Computer Physics Communications 319 (2026) 109939

Contents lists available at ScienceDirect

Computer Physics Communications

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

Computational Physics

GPU-optimized adaptive mesh refinement for scalable two-phase resolved CFD-DEM simulations on unstructured hexahedral grids

Tao Yu *, Jidong Zhao *

Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong

ARTICLE INFO

The review of this paper was arranged by Prof. Andrew Hazel

Keywords:
Adaptive mesh refinement
GPU parallel algorithm
Resolved coupled CFD-DEM
Two-phase CFD

ABSTRACT

Adaptive mesh refinement (AMR) is essential for accurately resolving interfacial dynamics in resolved coupled computational fluid dynamics-discrete element method (CFD-DEM) and two-phase CFD simulations. However, traditional methods struggle with logical complexity and memory inefficiency when applied to unstructured grids on GPU architectures. This paper presents a novel GPU-accelerated AMR algorithm that eliminates CPU-GPU data transfers and minimizes grid manipulation overhead through a compressed data format and topology-aware reuse strategies. By reconstructing mesh topology entirely on the GPU and retaining parent-mesh indexing, our method reduces AMR-related computational overhead to less than 25% of the total simulation time while ensuring full compatibility with unstructured granular domains. A CUDA-centric implementation, validated across five benchmarks and two powder-based additive manufacturing applications, demonstrates that our framework achieves accuracy comparable to uniformly refined grids with 50% lower computational effort. Furthermore, it exhibits near-linear throughput performance with increasing problem size and achieves over 20 × speedup in large-scale laser powder bed fusion simulations when integrated with GPU-accelerated CFD-DEM solvers. The scalability of the algorithm is further highlighted through hexahedral mesh case studies, with extensibility to general unstructured grids via sub-mesh templating. These advancements enable high-fidelity, GPU-native simulations of complex fluid-particle systems, effectively bridging the gap between adaptive resolution and large-scale parallelism in complex two-phase resolved CFD-DEM simulations.

1. Introduction

The growing complexity of fluid-particle interaction problems in engineering applications, such as laser additive manufacturing [1], sediment transport [2], pharmaceutical processing [3], and multiphase reactor design [4], has intensified the demand for high-fidelity simulations of coupled computational fluid dynamics and discrete element method (CFD-DEM). While such simulations offer unparalleled insights into multiphysics phenomena, their computational cost remains prohibitive, particularly when resolving dynamically evolving interfaces or granular assemblies. Graphics processing units (GPUs), with their massively parallel architecture and high memory bandwidth, have emerged as a transformative tool for accelerating scientific computing, enabling order-of-magnitude speedups over traditional CPU-based approaches [5]. However, harnessing GPU potential for mesh-based methods like the finite volume method (FVM), the backbone of industrial CFD software such as OpenFOAM and Ansys Fluent, remains challenging, especially when adaptive mesh refinement (AMR) is required for efficiency [6].

Particle-based techniques, such as Smoothed Particle Hydrodynamics (SPH) [7–9] and the Lattice Boltzmann Method (LBM) [10,11], have readily capitalized on GPU parallelism due to their naturally decoupled computations, where particles or lattice nodes operate independently [10–14]. Similarly, Discrete Element Method (DEM) [15–17] and Material Point Method (MPM) [18,19] have achieved significant GPU-driven performance gains by exploiting localized particle interactions. In contrast, the reliance of FVM on unstructured mesh connectivity introduces inherent dependencies between cells and faces, complicating parallelization. Specifically, there are three major challenges hindering GPU-accelerated FVM framework: (1) Architectural misalignment: Traditional CPU-based domain decomposition strategies fail to exploit modern GPUs' thousands of CUDA cores [20,21], demanding ground-up algorithmic redesign for fine-grained parallelism. (2) Unstructured grid complexity: Widely used in industrial CFD, unstructured grids introduce intricate cell-face-node dependencies that degrade thread independence during equation discretization [22], in

* Corresponding authors.
E-mail addresses: tyuak@connect.ust.hk (T. Yu), jzhao@ust.hk (J. Zhao).

https://doi.org/10.1016/j.cpc.2025.109939
Received 18 May 2025; Received in revised form 25 October 2025; Accepted 2 November 2025
Available online 3 November 2025

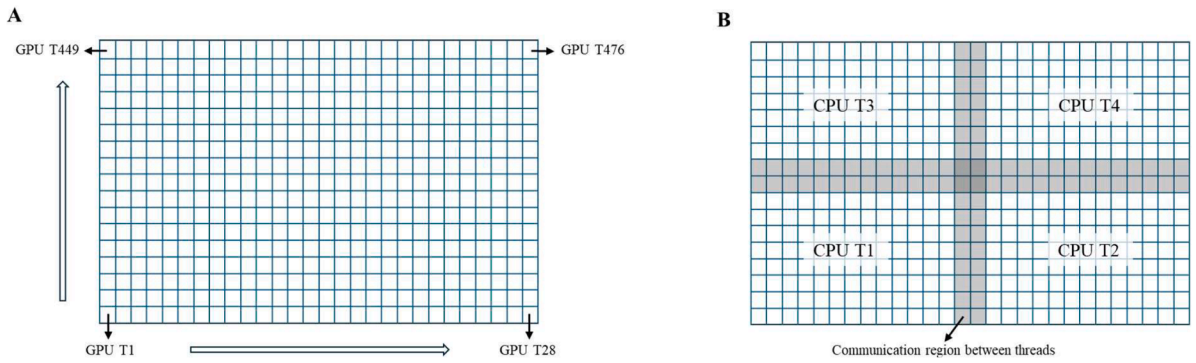


Fig. 1. Schematic illustration of the cell-based GPU parallel algorithm (A) and the CPU parallel algorithm (B). (A) Each GPU thread is responsible for the computational tasks of one mesh cell for the cell-based GPU parallel algorithm. (B) The domain is divided into multiple parts according to the utilized number of CPU threads. The highlighted elements are utilized for communications between threads. Note that, T*i* means the thread #*i* of the CPU or GPU.

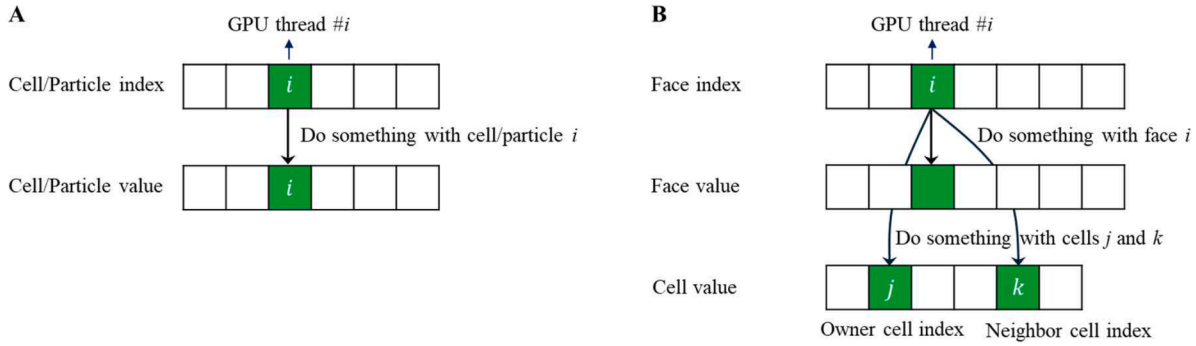


Fig. 2. Schematic illustration of the cell/particle-based GPU parallel algorithm (A) and the face-based GPU parallel algorithm (B). Note that the boxes in the figure represent data lists rather than mesh elements.

sharp contrast to the highly decoupled computations typically achieved by particle-based methods on structured grids [23,24]. (3) Lack of Auxiliary GPU algorithms: Critical tools like AMR, dynamic meshes, and overlapping grids, which form the cornerstone of efficient CFD [22,25], remain underdeveloped for GPUs, forcing trade-offs between adaptability and performance.

Recent advances partially address these issues. FVM solvers with structured grids [23,26] may leverage GPU parallelism but encounter difficulties in geometrically complex domains. For unstructured grids, modular GPU acceleration of linear solvers (e.g., via NVIDIA's AMGx) [20] and vertex-centered Navier-Stokes solver [27–29] show great promise, yet these efforts either assume static grids [30] or offload adaption to CPUs [22,25], incurring limitations on scalability and adaptability and data-transfer overhead. Meanwhile, AMR implementations for GPUs remain confined to structured grids or 2D domains [31–33], relying on octree-based hierarchy algorithms [34–36] that introduce excessive logical processing and branching inefficient for GPU warps.

This paper proposes a GPU-native AMR framework for unstructured hexahedral grids that eliminates CPU intervention and octree-derived bottlenecks, featured by key innovations including: (1) A compressed format that significantly reduces memory footprint compared to conventional unstructured grid storage to enable efficient topology updates without global reindexing, (2) parent-mesh anchoring that localizes refinement operations to minimize thread divergence and remarkably cut logical overhead versus octree-based methods, and (3) dynamic variable remapping that retains simulation state continuity entirely on the GPU to avoid CPU-GPU transfers responsible for expensive runtime in hybrid approaches. The method is rigorously validated against five benchmark cases, including 3D dam breaks and particle sedimentation, and further demonstrated in two powder-based additive manufacturing processes: binder jetting and laser powder bed fusion. In addition, we extend the algorithm to unstructured tetrahedral grids and provide one illustrative example to demonstrate its generalizability beyond hexahedral grids. These examples confirm the accuracy, efficiency, and robustness of the proposed method in complex engineering applications. By extending hexahedral sub-mesh templates to general unstructured topologies, we further demonstrate a path toward GPU-accelerated AMR for industrial-grade CFD-DEM.

The paper is structured as follows: Section 2 details the GPU-optimized AMR methodology, contrasting CPU/GPU parallelism and presenting the CFD-DEM-AMR integration. Section 3 validates the framework across multiphase and granular flow benchmarks, quantifying the accuracy and speedups of the framework. Section 4 discusses broader implications for high-performance CFD and outlines future extensions.

2. Methodology

2.1. Parallel algorithms for GPU-accelerated CFD

CPUs and GPUs have distinct architectures [37] suited for different tasks: CPUs usually have a few powerful cores for sequential processing and complex control, while GPUs feature thousands of simpler cores optimized for parallel tasks like matrix computations or rendering. This makes GPUs ideal for high-throughput workloads, such as large-scale scientific simulations like CFD [37,38], where solving equations over large grids benefits from massive parallelism, enabling faster and more scalable computations. Parallel programming on CPUs and GPUs relies on distinct platforms. For CPUs, OpenMP and MPI enable multi-threading and task scheduling, simplifying parallel application development. For GPUs, CUDA (NVIDIA-specific) and OpenCL

(cross-platform) are the dominant frameworks. CUDA provides optimized libraries like cuBLAS, cuSPARSE, and Thrust for scientific computing, while OpenCL offers broader hardware support but requires more manual tuning.

Compared to CPU-based parallel algorithms, GPU parallel algorithms demonstrate significant advantages in terms of parallel efficiency [27, 39] and mesh postprocessing. Specifically, GPU parallel algorithms excel in handling large-scale CFD problems due to their ability to execute thousands of threads simultaneously without being constrained by an optimal thread count [37], unlike CPU-based algorithms, which face diminishing returns beyond a certain number of threads due to inter-core communication. On mesh postprocessing, GPU parallel algorithms simplify the process by performing computations globally at the level of mesh cells and faces (Fig. 1A), automatically assigning tasks to available threads without requiring subdomain partitioning. This approach avoids the use of ghost cells, which are necessary in CPU-based domain decomposition algorithms (Fig. 1B) to manage data exchange across subdomain boundaries [40]. This automatic task assignment minimizes workload imbalance and eliminates efficiency losses while making processing significantly less complex, even for intricate domains. This work focuses on developing parallel algorithms for GPU-accelerated CFD and CFD-DEM framework, in conjunction with the adaptive mesh refinement algorithm.

We propose two foundational GPU-parallel algorithms for finite volume method (FVM) computations on unstructured grids: a cell-based and a face-based strategy, as illustrated in Fig. 2. In Fig. 1A and Fig. 2A, the cell-based approach assigns each GPU thread to a unique control volume (cell or particle), allowing it to independently access and update local data. This thread-level parallelism is particularly effective for localized FVM operations, such as source term evaluations and explicit time integration of cell-centered quantities. Moreover, this pattern is directly applicable to the discrete element method (DEM), where each thread independently updates the motion and interaction forces of a single particle, facilitating efficient DEM coupling.

Fig. 2B presents the face-based parallel strategy, in which each GPU thread is assigned to a unique mesh face and is responsible for computations such as numerical flux evaluation or gradient reconstruction. Compared to the cell-based approach, the face-based strategy is more complex and involves two distinct scenarios. The first is analogous to the cell-based strategy, where each thread performs operations directly on data associated with the face itself. The second, and more involved case, requires the thread to access and update data associated with the cells adjacent to the face. Owing to the unstructured nature of the mesh, each face stores information about its neighboring control volumes (commonly referred to as the owner and neighbor cells), thereby enabling efficient access to surrounding cell values needed for operations such as interpolation or face-to-cell flux contribution. This added layer of complexity necessitates the use of atomic operations to ensure data consistency, as multiple threads may concurrently access and update the same cell-level quantities through different adjacent faces, leading to potential race conditions in the absence of proper synchronization.

These two parallelization strategies constitute the core computational framework for GPU-accelerated multiphysics simulations on unstructured meshes. By leveraging thread-level locality and efficient memory access, they enable scalable, high-performance computation for coupled CFD-DEM simulations involving complex particle-fluid interactions. Compared to traditional CPU-based parallelization, this GPU-based approach offers a fundamentally different paradigm. Even with high-end GPUs such as the RTX 5090, which features 21,760 CUDA cores, the number of threads remains significantly smaller than the millions of control volumes typically found in large-scale CFD simulations. This imbalance ensures high occupancy and throughput, allowing the GPU to be fully utilized. In contrast, CPU-based methods rely on domain decomposition, where each processor core is assigned a large subdomain (Fig. 1B). This process entails nontrivial overheads, including mesh partitioning, ghost-cell setup, and intensive inter-core communication, all of which can become significant bottlenecks in scaling. By eliminating the need for such decomposition and enabling fine-grained parallelism, the proposed GPU framework offers enhanced efficiency and scalability for multiphysics simulations on unstructured grids.

2.2. A GPU-accelerated two-phase resolved CFD-DEM framework

In this study, a fully resolved coupled CFD-DEM approach based on the Immersed Boundary (IB) method is employed to simulate the interaction between fluid and particles. Within this framework, accurate two-way fluid-particle coupling is achieved using a single background Eulerian mesh for the fluid, in which particles are embedded. To resolve the flow around individual particles and capture detailed interaction forces, the mesh size must be significantly smaller than the particle diameter. Although no universally applicable threshold exists, studies have suggested that a ratio (defined as mesh size divided by particle diameter) on the order of 1/8 [41] or 1/10 [42] is required. This resolution enables accurate estimation of the full spectrum of particle-fluid interaction forces, including drag, lift, and buoyancy, following the method proposed by Lai et al. [43], leading to improved predictions of the overall flow behavior. However, the high grid resolution results in a substantial increase in computational cost, making AMR a critical technique for enhancing computational efficiency. This section presents the numerical algorithm underlying the resolved CFD-DEM approach.

2.2.1. Momentum equation of CFD

Building upon the Navier-Stokes equation, the following momentum equation is formulated to account for various physical phenomena, such as surface tension and fluid- particle interaction forces, as represented on the right-hand side of the equation.

$$\frac{\partial}{\partial t}(\rho\mathbf{u}) + \nabla\cdot(\rho\mathbf{u}\otimes\mathbf{u}) = -\nabla p + \rho\mathbf{g} + \nabla\cdot(\mu\cdot(\nabla\mathbf{u})) + c\sigma|\nabla\alpha_1|\mathbf{n} + \boldsymbol{f}_s + \boldsymbol{f}_T, \tag{1}$$

where ρ, $\mathbf{u}$ and p represent the density, velocity, and pressure of fluid, respectively. p is defined as $p = p_d + \rho g h$, where p_d is the dynamic pressure, g is the magnitude of gravitational acceleration $\mathbf{g}$, and h is the reference height [44]. c represents the curvature of the fluid interface, expressed as $c = -\nabla\cdot\mathbf{n}$. $\mathbf{n}$ is the unit normal vector at the interface, calculated as $\mathbf{n} = \nabla\alpha_1/|\nabla\alpha_1|$. σ denotes the surface tension coefficient and $\boldsymbol{f}_s$ represents the interaction force between the fluid and particles, and $\boldsymbol{f}_T$ denotes the thermal dynamic forces.

The momentum equation (Eq. (1)) can be discretized based on the FVM, as shown in Eq. (2). The Pressure Implicit with Splitting of Operators (PISO) algorithm, a widely used pressure correction method for coupling velocity and pressure in CFD simulations [45], is employed to solve for velocity and pressure. The PISO algorithm involves a predictor step, where the pressure from the latest timestep is utilized to compute an intermediate velocity, followed by corrector steps that iteratively update and refine the velocity and pressure fields to obtain the final solution [46].

$$\frac{\rho V}{\Delta t}\left(\mathbf{u}^{n+1} - \mathbf{u}^{n}\right) + \sum \rho\varphi\mathbf{u}_f^{n+1} = \sum \mu(\nabla\mathbf{u})_f^{n+1}\cdot\mathbf{n}_f A + (c\sigma|\nabla\alpha_1|\mathbf{n} + \boldsymbol{f}_s + \boldsymbol{f}_T - \nabla p)V, \tag{2}$$

where V is the cell volume, Δt is the time step of CFD, $\varphi = \left(\mathbf{u}_f^n\cdot\boldsymbol{n}\right)A$ is the flux across the mesh face and A denotes the face area. $\mathbf{u}_f$ and $\nabla\mathbf{u}_f$ are the velocity and the velocity gradient at the mesh face, and $\mathbf{n}_f$ is the face normal.

For the interaction force in resolved CFD-DEM, two mainstream approaches are commonly adopted. The first method employs an explicit distributed processing strategy, where the interaction force is initially

excluded from the momentum equation [47]. The fluid velocity and pressure are then corrected based on particle velocities by solving an additional approximate Poisson-like correction equation. This approach offers higher numerical stability but requires the solution of an extra correction equation. The second method incorporates the interaction force implicitly within the momentum equation, solving it in a single step [43,48]. Since it rigorously satisfies the velocity conditions at the particle-fluid interface and is simpler to implement in code [43], the second approach is adopted in this work and the interaction force $\boldsymbol{f}_s$ is calculated based on the model proposed by Lai et al. [43].

$$\boldsymbol{f}_s = \chi\left(\left(1-\frac{\rho_s}{\rho}\varepsilon_p\right)\mathbf{u}^n + \frac{\rho_s}{\rho}\varepsilon_p\left(\mathbf{v}_p + \boldsymbol{\omega}_p \times \boldsymbol{r}\right) - \mathbf{u}^{n+1}\right)\mathbf{A}_u, \tag{3}$$

where ε_p denotes the solid fraction, representing the volume fraction occupied by particles within a given CFD cell. The solid fraction can be computed using a signed distance function, with detailed computational procedures provided in Ref. [43]. For the cell under observation, a solid fraction $\varepsilon_p = 1$ indicates that the cell is fully within a DEM particle, while $\varepsilon_p = 0$ signifies that the cell is entirely outside the particle. χ is a smooth masking function, often associated with the hyperbolic tangent function (tanh) [48,49]. In this study, $\chi = \varepsilon_p(1.0 + \tanh(100(\varepsilon_p - 0.5)))$. ρ_s and ρ represent the densities of the particle and fluid, respectively, while $\mathbf{v}_p$ and $\mathbf{u}$ denote their corresponding velocities. $\boldsymbol{\omega}_p$ represents the angular velocity and $\boldsymbol{r}$ represents the position vector extending from the particle's centroid to the center of the mesh cell. $\mathbf{A}_u$ represents the diagonal elements of the coefficient matrix formed after the discretization of Eq. (2).

In this work, three physical phenomena are considered to calculate the thermal dynamic force: Darcy's effect, recoil pressure, and Marangoni flow. These respectively represent the influence of solid-liquid phase change, evaporation, and the variation of surface tension with temperature.

$$\begin{aligned}\boldsymbol{f}_T = &-K_c\frac{(\alpha_1-\alpha_m)^2}{\alpha_m^3+C_k}\mathbf{u} + 0.54p_0\exp\left(L_v\cdot M\frac{T-T_b}{RTT_b}\right)|\nabla\alpha_1|\frac{2\rho}{\rho_1+\rho_2}\mathbf{n}\\ &+\frac{d\sigma}{dT}(\nabla T - \mathbf{n}(\mathbf{n}\cdot\nabla T))|\nabla\alpha_1|\frac{2\rho}{\rho_1+\rho_2},\end{aligned} \tag{4}$$

where $|\nabla\alpha_i|$ is an interface term used to convert a surface force per unit area into a volumetric surface force [50,51]. ρ_1 and ρ_2 denote the densities of the two phases. $2\rho/(\rho_1+\rho_2)$ represents a sharp surface force used to diffuse the interphase region [52]. K_c is the permeability coefficient, and C_k is a constant (set to 1×10^{-5} in this study)) to avoid division by zero. α_m is the volume fraction of molten metal, which can be approximated by a Gaussian error function [53] as $\alpha_m = \frac{\alpha_1}{2}\left[1+\mathrm{erf}\left(\frac{4}{T_l-T_s}\left(T-\frac{T_l+T_s}{2}\right)\right)\right]$, where T_l and T_s are the liquidus and solidus temperatures, respectively. p_0 is the atmospheric pressure and L_v is the latent heat of vaporization. The term $d\sigma/d$T denotes the temperature dependence of the surface tension coefficient. M is the molar mass, T is the temperature, T_b is the boiling temperature, and R is the universal gas constant.

2.2.2. Multiphase consideration in CFD

The simulation of multiphase flow has commonly been addressed through the Volume of Fluid (VOF) method, utilizing volume fractions α_i to characterize the presence of distinct phases within the domain. Nevertheless, VOF tends to yield a diffused interphase between different phases, necessitating a specialized and intricate approach to capture sharp interfaces. In this study, the default scheme in OpenFOAM [54], MULES, is utilized to sharpen the interface. MULES integrates a compressive flux term into the advection equation to enhance the interface resolution. The following continuity equation with a compressive flux term (Eq. (5)) is utilized to solve the volume fraction field. The fluids are assumed to be incompressible, and their continuity equation can be written as $\nabla\cdot\mathbf{u} = 0$.

$$\frac{\partial\boldsymbol{\alpha_i}}{\boldsymbol{dt}} + \nabla\cdot(\boldsymbol{\alpha_i}\mathbf{u}) + \nabla\cdot(\boldsymbol{\alpha_i}(1-\boldsymbol{\alpha_i})\mathbf{u_c}) = 0, \tag{5}$$

where $\mathbf{u}_c = c|\mathbf{u}|\nabla\alpha_i/|\nabla\alpha_i|$ is the compressed velocity and $c\in[0,\ 1]$ is the coefficient to control the compressed velocity. The compressive velocity is an artificially augmented velocity field specifically designed to counteract numerical diffusion at phase interfaces [55]. It acts as a sharpening term applied to the relative velocity between phases, enhancing interface resolution while maintaining boundedness through flux limiting.

The discretization of Eq. (5) is shown in Eq. (6). The term $\varphi^n_{\alpha_i f}$ signifies the cumulative volume fraction flux at the mesh face at time step n, encompassing both the flux induced by the fluid velocity and the compressed velocity. The core of the MULES algorithm lies in the method used to update this flux term $\varphi^n_{\alpha_i f}$. Further details about the MULES algorithm can be found in the literature [56].

$$\boldsymbol{\alpha_i}^{n+1} = \boldsymbol{\alpha_i}^n - \boldsymbol{\Delta t}\cdot\sum\boldsymbol{\varphi}^n_{\boldsymbol{\alpha_i f}}, \tag{6}$$

The equivalent density ρ and viscosity μ over the entire CFD domain could be updated:

$$\begin{cases}\rho = \alpha_1\rho_1 + \alpha_2\rho_2\\ \mu = \alpha_1\mu_1 + \alpha_2\mu_2\end{cases}, \tag{7}$$

where the subscripts (1 and 2) denote two fluid phases, respectively.

