anticipated to achieve a nodal memory savings of 57% due to the dynamic activation of node spaces linked to the material points. The validation of this example shows that the contact algorithm operates effectively under self-gravity conditions for nearly rigid material contact.

4.4. Digital terrain contact: a cylinder touching a plane

The Hertz contact problem of a cylinder in contact with a plane is used to validate the force response under a quasi-static, small deformation regime. The geometrical setup is illustrated in Fig. 13(a). A cylinder with a radius of $R = 8$ m and a limited length in the Y direction makes contact with the plane under a downward surcharge pressure of $p = 0.625$ Pa. The analytical pressure distribution around the contact area is given by:

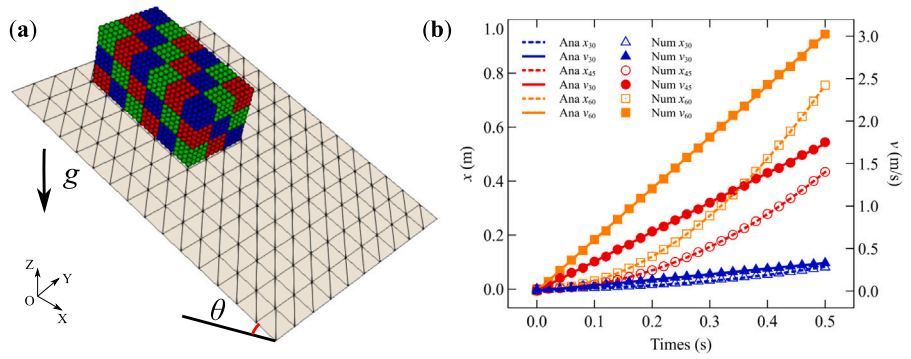


Fig. 12. Sliding block on a slope formed by digital terrain: (**a**) initial configuration and (**b**) variations in velocity and displacement along the slope direction.

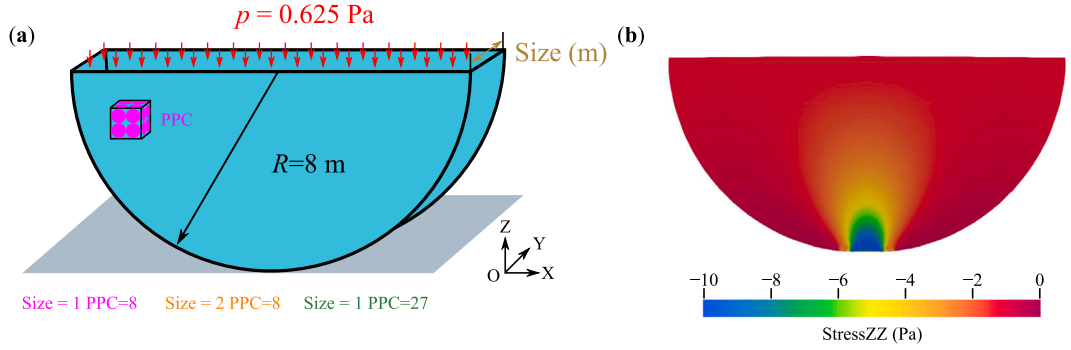


Fig. 13. Hertz contact: (**a**) settings of the numerical simulation and (**b**) vertical stress distribution.

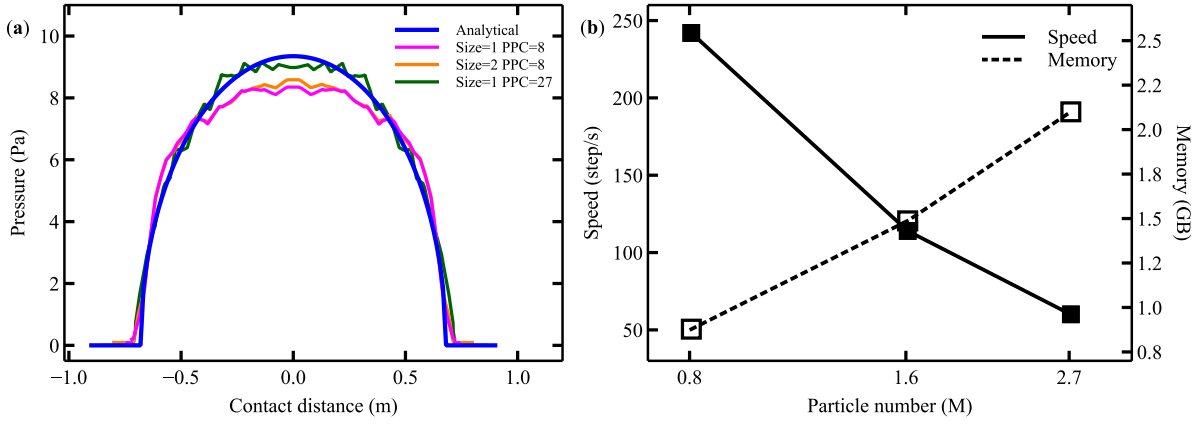


Fig. 14. Hertz contact benchmark and profiling: (**a**) contact pressure distribution and (**b**) evolution of speed and memory consumption versus the number of material points.

$$p_{ana} = \frac{4Rp}{\pi b^2}\sqrt{b^2 - x^2} \tag{16a}$$

$$b = 2\sqrt{\frac{2R^2 p(1-\nu^2)}{E\pi}} \tag{16b}$$

where E and ν denote the Young's modulus and Poisson's ratio of the elastic cylinder, respectively; b represents the contact area with a value of $b = 0.681$ m, as referenced in Johnson (1987).

To determine both the accuracy and efficiency of proposed contact algorithm, three options combining the length of Y as 1 m and 2 m, and particle per cell (PPC) of 8 and 27 are listed, and counting to particle numbers as SIZE 1 + PPC 8 = 804,520, SIZE 2 + PPC 8 = 1,609,040 and SIZE 1 + PPC 27 = 2,714,400, respectively. The simulation runs for 50 s, with a ramped surcharge and a damping coefficient of 0.1 applied to ensure a quasi-static regime.