2.2.3. Momentum equations of DEM

DEM is used to solve the linear and angular momentum equations governing the motion of individual particles (Eq. (8)), which are derived based on the Newton-Euler equations. Multiple interaction forces [57,58], including the particle-particle collision force $\mathbf{F}_{p-p}$, the particle-wall collision $\mathbf{F}_{p-w}$, and the fluid-particle interaction $\mathbf{F}_f$ [43], have been considered. The term $\boldsymbol{\omega}_p\times(\mathbf{I}_p\boldsymbol{\omega}_p)$ represents the gyroscopic torque arising from the interaction between the inertia tensor distribution and angular velocity. In the context of DEM, spherical particles are typically assumed to be homogeneous and isotropic, with an inertia tensor $\mathbf{I}_p = I\cdot\mathbf{I}_3$, where $\mathbf{I}_3$ is the identity matrix and I is a scalar constant. As a result, the gyroscopic term $\boldsymbol{\omega}_p\times(\mathbf{I}_p\boldsymbol{\omega}_p) = \boldsymbol{\omega}_p\times(I\boldsymbol{\omega}_p) = I(\boldsymbol{\omega}_p\times\boldsymbol{\omega}_p) = 0$ identically vanishes for spherical particles, and thus only contributes to simulations involving non-spherical particles. The fluid-particle interaction $\mathbf{F}_f$ consists of two components: one arising from the fluid pressure gradient and viscous forces, and the other, associated with $\boldsymbol{f}_s$, accounting for errors due to the averaging of fluid and solid velocities in partially overlapped cells [43].

$$\begin{cases} m_p\dfrac{d\mathbf{v}_p}{dt} = \mathbf{F}_f + m_p g + \sum\mathbf{F}_{p-p} + \sum\mathbf{F}_{p-w}\\ \mathbf{I}_p\dfrac{d\boldsymbol{\omega}_p}{dt} + \boldsymbol{\omega}_p\times(\mathbf{I}_p\boldsymbol{\omega}_p) = \sum\mathbf{M}_t + \sum\mathbf{M}_r\\ \mathbf{F}_f = \sum\limits_{j\in T_h}\left(\dfrac{\rho_s}{\rho}\varepsilon_p\left(-\nabla p + \mu\rho\nabla^2\mathbf{u}_f\right) + \left(1-\dfrac{\rho_s}{\rho}\varepsilon_p\right)\boldsymbol{f}_s\right)_j\cdot V_j\end{cases} \tag{8}$$

where m_p denotes the mass of Particle p. $\mathbf{M}_t$ and $\mathbf{M}_r$ correspond to the torque generated by the tangential force and the rolling friction torque, respectively [59]. The subscript j denotes the cell index in CFD, T_h represents a set of grid cells covered by Particle p, and V indicates the cell volume. The interparticle contact behavior is modeled using the classical Hertzian contact law in combination with Coulomb's friction law [60]. In this study, the particle-particle collisions are characterized by a contact stiffness of 2×10^6 N/m, a friction coefficient of 0.5, a restitution coefficient of 0.3, and a Poisson's ratio of 0.35.

This work employs explicit schemes to discretize the governing Eq. (8) into Eq. (9), facilitating the development of a particle-based parallel

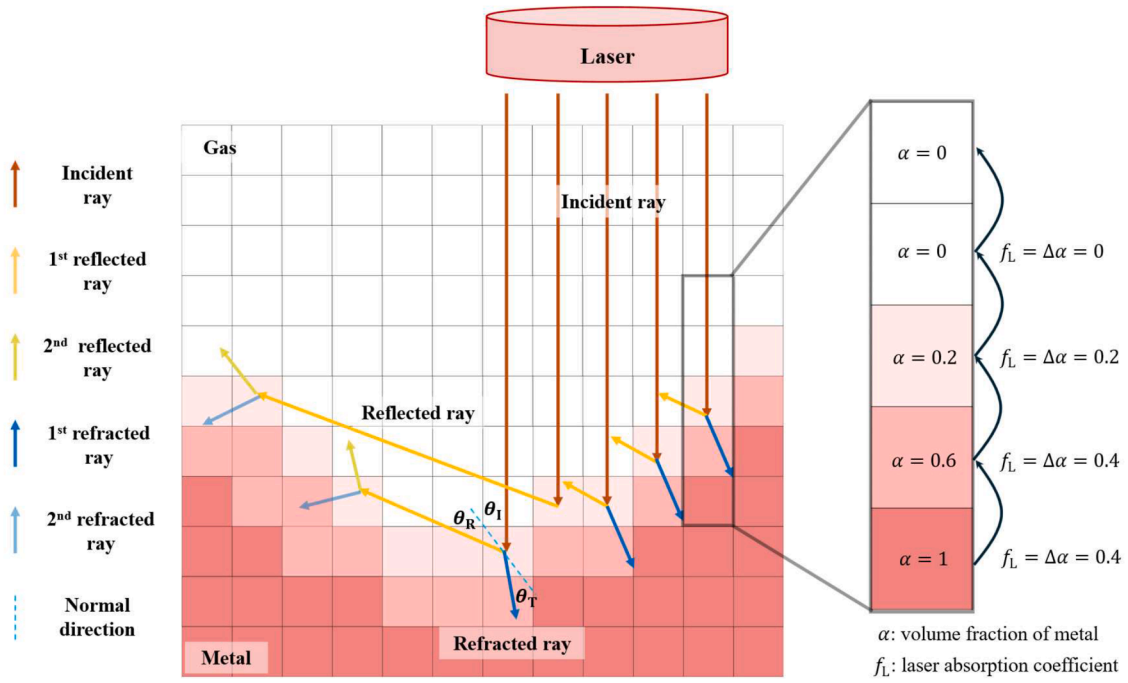


Fig. 3. A schematic illustration of the VOF-compatible ray tracing model.

algorithm compatible with GPU implementation, analogous to the cell- or face-based parallel algorithm illustrated in Fig. 2.

$$\begin{cases} \mathbf{v}_p^{n+1} = \left(\mathbf{v}_p^{\,n} + \left(\mathbf{F}_f + m_p\mathbf{g} + \sum \mathbf{F}_{p-p} + \sum \mathbf{F}_{p-w}\right)^n \Delta t\right) \Big/ m_p \\ \boldsymbol{\omega}_p^{n+1} = \left(\boldsymbol{\omega}_p^{\,n} + \left(\mathbf{I}_p^{-1}\left(\sum \mathbf{M}_t + \sum \mathbf{M}_r - \boldsymbol{\omega}_p \times (\mathbf{I}_p\boldsymbol{\omega}_p)\right)\right)^n \Delta t\right) \end{cases} \tag{9}$$

where the superscript n denotes the n^{th} time step. Δt indicates the time step of DEM.

2.2.4. Temperature equation of resolved CFD-DEM

In this study, a fictitious CFD domain consisting of pure fluid phases and virtual DEM particles is constructed to resolve the high-resolution thermal field for both fluids and particles in laser powder bed fusion (LPBF). This approach, originally proposed in 2021 [1], has been validated through multiple benchmark cases [1,61]. It overcomes the limitation of the homogeneous particle assumption in traditional DEM by capturing rigorous intra-particle temperature gradients under laser heating. Furthermore, once local melting occurs within a particle, the DEM particle is replaced by CFD fluid to enable continued multiphase thermal-fluid simulation. For the CFD fluid, the solid phase is modeled as a highly viscous fluid, and the melting process is captured by a temperature-dependent metal viscosity μ_1, as defined by the following expressions.

$$\ln\mu_1 = \frac{1}{2}\mathrm{erfc}\left[\frac{4}{\ln T_l - \ln T_s}\cdot\left(\ln T - \frac{\ln(T_l)+\ln(T_s)}{2}\right)\right]\cdot(\ln\mu_s - \ln\mu_l) + \ln\mu_l \tag{10}$$

where μ_s and μ_l are the viscosities at solidus temperature T_s and liquidus temperature T_l, and erfc denotes the complementary error function.

The thermal field is governed by the temperature equation (Eq. (11)), which is derived from the principle of energy conservation. The seven terms on the right-hand side of the equation represent the contributions from laser heating, conduction, dissipation, fusion, convection, radiation, and vaporization, respectively.

$$\begin{aligned} &\frac{\partial}{\partial t}(C\rho_T T) + \nabla\cdot(C\rho_T \mathbf{u}T) = S_l + \nabla\cdot\nabla(kT) + \mu(\nabla\mathbf{u} + \mathbf{u}\nabla) : \nabla\mathbf{u} \\ &-L_f\left[\frac{\partial}{\partial t}(\rho_T\alpha_m) + \nabla\cdot(\rho_T\mathbf{u}\alpha_m)\right] \\ &-h_c(T - T_{ref})|\nabla\alpha'_1|\frac{2C\rho_T}{C_1\rho_1 + C_2\rho_2} \\ &-\sigma_{sb}\left(T^4 - T_{ref}^4\right)|\nabla\alpha'_1|\frac{2C\rho_T}{C_1\rho_1 + C_2\rho_2} \\ &-0.82\frac{p_0 L_v M}{(2\pi MRT)^{0.5}}\exp\left(L_v M\frac{T - T_b}{RTT_b}\right)|\nabla\alpha'_1|\frac{2C\rho_T}{C_1\rho_1 + C_2\rho_2} \end{aligned} \tag{11}$$

where L_f is the latent heat of fusion, h_c is the convective heat transfer coefficient, T_{ref} is the reference temperature, and σ_{sb} is the Stefan-Boltzmann constant. ρ_T, C and k denote the effective density, heat capacity, and thermal conductivity, respectively, and are computed as follows: $\rho_T = \varepsilon_p\rho_p + (1-\varepsilon_p)(\alpha_1\rho_1 + \alpha_2\rho_2)$, $k_T = \varepsilon_p k_p + (1-\varepsilon_p)(\alpha_1 k_1 + \alpha_2 k_2)$, and $C = \varepsilon_p\rho_p C_p/\rho_T + (1-\varepsilon_p)(\alpha_1 C_1\rho_1 + \alpha_2 C_2\rho_2)/\rho_T$. The subscripts p, 1, and 2 represent the particle phase, first fluid phase, and second fluid phase, respectively. $\alpha'_1 = \varepsilon_p + \alpha_1$ denotes the effective volume fraction of metal during the LPBF process, accounting for both the solid particle fraction and the fluid phase metal fraction. S_l represents the laser energy input, which is calculated using a ray tracing model [61] specifically tailored for the VOF method.

Fig. 3 illustrates the ray tracing model adopted in this study, which is specifically designed to accommodate the diffuse interfaces generated by the VOF method and accounts for multiple reflections and refractions. The model involves two levels of energy discretization. First, the continuous laser beam is discretized into a set of incident rays. Second, at the interface between fluid phases, each incident ray is further partitioned into sub-rays based on the local volume fraction, distributing energy accordingly. These sub-rays are then individually subjected to ray tracing operations, including reflection and refraction. Fresnel reflection and refraction laws are employed to calculate multiple reflections and refractions at the phase interfaces. It is assumed that the incident angle θ_I is equal to the reflection angle θ_R, and the refraction

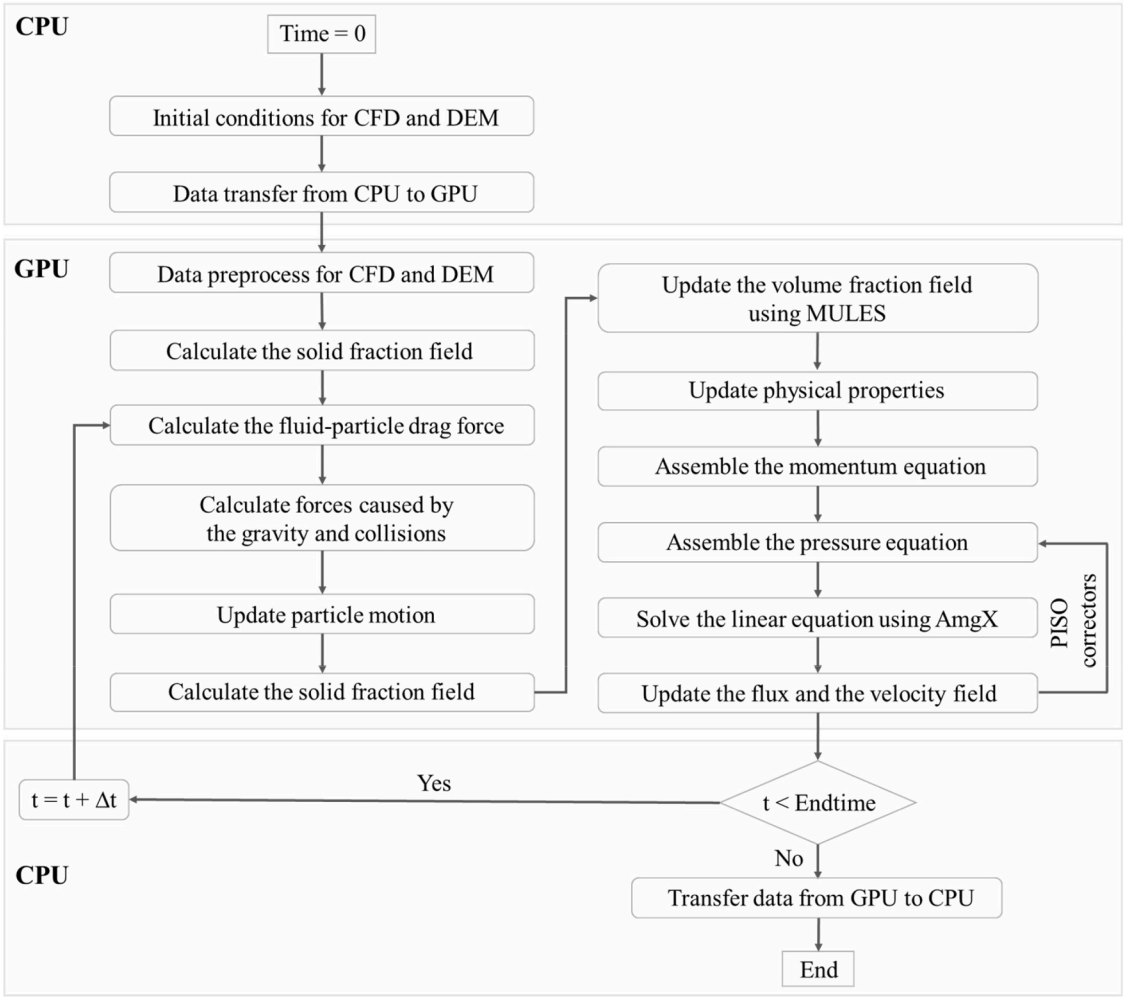


Fig. 4. Flowchart of the full-process GPU accelerated coupled CFD-DEM framework.

angle θ_T satisfies Snell's law: $n_I \sin\theta_I = n_T \sin\theta_T$, where n_I and n_T are the refractive indices of the two respective media.

The incident, reflected, and transmitted energies of each sub-ray are calculated using Eqs. (12)–(14) [62]. In particular, the laser absorption coefficient f_L in Eq. (12) represents the second level of discretization, which is determined based on the difference in the volume fraction α at the fluid interface (Fig. 3). The remaining terms in Eq. (12) correspond to the first level of discretization, where the continuous laser beam is decomposed into incident rays according to the mesh resolution. Additionally, the model accounts for the attenuation of laser energy within the metal phase, with the energy absorbed by each traversed cell computed as described in Eq. (15). Detailed algorithmic procedures and validation cases can be found in Ref. [61].

$$q_I = f_L \frac{2P}{\pi\left(R_0^2 + \left[\frac{\lambda}{\pi R_0}(z - z_f)\right]^2\right)\Delta L} \exp\left[\frac{-2\left[(x - X_l(t))^2 + (y - Y_l(t))^2\right]}{R_0^2 + \left[\frac{\lambda}{\pi R_0}(z - z_f)\right]^2}\right] \tag{12}$$

$$q_R = \frac{1}{2}\left(\frac{1 + (1 - \varepsilon\cos\theta_I)^2}{1 + (1 + \varepsilon\cos\theta_I)^2} + \frac{\varepsilon^2 - 2\varepsilon\cos\theta_I + 2\cos^2\theta_I}{\varepsilon^2 + 2\varepsilon\cos\theta_I + 2\cos^2\theta_I}\right) q_I \tag{13}$$

$$q_T = \left(1 - \frac{1}{2}\left(\frac{1 + (1 - \varepsilon\cos\theta_I)^2}{1 + (1 + \varepsilon\cos\theta_I)^2} + \frac{\varepsilon^2 - 2\varepsilon\cos\theta_I + 2\cos^2\theta_I}{\varepsilon^2 + 2\varepsilon\cos\theta_I + 2\cos^2\theta_I}\right)\right) \tag{14}$$

$$q_C = \left(e^{-\gamma l_p} - e^{-\gamma(l_p + \Delta L)}\right) q_T \tag{15}$$

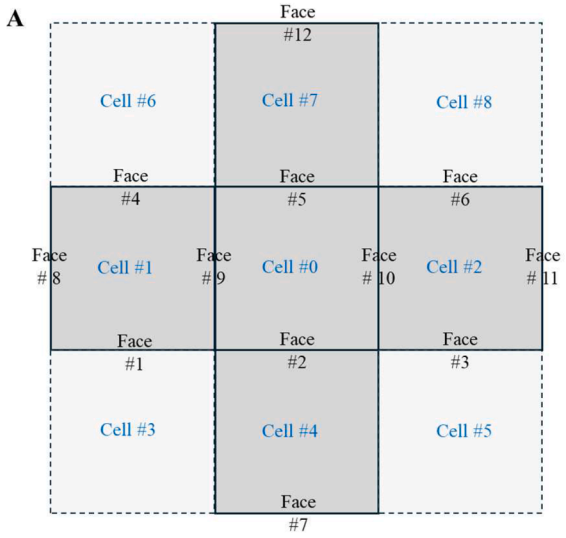
Where P is the laser power, R_0 is the laser beam radius, and z_f is the z-coordinate of the lens focus. The laser is assumed to irradiate vertically along the z-axis. λ is the laser wavelength, and ΔL is the cell size. (x, y, z) denotes the coordinates of the target cell and $(X_l(t), Y_l(t))$ specifies the laser scanning center on the x-y plane. ε is a material-dependent parameter related to electrical conductivity [63,64]. γ is the attenuation coefficient governing laser energy decay during penetration [65], and l_p is the distance from the point of laser incidence to the target cell along the ray path.

2.2.5. *Full-process GPU parallel framework*

Full-process GPU-accelerated CFD-DEM refers to executing all steps, except for mesh reading and results export, entirely on the GPU. This includes mesh postprocessing, equation discretization and assembly, and linear system solving. This section focuses on presenting a basic parallel framework without delving into the detailed algorithm implementation. It is important to note that in full-process GPU-accelerated CFD-DEM, all computational modules are executed on the GPU. Therefore, the implementation of adaptive mesh refinement algorithms must also be fully GPU-based. Any data transfer between the CPU and GPU during the adaptive mesh refinement process would significantly reduce overall computational efficiency.

The computational modules of the full-process GPU-accelerated CFD-DEM framework primarily consist of the following components:

A



B

Inner face index	Owner cell index	Neighbor cell index
1	1	3
2	0	4
3	2	5
4	1	6
5	0	7
6	2	8
9	0	1
10	0	2

C

Boundary face index	Owner cell index	Neighbor cell index
7	4	-
8	1	-
11	2	-
12	7	-

Fig. 5. Schematic representation of the mesh data structure: (A) face and cell indices; (B) owner and neighbor cell indices for selected interior faces; (C) owner cell indices for selected boundary faces. For interior faces, the owner cell index is always smaller than the neighbor cell index. Boundary faces have no neighbor cell, as each boundary face is adjacent to only one cell.

(1) *Basic Mathematical Operations*: Fundamental arithmetic operations such as addition, subtraction, multiplication, and division.
(2) *Explicit Computations*: Calculations including gradients, face interpolation, divergence terms, and particle interaction forces.
(3) *Equation Assembly*: Construction of linear systems, incorporating terms such as convective, diffusive, transient, and source terms.

For Components (1) and (2), parallel algorithms based on cells, faces, or particles (Fig. 2) can be directly employed. For component (3), cell- or face-based parallel algorithms must be utilized to assemble the coefficient matrix **A** and constant vector **B** of the linear equations ($\mathbf{A}x = \mathbf{B}$), depending on the chosen discretization scheme. To reduce GPU memory usage, the matrix **A** is stored in the Compressed Sparse Row (CSR) format [66], which is widely used in sparse linear algebra. For instance, in a mesh with one million cells, the resulting sparse matrix contains fewer than seven million nonzero double-precision entries, corresponding to roughly 56 MB of memory. To clarify the distinction between the abstract mathematical formulation and the actual implementation, we provide a theoretical comparison with a full matrix representation. While not used in practice, such a representation would require on the order of 10^{12} double-precision entries, approximately 8 terabytes of memory, rendering it computationally infeasible for GPU-based simulations. This contrast underscores the necessity of adopting sparse matrix storage formats such as CSR in large-scale simulations. Moreover, it is important to note that for face-based parallel algorithms, atomic operations are necessary to prevent memory conflicts that could lead to incorrect results. In this study, the algebraic multigrid solver (AMGx) library, developed by NVIDIA, is employed to solve the linear equations.

2.2.6. Overall solution procedure

The fully GPU-accelerated resolved coupled CFD-DEM framework proposed in this study has been implemented in the CFD-DEM software TFluid (*www.t-fluid.com*), developed on the CUDA *C++* platform. Both the CFD and DEM solvers are executed entirely on the GPU, with all data residing in the global memory. As a result, during coupling, the required information, such as fluid velocity fields, particle positions, and velocities, can be directly accessed from the global memory without CPU-GPU communication overhead. It is important to note that the coupling procedure typically involves neighborhood search algorithms, including both particle-particle and particle-cell proximity queries. In this work, a classical spatial hashing technique is employed, wherein the computational domain is divided into uniform cubic bins to facilitate efficient neighbor searching. The algorithm's core solution procedure is outlined below, in conjunction with the flowchart depicted in Fig. 4:

(1) Initialization: Set the initial conditions for the CFD and DEM domains, including mesh data, boundary conditions, initial conditions, particle data. Transfer the input data from CPU memory to GPU memory.
(2) Preprocess: Compute mesh-related information and particle-specific data on the GPU using the cell-/face-/particle-based parallel algorithm (Fig. 2), such as cell volumes, face normal vectors, and particle volumes.
(3) DEM: Evaluate forces due to gravity, particle-fluid interaction (Eq. (8)), and particle collisions. Update particle motion using the explicit scheme (Eq. (9)).
(4) Multiphase model: Solve the multiphase model using the MULES algorithm (Eqs. (5) and (6)), updating the volume fraction field for the two phases. Adjust the physical parameters in Eq. (7), including density and viscosity, based on the updated volume fraction field.
(5) PISO solver: Assemble the momentum equations (Eqs. (1) and (2)) without the pressure term, and construct the pressure equation. Solve the pressure equation using AmgX, then update the pressure, flux, and velocity fields accordingly.
(6) Iteration: Return to Step (3) and repeat the simulation until the final time step is reached.

This streamlined iteration ensures efficient simulation performance entirely on the GPU, leveraging the framework's full computational potential.

2.3. A GPU-accelerated AMR algorithm

Three essential requirements must be satisfied for a GPU-accelerated AMR algorithm to be practical and efficient: (1) The entire AMR algorithm must be implemented exclusively on the GPU, completely avoiding data transfer between the CPU and GPU; (2) The AMR algorithm should exhibit scalability and not be restricted to specific mesh structures. For example, octree structures are generally limited to hexahedral meshes; (3) The implemented AMR algorithm must be sufficiently efficient, ensuring that its computational cost constitutes only a small fraction of the total duration for one time step. By fulfilling these three

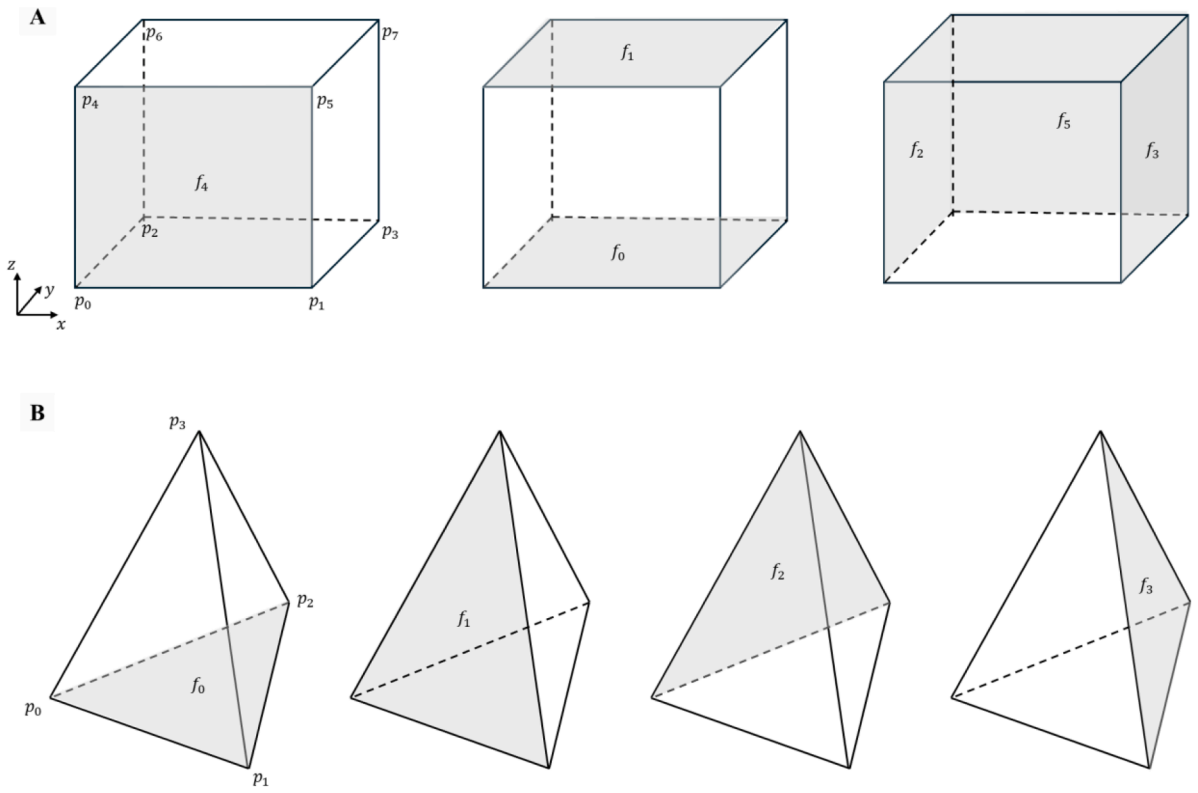


Fig. 6. Point and face index conventions for parent mesh cells used in refinement: (A) regular hexahedron; (B) tetrahedron.

requirements, the proposed algorithm ensures high efficiency and strong scalability, making it suitable for integration into various CFD and CFD-DEM algorithms.

The primary challenge involves refining the mesh, reconstructing its topology, rebuilding data structures, and remapping data from the original mesh to the updated mesh on the GPU. The following sections present a comprehensive explanation of how the proposed algorithm effectively addresses these challenges.

2.3.1. *Unstructured mesh*

Mesh storage structures are generally categorized into two types: structured and unstructured meshes. Structured meshes allow for efficient identification of neighboring cells, facilitating the assembly of linear equation systems. However, their application is limited to computational domains where the curved boundaries are not readily amenable to alignment with a structured grid. Consequently, most commercial and open-source software employs unstructured meshes, which can accommodate arbitrary computational domains. For instance, the widely used open-source software OpenFOAM employs a face-based unstructured mesh structure. This structure represents the mesh topology through three primary lists (Fig. 5): point coordinates (*P*), point indices of faces (*F*), and boundary information (*B*), along with two lists defining face relationships: the owner (*O*) and neighbor (*N*) cells of the faces. This study employs the same unstructured mesh framework as OpenFOAM to ensure broader applicability across diverse scenarios. As the primary focus is on the adaptive mesh refinement algorithm, mesh generation is not addressed in detail. It is assumed that the five previously mentioned lists are already provided.

The proposed AMR algorithm is primarily designed for unstructured hexahedral grids. In addition to regular cuboid domains or those composed of multiple adjoining cuboids, it can also be combined with the cut-cell method [67,68] to accommodate computational domains containing curved or inclined boundaries commonly encountered in industrial or laboratory scenarios [69–71], such as cylindrical walls, trapezoidal enclosures, or undulating terrain surfaces. In such cases, the boundary region is represented by polyhedral cells formed by cutting the base hexahedral grids, which are then seamlessly connected to the interior hexahedral mesh. This results in a hybrid grid structure where the interior retains hexahedral regularity while the boundary is represented by locally refined polyhedral cells. Consequently, structured grids cannot be directly employed, and an unstructured representation becomes necessary to preserve the geometric fidelity of the boundary.

It should be emphasized that, in the resolved CFD-DEM framework, the particle diameter is typically at least 8–10 times [41,42] larger than the local cell size. Therefore, the cut-cell boundaries only affect particles located near the wall region, typically within a range below 10% of the computational domain. Moreover, multiple levels of refinement [67,68] can be applied prior to the cutting operation to further minimize the influence of boundary cells on particle-wall interactions. Consequently, under these conditions, the core focus of the AMR scheme can remain on the dynamic adaptation of the internal hexahedral cells, largely independent of the specific cell types at the boundary. Accordingly, both rectangular and cylindrical computational domains are used in this study to evaluate the core performance of the AMR scheme.

2.3.2. *Mesh refinement*

The AMR algorithm focuses on refining selected regions and reconstructing the mesh topology to generate updated unstructured mesh information, as described in the previous section. This chapter specifically introduces the first part of the algorithm, which concentrates on refining selected regions and consists of four key steps: (1) Copy the initial mesh at $t = 0$ s as the parent mesh; (2) Establish criteria to identify regions requiring mesh refinement; (3) Determine the mesh type within the selected regions and the corresponding refined sub-mesh type; (4) Construct cell- and face-based number lists to rebuild the counts of points, faces, and cells; The following part will provide a detailed explanation of these four steps.

(1) Parent mesh preparation

Algorithm 1
Mapping of the refinement index.

Input: Refinement index for the parent mesh I_0, refinement index for the current mesh I_t, cell number of the current mesh N_{Ct}, and mapping array I_{map}.
Output: Refinement index for the parent mesh I_0.
1st GPU kernel:
for $i < N_{Ct}$ **do**
  Get the cell index $j = I_{\text{map}}[i]$ in the parent mesh corresponding to cell index i in the current mesh;
  Calculate the refinement index of the parent mesh: $I_0[I_{\text{map}}[i]] = \max\{I_0[I_{\text{map}}[i]], I_t[i]\}$;

The initial mesh is saved as the parent mesh, serving as the foundation for AMR processing at each subsequent time step. In each time step, specific regions are selected for mesh refinement based on this parent mesh. An alternative approach involves using the mesh from the previous time step as the base, refining the mesh in selected regions or removing previously refined mesh. While this method reduces the number of mesh elements being processed, it requires handling both the addition and removal of mesh elements and limits the ability to fully leverage the GPU's parallel processing capabilities. In contrast, using a single parent mesh requires only the addition of mesh elements, simplifying the logic and ensuring a uniform algorithm for every time step. This approach is better suited for large-scale parallelism on GPUs.

Fig. 6 provides a schematic illustration of the point and face indexing schemes for a single parent-cell, using both a regular hexahedral and a tetrahedral element as examples. Specifically, point indexing follows the convention used in the VTK library [72] for voxel and tetrahedral elements. For face indexing, a mapping between point indices and face indices is explicitly defined. Fig. 6A illustrates the point-to-face mapping for a hexahedral element; for example, face f_0 is defined by the quadrilateral formed by points p_0, p_1, p_2, and p_3, while face f_1 is formed by points p_4, p_5, p_6, and p_7. Fig. 6B shows the corresponding mapping for a tetrahedral element; for instance, face f_0 is formed by points p_0, p_1, and p_2, and face f_1 is formed by points p_0, p_1, and p_3. With this mapping, the local face index within a given cell can be determined by matching the vertex indices of a face during face-based traversals. These indexing conventions serve as the basis for subsequent mesh refinement operations.

(2) Refinement index computation

Based on the current mesh, a refinement index I_t is computed to identify cells requiring refinement. This index is formulated as a Boolean variable, with 1 indicating a cell marked for refinement and 0 indicating no refinement needed. Common physical criteria for I_t include the gradient of the volume fraction in general two-phase flows or the gradient of the solid fraction in CFD-DEM simulations. For the present work, we adopt these criteria to enable the selective application of grid refinement specifically around the particle-fluid and two-phase interfaces. This targeted refinement is critical for achieving the accuracy required in force computations.

Under this criterion, cells entirely inside the particles are maintained at a coarse level since the fluid velocity in these regions is prescribed by

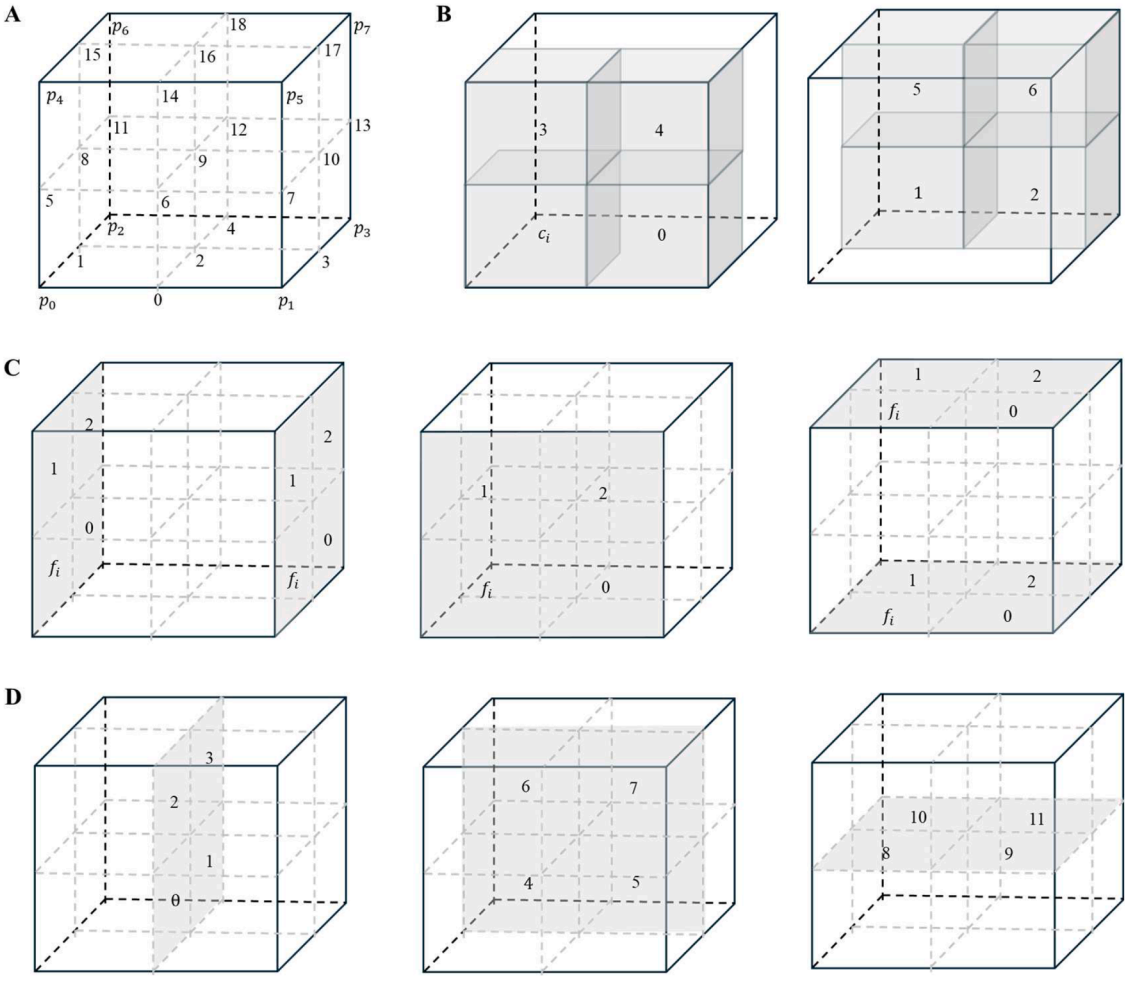


Fig. 7. Local index mapping for a hexahedral sub-mesh: (A) vertex indices, (B) sub-cell indices, (C) face indices on the parent mesh, and (D) face indices in the sub-mesh. $\boldsymbol{p}_0$ to $\boldsymbol{p}_7$, $\boldsymbol{c}_i$, and $\boldsymbol{f}_i$ represent the initial point indices, cell index, and face index for the parent mesh.

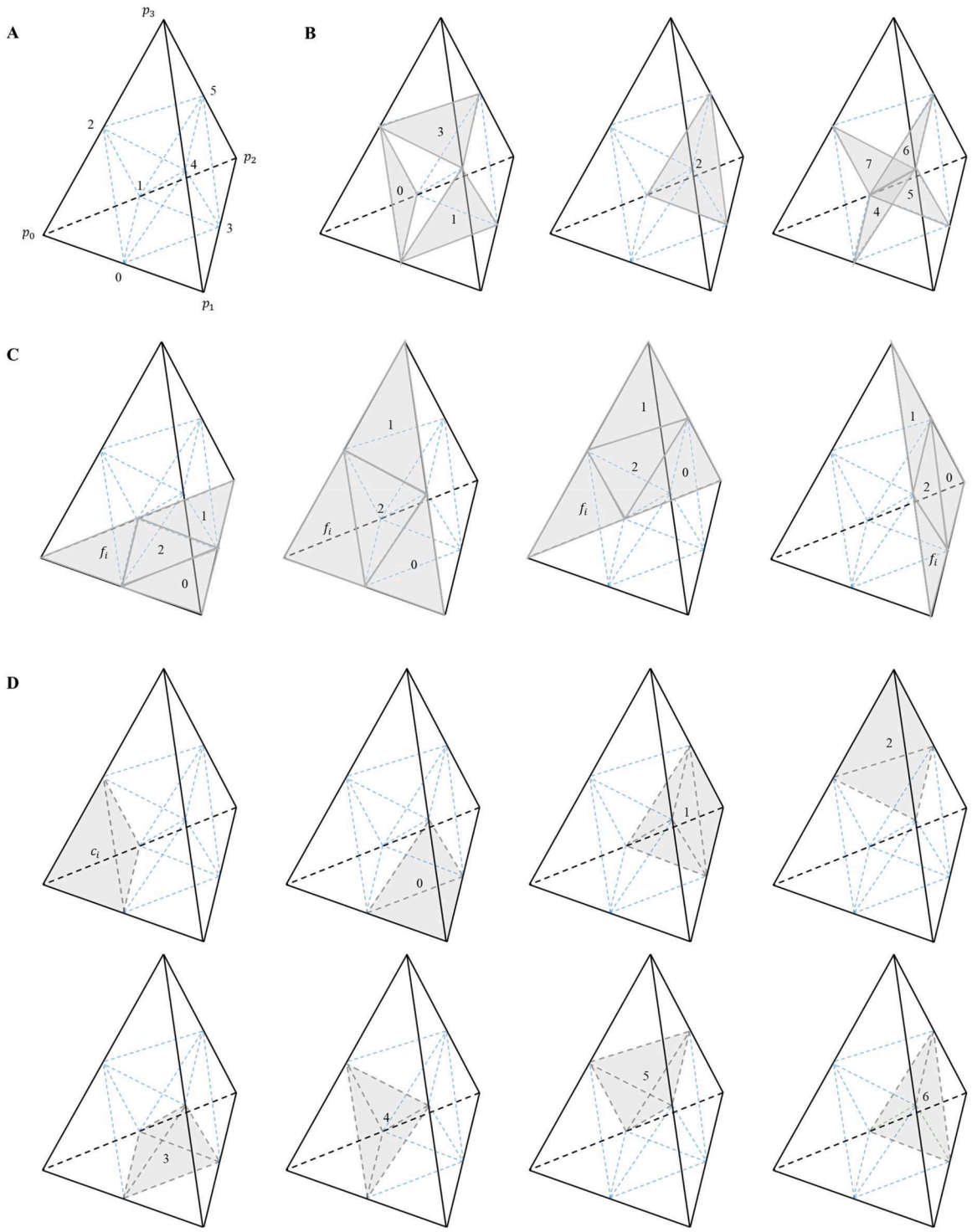


Fig. 8. Local index mapping for a tetrahedral sub-mesh: (A) vertex indices, (B) face indices in the sub-mesh, (C) face indices on the parent mesh, and (D) sub-cell indices. $\boldsymbol{p}_0$ to $\boldsymbol{p}_3$, $\boldsymbol{c}_i$, and $\boldsymbol{f}_i$ represent the initial point indices, cell index, and face index for the parent mesh.

Eq. (3) to match the particle velocity, and is thus independent of the CFD solution, rendering further refinement unnecessary. Although the refinement criterion could be extended to include particle interiors, the strategy employed here focuses selectively on the interfaces to achieve an optimal balance between numerical accuracy and computational efficiency.

Next, the refinement index I_t from the current mesh must be mapped onto the parent mesh, denoted as I_0, to determine which cells of the parent mesh require refinement. This mapping process involves a mapping array, I_{map}, of size N_{Ct}, where $I_{\text{map}}(i) = j$ indicates that cell i in the current mesh corresponds to cell j in the parent mesh, and N_{Ct} denotes the cell number of the current mesh. Using this mapping, the refinement index for the parent mesh, I_0, can be updated as $I_0\left[I_{\text{map}}[i]\right] = \max\left\{I_0\left[I_{\text{map}}[i]\right], I_t[i]\right\}$. Algorithm 1 presents the GPU kernel function utilized in this process. For the initial state, $I_{\text{map}}[i] = i$. For the other time steps, the computation of the mapping array I_{map} will be explained in the next sections.

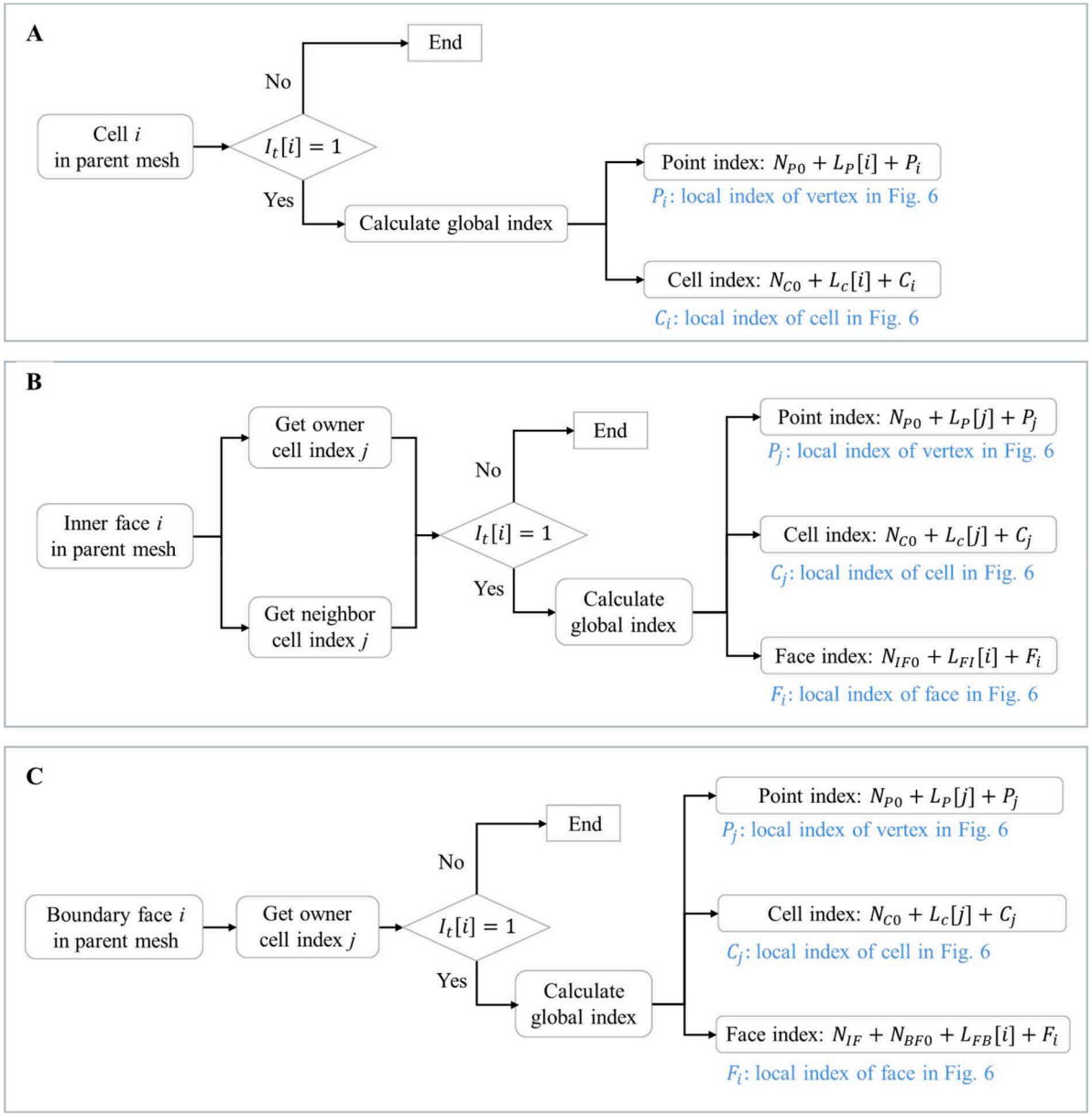


Fig. 9. Schematic illustration of the global index computation for newly added mesh points, cells, and faces by traversing cells (A), interior faces (B), and boundary faces (C). The traversal pattern corresponds to the two fundamental parallelization strategies introduced in Fig. 2.

(3) Sub-mesh structure

After identifying the parent mesh cells requiring refinement, their cell types are determined, and corresponding sub-meshes are assigned. For programming convenience, the sub-mesh structure for each cell type can be predefined, including sub-cells and sub-faces. Specifically, the sub-mesh must predefine the number and local indices of new points and faces added to the parent mesh face, as well as the number and local indices of new points, faces and cells within the parent mesh cell. This facilitates subsequent face numbering and mesh renumbering.

Taking a hexahedron as an example, a classical refinement approach is similar to an octree structure, where the cell is uniformly divided into 8 smaller hexahedra, as illustrated in Fig. 7. However, it is important to note that this study does not adopt the octree structure. Instead, the approach shown in Fig. 7 is used, which employs local point indices (Fig. 7A), local cell indices (Fig. 7B), and local face indices (Fig. 7C and D). The numbering of sub-cells and sub-faces in the proposed sub-mesh framework must satisfy the following requirements to ensure algorithmic generality and efficient execution: (1) Maximally reuse information from the parent mesh to minimize the computational load required at each time step; (2) Numbering scheme should be compatible with mesh types, such as tetrahedral and hexahedral meshes.