Fig. 13(b) illustrates the vertical stress distribution within the half-circle section, while Fig. 14(a) presents the evolution of contact pressure around the central contact points. As the dimension in the y-direction increases, the numerical pressure results exhibit a subtle upward trend. In contrast, the PPC refinement shows a significant convergence towards the analytical solutions. The discrepancies observed in SIZE 1 and PPC 27 compared to the analytical solution can be attributed to the method of imposing contact forces. To enhance efficiency, we employ body forces applied to a thin ghost layer of material points instead of utilizing surcharge and contact response as traction forces from the plane. This approach causes the shape function to taper off near the top or bottom boundaries, leading to an incomplete mapping of the body force onto the material points due to the truncation of the shape function. Consequently, this results in a reduced peak force. We plan to address this limitation in future work. Fig. 14(b) demonstrates basic scalability, revealing a nearly linear relationship

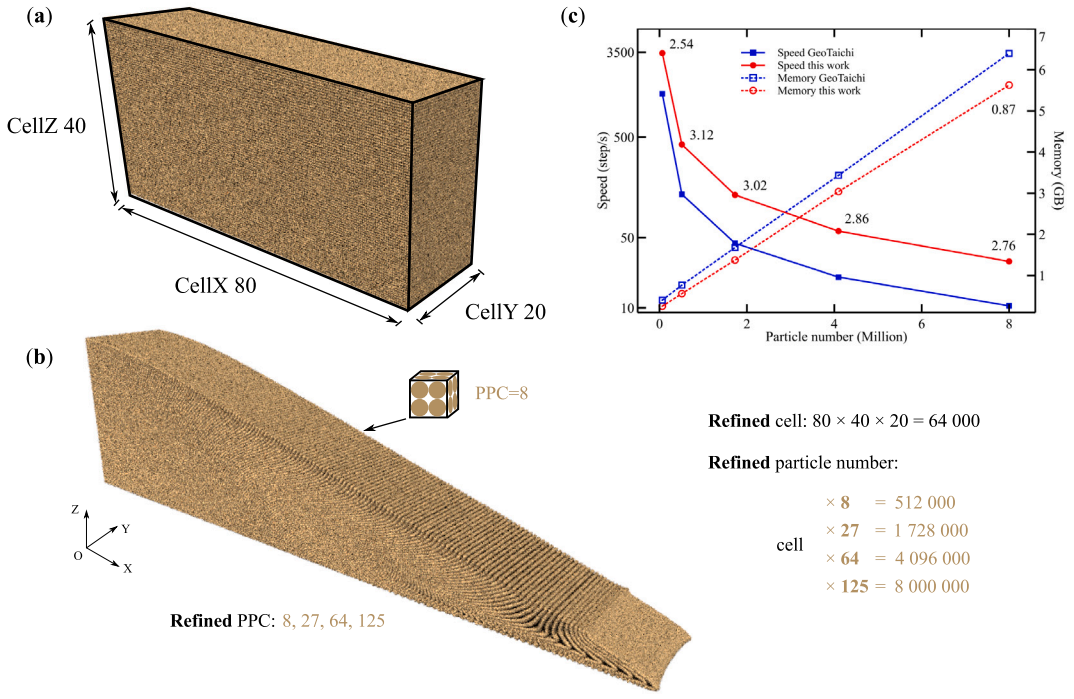


Fig. 15. Scalability test configurations of a large-scale granular collapse with (**a**): a refined eight times number of cells, (**b**): four refined resolutions of material points, and (**c**): scalability test results in terms of speed and memory consumption, along with a comparison against the open-source code GeoTaichi (Shi et al., 2024).

between speed/memory consumption and computational tasks, which indicates robust scalability of our framework.

4.5. Scalability of the GPU-based MPM

Granular materials can undergo large deformations in various modes. The first type may exhibit a continuous spatial distribution; for example, slope instability often maintains geometric continuity even when failure occurs across a widespread area. Alternatively, another mode may demonstrate highly discrete behavior, where the solution domain is sparsely distributed, and each sub-domain loses connection with its neighboring regions. This section will assess scalability in terms of these two dramatically different deformation modes.

4.5.1. Granular collapse

The case presented in Section 4.2, which involves dense spatial distributions of MPs within a large deformation regime, is revisited here to establish a baseline for a scalability test against the open-source framework GeoTaichi (Shi et al., 2024). The geometrical dimensions depicted in Fig. 15(a) are consistent with those outlined in Section 4.2, though the resolutions for both the cells and the MPs have been refined. Specifically, the number of cells has been doubled in each dimension, resulting in a total of $80\times40\times20 = 64{,}000$ cells. Meanwhile, three new configurations for particles per cell (PPC) are introduced, specifically PPC values of 27, 64, and 125, to create a more intensive computational task. The various combinations of refined cells and PPC options yield MP totals of 512,000, 1,728,000, 4,096,000, and 8,000,000. The baseline for the scalability test is established by the configurations described in Section 4.2, where the cell count is 8000 and the number of MPs is $8000 \times 8 = 64{,}000$. For consistency, all data types in this study and GeoTaichi are set to float32. The material properties remain the same as those in Section 4.2. After a geostatic process, damping is set to 0.1, and the full simulation duration is 0.6 s.

In Fig. 15(c), the solid lines illustrate the overall efficiency comparison between this study (red line) and GeoTaichi (blue line). The denoting accelerations adjacent to the line indicate an average acceleration ratio of 2.86 for this framework. Regarding memory consumption (dashed lines), both this framework and GeoTaichi exhibit a linear relationship between memory allocation and particle number; however, the slope for this framework is lower, demonstrating an 13% memory cost reduction when the particle count reaches 8 million. This suggests that, in scenarios involving extremely large scales, this approach provides superior memory compatibility. The differences in memory efficiency and cost may be attributed to the additional memory abstraction layer connecting C++ and Python. For example, advanced features in GeoTaichi might necessitate greater Python functionality, which may not be fully optimized at the intermediate program level, particularly evident during GPU kernel execution. In contrast, this framework's direct memory manipulation allows for faster large-memory operations, such as copying, writing, and resetting, thereby enhancing performance more effectively than higher-level, abstracted Python calls.

4.5.2. Snow packing

The original simulation prototype described in Stomakhin et al. (2013) is revisited in this study to test the scalability of the sparsity mode (Fig. 16). In the simulation, a hanging cubic snow block falls freely under gravity, impacts a rigid and sticky roof, and finally settles onto a sticky base. *Sticky* indicates that both the roof and base impose displacement constraints in the normal and tangential directions simultaneously. The cell discretization is set at 0.0125 m, with 8 material points per cell, resulting in a total of 1.7 million material points. The snow material is modeled to exhibit hyper-plastic behavior. The elasto-plastic energy density function (Φ), as a function of the deformation matrix ($F = F_E F_P$, where F_E represents elastic deformation and F_P denotes plastic deformation), is expressed as follows:

$$\Phi = A\|F_E - R_E\|^2 + \frac{B}{2}(J_E - 1)^2 \tag{17a}$$

$$A = \frac{E}{2(1+\nu)} e^{\lambda(1-J_P)} \tag{17b}$$

$$B = \frac{E\nu}{(1+\nu)(1-2\nu)} e^{\lambda(1-J_P)} \tag{17c}$$

$$J_E = det(F_E) \tag{17d}$$

$$J_P = det(F_P) \tag{17e}$$

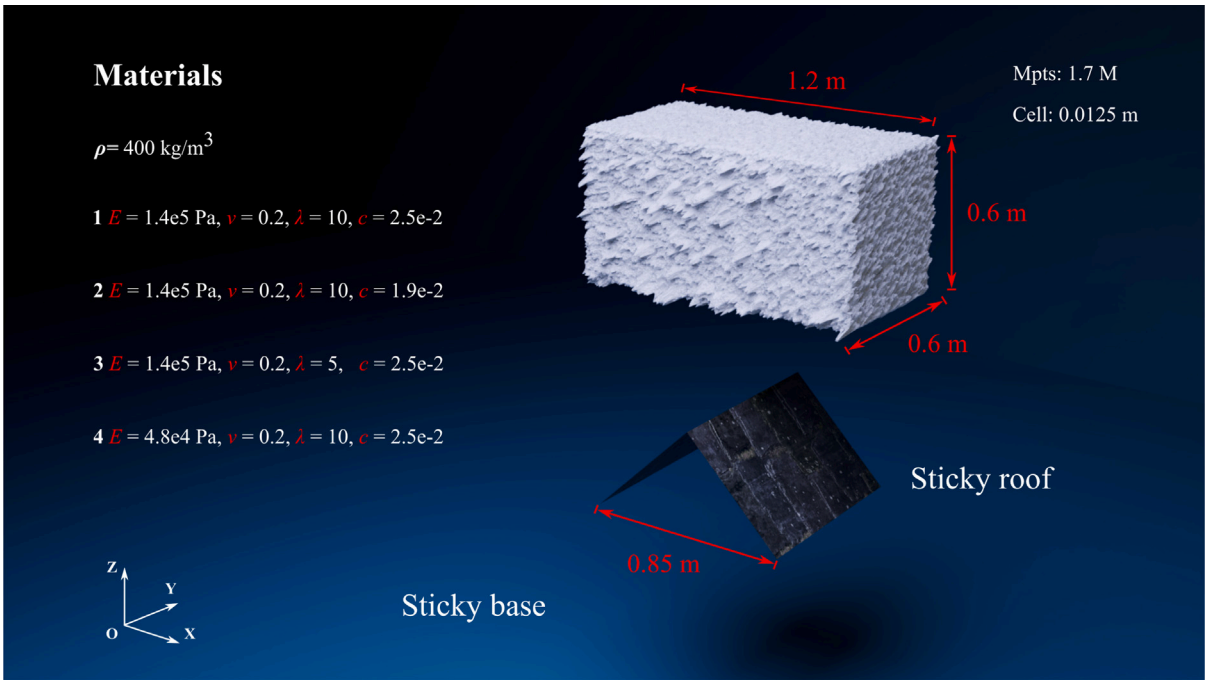


Fig. 16. Schematic representation of the snow packing configuration.

$$F_E = R_E U_E \tag{17f}$$

$$U_{E,i} \in (1-c, 1+c) \tag{17g}$$

where E and ν represent Young's modulus and Poisson's ratio, respectively; λ denotes a plastic hardening multiplier; and R_E and U_E represent the left polar decomposition of the elastic deformation matrix. The parameter c constrains the minimum and maximum values of the diagonal stretch components of U_E.

Given that the three principal material properties, E, λ and c, significantly influence snow behavior as noted by Stomakhin et al. (2013), the sparsity performance is averaged across four cases. Among them, **Material 1** serves as the reference baseline, while **Materials 2-4** provide variations in E, λ and c, respectively (see the Materials section in Fig. 16). The timestep is set at 5×10^{-4} s, and the simulation with damping of 0.7 concludes after 60,000 steps, equivalent to 30 s. The final time states for the four materials are displayed in Fig. 17. Lowering the parameter c results in slimmer falling curtains, reducing λ sharpens the remaining snow caps on the roof, and a decrease in E leads to degradation in snow accumulation both on the roof and the ground.

Since all four features are integrated into the sparsity of the MP distribution, the profiling time is averaged based on MPM iteration stages: Reorder, GIMP, Stress, P2G, G2P, and auxiliary processes such as Memset, Memcpy, and Contact with the digital terrain. Note that the legend for Reorders includes values of 1, 10, and 20, as described in Section 3.6, where periodic reordering aims to optimize the program. These values indicate the iteration step intervals for reordering. For a sparse distribution, a very high interval, as seen in DEM reorder (100 in Zhao et al. (2023)), is not recommended. Due to the significant dynamic regime, an excessively high interval can cause particles to jump outside the GIMP interpolation domain, disconnecting material points from the anchoring nodes within a reorder period. The limitation for reordering in this snow packing case is approximately 24, with 20 chosen as the maximum reorder interval for comparison.

The performance comparison of the snow packing simulation is shown in Fig. 18. If reordering is performed at every step, the total time cost would be 731 s, resulting in a speed of 82 steps/s. Referring to the linear scalability of the dense mode discussed in Sections 4.4 and 4.5.1, the expected speed for 1.7 million particles would be 112 steps/s and 115 steps/s, which is significantly faster than the sparse configuration. By increasing the reorder interval from 1 to 10 and then to 20, efficiency gains of 108% and 121% are achieved, indicating the limitations of the reordering technique in sparse memory mode.

When breaking down the simulation into principal components such as reorder, GIMP shape function calculation, material constitutive modeling, P2G, and G2P, it is evident that P2G dominates the other processes. This dominance is primarily due to the unavoidable many-to-one memory write operations required when multiple particles write to the same nodes in the scatter-dominated P2G stage. Although warp intrinsics help alleviate the serialization burden to some extent, P2G remains the primary bottleneck. A potential solution for the sparse mode could be implementing a pure gather P2G method from the node side, as the mismatch in the degrees of freedom between nodes and material points is within acceptable limits.