Firstly, when defining local indices, we opt to reuse the initial mesh indices, as illustrated by p_0 to p_7, c_i, and f_i in Fig. 7. The remaining local indices are appended to the end of the mesh topology data, including the owner array, neighbor array, point array, face array, and boundary face array. This approach eliminates the need to reorder or update information for cells that do not require refinement, allowing the original data to be directly copied while updating only the information for refined cells. Through this operation, the aforementioned requirement (1) can be effectively satisfied.

Secondly, for vertices, faces, and cells that are not inherited from the initial mesh indices, custom numbering rules are applied. For hexahedral meshes, indices are assigned to the sub-mesh elements based on the magnitude of the three-dimensional coordinates of vertices, face centers, and cell centers, as illustrated in Fig. 7. Additionally, we further present

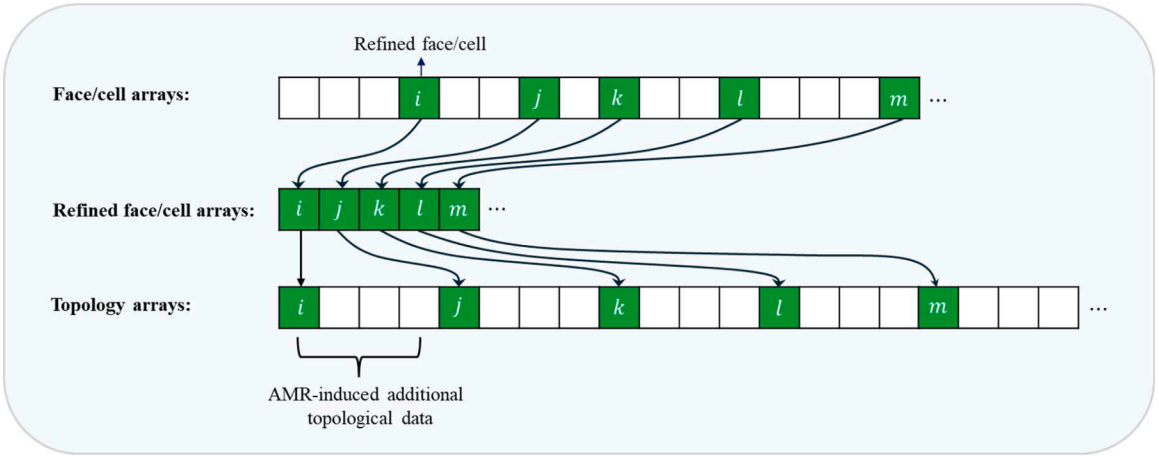


Fig. 10. Schematic illustration of the topology reconstruction process. The refined face and cell arrays are compressed representations of the original arrays, designed to improve GPU parallel efficiency during the construction of topology arrays. Note that the amount of AMR-induced additional topological data shown in the figure is for illustrative purposes only; the actual data depends on the specific refinement strategy and the variables involved. Details can be found in the subsequent algorithm descriptions.

a feasible example of a tetrahedral sub-mesh, as illustrated in Fig. 8, which includes the vertex indices, face indices, and sub-cell indices. These indices are systematically assigned based on the ordering of the four vertices of the parent tetrahedron. Specifically, given the set of three vertex indices that define a face and the full set of four vertex indices of the tetrahedron, the local indices of the sub-cells adjacent to the corresponding sub-face can be efficiently identified. For example, if the input face is defined by the vertex set $\{p_1, p_2, p_3\}$, the four sub-cells adjacent to this face can be directly mapped to local indices 0, 1, 2, and 6. Furthermore, based on the vertex coordinates and geometric relationships, important geometric information such as the centroids of sub-cells and sub-faces can be rapidly determined. For other types of meshes, a custom numbering scheme can be similarly adopted, ensuring compliance with the aforementioned requirement (2). However, it is important to note that the numbering scheme must satisfy the condition that a refined sub-face index can be quickly mapped to its adjacent sub-cell index. The significance of this requirement will be elaborated upon in the following steps.

(4) Numbering list construction

Following the determination of refinement indices and sub-mesh configurations, the cardinality of new geometric entities (vertices, faces, and cells) can be systematically quantified. Taking the hexahedral mesh used in this work as an example, a single first-level refined sub-mesh results in an increase of 19 vertices and 7 cells. Additionally, the newly generated faces are categorized into three groups based on their location: on the inner face of the parent mesh (FI), on the boundary face of the parent mesh (FB), and inside the sub-mesh (FC). For faces located on the inner or boundary faces of the parent mesh, the face count increases by 3, whereas for faces inside the sub-mesh, the face count increases by 12. The construction process of the numbering lists of points (L_P), cells (L_C), FI faces (L_{FI}), FB faces (L_{FB}), and FC faces (L_{FC}) can be found in Algorithm A1.

The total number of point (N_P), cell (N_C), inner face (N_{IF}), boundary face (N_{BF}) for the refined mesh can also be obtained by the sum of the numbering lists of L_P, L_C, L_{IF}, L_{BF}, L_{FC}, and initial number of point (N_{P0}), cell (N_{C0}), inner face (N_{IF0}), boundary face (N_{BF0}) for the original mesh. Drawing inspiration from CSR format efficiency, we implement inclusive scan operations to dynamically update indexing structures. This optimization enables efficient index mapping between parent mesh subdomains and their refined counterparts during subsequent computational phases.

Our parallel processing framework employs a dual traversal strategy aligned with fundamental GPU parallelism paradigms illustrated in Fig. 2. By systematically traversing mesh cells, interior faces, and boundary faces, we efficiently compute global indices for all refined mesh entities through coordinated utilization of three critical components: the parent mesh's original entity counts, dynamically updated numbering lists from current refinement operations, and local-to-global mapping schematics depicted in Fig. 7. This integrated approach ensures deterministic index resolution for newly generated geometric entities while rigorously preserving topological relationships inherited from the parent mesh. The computational efficacy of this methodology is visually demonstrated in Fig. 9, where the derived global indices maintain both geometric consistency and hierarchical connectivity across refinement levels. The traversal mechanism inherently aligns with GPU-optimized memory access patterns, enabling concurrent processing of mesh entities without compromising data dependency constraints.

2.3.3. *Topology reconstruction*

The objective of this step is to reconstruct the five key arrays that define the topology of unstructured meshes, as described in Section 2.3.1. A schematic overview of the topology reconstruction process is presented in Fig. 10. Prior to this reconstruction, three auxiliary arrays are introduced to store the indices of all cells (*RC*), interior faces (*RIF*), and boundary faces (*RBF*) that require refinement. Their respective sizes are $NL_C/19$, $NL_{FI}/3$, and $NL_{FB}/3$, where NL_C, NL_{FI}, and NL_{FB} denote the total number of cells, interior faces, and boundary faces before refinement. These refinement arrays explicitly identify the mesh entities that must be updated following adaptive refinement. The construction process of these refinement arrays is detailed in Algorithm A2 in the Appendix.

This design is critical for maintaining high parallel efficiency on NVIDIA GPUs, where 32 threads typically form a warp that executes the same instruction simultaneously. By eliminating the conditional branching illustrated in Fig. 9, which would otherwise arise from checking whether each entity requires refinement, our pre-filtering arrays ensure warp-level coherence. Without these pre-filtering arrays, the presence of only a few cells or faces requiring refinement within a warp would lead to significant thread divergence and reduced computational throughput.

Subsequent sections will present a systematic reconstruction methodology for these core topological arrays, comprising point coordinates, face connectivity, owner-neighbor associations, and boundary definitions. The reconstruction process fundamentally relies on the global

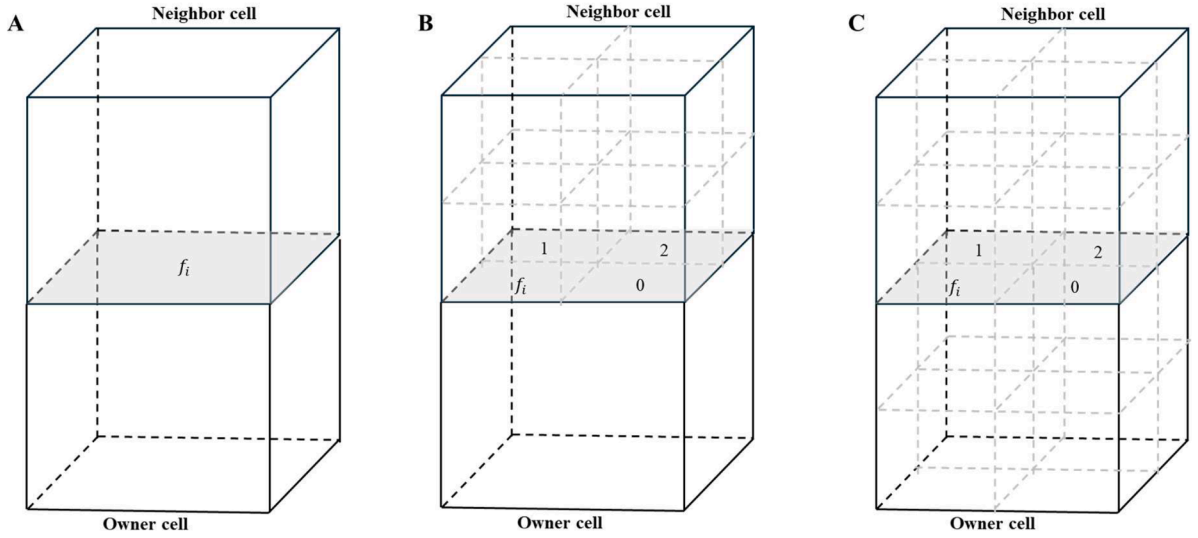


Fig. 11. Schematic illustration of three typical scenarios for an inner face in a hexahedral parent mesh: (A) Neither the owner cell nor the neighbor cell is refined; (B) Only the owner cell is refined; (C) Both the owner cell and the neighbor cell are refined.

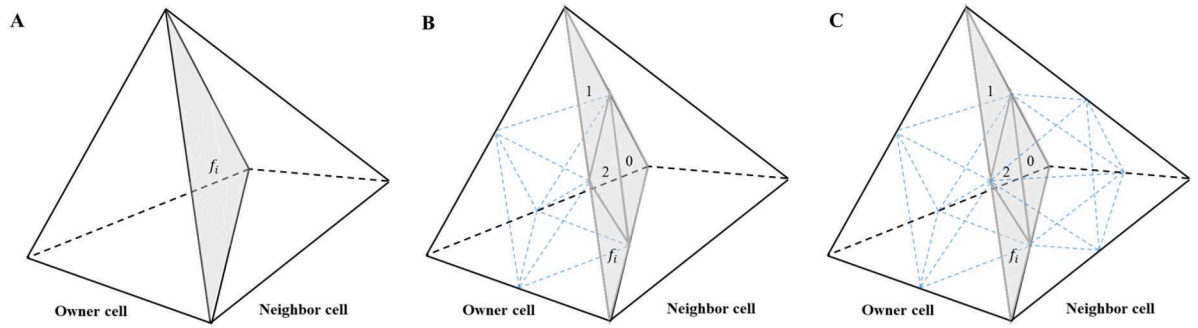


Fig. 12. Schematic illustration of three typical scenarios for an inner face in a tetrahedral parent mesh: (A) Neither the owner cell nor the neighbor cell is refined; (B) Only the owner cell is refined; (C) Both the owner cell and the neighbor cell are refined.

index construction framework demonstrated in Fig. 9, which enables efficient cross-reference between pre-refinement and post-refinement topological relationships through deterministic mapping operations.

(1) Point array

The list of newly added points is appended to the original point array. These new points correspond to the midpoints of edges and faces, as well as the centroids of cells in the original mesh, and their coordinates can be readily computed from the coordinates of the original mesh points. Given the total number of points in the parent mesh N_{P0} and the local numbering list L_P, obtained via an inclusive scan, the global indices of the newly added points can be systematically determined (Fig. 9). The coordinate construction process follows the local numbering convention illustrated in Fig. 7A and is implemented according to the procedure detailed in Algorithm A3 in the Appendix. It is worth noting that the resulting point array may contain duplicate entries, corresponding to geometrically identical points shared across neighboring mesh elements. These redundant points can be removed using established geometric filtering algorithms. However, such duplicates do not interfere with the correctness of subsequent numerical computations, and their removal is not strictly necessary for the method to function correctly.

(2) Face array and boundary array

The reconstruction of the boundary array is relatively straightforward, as the boundary condition type for each newly added boundary face must remain consistent with that of the corresponding parent face in the original mesh. In reconstructing the face array, the newly introduced faces are categorized into three groups to assign their global face indices: (1) faces that coincide with the interior faces of the parent cell (FI), (2) faces corresponding to the parent's boundary faces (FB), and (3) internal faces formed entirely within the refined sub-cell mesh (FC).

The face indices in the refined mesh are composed of two parts: the number of pre-existing faces in the parent mesh and the local indices of the newly added faces, as defined by the corresponding numbering lists. For the first category, global indices are determined using the total number of interior faces in the parent mesh and the numbering list L_{FI}. For the second category, the global indices are computed by considering the sum of the original interior faces and the first set of added faces, referencing the boundary face numbering list L_{FC}. For the third category, indices are assigned by incorporating all previous face counts and using the sub-mesh face numbering list L_{FC}. With the global indices of newly added points and faces established, and by referring to the local face and point numbering illustrated in Fig. 7, the face and boundary arrays can be reconstructed (Fig. 9). The full procedure is described in Algorithms A4 and A5 in the Appendix.

(3) Owner array and neighbor array

To reconstruct the owner and neighbor arrays, we reuse the global

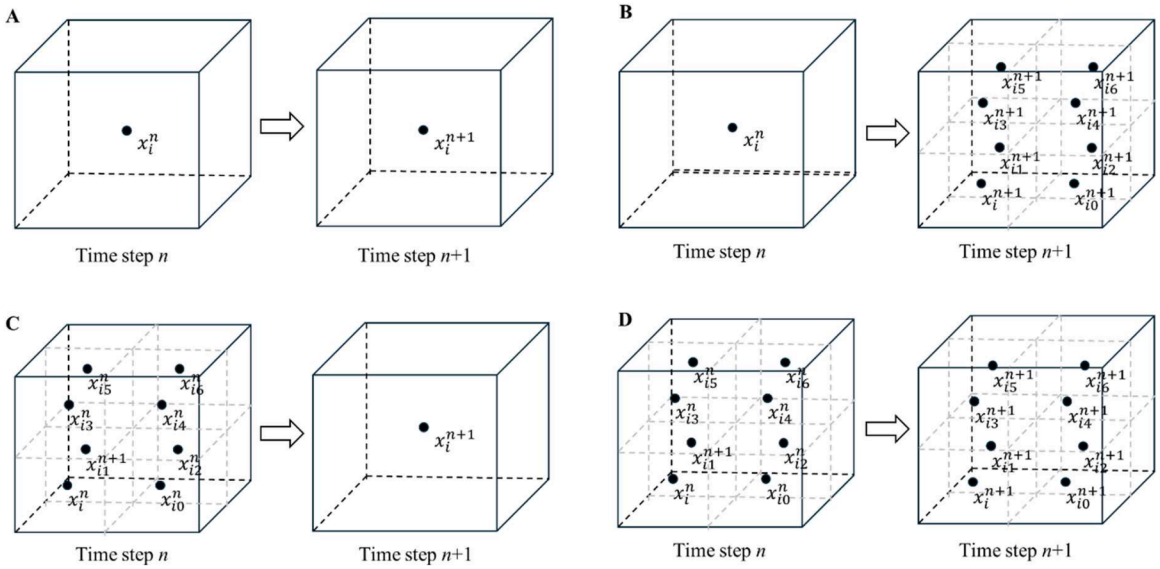


Fig. 13. Schematic representation of the four typical scenarios encountered during field remapping in AMR: (A) Cell remains unrefined; (B) Cell is refined; (C) Cell is coarsened; (D) Cell remains refined.

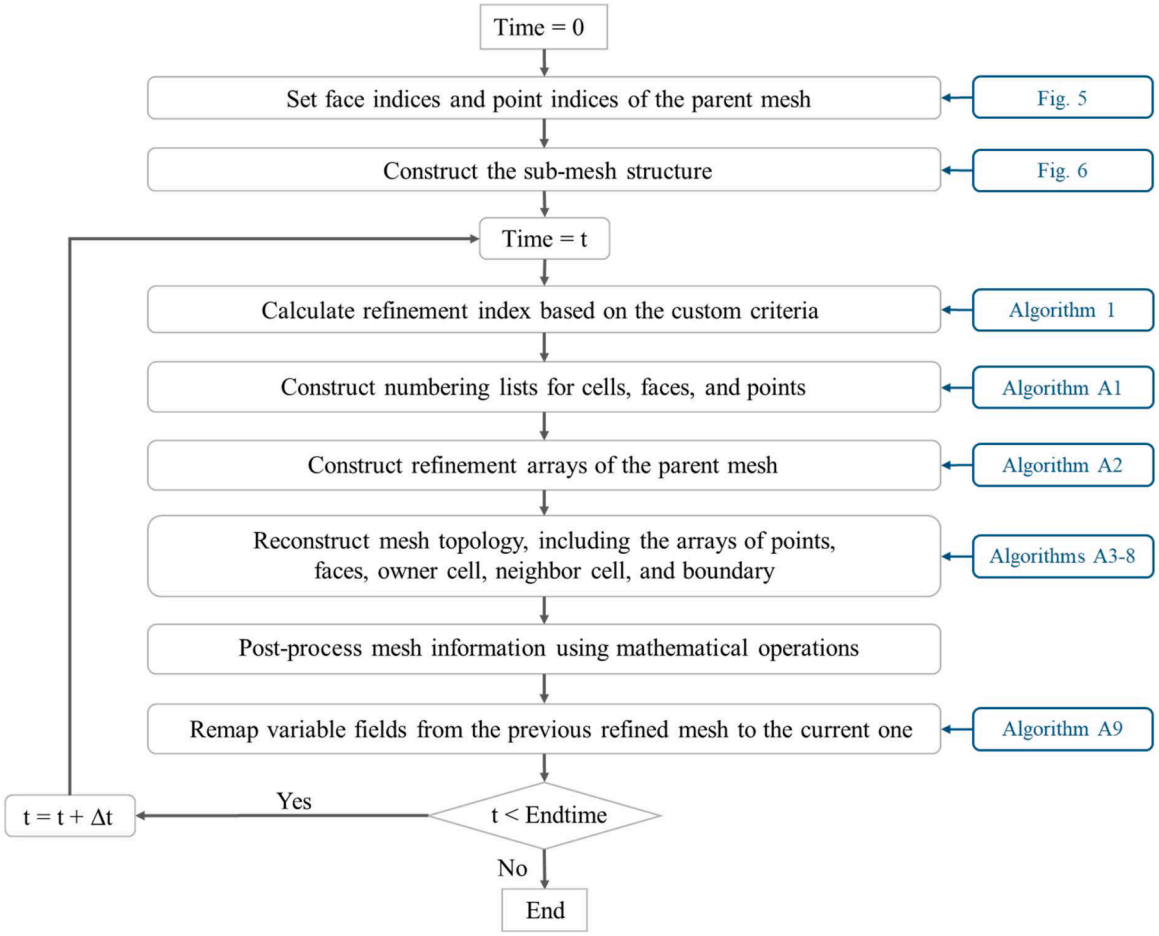


Fig. 14. Workflow of the proposed GPU-accelerated AMR algorithm.

indices of the three categories of newly added faces defined in the previous section. By iterating over the parent mesh faces marked for refinement, the global indices of the new faces, as well as their associated owner and neighbor cell indices, can be determined using the face numbering lists and the local indexing schemes illustrated in Figs. 6 and 7. These mappings shown in Fig. 9 enable the accurate reconstruction of the owner and neighbor arrays.

For internal faces of the parent mesh, the post-refinement

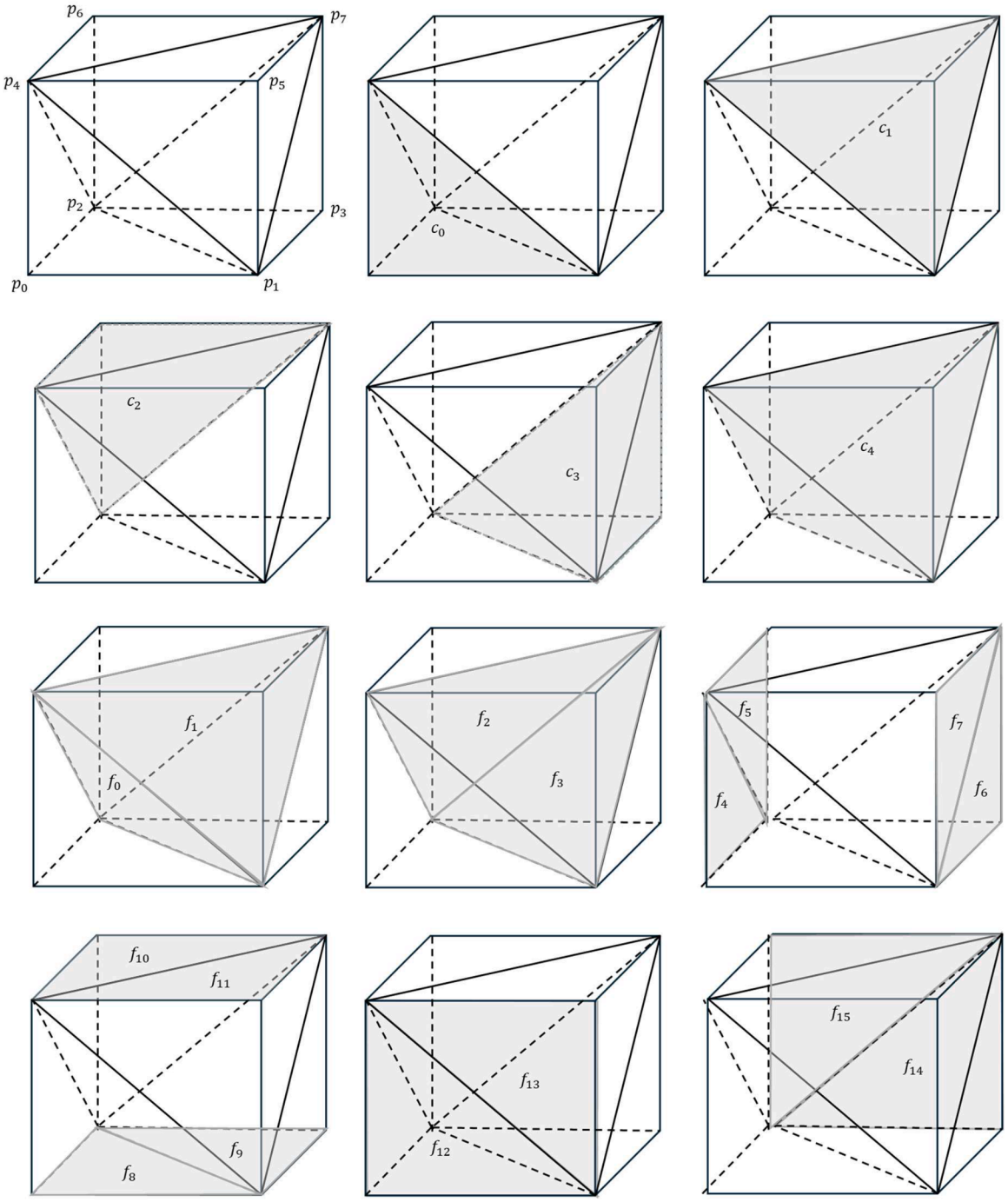


Fig. 15. Point, cell, and face indices for the minimal AMR case. Here, $\boldsymbol{p_i}$, $\boldsymbol{c_i}$, and $\boldsymbol{f_i}$ denote the indices of points, cells, and faces, respectively.

connectivity between owner and neighbor cells falls into three typical cases, as shown in Figs. 11 and 12: (1) neither the owner nor the neighbor cell is refined; (2) only one of the two is refined; or (3) both are refined. Each case is handled separately to preserve face-cell consistency, with the detailed procedure presented in Appendix, Algorithm A6.

For faces located entirely within sub-meshes, the owner and neighbor cell indices are directly derived from the predefined numbering schemes in Fig. 7. This step is outlined in Appendix, Algorithm A7. For parent boundary faces, which only have an owner cell, reconstruction depends solely on the refinement status of that cell. This is effectively a simplified case of the internal face handling and is treated accordingly, as described in Appendix, Algorithm A8.

(4) Mesh postprocess

After reconstructing the mesh topology, post-processing of the mesh information is required, including basic attributes such as face area, cell volume, and face normal. These calculations involve straightforward mathematical operations and can be efficiently performed using GPU parallel algorithms based on faces or cells which are not detailed here.

2.3.4. *Variable field remapping*

After obtaining the newly refined mesh for the current time step, it is necessary to map the variable fields from the previous time step's mesh to the updated mesh, including the pressure p, velocity $\mathbf{u}$, and volume fraction α. During this process, cells in the parent mesh may fall into one

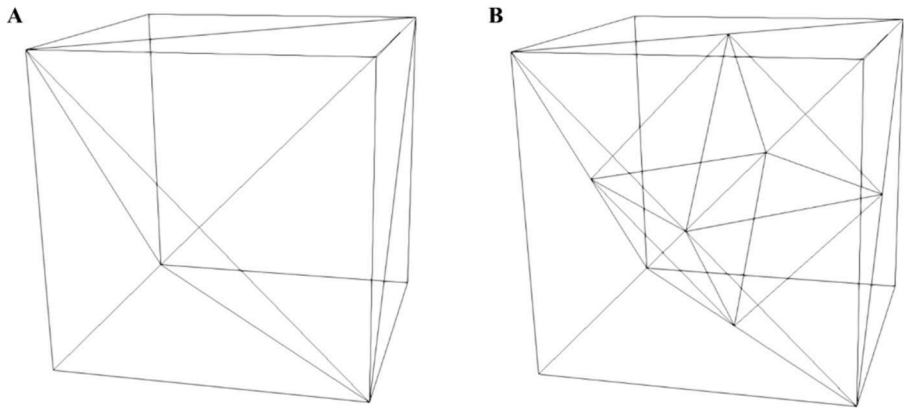


Fig. 16. Output of the minimal case demonstration code: (A) initial tetrahedral mesh; (B) refined mesh after applying AMR to the central tetrahedron. GitHub: https://github.com/TFluid/GPU_AMR_Tetrahedral.

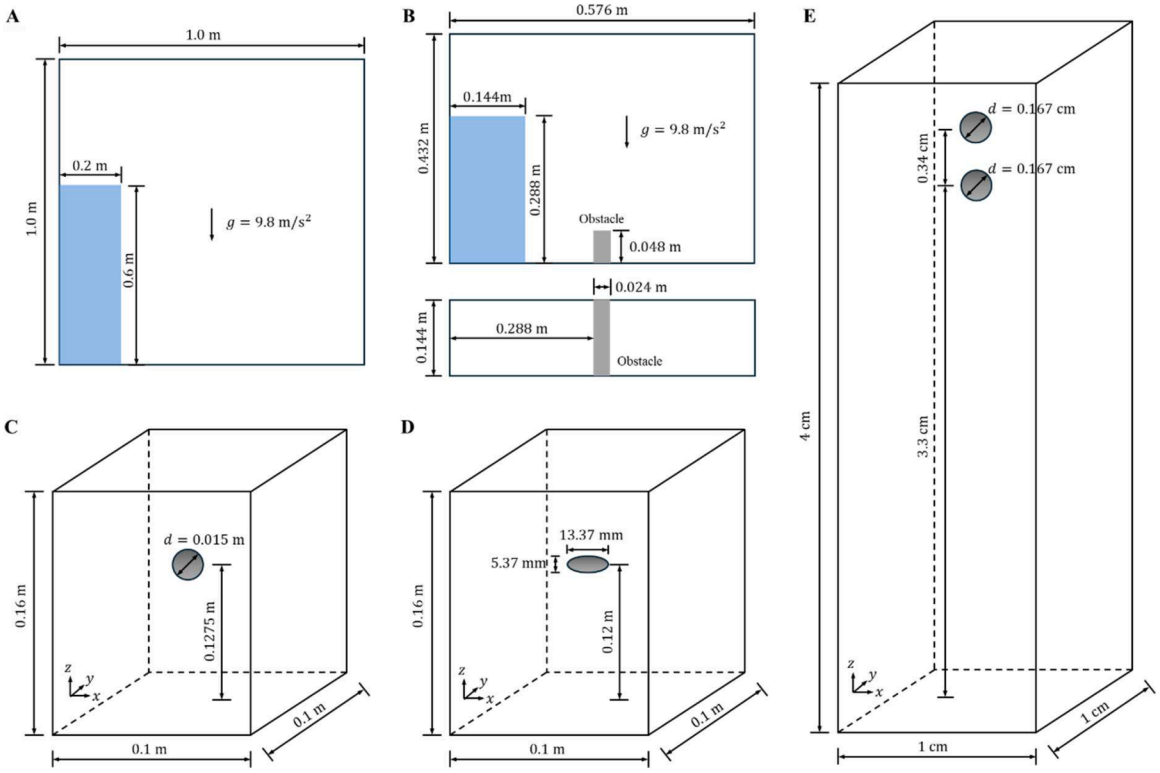


Fig. 17. Schematic diagram of five benchmark cases. (A) Benchmark I: 2D dam break; (B) Benchmark II: 3D dam break with obstacle [73]; (C) Benchmark III: settling of spherical particle [74]; (D) Benchmark IV: Settling of ellipsoidal particle [75]. (E) Benchmark V: Drafting-kissing-tumbling of two particles [43].

of four categories (Fig. 13): (1) the cell remains unrefined in both the previous and current time steps; (2) the cell was unrefined in the previous time step but becomes refined in the current time step; (3) the cell was refined in the previous time step but becomes unrefined in the current time step; (4) the cell remains refined in both the previous and current time steps.

The first and fourth cases are straightforward, as the mesh connectivity remains unchanged between time steps, allowing the solution variables to be directly copied. The third case requires averaging the values of all sub-cells to assign a representative value to the coarsened parent cell. The second case involves distributing the values from a previously unrefined parent cell to its newly created sub-cells.

Several interpolation strategies may be employed for this mapping task, such as uniform mapping, linear interpolation, or higher-order schemes. In this study, we adopt uniform mapping to preserve global conservation properties: each sub-cell inherits the same value as its parent cell from the previous time step. In addition to value mapping, index remapping is essential, since the global indices of cells occupying the same physical location may differ between time steps. To maintain consistency, we employ a parent-mesh-based indexing scheme that provides a unified reference framework across time steps. As detailed in the previous section, the refined mesh's global indices are derived from the parent mesh, supplemented by time-step-specific numbering lists for localization. Our GPU-parallel remapping algorithm traverses the parent mesh and utilizes the numbering lists (L_C^n and L_C^{n+1}) to locate and map corresponding parent and sub-cell indices. The complete remapping procedure, encompassing both index and value transfer, is provided in Algorithm A9 in the Appendix.

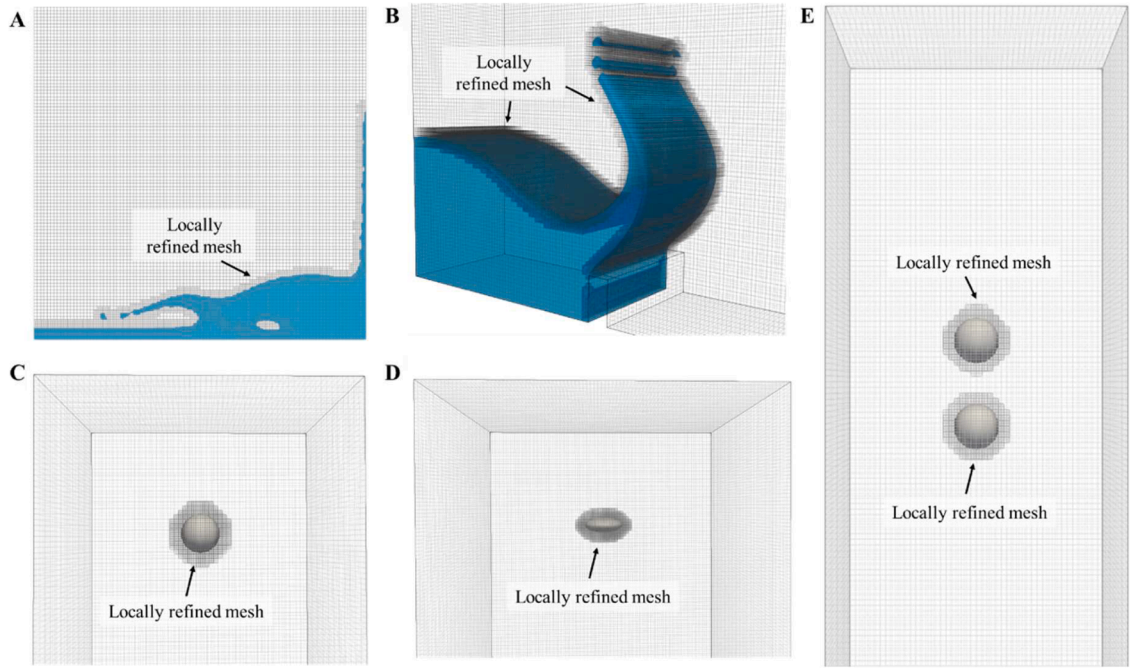


Fig. 18. Adaptive refined grids used in Benchmark I (A), Benchmark II (B), Benchmark III (C), Benchmark IV (D), and Benchmark V (E). Mesh refinement is applied at the two-phase interfaces and particle boundaries.

Table 1
Mesh information of five benchmark cases.

Benchmark index	Case index	Grid type	Mesh size	Mesh number
Benchmark I	Case 1	Uniform	10 mm	10,000
	Case 2	AMR	5–10 mm	11,700
	Case 3	Uniform	5 mm	40,000
Benchmark II	Case 1	Uniform	4.5 mm	400,672
	Case 2	AMR	2.25–4.5 mm	710,534
	Case 3	Uniform	2.25 mm	3164,160
Benchmark III	Case 1	Uniform	1.33 mm	675,000
	Case 2	Uniform	1.00 mm	1600,000
	Case 3	Uniform	0.833 mm	2822,400
	Case 4	AMR	0.667–1.33 mm	686,424
Benchmark IV	Case 1	Uniform	1.00 mm	1600,000
	Case 2	Uniform	0.667 mm	5400,000
	Case 3	Uniform	0.571 mm	8575,000
	Case 4	AMR	0.5–1.0 mm	1610,332
Benchmark V	Case 1	Uniform	0.2 mm	500,000
	Case 2	AMR	0.1–0.2 mm	510,150
	Case 3	Uniform	0.1 mm	4000,000

2.3.5. *Overall AMR procedure*

From Section 2.3.2–Section 2.3.4, a complete AMR algorithm fully implemented with GPU parallelism has been developed. Unlike traditional octree-based approaches, this algorithm minimizes the logical operations that GPUs handle less efficiently, with the most complex logic involving only the classification of four categories. Moreover, the proposed AMR framework is more versatile, requiring only the construction of a sub-mesh framework. Additionally, we introduced compressed refinement arrays similar to the CSR format to parallelize the refinement of grids, effectively avoiding the waste of GPU thread resources. Fig. 14 summarizes the workflow of the proposed GPU-accelerated AMR algorithm.

2.3.6. *Minimal example and implementation of AMR for tetrahedral grids*

To demonstrate the extensibility of the proposed AMR framework to general unstructured meshes beyond hexahedral grids, we present a simplified case based on a tetrahedral mesh. The simplified case consists of 8 vertices, 5 tetrahedral elements, and 16 faces, as illustrated in Fig. 15. Sub-mesh templating is applied to adaptively refine a parent tetrahedron using predefined subdivision patterns shown in Fig. 8. For ease of understanding and reproducibility, the implementation of this tetrahedral AMR case is made publicly available via GitHub.[1] As illustrated in Fig. 16, the demonstration code written in CUDA *C*++ performs adaptive refinement on the central tetrahedron, showing the transition from the initial mesh to the locally refined configuration. This example serves as a proof of concept that the proposed AMR algorithm can be extended to more general unstructured grids, laying the groundwork for future applications to polyhedral or hybrid meshes.

Unlike hexahedral grids, tetrahedral elements offer greater flexibility for representing complex geometries. Meanwhile, due to the relatively fewer points and faces involved in a tetrahedral cell, we selected this mesh structure for the demonstration code, which also facilitates readers' understanding of the algorithm. However, although the current AMR framework is capable of performing adaptive refinement on tetrahedral grids, generating high-quality refined meshes for tetrahedra, particularly in complex geometries, remains a significant challenge and warrants further investigation. Therefore, in the remainder of this study, we adopt hexahedral meshes, which allow for more controllable refinement quality and are more commonly used in resolved CFD-DEM simulations.

3. Benchmark and validation

Five benchmark cases are designed to rigorously evaluate the performance of the proposed GPU-accelerated AMR algorithm in coupled simulations of two-phase resolved CFD and DEM involving both spherical and non-spherical particles. Furthermore, two powder-based additive manufacturing processes, including the binder jetting and the laser powder bed fusion, are employed to further validate the applicability and robustness of the proposed method in complex engineering scenarios.

[1] https://github.com/TFluid/GPU_AMR_Tetrahedral

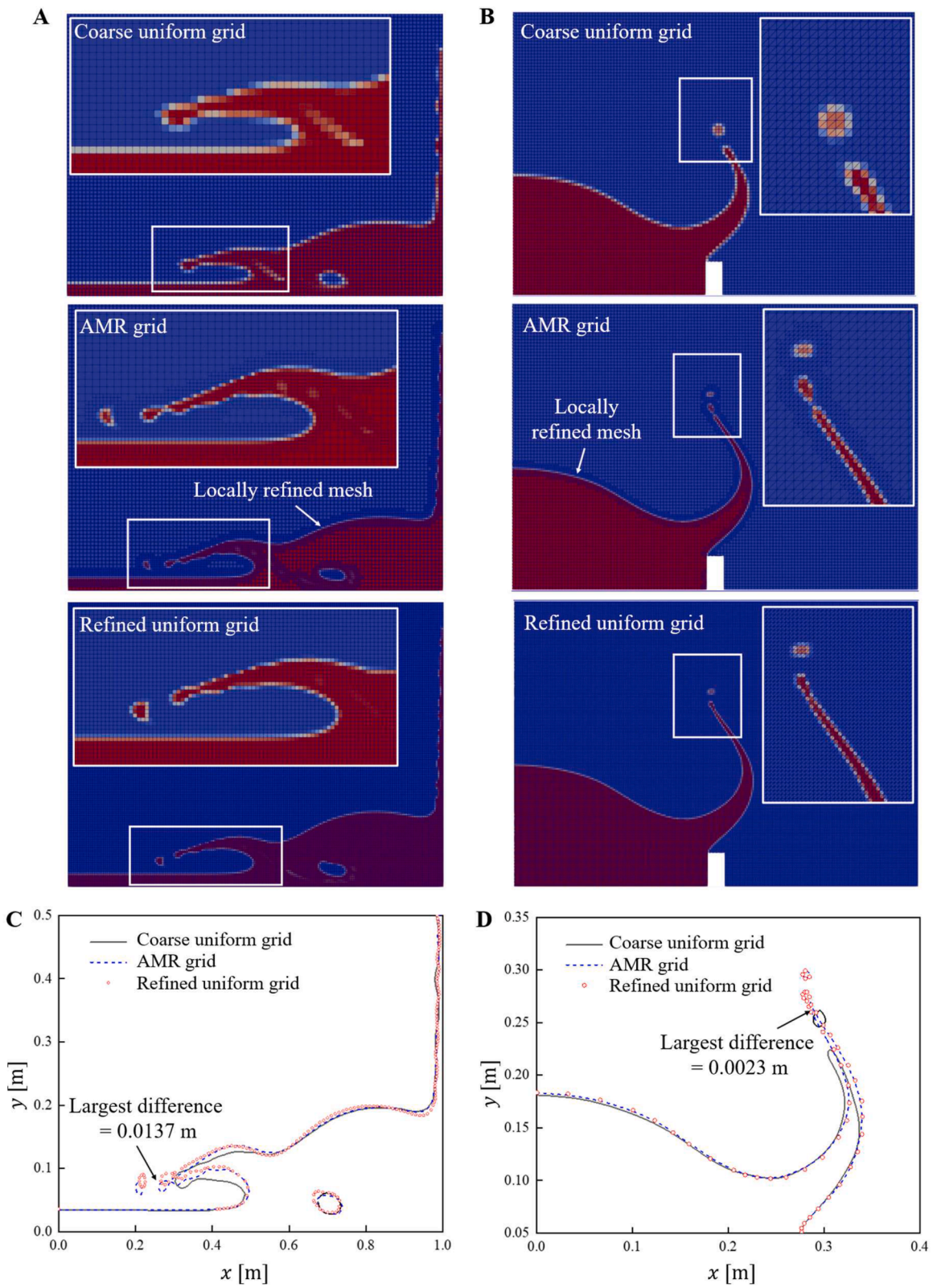


Fig. 19. (A and B) Volume fraction fields for Benchmark I (at 1.1 s) and Benchmark II (at 0.2 s), simulated using three grids, accompanied by their corresponding grid structures. Top row: Uniform coarse grid ($\Delta x_{\mathrm{I}} = 10$ **mm** for Benchmark I, $\Delta x_{\mathrm{II}} = 4.5$ **mm** for II), Middle row: Adaptive refined grid ($\Delta x_{\mathrm{I}} = 5 - 10$ **mm**, $\Delta x_{\mathrm{II}} = 2.25 - 4.5$ **mm**), Bottom row: Uniformly refined grid ($\Delta x_{\mathrm{I}} = 5$ **mm**, $\Delta x_{\mathrm{II}} = 2.25$ **mm**). Subscripts I and II indicate Benchmark I and Benchmark II, respectively. Red and blue colors represent water volume fraction of 1 (pure water) and 0 (pure gas), respectively. (C and D) Gas-liquid interface contours for Benchmark I (at 1.1 s) and Benchmark II at the $y = 0.072$ m section (at 0.2 s), comparing results from the three grid configurations.

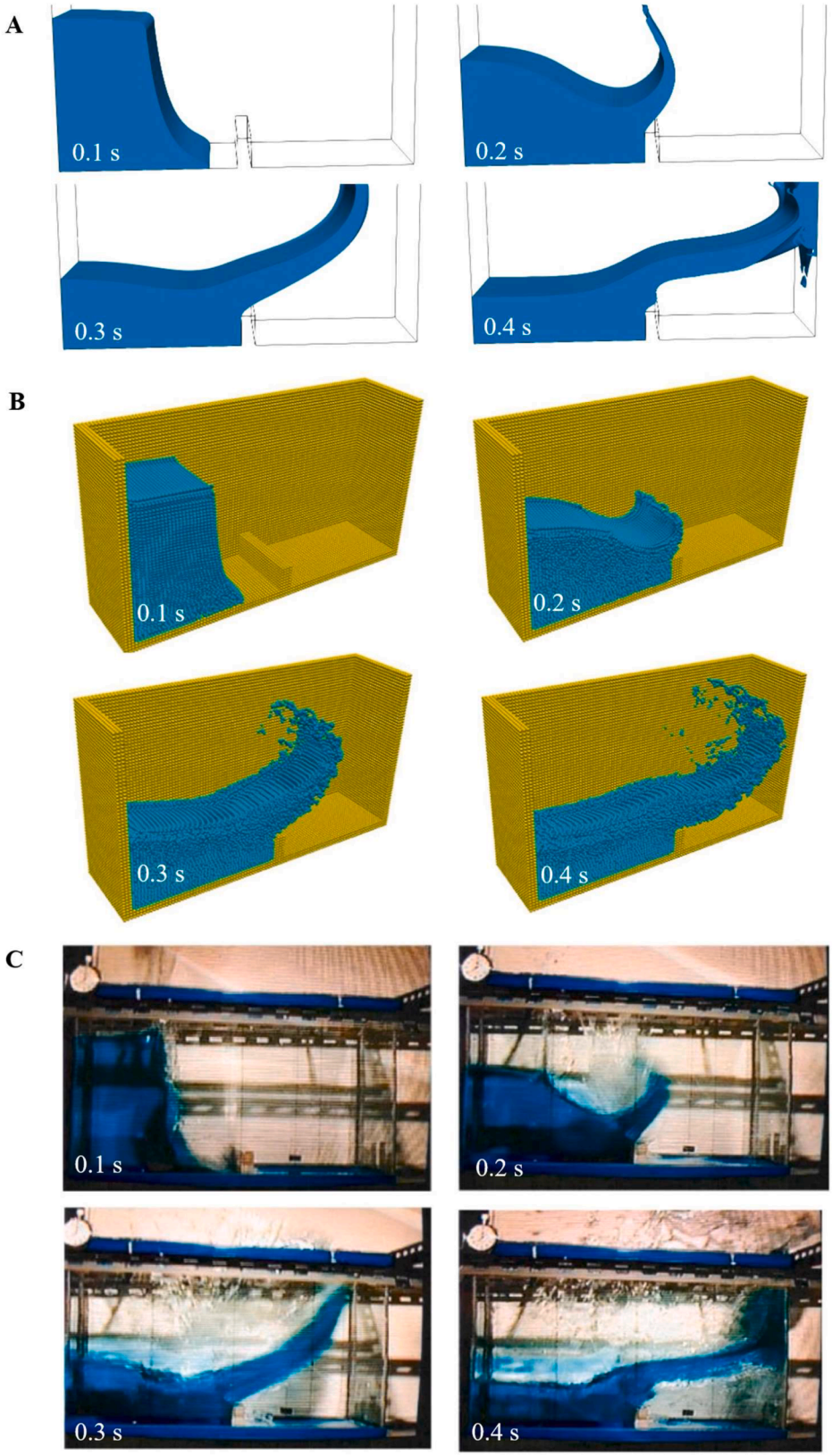


Fig. 20. Comparison of simulation results from this study (A), existing simulation results obtained by peridynamics [76] (B), and experimental results [73] (C) from 0.1 s to 0.4 s.

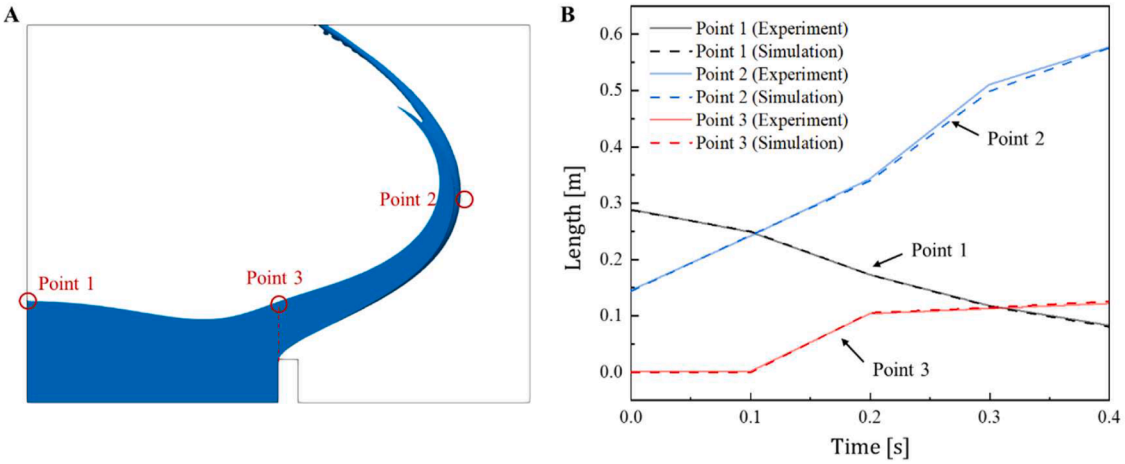


Fig. 21. (A) Locations of the three probe points on the water-air interface used for quantitative comparison. Point 1: the leftmost point; Point 2: the rightmost point; Point 3: the point above the left edge of the obstacle. (B) Comparison between experimental [73] and numerical results at the three points. Point 1 and Point 3 are evaluated based on vertical height, while Point 2 is evaluated based on horizontal displacement.