The rose polar plot shows that Memset, Memcpy, and Contact processes occupy a minimal fraction across all three reorder intervals, demonstrating that modern GPUs (such as the NVIDIA RTX 4070 Ti used in this study) efficiently manage memory operations. This also indicates that the digital terrain contact handling method employed here does not significantly increase computational cost and allows for greater flexibility in mimicking complex boundary conditions.

5. Engineering-scale case: Baige landslide

5.1. Background

This analysis focuses on the Baige landslide, which occurred on the right bank of the Jinsha River at coordinates (31°4′56.41″ N, 98°42′17.98″ S), near Baige village along the border of Sichuan and Tibet in southwest China. The area is characterized by developed faults and folds within the strata, a consequence of multi-phase tectonic movements associated with the freeze-thaw plateau. The average and maximum annual precipitation recorded in the landslide vicinity are 650 mm and 1067.7 mm, respectively (Wang et al., 2020a). Notably,

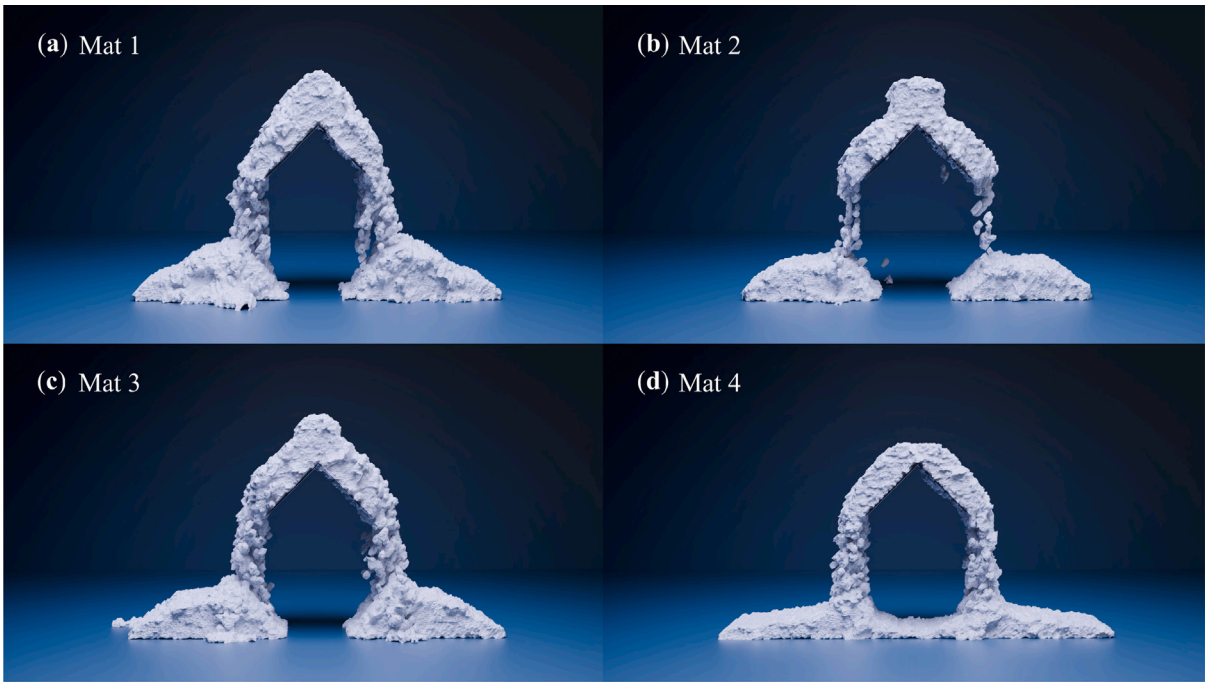


Fig. 17. Final packing profiles of snow for different sets of material properties.

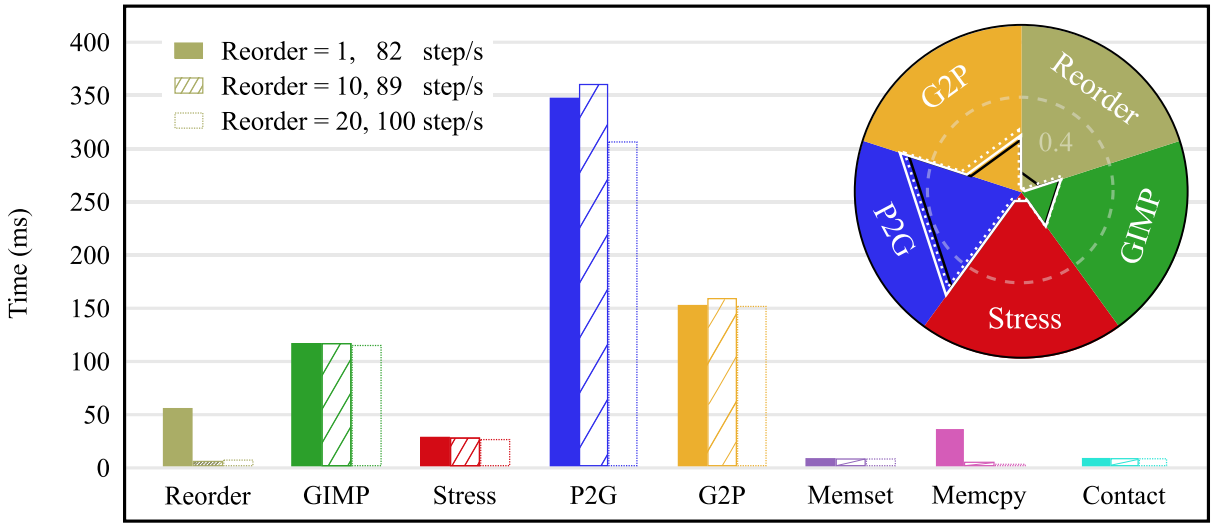


Fig. 18. Computational profiling of the GPU-based MPM for snow packing.

cumulative rainfall at the nearest monitoring station surpassed 590 mm between mid-June and early October 2018. This persistent heavy rainfall led to elevated water content and the saturation of bed deposits, which are identified as the primary triggers of the catastrophic landslide (Zhang et al., 2019). During the event, approximately 27.6 million m^3 of large rock masses collapsed into the Jinsha River, resulting in a blockage of the river channel (Fig. 19(a)).