3.1. Model setup of five benchmark cases

Five common benchmark cases shown in Fig. 17 are designed to rigorously evaluate the performance of the proposed GPU-accelerated AMR algorithm for coupled simulations of two-phase resolved CFD and DEM with spherical/non-spherical particles. These cases include: (I) a two-dimensional dam-break problem, serving as a baseline to validate interfacial sharpness and mass conservation in multiphase flow; (II) a three-dimensional dam-break scenario with an obstacle according to [73], designed to assess geometric adaptivity and dynamic refinement in complex domains; (III) the settling of a spherical particle [74], used to quantify the accuracy of fluid–particle coupling and drag resolution; (IV) the settling of an ellipsoidal particle [75], extending the validation to non-spherical particles by incorporating torque and orientation effects; and (V) the drafting-kissing-tumbling (DKT) motion of two particles [43], used to validate both fluid-particle and particle-particle interactions. Benchmarks I and II are employed to validate the two-phase AMR CFD only, while the other three benchmarks are used to validate CFD-DEM coupling.

Benchmarks I and II represent a similar dam-break problem conducted with identical physical properties but with different computational domain sizes and obstacle presence (Fig. 17A and B). Both cases adopt the following physical properties: the density and kinematic viscosity of water are 1000 kg/m^3 and 10^{-6}m^2/s, respectively, while those of air are 1 kg/m^3 and 1.48 $\times$ 10^{-5}m^2/s. To assess mesh sensitivity, both cases are simulated using three types of computational grids: a coarse uniform grid, a refined uniform grid, and an adaptively refined grid based on the coarse mesh, as shown in Fig. 18A. The mesh resolutions for Benchmark I are $\Delta x = 10$ mm and 5 mm for the coarse and refined grids, respectively, while Benchmark II uses finer resolutions of $\Delta x = 4.5$ mm and 2.25 mm (Fig. 18B).

Benchmark III and Benchmark IV share identical computational domain size (Fig. 17C and D) but differ in fluid properties and particle shapes. In Benchmark III, the fluid has a density of 962 kg/m^3 and a kinematic viscosity of 1.175 $\times$ 10^{-4}m^2/s. The diameter of the spherical particle is 15 mm and the particle Reynolds number is 11.6. In Benchmark IV, the fluid density is 1120 kg/m^3 and the viscosity is 6.67 $\times$ 10^{-6}m^2/s. The ellipsoidal particle measures 13.37 mm along both the x- and y-axes, and 5.37 mm along the z-axis. Both benchmarks are evaluated using four mesh configurations: three uniform grids and one adaptively refined grid (Fig. 18C and D). For Benchmark III, the uniform grid resolutions are 1.33 mm (Case 1), 1.00 mm (Case 2), and 0.833 mm (Case 3). These grid sizes correspond to approximately 11.3, 15.0, and 18.0 cells per particle diameter (15 mm), respectively. For Benchmark IV, the corresponding grid resolutions are 1.00 mm (Case 1), 0.667 mm (Case 2), and 0.571 mm (Case 3). These values yield approximately 13.4, 20.0, and 23.4 grid cells per particle length in the x/y directions, and 5.4, 8.1, and 9.4 cells along the z-axis, respectively, for an ellipsoidal particle with dimensions of 13.37 mm $\times$ 13.37 mm $\times$ 5.37 mm. This level of spatial resolution ensures an accurate representation of both spherical and ellipsoidal particle geometries, which is essential for resolving fluid-particle interactions with high fidelity. In both cases, the adaptive mesh is generated based on the coarsest uniform grid as the parent. The finer mesh in Benchmark IV is due to the smaller thickness of ellipsoidal particles compared to the spherical ones in Benchmark III, requiring higher spatial resolution.

For Benchmark V, the two particles possess identical properties, each with a density of 1140 kg/m^3. Both particles are fully submerged in water, with fluid properties identical to those used in Benchmark I. The dimensions of computational domain and particles are shown in Fig. 17E, and the mesh configuration of the adaptively refined grid is presented in Fig. 18E. The spherical particle has a diameter of 16.7 mm. The uniform refined grids are configured with resolutions of 0.1 mm and 0.2 mm, corresponding to particle-to-grid size ratios of 16.7 and 8.35, respectively. Detailed mesh configurations for the five benchmark cases are summarized in Table 1.

3.2. Benchmarking analyses

(1) Benchmarks I and II: dam break without and with barrier

Fig. 19A and B shows the volume fraction field of water and the corresponding grid for Benchmark I at 1.1 s and Benchmark II at 0.2 s, where the adaptive refinement is applied only to grids around the air-water interface in the CFD simulations. Evidently, the coarse uniform grid lacks the necessary resolution to resolve the detailed spatter effects. Fig. 19C and D further presents the comparison of fluid contour profiles for Benchmark I at 1.1 s and Benchmark II at 0.2 s. The maximum differences in the air-water interface profiles between the adaptive refined grid and the uniformly refined mesh are 0.0137 m for Benchmark I and 0.0023 m for Benchmark II, corresponding to 1.37% and 0.6% of the computational domain size, respectively. It can be demonstrated that the results obtained using AMR closely match those of the refined uniform grid case, providing a sharper air-water interface.

Fig. 20 further compares the obtained simulation results of Benchmark II with the existing simulation results obtained by peridynamics [76] and experimental results [73] from 0.1 s to 0.4 s. The results show that the outcomes obtained using the uniform refined grid in this study

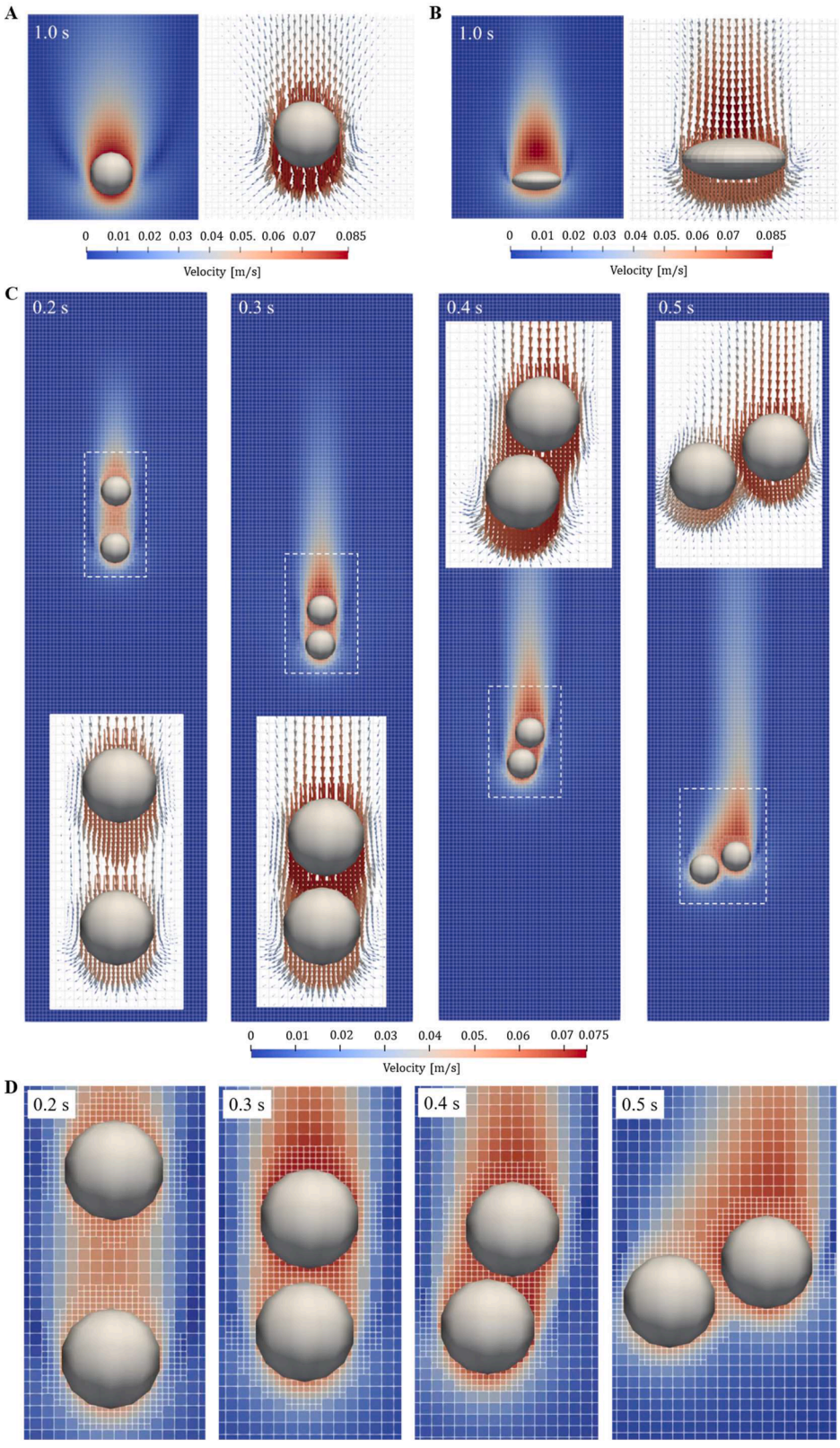


Fig. 22. Velocity magnitude and vector fields for the adaptive mesh cases of Benchmark III (A), Benchmark IV (B), and Benchmark V (C). Subfigure (D) shows a close-up view of the locally refined region in Benchmark V (C), highlighting the adaptive refinement near the particles. White lines represent the mesh structure. The denser lines around particles indicate locally refined regions due to adaptive mesh refinement.

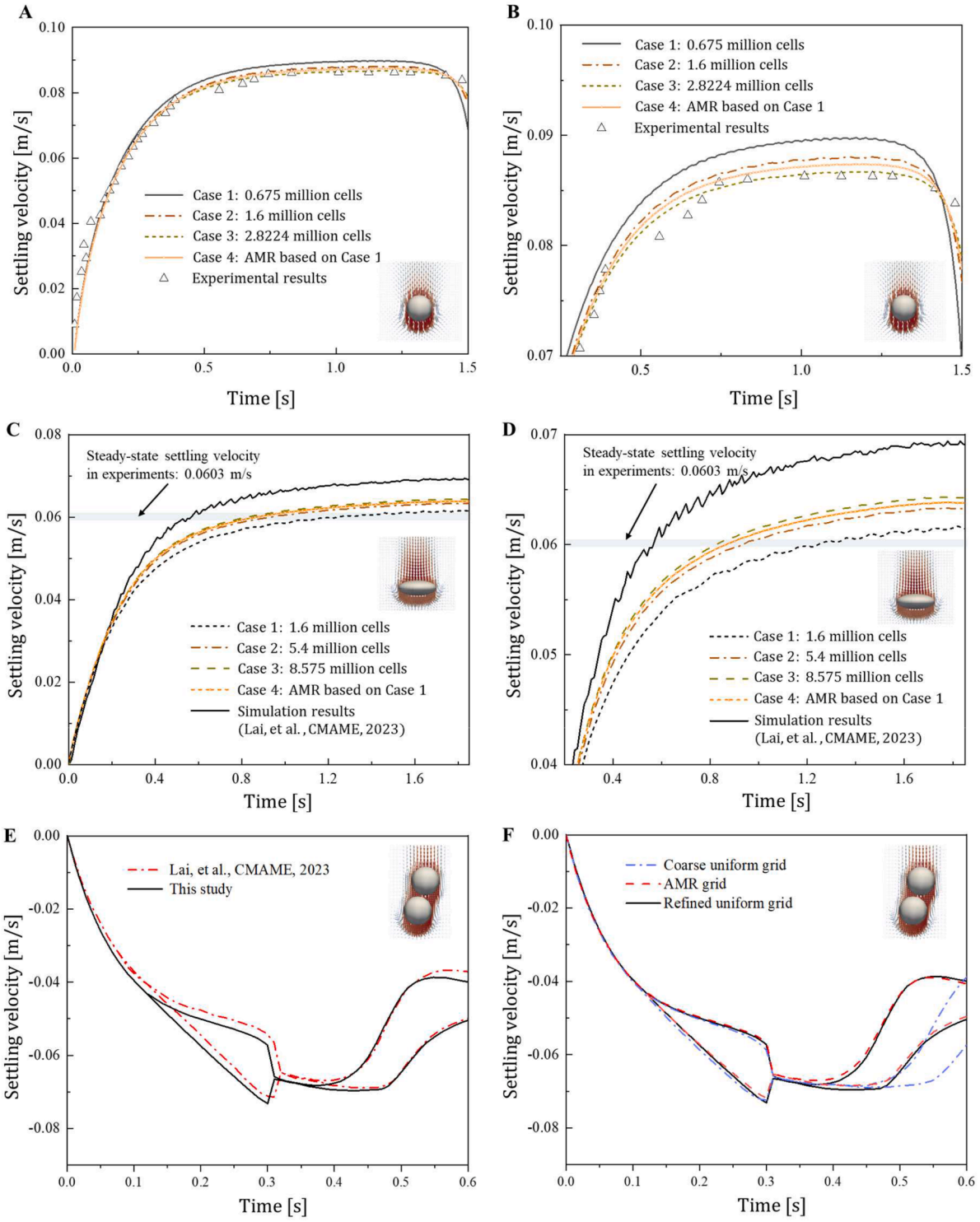


Fig. 23. Comparative analysis of simulation accuracy across Benchmarks III-V. (A-D) Comparison of simulation results for Benchmark III (A, B) and Benchmark IV (C, D), including experimental data [74,75], prior simulations [74,75], results from three uniform grids (Cases 1–3), and an AMR grid (Case 4). Panels A and C show full time histories, while B and D provide zoomed-in views for detailed comparison (0.25–1.5 s for B; 0.2–1.6 s for D). (E) Validation of Benchmark V (refined uniform grid) against existing simulation results [43]; (F) Grid resolution study for Benchmark V using two uniform grids and one AMR grid.

are in high agreement with the experimental results and exhibit higher accuracy compared to those obtained with peridynamics. Together with Fig. 19, these comparisons verify the performance of the proposed AMR approach by showing that it achieves results comparable to those obtained with uniform refinement. Fig. 21 further presents a quantitative comparison of the vertical height and horizontal displacement at three selected points. The maximum deviation among these three points between the experimental and numerical results is approximately 9.9 mm, accounting for only 2.3% of the total domain height, which falls within an acceptable error range. The root mean square error (RMSE) obtained from the data in Fig. 21B is 3.93 mm, which corresponds to approximately 0.91% of the domain height. This figure demonstrates the accuracy of the two-phase CFD algorithm with full GPU parallelization employed in this study.

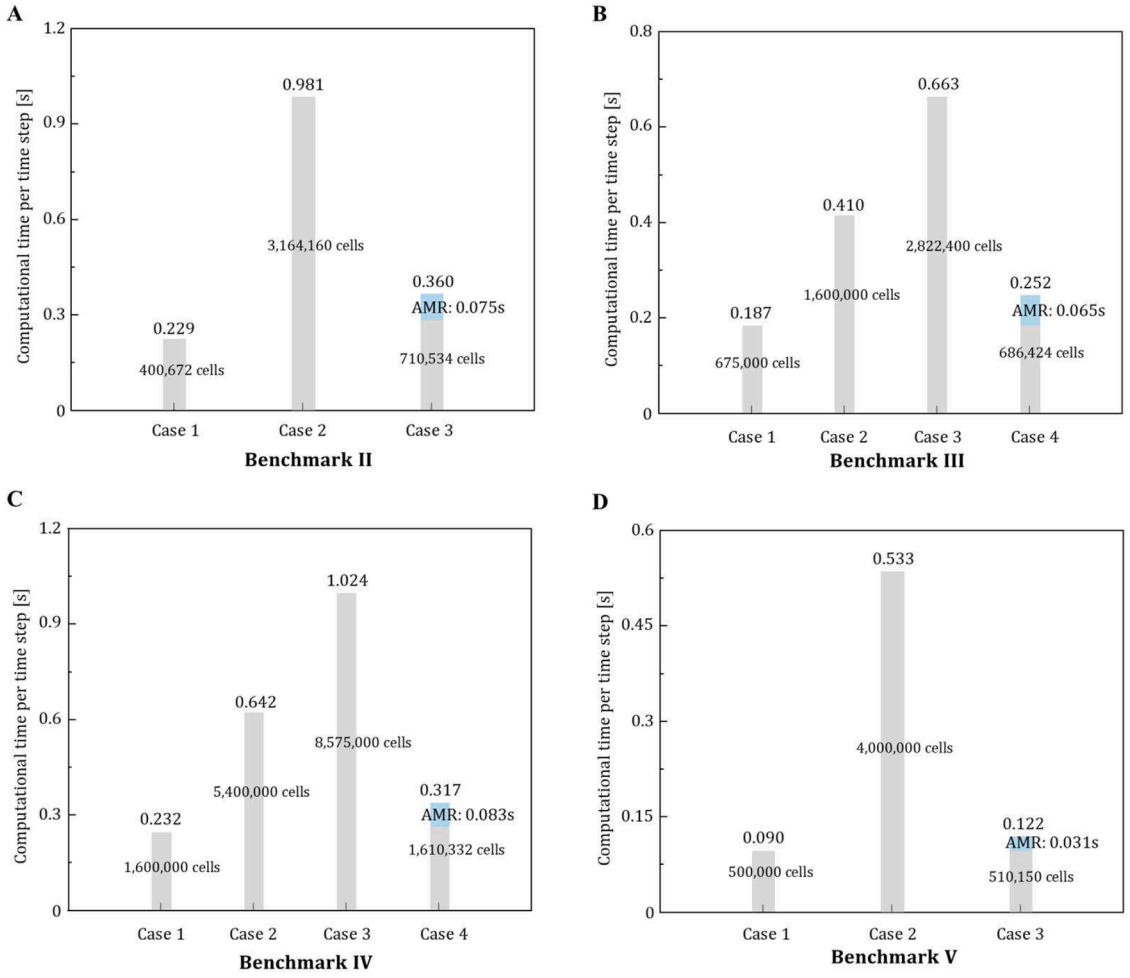


Fig. 24. Averaged computational time per time step for Benchmarks II to V with various grids. Since the two-phase cases involve fewer computational cells, simulations were performed on an RTX 3060 Ti GPU. In contrast, the coupled CFD-DEM cases require significantly more computational resources due to larger grid sizes and thus were tested on an RTX 4070 Ti Super GPU. The detailed mesh configurations for the benchmark cases can be found in Table 1.

(2) Benchmarks III-V on particle-fluid coupling

This section presents a quantitative evaluation of simulation accuracy against experimental data for three standard benchmark cases simulated using the coupled CFD-DEM approach. Fig. 22 visualizes the velocity fields, near-particle velocity vectors, and mesh structures (indicated by white lines) across all three Benchmarks. The results demonstrate adaptive mesh refinement around the particles, with the surrounding velocity field transitioning smoothly without introducing discontinuities arising from mesh resolution variations. A notable feature in Fig. 22C is the clear depiction of the classical drafting, kissing, and tumbling (DKT) phenomena observed between two settling particles, further validating the model's capability to resolve complex particle-fluid interactions.

Fig. 23 (A and B) compares the simulation results of four grid resolutions for Benchmark III with the experimental data reported in [74]. As the mesh is refined, the numerical results exhibit improved agreement with the experimental observations, with Case 3 showing excellent consistency. The simulation results for Case 4, which employs adaptive mesh refinement (AMR), fall between those of Case 2 and Case 3. Fig. 23 (C and D) present a similar comparison for Benchmark IV, including experimental data [75] and reference simulation results [43]. Evidently, the present study achieves a closer match with the experimental measurements than the reference simulations do. Compared to spherical particles, the terminal velocity of non-spherical particles exhibits a slightly larger deviation from the experimental results, but the error remains within approximately 6%, demonstrating that the proposed algorithm maintains sufficient accuracy. This discrepancy may stem from measurement uncertainties or slight geometric deviations in the particle shape. Nonetheless, the possibility that further optimization of the algorithm may be required cannot be entirely excluded. Additionally, for the AMR grid (Case 4) and the fully refined grid (Case 3), the errors in terminal velocity for Benchmark III and Benchmark IV are as low as 0.8% and 0.77%, respectively, demonstrating excellent quantitative agreement with the reference solutions.

Fig. 23 (E and F) further validates the accuracy of the simulation results for the drafting-kissing-tumbling (DKT) behavior of two particles. To facilitate quantitative comparison with the literature, the time of particle contact is defined as the instant when the settling velocity changes abruptly. Compared to the work of Lai et al. [43], the time of particle contact differs by 3.3% (0.30 s vs. 0.31 s), and the velocity at collision differs by 1.6% (0.07257 m/s vs. 0.07139 m/s), which are within an acceptable range of numerical errors. The results obtained using three different grid types indicate that the predicted collision time and velocity are highly consistent, with errors within 1%. However, due to the limited number of cells, the simulation using the coarse uniform grid deviates significantly in the final particle separation state compared to the results from the other two grids.

3.3. *Numerical accuracy and computational performance*

This section provides a comprehensive evaluation of the proposed

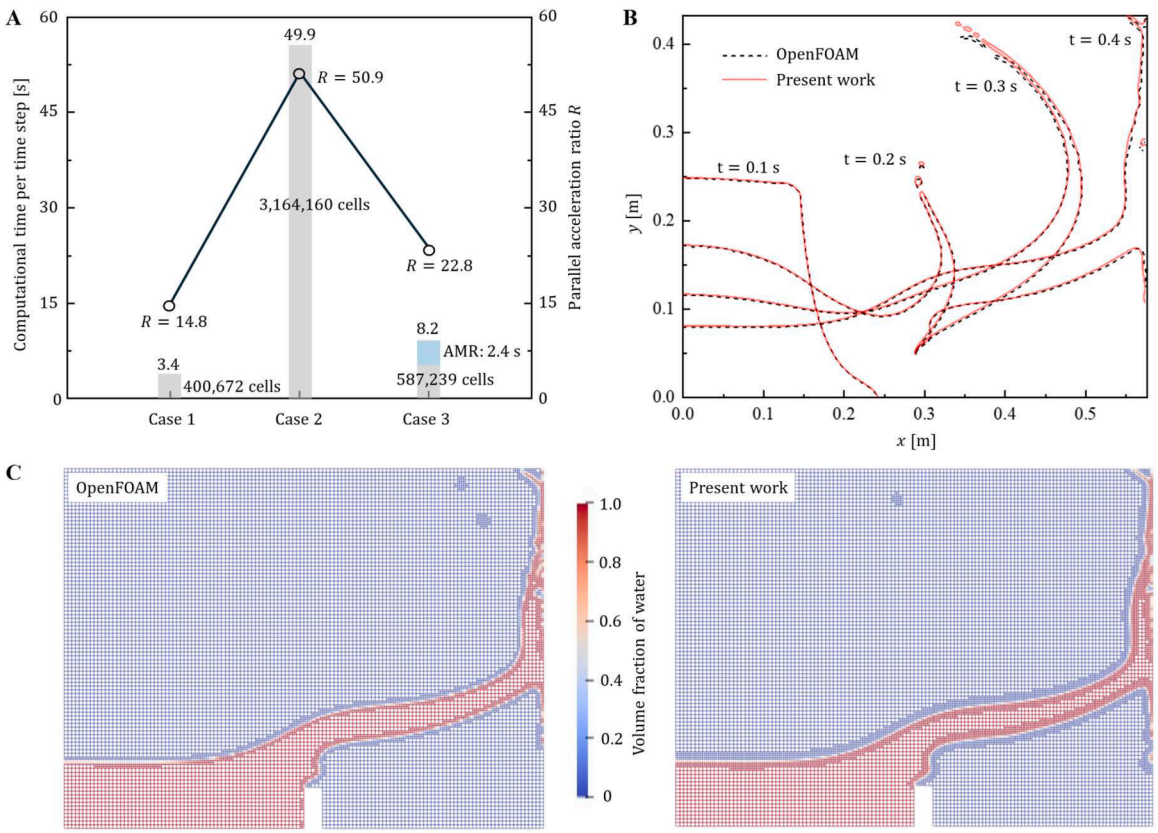


Fig. 25. Benchmark II: (A) Computational time per time step at $\mathbf{t} = 0.3$ s obtained using OpenFOAM, along with the corresponding GPU acceleration ratio relative to the result shown in Fig. 24A; (B) Comparison of fluid contours between OpenFOAM and the present study, both using adaptive mesh refinement with identical resolution; (C) Water volume fraction and corresponding mesh distribution at $\mathbf{t} = 0.4$ s obtained from OpenFOAM and the present study. The simulations were performed on an RTX 3060 Ti GPU and a single-core Intel i5–13600KF CPU.