The Baige landslide case has been widely utilized to validate numerical simulations employing various methods, including MPM (Zhang et al., 2023) and Smoothed Particle Hydrodynamics (SPH) (Peng et al., 2022). For instance, Zhang et al. (2023) performed a GPU-based MPM simulation of the Baige landslide, utilizing a digital elevation model and material point sampling data to showcase the capabilities of GPU MPM in addressing large-scale problems. This study aims to revisit the Baige landslide simulation using our proposed optimized GPU MPM framework, incorporating a BC handler for improved accuracy and performance.

5.2. Model setup

The raw data for the Baige landslide solution domain is sourced from Zhang et al. (2023), tailored, and subsequently formatted for input into the MPM solver in STL format. As illustrated in Fig. 19(b), the solution domain measures 3380 m × 3170 m × 790 m, structured using a uniform orthogonal cell grid with a resolution of 10 m. The black outline indicates the area of the sliding mass, as identified through field investigations. Prior to initiating the simulation, the source mass is initialized with the refined red MPs positioned at their original locations, encompassing a total of 4,329,288 MPs (Fig. 19(c)), with each cell containing $6 \times 6 \times 6 = 216$ MPs. Material types and parameters are consistent with those reported by Zhang et al. (2023), utilizing the Drucker–Prager model (Eq. (14)). The specified material properties are

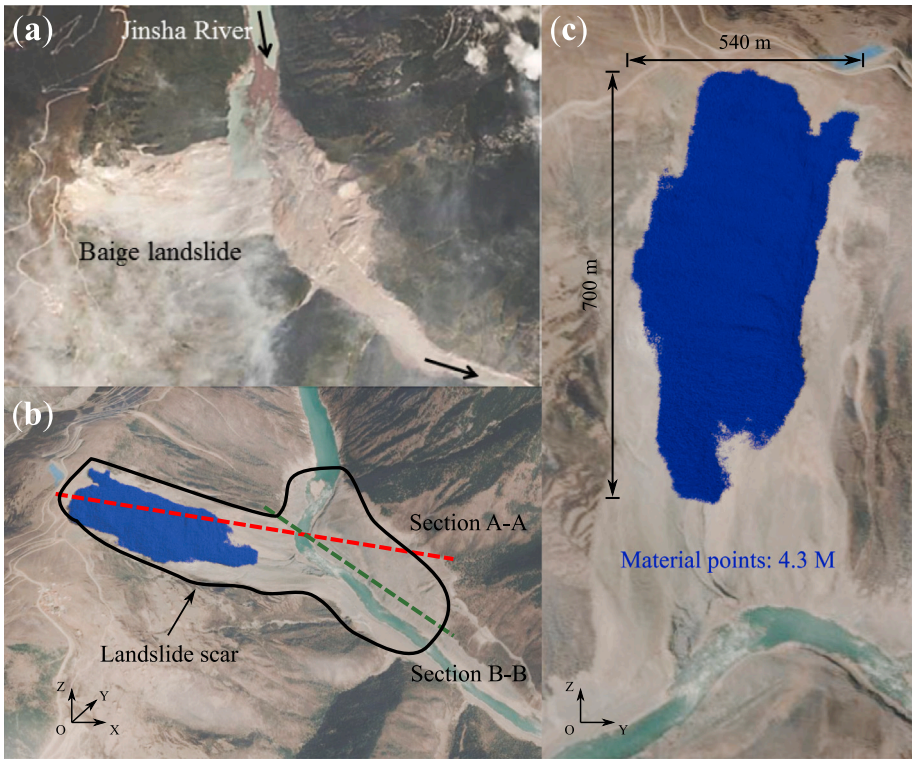


Fig. 19. Baige landslide: (**a**) oblique photograph obtained from Zhong et al. (2020), (**b**) three-dimensional representation of the numerical simulation settings; and (**c**) inset demonstration of the initialization of material points in the front view (Map data provided by OpenTopography).

as follows: density $\rho = 1850$ kg/m^3, Young's modulus $E = 2 \times 10^8$ Pa, Poisson's ratio $\nu = 0.33$, cohesion $c = 10$ kPa, friction angle $\phi = 28°$, dilation angle $\psi = 5°$, and a modified friction coefficient of $\mu = 0.25$ between MPs and the base. The total duration of the simulation is set at 100 s, with a timestep of 0.001 s. A minor damping factor of 0.01 is applied to ensure numerical stability throughout the simulation.

5.3. Simulation results

The landslide process is comprehensively depicted in Fig. 20. The left column presents a top view of the results, featuring a contour of deposition that quantifies the characteristics of the sliding mass. In contrast, the right column provides an inset from a front view. The results reveal a rapid downward acceleration of the sliding mass during the initial 40 s, which subsequently transitions into a quasi-static packing state that continues until the simulation concludes at 100 s. Notably, Fig. 20(d) demonstrates that the accumulation area identified through the MPM simulation closely aligns with the light-gray region observed in field investigations. However, a discrepancy arises: the MPM model does not account for the ascent of the debris flow onto the riverbed, likely due to the single-phase nature of the dry debris flow, which limits its rheological response.

In the packing region of the MPM results, a deposition profile is illustrated in Fig. 21 for quantitative comparison with field data. The red and green lines represent the test data from Sections A-A and B-B, respectively, while circular and square markers indicate the numerical results. Notably, Section B-B shows a strong agreement in trends, whereas Section A-A reveals a leftward shift in the peak of the numerical data, suggesting an earlier cessation of the dry flow. The simulation was completed in approximately 2777 s, executing 100,000 steps on a workstation equipped with an NVIDIA RTX 4070 Ti, achieving an effective performance rate of 36 steps/s. In contrast, Zhang et al. (2023) reported a runtime of 58.63 h for 15,000 steps on an NVIDIA RTX 3090, which has a similar overall performance to the NVIDIA RTX 4070 Ti, employing nearly the same number of material points, 4,135,539. This comparison highlights a significant improvement in efficiency provided by the current GPU MPM framework.

6. Concluding remarks

We present a novel sparse memory-driven GPU-based MPM framework designed to optimize the balance between memory efficiency and computational performance. This framework effectively addresses the significant memory demands associated with large-scale simulations, particularly for materials with sparse spatial distributions. To achieve this, we introduce a hierarchical block-cell-material point structure that enables an atomic-free dual mapping between material points and nodes at each MPM iteration. This structure supports warp-level intrinsic acceleration during the particle-to-grid and grid-to-particle processes, facilitating efficient coalesced memory access and minimizing latency through staggered execution. Furthermore, our framework is adaptable, accommodating for a variety of iteration schemes and boundary condition types, making it suitable for engineering-scale challenges involving complex materials and boundary conditions. The two key features of the proposed framework are summarized as follows: (1) Block-cell-MP hierarchy: this hierarchy is pivotal to the overall execution framework. At the cell-material point level, it serves as the foundation for the atomic-free dual mapping between material points and nodes. The block-cell level ensures excellent compatibility with low-level programming intrinsics, thereby enhancing the efficiency of attribute communication, which is crucial for both particle-to-grid and grid-to-particle processes. (2) Memory shift for material properties: the memory shift in key components of material properties enhances the capacity of this framework to accommodate a broader range of material types. Moreover, the incorporation of arbitrarily shaped boundary

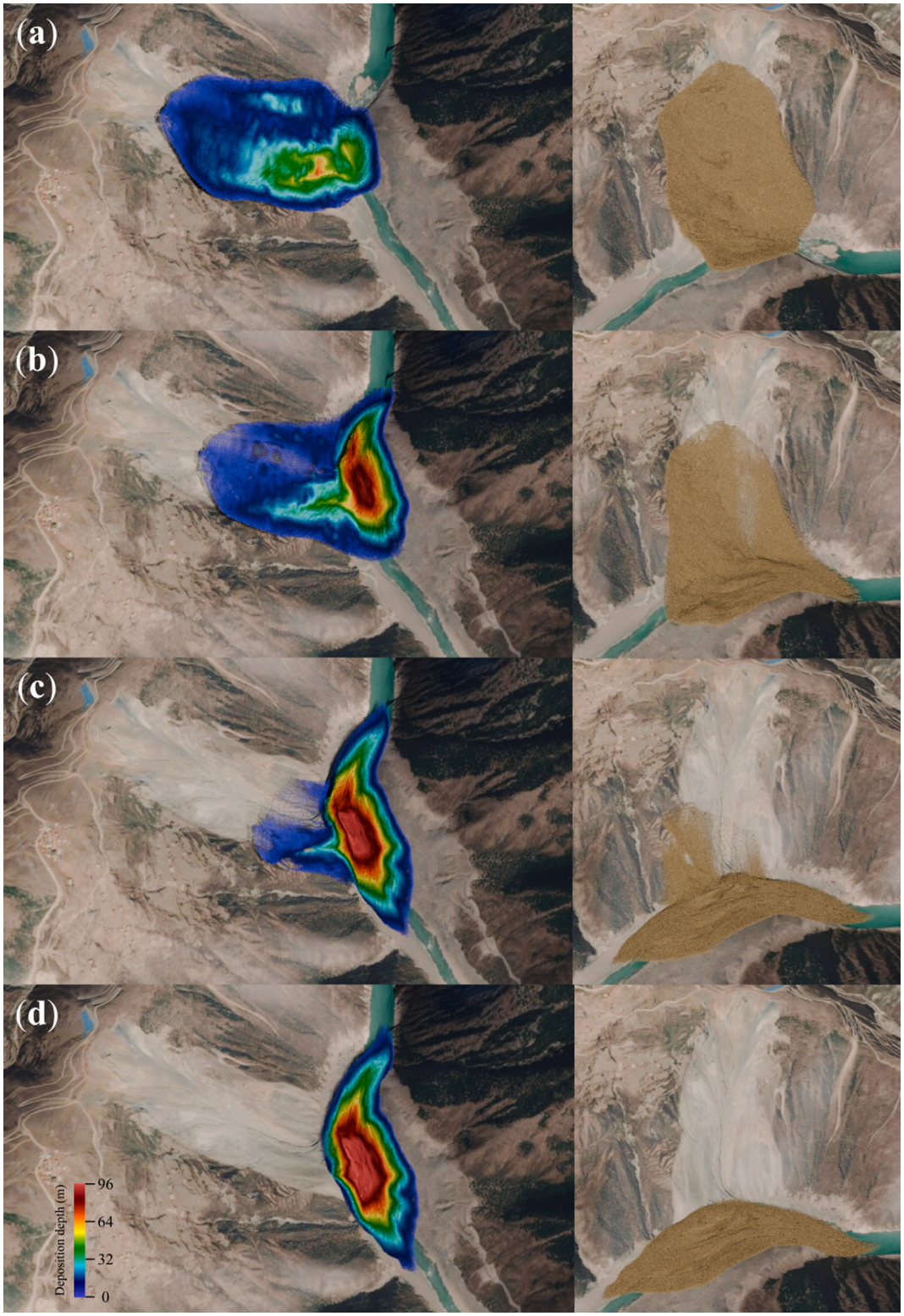


Fig. 20. Simulation results (top and front view) of the dry debris flow on the Baige digital terrain: (**a**) t = 20 s, (**b**) t = 30 s, (**c**) t = 40 s, and (**d**) t = 100 s.

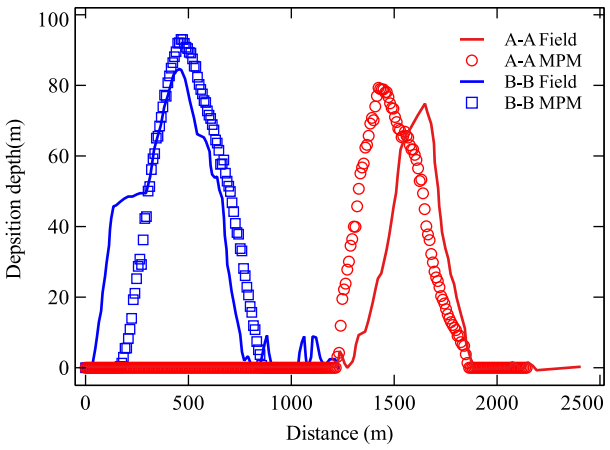


Fig. 21. Deposition profile in Sections A-A and B-B.

condition engines further extends the applicability of this framework.

While the proposed framework demonstrates significant efficiency in simulating engineering-scale problems; however, it also has several limitations that warrant further investigation. Firstly, the current implementation is restricted to single-phase problems that focus exclusively on the mechanical responses of dry granular materials. Future research will aim to broaden this framework to encompass multiphysics interactions, including thermo-hydro-mechanical coupling (Yu et al., 2024). Secondly, due to the complex iteration schemes and boundary conditions inherent in realistic scenarios, the existing serialization order of the GPU kernel may not be optimal. To address this challenge, a graph-based approach could be utilized to maintain a repository of GPU kernels, facilitating asynchronous scheduling of computational and memory tasks. This strategy has the potential to enhance the balance between simulation efficiency and critical I/O operations, such as the frequent archiving of results that is essential for near real-time simulation applications.