GPU-accelerated AMR algorithm in terms of numerical accuracy and computational efficiency. Discrepancies in *numerical accuracy* observed between adaptively refined meshes and uniformly fine meshes are attributed to three factors: (1) Refinement scope limitations: The current AMR strategy focuses on refinement near fluid-fluid and particle-fluid interfaces, but may overlook dynamically critical regions (e.g., high velocity gradients). Future implementations could integrate refinement criteria based on additional physical indicators to enhance solution fidelity. (2) Geometric errors at interfaces: Non-orthogonal cell geometries at coarse-fine grid boundaries introduces residual numerical errors, even with non-orthogonal correction schemes. (3) Interpolation scheme constraints: First-order interpolation for field remapping adopted in current implementation may reduce precision; future work may explore higher-order interpolation methods to mitigate this limitation.

As for computational performance, Fig. 24 benchmarks the efficiency of the AMR algorithm across varying problem complexities (Benchmarks II-V). Benchmark I, omitted due to its small mesh size, served solely for accuracy validation. In all four cases, the AMR approach consistently reduces total computational time compared to uniformly refined grids while maintaining comparable accuracy. Notably, the AMR module accounts for only 21–26% of the computation time per time step, which is significantly lower than the 35% typically required in CPU-based AMR implementations [38]. This highlights the enhanced parallel efficiency of our GPU-accelerated algorithm. Across different hardware configurations (RTX 3060Ti and 4070Ti Super), the AMR algorithm also demonstrates robust performance regardless of cell count or refinement region size. Compared to refined uniform grids, AMR achieves equivalent accuracy at 23–61% of the computational cost, corresponding to speedup ratios of 1.63–4.37. These results confirm the method's efficiency and scalability, though the achievable acceleration depends heavily on the fraction of the domain requiring refinement.

To further validate the computational efficiency and accuracy of the proposed GPU-AMR method, we reproduced Benchmark II using OpenFOAM, one of the most widely adopted open-source CFD software packages [77]. Based on the model setup shown in Fig. 17B, a simplified modification of OpenFOAM's standard test case "*damBreakWithObstacle*" was performed to match the configuration of Benchmark II. For a fair comparison, this OpenFOAM simulation was conducted using a single core of an Intel i5–13600KF CPU, in order to avoid the load imbalance issues typically introduced by AMR in multi-core environments.

In addition, it is important to clarify why a pure-fluid benchmark was selected for this comparison. The computational efficiency of resolved CFD-DEM simulations is overwhelmingly dominated by the CFD part: for instance, the grid resolution requirement (cell size $< 1/10$ of particle diameter) [42] leads to approximately 1000 cells per particle volume, and in a naturally packed bed of 200 particles, at least one million fluid cells are required, which is two orders of magnitude more than the number of particles. Moreover, since the CFD equations are solved implicitly while the DEM is solved explicitly, the computational cost of the DEM typically represents less than 5% of the total runtime. Since the AMR framework primarily affects the number of fluid cells and thus the CFD solver performance, while DEM calculations are mesh-independent, focusing on the CFD part provides a clearer and fairer assessment of the computational efficiency of the proposed AMR method.

Fig. 25A presents the averaged computational time per time step at $t = 0.3$ s obtained using OpenFOAM, along with the GPU acceleration ratio calculated with respect to Fig. 24A, defined as $= t_{CPU}/t_{GPU}$. As the number of mesh cells increases, the acceleration ratio rises rapidly, reaching approximately 51 times at 3.16 million cells. Notably, since

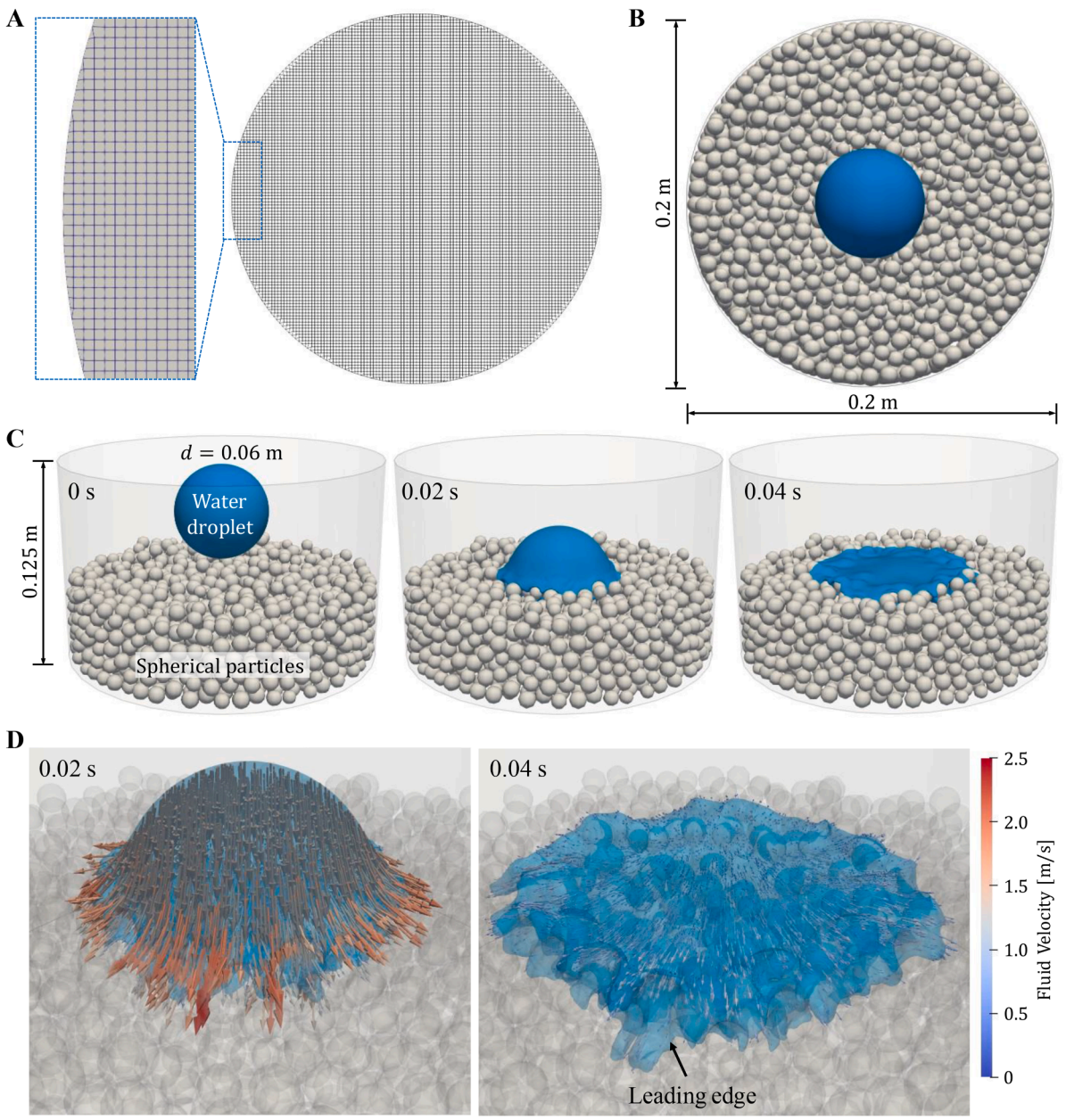


Fig. 26. Computational domain and simulation results: (A) Mesh of the cylindrical domain generated using the cut-cell method. (B-C) Schematic illustration of the computational domain dimensions and simulation results at 0 s, 0.02 s, and 0.04 s. (D) Velocity fields of the water droplet at 0.02 s and 0.04 s. The simulation results are obtained with the AMR scheme.

this study focuses on the performance of the AMR algorithm, we further analyzed the fraction of AMR-related runtime. In this benchmark, the AMR procedure in OpenFOAM accounts for 29.3% of the total time per step, which is about 41% higher than the 20.8% observed with our proposed method, demonstrating the superior efficiency of the GPU-AMR implementation.

Fig. 25B shows the comparison of fluid contours at four time instances, using identical mesh resolution with AMR. The results indicate an excellent match in the majority of the flow domain, with only minor differences observed in the splash regions. The maximum deviation between the fluid-gas interface contours from the two simulations is 9.2 mm, which corresponds to only 2.1% of the total domain height. Fig. 25C further illustrates the water volume fraction and the corresponding mesh distribution at $t = 0.4$ s obtained from both OpenFOAM and the present study. Due to different refinement criteria, OpenFOAM adopts a threshold of $\alpha \in [0.001,\ 0.009]$, whereas our method uses $|\nabla\alpha| > 0.001/\Delta x$ with Δx denoting local mesh size. As a result, the adaptively refined region in our simulations tends to be slightly larger than that in OpenFOAM. Taken together, these comparisons provide strong evidence for the high accuracy and computational efficiency of the proposed GPU-AMR algorithm.

Performance profiling indicates that over 90% of the total computation time is consumed during the grid topology reconstruction phase, which involves operations such as conditional branching, atomic additions, and inclusive scans-tasks that are inherently less efficient on GPU architectures due to limited parallelism. Furthermore, several GPU-specific optimization strategies, including memory coalescing and shared memory utilization, have not yet been integrated into the current implementation. As such, there remains substantial room for further performance enhancement.

In terms of memory utilization, simulations using uniform grids require an average of 2 GB of GPU memory per million cells. In contrast,

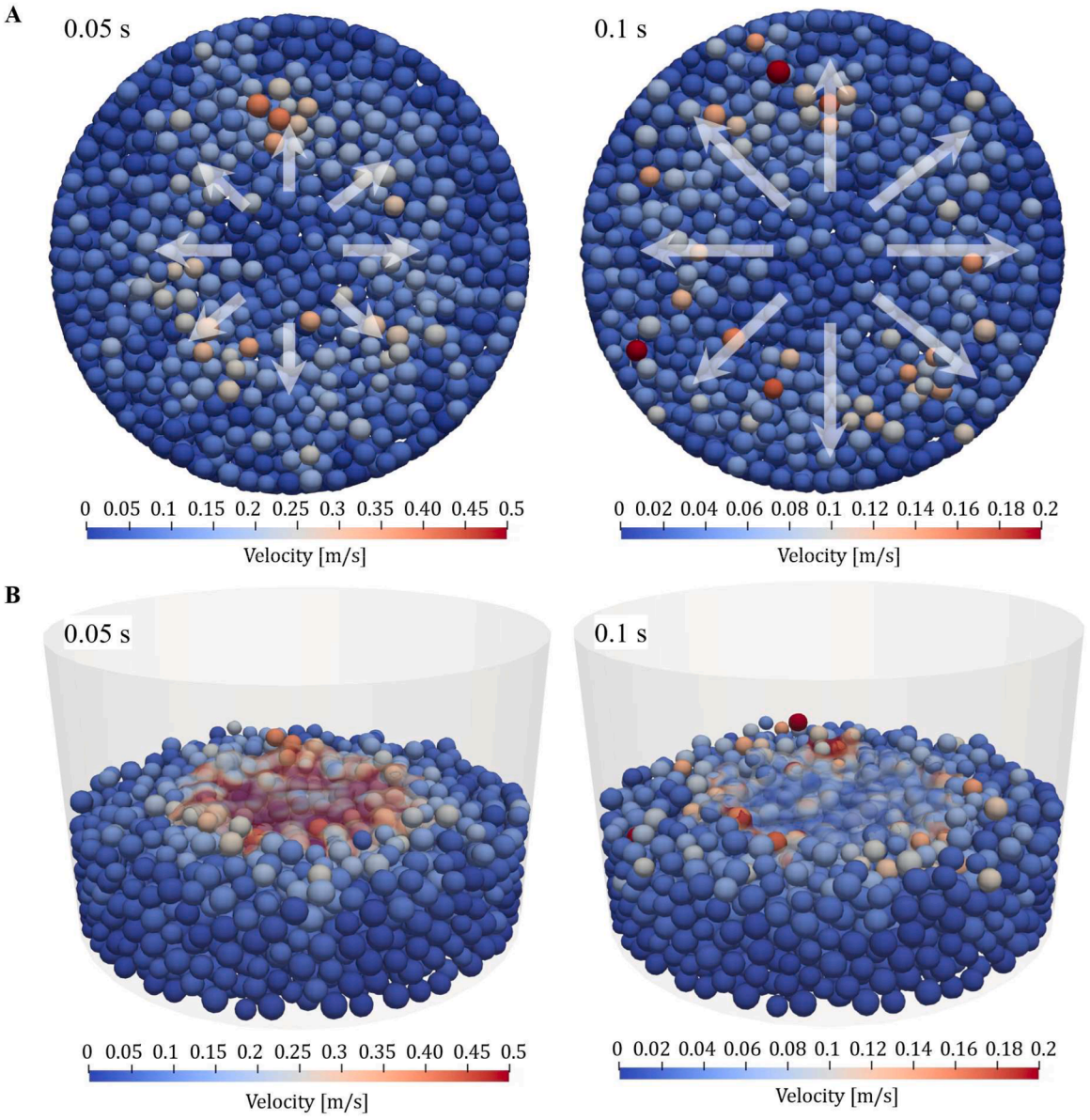


Fig. 27. (A) Top view of particle distribution and its velocity field at 0.05 s and 0.1 s; (B) Distribution of water and particles with the corresponding velocity field at 0.05 s and 0.1 s. The simulation results are obtained with the AMR scheme.

the present AMR implementation consumes approximately 3.1 GB per million cells, representing a 55% increase (1.1 GB additional memory per million cells) compared to uniform grid simulations. The additional memory consumption primarily arises from the need to store not only the background mesh and the previous time-step mesh but also the current refined mesh during grid information mapping. Furthermore, variable field remapping operations necessitate retaining field variables, such as velocity, pressure, volume fraction, and fluxes on the previous mesh. Future work could mitigate this through dynamic memory management strategies to reduce the AMR memory footprint.

The goal of dynamic memory management strategies is to alleviate pressure from temporary memory peaks, which often occur during key operations such as AMR and linear system solves. For example, when solving the linear systems using algebraic multigrid (AMG) methods, the construction of multilevel coarse grids and temporary residual vectors requires substantial memory allocation. Similarly, during AMR procedures, intermediate data structures are temporarily introduced to support mesh adaptation and refinement operations. To address this, manually manage memory allocation and deallocation for auxiliary mesh-related variables can decrease the peak memory usage. Specifically, variables required only during the AMR generation phase are explicitly released immediately after mesh construction, and memory is reallocated only when needed prior to the next AMR operation. This deliberate strategy of memory reuse and early deallocation helps reduce the GPU's peak memory load, thereby enabling larger-scale simulations on limited-memory hardware. Future work may further improve robustness and scalability by automating these strategies and integrating more advanced memory management policies, such as memory pooling or streaming techniques.

3.4. Limitations of the current AMR implementation

While the proposed AMR framework demonstrates substantial computational advantages, several limitations remain that warrant further development:

(1) High memory consumption: As discussed in the preceding section, the current AMR implementation incurs significant GPU

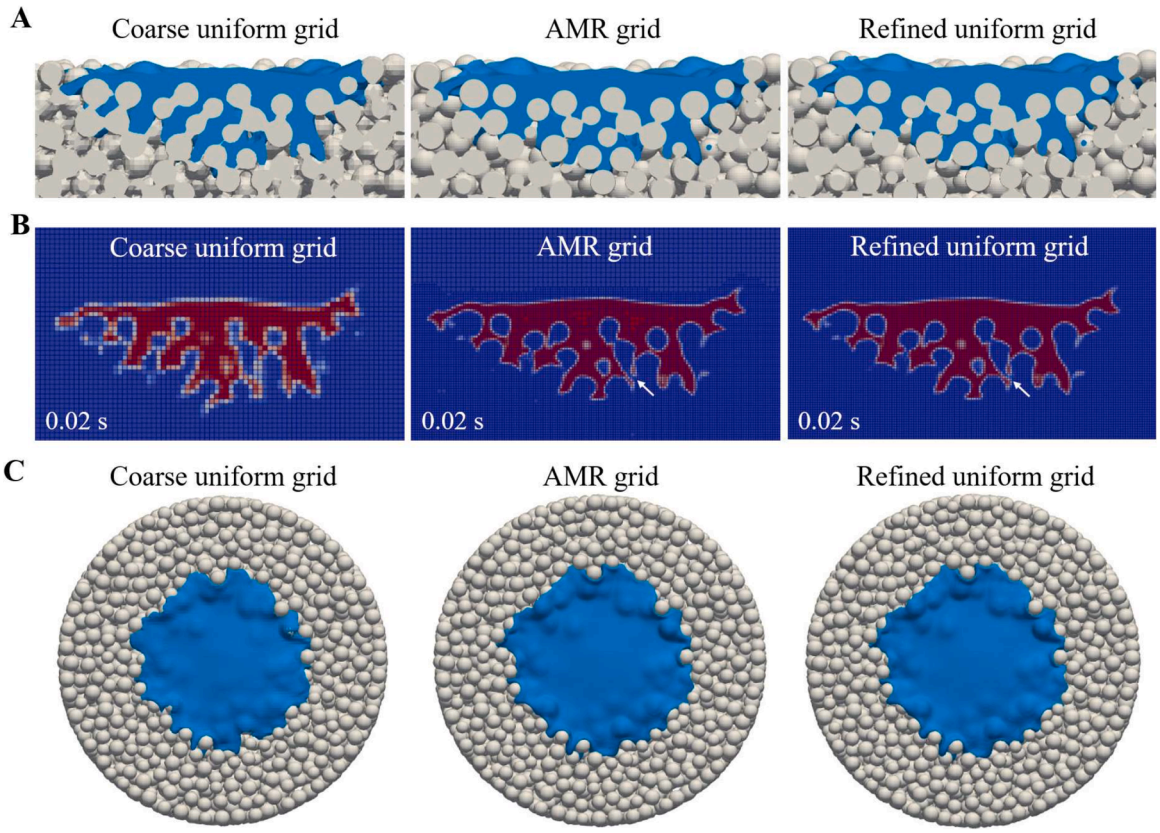


Fig. 28. Comparative study of simulation results using different grids at 0.02 s. (A) Half-section view of the particle and water distribution. (B) The grid structure and fluid volume fraction field of the half section. (C) Distribution of particles and water. Left: Uniform coarse grid, Middle: Adaptive refined grid, Right: Uniform refined grid. The red and blue colors represent a volume fraction of 1 and 0 for water, respectively.

memory overhead, with a usage of approximately 3.1 GB per million cells. On a 32 GB NVIDIA RTX 5090 GPU, this restricts the maximum feasible mesh size to roughly 10 million cells, thereby limiting its applicability in large-scale simulations. Reducing peak memory usage through more efficient memory management strategies remains an urgent challenge.

(2) Single-GPU restriction: At present, the AMR framework is designed for single-GPU execution and does not yet support multi-GPU parallelism. Although multi-GPU computing offers a potential path to overcome memory constraints, achieving efficient inter-GPU scalability and data distribution requires further algorithmic innovations and communication optimization.

(3) Mesh anisotropy and interface treatment: Under conditions of extreme mesh anisotropy, additional care must be taken to construct appropriate sub-mesh templates that minimize mesh distortion and non-orthogonality in the refined regions. Furthermore, at the interface between refined and unrefined regions, the current method does not yet provide an effective strategy to handle geometric discontinuities or non-orthogonal faces, which may degrade numerical accuracy. Future efforts should focus on robust interface treatments to address this issue.

4. Illustrative simulations of complex additive manufacturing processes

This section presents the application of our fully GPU-parallelized two-phase CFD-DEM solver integrated with the proposed GPU-optimized AMR technique to simulate two representative powder-based additive manufacturing processes: binder jetting and laser powder bed fusion (LPBF). The developed methodology demonstrates apparent advantages for these applications due to several inherent characteristics of the physical processes and computational requirements.

Specifically, the proposed method is particularly well suited to the simulation demands of such processes for the following three reasons. First, the complex powder-bed and two-phase flow interactions characteristic of both binder jetting and LPBF processes demand substantial computational resources. In this context, the additional computational cost introduced by AMR is relatively minor, while the resulting improvements in efficiency are significant. Second, the simulation domains associated with binder jetting and LPBF are typically geometrically regular, enabling seamless integration of the orthogonal grid-based AMR algorithm developed in this study. Third, the regions of physical interest are often spatially localized, for instance, the binder-saturated zones in binder jetting or the melt pool in LPBF, making localized mesh refinement via AMR especially effective for capturing fine-scale dynamics without incurring excessive computational cost.

4.1. Binder jetting

Given the inherent complexity of fluid dynamics in binder jetting processes, including non-Newtonian rheology and phase change phenomena, the present study employs a simplified model to facilitate development and evaluation of the AMR methodology. The jetting process is modeled as a water droplet with a dropping velocity of 2 m/s impinging on a 5.5 cm-thick bed of spherical particles (Fig. 26). The particle bed comprises three distinct particle sizes, 4 mm, 4.5 mm, and 5 mm in radius, with a uniform density of 1200 kg/m^3. These particles are distributed by number in proportions of 20% (4 mm), 40% (4.5 mm), and 40% (5 mm), respectively, summing to 100% of the total particle count. Fluid properties for both water and air maintain consistency with previous benchmark cases. While simplified, this configuration

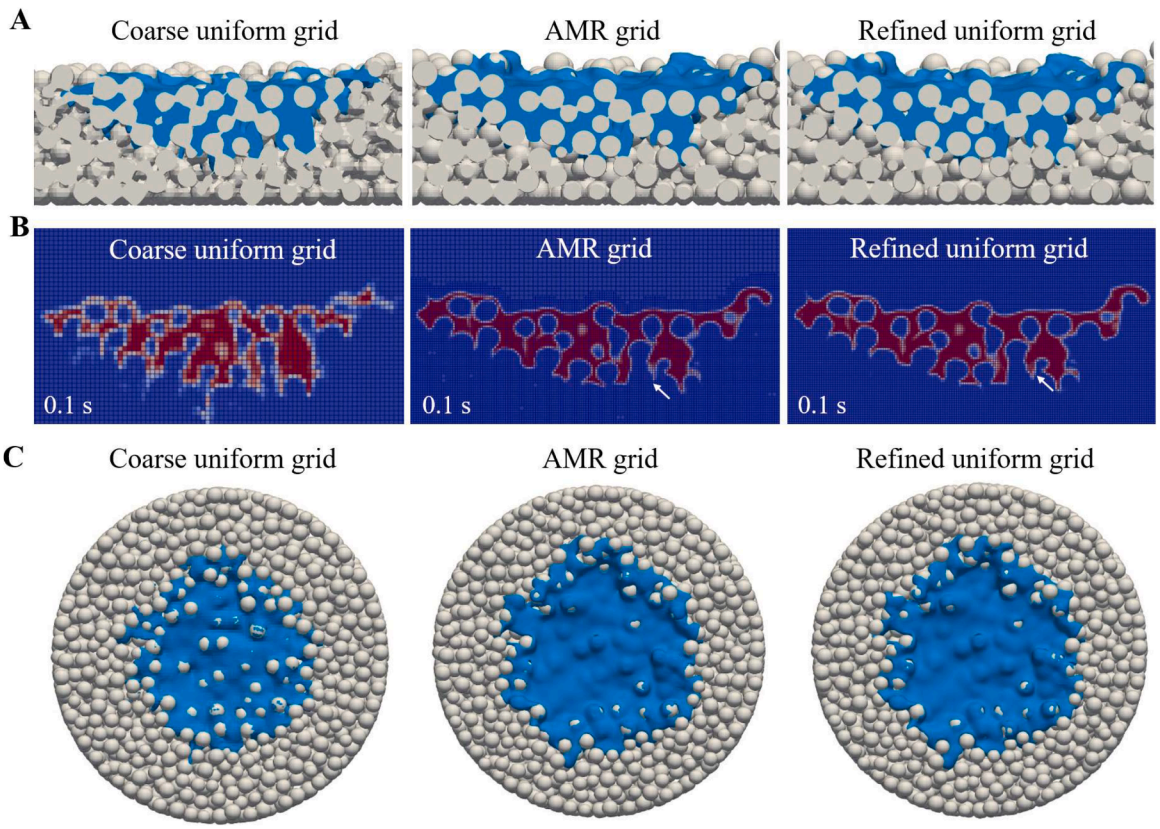


Fig. 29. Comparative study of simulation results using different grids at 0.1 s. (A) Half-section view of the particle and water distribution. (B) The grid structure and fluid volume fraction field of the half section. (C) Distribution of particles and water. Left: Uniform coarse grid, Middle: Adaptive refined grid, Right: Uniform refined grid. The red and blue colors represent a volume fraction of 1 and 0 for water, respectively.