CRediT authorship contribution statement

Hao Chen: Writing – original draft, Software, Investigation. **Shiwei Zhao:** Writing – review & editing, Supervision, Funding acquisition. **Jidong Zhao:** Writing – review & editing, Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was financially supported by National Natural Science Foundation of China (by Project No. 52439001) and Research Grants Council of Hong Kong (by GRF Projects No. 16206322 and No. 16211221, by CRF Project No. C7082-22G, and by TRS Projects No. T22-606/23-R and No. T22-607/24-N). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the financial bodies.

Data availability

No data was used for the research described in the article.

References

Anura3D MPM Research Community, 2024. Anura3D source code. https://www.anura3d.com. (Accessed 07 October 2024).
Bardenhagen, S.G., Kober, E.M., et al., 2004. The generalized interpolation material point method. Comput. Model. Eng. Sci. 5, 477–496.
Bing, Y., Cortis, M., Charlton, T., Coombs, W., Augarde, C., 2019. B-spline based boundary conditions in the material point method. Comput. Struct. 212, 257–274.
Bird, R.E., Pretti, G., Coombs, W.M., Augarde, C.E., Sharif, Y.U., Brown, M.J., Carter, G., Macdonald, C., Johnson, K., 2024. An implicit material point-to-rigid body contact approach for large deformation soil–structure interaction. Comput. Geotech. 174, 106646.
Brackbill, J.U., Ruppel, H.M., 1986. FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. J. Comput. Phys. 65, 314–343.
Buckland, E., Nguyen, V.P., de Vaucorbeil, A., 2024. Easily porting material point methods codes to GPU. Comput. Part. Mech. 1–16.
Bui, H.H., Fukagawa, R., Sako, K., Ohno, S., 2008. Lagrangian meshfree particles method (SPH) for large deformation and failure flows of geomaterial using elastic–plastic soil constitutive model. Int. J. Numer. Anal. Methods Geomech. 32, 1537–1570.
Cao, Y., Zhao, Y., Li, M., Yang, Y., Choo, J., Terzopoulos, D., Jiang, C., 2024. Unstructured moving least squares material point methods: a stable kernel approach with continuous gradient reconstruction on general unstructured tessellations. Comput. Mech. 1–24.
Chen, W.F., Baladi, G.Y., 1985. Soil Plasticity: Theory and Implementation. Elsevier.
Chen, H., Zhao, S., Zhao, J., Zhou, X., 2023. DEM-enriched contact approach for material point method. Comput. Methods Appl. Mech. Engrg. 404, 115814.
Clarke, L., Glendinning, I., Hempel, R., 1994. The MPI message passing interface standard. In: Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994. Springer, pp. 213–218.
Coombs, W.M., Charlton, T.J., Cortis, M., Augarde, C.E., 2018. Overcoming volumetric locking in material point methods. Comput. Methods Appl. Mech. Engrg. 333, 1–21.
Cundall, P.A., Strack, O.D., 1979. A discrete numerical model for granular assemblies. Geotechnique 29, 47–65.
Dagum, L., Menon, R., 1998. OpenMP: an industry standard API for shared-memory programming. IEEE Comput. Sci. Eng. 5, 46–55.
De Vaucorbeil, A., Nguyen, V.P., Sinaie, S., Wu, J.Y., 2020. Material point method after 25 years: theory, implementation, and applications. Adv. Appl. Mech. 53, 185–398.
Dong, Y., Cui, L., Zhang, X., 2022. Multiple-GPU parallelization of three-dimensional material point method based on single-root complex. Internat. J. Numer. Methods Engrg. 123, 1481–1504.
Dong, Y., Wang, D., Randolph, M.F., 2015. A GPU parallel computing strategy for the material point method. Comput. Geotech. 66, 31–38.
Drucker, S., Prager, W., 1952. Mechanics and plastic analysis or limit design. Quart. Appl. Math. 2, 157–165.
Feng, Z.K., Xu, W.J., 2021. GPU material point method (MPM) and its application on slope stability analysis. Bull. Eng. Geol. Environ. 80, 5437–5449.
Gao, M., Wang, X., Wu, K., Pradhana, A., Sifakis, E., Yuksel, C., Jiang, C., 2018. GPU optimization of material point methods. ACM Trans. Graph. 37, 1–12.
Gaume, J., Gast, T., Teran, J., van Herwijnen, A., Jiang, C., 2018. Dynamic anticrack propagation in snow. Nat. Commun. 9, 3047.
Guilkey, J., Lander, R., Bonnell, L., 2021. A hybrid penalty and grid based contact method for the material point method. Comput. Methods Appl. Mech. Engrg. 379, 113739.
Hammerquist, C.C., Nairn, J.A., 2017. A new method for material point method particle updates that reduces noise and enhances stability. Comput. Methods Appl. Mech. Engrg. 318, 724–738.
Harlow, F.H., 1964. The particle-in-cell computing method for fluid dynamics. Methods Comput. Phys. 3, 319–343.
Hrennikoff, A., 1941. Solution of problems of elasticity by the framework method. J. Appl. Mech..
Hu, Y., Fang, Y., Ge, Z., Qu, Z., Zhu, Y., Pradhana, A., Jiang, C., 2018. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. ACM Trans. Graph. 37, 1–14.
Hu, Y., Li, T.M., Anderson, L., Ragan-Kelley, J., Durand, F., 2019. Taichi: a language for high-performance computation on spatially sparse data structures. ACM Trans. Graph. 38, 201.
Iverson, R.M., 2012. Elementary theory of bed-sediment entrainment by debris flows and avalanches. J. Geophys. Res.: Earth Surf. 117.
Jiang, Y., Zhao, Y., Choi, C.E., Choo, J., 2022. Hybrid continuum–discrete simulation of granular impact dynamics. Acta Geotech. 17, 5597–5612.
Jin, Y.F., Yin, Z.Y., Zhou, X.W., Liu, F.T., 2021. A stable node-based smoothed pfem for solving geotechnical large deformation 2d problems. Comput. Methods Appl. Mech. Engrg. 387, 114179.
Johnson, K.L., 1987. Contact Mechanics. Cambridge University Press.
Kamrin, K., Koval, G., 2012. Nonlocal constitutive relation for steady granular flow. Phys. Rev. Lett. 108, 178301.

Kumar, K., Salmond, J., Kularathna, S., Wilkes, C., Tjung, E., Biscontin, G., Soga, K., 2019. Scalable and modular material point method for large-scale simulations. arXiv preprint arXiv:1909.13380.

Li, J., Wang, B., Wang, D., Zhang, P., Vardon, P.J., 2023. A coupled MPM-DEM method for modelling soil-rock mixtures. Comput. Geotech. 160, 105508.

Liang, W., Fang, H., Yin, Z.Y., Zhao, J., 2024. A mortar segment-to-segment frictional contact approach in material point method. Comput. Methods Appl. Mech. Engrg. 431, 117294.

Liang, Y., Given, J., Soga, K., 2023. The imposition of nonconforming Neumann boundary condition in the material point method without boundary representation. Comput. Methods Appl. Mech. Engrg. 404, 115785.

Museth, K., 2013. VDB: high-resolution sparse volumes with dynamic topology. ACM Trans. Graph. 32, 1–22.

Nairn, J.A., Hammerquist, C.C., 2021. Material point method simulations using an approximate full mass matrix inverse. Comput. Methods Appl. Mech. Engrg. 377, 113667.

NVIDIA, Vingelmann, P., Fitzek, F.H., 2020. CUDA, release: 10.2.89. URL: https://developer.nvidia.com/cuda-toolkit.

Peng, C., Li, S., Wu, W., An, H., Chen, X., Ouyang, C., Tang, H., 2022. On three-dimensional SPH modelling of large-scale landslides. Can. Geotech. J. 59, 24–39.

Pudasaini, S.P., Krautblatter, M., 2021. The mechanics of landslide mobility with erosion. Nat. Commun. 12, 6793.

Remmerswaal, G., 2023. The random material point method for assessment of residual dyke resistance: investigating the influence of soil heterogeneity on slope failure processes.

Setaluri, R., Aanjaneya, M., Bauer, S., Sifakis, E., 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. ACM Trans. Graph. 33, 1–12.

Shi, Y., Guo, N., Yang, Z., 2024. GeoTaichi: A Taichi-powered high-performance numerical simulator for multiscale geophysical problems. Comput. Phys. Comm. 301, 109219.

Steffen, M., Kirby, R.M., Berzins, M., 2008. Analysis and reduction of quadrature errors in the material point method (MPM). Internat. J. Numer. Methods Engrg. 76, 922–948.

Stomakhin, A., Schroeder, C., Chai, L., Teran, J., Selle, A., 2013. A material point method for snow simulation. ACM Trans. Graph. 32, 1–10.

Sulsky, D., Chen, Z., Schreyer, H.L., 1994. A particle method for history-dependent materials. Comput. Methods Appl. Mech. Engrg. 118, 179–196.

de Vaucorbeil, A., Nguyen, V.P., Nguyen-Thanh, C., 2021. Karamelo: an open source parallel C++ package for the material point method. Comput. Part. Mech. 8, 767–789.

Wang, B., Chen, P., Wang, D., Liu, L.L., Zhang, W., 2024. Development of a GPU-accelerated implicit material point method for geotechnical engineering. Acta Geotech. 19, 3729–3749.

Wang, L., Coombs, W.M., Augarde, C.E., Cortis, M., Brown, M.J., Brennan, A.J., Knappett, J.A., Davidson, C., Richards, D., White, D.J., et al., 2021. An efficient and locking-free material point method for three-dimensional analysis with simplex elements. Internat. J. Numer. Methods Engrg. 122, 3876–3899.

Wang, X., Qiu, Y., Slattery, S.R., Fang, Y., Li, M., Zhu, S.C., Zhu, Y., Tang, M., Manocha, D., Jiang, C., 2020b. A massively parallel and scalable multi-GPU material point method. ACM Trans. Graph. 39, 30–31.

Wang, W., Yin, Y., Zhu, S., Wang, L., Zhang, N., Zhao, R., 2020a. Investigation and numerical modeling of the overloading-induced catastrophic rockslide avalanche in Baige, Tibet, China. Bull. Eng. Geol. Environ. 79, 1765–1779.

Wyser, E., Alkhimenkov, Y., Jaboyedoff, M., Podladchikov, Y.Y., 2021. An explicit GPU-based material point method solver for elastoplastic problems (ep2-3de v1. 0). Geosci. Model Dev. 14, 7749–7774.

Yamaguchi, Y., Moriguchi, S., Terada, K., 2021. Extended B-spline-based implicit material point method. Internat. J. Numer. Methods Engrg. 122, 1746–1769.

Yu, J., Zhao, J., Liang, W., Zhao, S., 2024. A semi-implicit material point method for coupled thermo-hydro-mechanical simulation of saturated porous media in large deformation. Comput. Methods Appl. Mech. Engrg. 418, 116462.

Zhang, D.Z., Perez, K.A., Barclay, P.L., Waters, J., 2024. Rapid particle generation from an STL file and related issues in the application of material point methods to complex objects. Comput. Part. Mech. 1–15.

Zhang, W., Wu, Z., Peng, C., Li, S., Dong, Y., Yuan, W., 2023. Modelling large-scale landslide using a GPU-accelerated 3D mpm with an efficient terrain contact algorithm. Comput. Geotech. 158, 105411.

Zhang, L., Xiao, T., He, J., Chen, C., 2019. Erosion-based analysis of breaching of Baige landslide dams on the Jinsha River, China, in 2018. Landslides 16, 1965–1979.

Zhao, S., Lai, Z., Zhao, J., 2023. Leveraging ray tracing cores for particle-based simulations on GPUs. Internat. J. Numer. Methods Engrg. 124, 696–713.

Zhao, S., Zhao, J., Liang, W., 2021. A thread-block-wise computational framework for large-scale hierarchical continuum-discrete modeling of granular media. Internat. J. Numer. Methods Engrg. 122, 579–608.

Zheng, X., Seaid, M., Pisanò, F., Hicks, M.A., Vardon, P.J., Huvaj, N., Osman, A.S., 2023. A material point/finite volume method for coupled shallow water flows and large dynamic deformations in seabeds. Comput. Geotech. 162, 105673.

Zhong, Q., Chen, S., Wang, L., Shan, Y., 2020. Back analysis of breaching process of Baige landslide dam. Landslides 17, 1681–1692.

Zhou, X.W., Jin, Y.F., He, K.Y., Yin, Z.Y., Liu, F.T., 2024. A novel implicit fem-mpm coupling framework using convex cone programming for elastoplastic problems. Comput. Methods Appl. Mech. Engrg. 429, 117153.