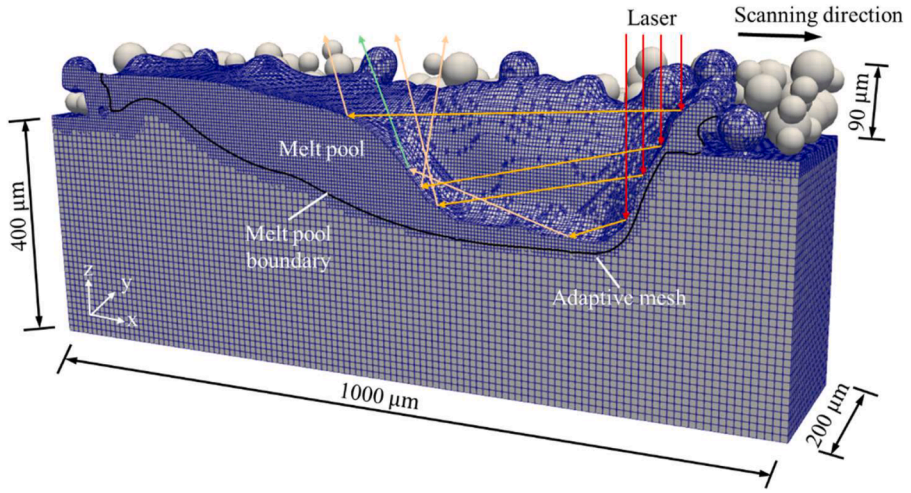


Fig. 30. Schematic of the multiphase and multiphysics LPBF simulation setup, illustrating the half-domain configuration, computational mesh, laser and scanning directions, and adaptive mesh refinement near the melt pool.

preserves the essential physics of binder jetting, including multiphase flow dynamics, fluid-particle coupling, and interparticle interactions. The framework remains fully extensible for incorporation of more sophisticated physical models in future investigations.

This case uses a cylindrical computational domain meshed with the cut-cell method, where the interior comprises a uniform hexahedral mesh while the boundary is resolved with polyhedral cells, as illustrated in Fig. 26A. Within this configuration, the AMR scheme is applied exclusively to the internal hexahedral cells. Three distinct grid configurations were evaluated: two uniform grids (1.67 mm and 0.83 mm cell sizes) and an adaptive grid with 1.67 mm parent cells. The maximum and minimum particle diameters are approximately 12 and 9.6 times the size of the refined mesh, respectively.

Fig. 26C demonstrates the temporal evolution of droplet impact and subsequent infiltration through the powder bed, while Fig. 26D reveals the corresponding velocity field. The velocity distribution shows maximum fluid velocities occurring within interparticle pores, a consequence of constrained particle motion that forces fluid flow

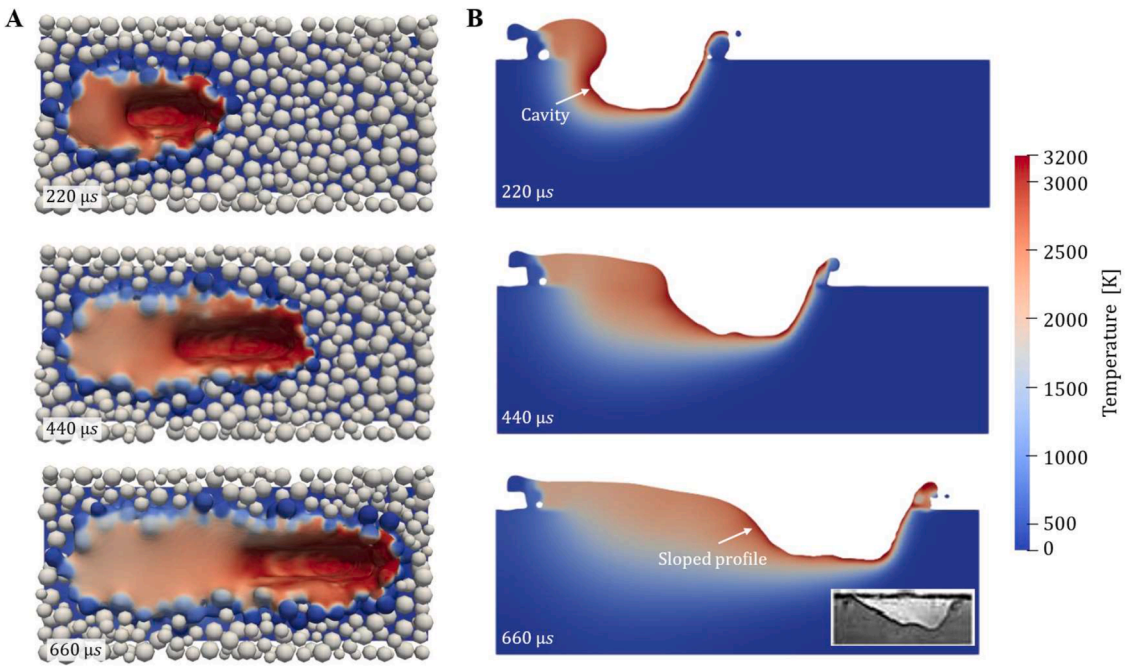


Fig. 31. Results simulated using the AMR scheme at 220 μs, 440 μs, and 660 μs: (A) melt pool and powder bed morphology; (B) central cross-section of the melt pool. The experimental image [80] in the bottom-right corner shows the melt pool shape in a pure plate under similar conditions (200 W laser power, identical metal, scanning speed, and spot size).

through interstitial pathways. Driven by the leading edge of the droplet, particles are radially displaced outwards through particle-particle interactions, reaching velocities up to 0.5 m/s, as shown in Fig. 27. Simultaneously, particles directly beneath the droplet's impact center are driven downward, forming a distinct depression, evident in both Fig. 27 and Fig. 28A. The resulting particle motion continuously modifies the pore geometry, which in turn governs the evolving water penetration path. These high-fidelity, fully resolved CFD-DEM simulations successfully capture the spatially varying particle dynamics during droplet infiltration, providing critical mechanistic understanding for process optimization.

Fig. 28A and Fig. 29A present half-section views of the particle and water distribution at 0.02 s and 0.1 s, respectively. Correspondingly, Fig. 28B and Fig. 29B display the underlying grid structure and fluid volume fraction field, highlighting local mesh resolution and phase distribution. The results demonstrate that both the AMR configuration and the fine uniform grid yield equally sharp fluid interfaces and particle boundaries, with only subtle local differences in the flow field, which are highlighted by white arrows in Fig. 28B and Fig. 29B. The coarse uniform grid, in contrast, produces results that deviate substantially from the refined and AMR cases. This discrepancy grows more pronounced over time, as clearly shown in the particle and fluid distributions at 0.02 s and 0.1 s (Fig. 28C and Fig. 29C). The divergence is attributable to the combined effects of grid resolution and the resulting differences in simulated particle motion.

4.2. Laser powder bed fusion

Fig. 30 presents the computational domain configuration for the LPBF simulation. The substrate measures 1000 μm (length) × 400 μm (width) × 400 μm (height), with a 90 μm thick powder layer deposited on top. The powder bed consists of particles with diameters of 30 μm, 40 μm, and 50 μm in a 3:4:3 ratio. To examine the influence of spatial resolution, we considered two uniform meshes with cell sizes of 4.17 μm and 8.33 μm, as well as an adaptive mesh based on 8.33 μm parent cells. In the refined regions, the particle diameters correspond to approximately 7.2–12 times the local grid spacing, ensuring adequate resolution of particle-scale phenomena. The laser parameters include a power output of 300 W, scanning velocity of 1 m/s, and spot size of 95 μm, with multiple reflections occurring within the melt pool. The AMR implementation focuses on critical regions: the melt pool (where temperatures exceed the liquidus point), particle boundaries, and fluid-fluid interfaces. The material system comprises Ti-6Al-4 V alloy with argon shielding gas, whose thermophysical properties are detailed in our previous work [78].

Fig. 31 illustrates the temporal evolution of the LPBF process at three characteristic time points. The left-to-right laser scanning initially melts the powder layer before penetrating the substrate. Upon reaching the evaporation temperature, recoil pressure creates a pronounced depression in the melt pool. During the initial phase, the confined heating zone causes molten material accumulation on the laser's trailing edge, forming a distinct cavity. As the process stabilizes, this cavity evolves into a characteristic sloped profile. Experimental validation (Fig. 31B) using a pure titanium plate under comparable conditions (with adjusted laser power) shows excellent agreement with our simulated melt pool morphology. Additionally, the use of a resolved CFD-DEM scheme, even with limited particle mobility in this case, is justified by the need for a robust and extensible computational foundation. This strategy ensures that the model can directly evolve to simulate more intense multiphase interactions, notably the pronounced particle entrainment by high-speed vapor streams anticipated in subsequent studies [79].

Fig. 32 compares melt pool morphologies at 450 μs using three mesh configurations. The uniform coarse mesh produces significantly different results due to insufficient resolution of particle geometry and temperature gradients. Quantitative analysis (Fig. 32D, E) reveals maximum contour deviations of 0.022 mm (5.5% of model width) at $y =$

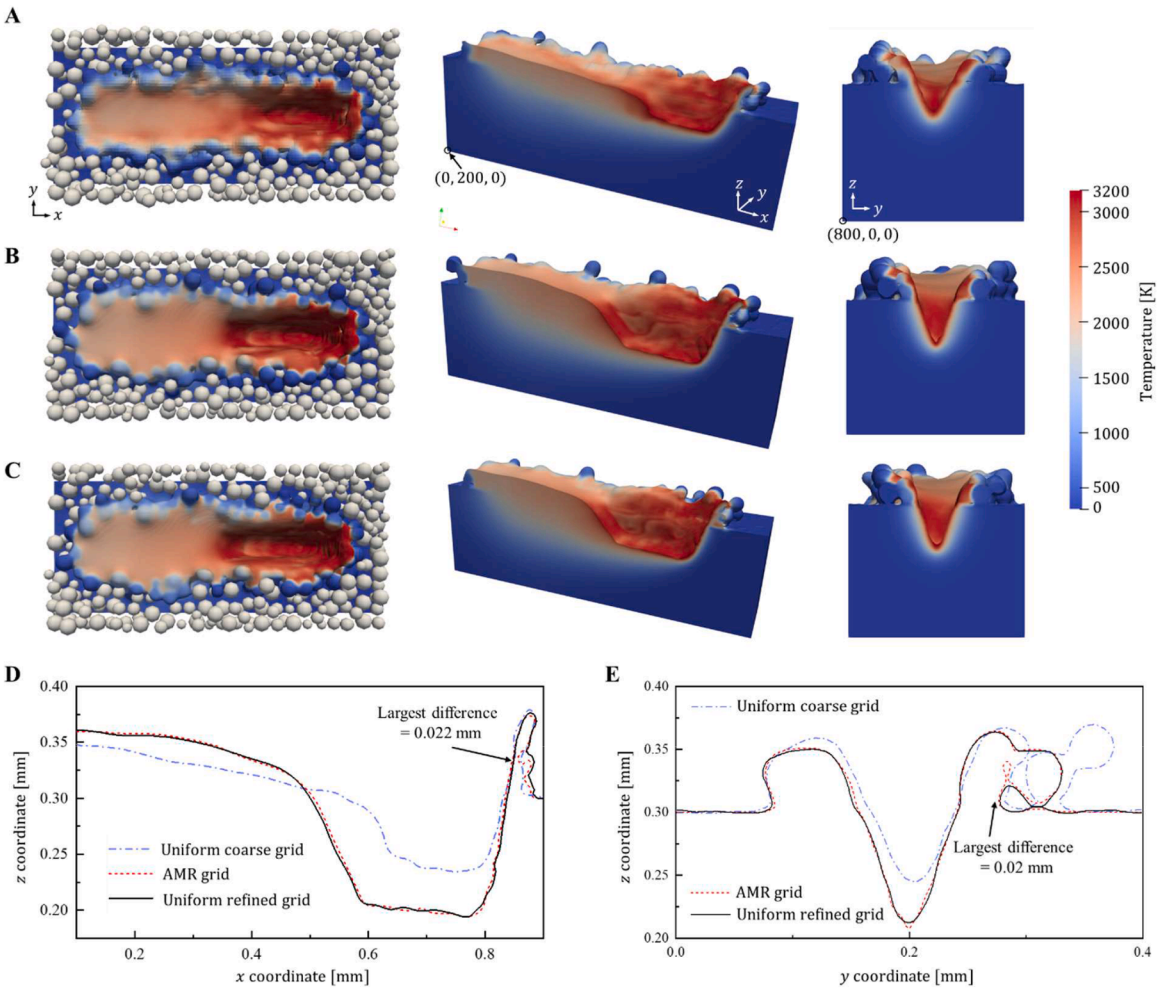


Fig. 32. Simulation results at 650 μs using (A) a uniform coarse grid, (B) an AMR grid, and (C) a uniform refined grid. For each case, the left panel shows a top view, the center panel shows a cross-section at $y = 200$ μm, and the right panel shows a cross-section at $x = 800$ μm. (D, E) Comparison of melt pool contours at $y = 200$ μm and $x = 800$ μm obtained from the three grid configurations. The coordinates shown in the figure are given in micrometers (μm).

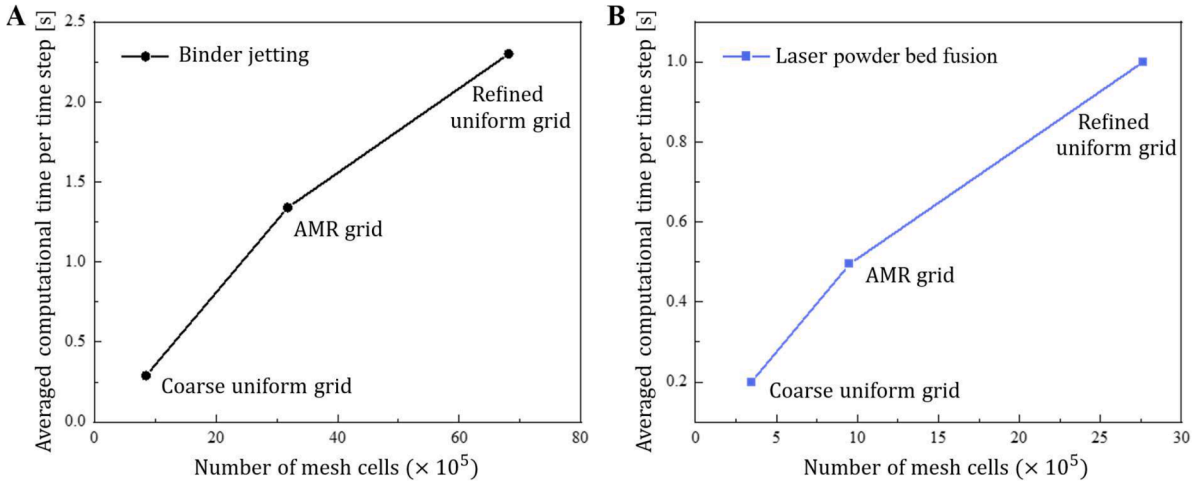


Fig. 33. Averaged computational time per time step for the binder jetting (A) and the LPBF (B) with three grid types: a coarse uniform grid, a refined uniform grid, and an adaptively refined grid based on the coarse mesh. All tests were conducted on the same GPU hardware (RTX 4070 Ti Super).

200 μm and 0.020 mm (5% of model width) at $x = 800$ μm between AMR and refined uniform meshes. However, the core melt pool region shows exceptional agreement, with deviations below 0.0044 mm (0.88% of model width). Notably, the surface contour irregularities caused by partial melting of particles at the melt pool periphery have been omitted from analysis, as these artifacts primarily stem from limitations in the particle replacement algorithm rather than physical phenomena.

To provide statistically robust validation, we calculated root mean square errors (RMSE) from 50 uniformly sampled points along each section. The resulting RMSE values of 0.0053 mm (1.3%) and 0.0033 mm (0.82%) for the $y = 200$ μm and $x = 800$ μm sections, respectively, confirm the AMR method's accuracy falls well within acceptable engineering tolerances. These results substantiate the reliability of our GPU-optimized AMR approach for complex multiphysics simulations.

4.3. Evaluation of computational performance

Fig. 33 summarizes the relationship between total number of cells and average computation time per timestep for three mesh configurations across both applications. The results demonstrate near-linear throughput performance with increasing problem size, particularly in the physically demanding LPBF case, confirming that the AMR algorithm introduces negligible computational overhead relative to the overall simulation cost while offering significant efficiency improvements. As the computational domain grows with a fixed refined region size, the benefits of efficiency gains by AMR scale multiplicatively, progressively approaching the theoretical maximum eightfold speedup achievable through single-level refinement.

The performance advantages are exemplified by a benchmark LPBF simulation involving approximately one million cells and ten thousand computational steps. The previously employed open-source CFDEM framework [81] with CPU-based AMR required two days to complete this simulation using 36 cores of the Tianhe-II supercomputer [78]. In striking contrast, our GPU-accelerated solver with GPU-optimized AMR completed the identical simulation in merely two hours, achieving a speedup factor exceeding twenty. For comprehensive details of the LPBF algorithm implementation, readers are directed to our previous work [1]. A more extensive quantitative comparison across varying problem sizes and hardware configurations will be presented in future studies.

5. Conclusions and outlooks

This study presents a GPU-accelerated adaptive mesh refinement (AMR) framework specifically optimized for unstructured hexahedral grids, addressing critical bottlenecks in large-scale computational fluid dynamics and discrete element method (CFD-DEM) simulations. By leveraging a compressed data format and GPU-tailored data reuse strategies, the proposed algorithm significantly streamlines grid topology management, reduces logical overhead, and minimizes memory reorganization costs. Implemented on hexahedral meshes using CUDA, the methodology demonstrates extensibility to unstructured tetrahedral grids through sub-mesh templating, offering a pathway for broader adoption in geometrically complex domains. Comprehensive validation through five benchmark cases, including 2D/3D dam breaks and sedimentation of both spherical and non-spherical particles, demonstrates that the proposed framework achieves significant computational efficiency gains compared to uniformly refined grids while preserving accurate interfacial resolution and force calculations. The methodology's practical applicability has been further verified through successful implementation in two complex powder-based additive manufacturing scenarios.

These results collectively demonstrate the algorithm's capability to resolve complex multiphysics phenomena, such as dynamic fluid-structure interactions, free-surface flow dynamics, and heterogeneous particle-laden systems. As the computational domain scales while maintaining a fixed refined region size, the overall computational efficiency increases multiplicatively, approaching the theoretical limit of eightfold speedup achievable through single-level refinement. When integrated with GPU-accelerated CFD-DEM solvers, the proposed AMR framework achieves over 20 × acceleration in large-scale LPBF simulations, underscoring its transformative potential for industrial-scale, high-fidelity multiphysics modeling.

Future research will focus on extending the framework to tetrahedral and polyhedral meshes prevalent in biomedical and aerospace applications, while implementing multi-level refinement to better resolve multiscale phenomena like turbulent boundary layers and granular segregation. Computational performance will be enhanced through asynchronous kernel execution and memory latency reduction, with particular attention to optimizing GPU memory management via dynamic memory pooling and just-in-time allocation strategies to address the current 1.1 GB/million-cell overhead. These improvements, coupled with kernel-specific memory reuse schemes to mitigate the 55% memory increase compared to uniform grids, will reduce the AMR module's runtime share from its current 25% of total computation while enabling larger-scale simulations on memory-constrained systems. Such advancements will strengthen the algorithm's viability for real-time simulation and digital twin applications, ultimately bridging the gap between high-fidelity modeling and practical engineering workflows.

CRediT authorship contribution statement

Tao Yu: Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Jidong Zhao:** Writing – review & editing, Supervision, 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

The study was financially supported by National Natural Science Foundation of China under Project 52439001 and Research Grants Council of Hong Kong through Nos. GRF 16203123, CRF C7085-24G, RIF R6008-24, TRS T22-607/24N, and T22-606/23-R.

Appendix. Algorithms of the GPU-accelerated AMR

Algorithm A1
Construction of numbering lists.

Input: Refinement index for the parent mesh I_0, and total number of inner faces N_{IF0}, cells N_{C0}, and points N_{P0} in the parent mesh.
Output: Numbering lists of points (L_P), cells (L_C), FI faces (L_{FI}), FB faces (L_{FB}), and FC faces (L_{FC}).

1st GPU kernel:
for $i < N_{C0}$ **do**
  if $I_0[i] = 1$ **then**
    $L_P[i] = 19$; $L_C[i] = 7$; $L_{FC}[i] = 12$;
  else:
    $L_P[i] = 0$; $L_C[i] = 0$; $L_{FC}[i] = 0$;
2nd GPU kernel:
for $i < N_{IF0}$ **do**
  Get the owner cell index $\mathrm{ID_o}$ and neighbor cell index $\mathrm{ID_n}$ for inner face i;
  if $I_0[\mathrm{ID_o}] = 1$ or $I_0[\mathrm{ID_n}] = 1$ **then**
    $L_{FI}[i] = 3$;
  else
  $L_{FI}[i] = 0$;
3rd GPU kernel:
for $i < N_{BF0}$ **do**
  Get the owner cell index $\mathrm{ID_o}$ for boundary face i;
  if $I_0[\mathrm{ID_o}] = 1$ **then**
    $L_{FB}[i] = 3$;
  else
    $L_{FB}[i] = 0$;
Sum up the numbering lists using the reduce algorithm from the Thrust library:
  $NL_P = \sum L_P$, $NL_C = \sum L_C$, $NL_{FC} = \sum L_{FC}$, $NL_{FI} = \sum L_{FI}$, $NL_{FB} = \sum L_{FB}$;
Update the total number of point (N_P), cell (N_C), inner face (N_{IF}), boundary face (N_{BF}) for the refined mesh:
  $N_P = N_{P0} + NL_P$, $N_C = N_{C0} + NL_C$, $N_{IF} = N_{IF0} + NL_{FC} + NL_{FI}$, $N_{BF} = N_{BF0} + NL_{FB}$;
Update the numbering lists using the exclusive scan algorithm from the Thrust library:
  $L_P[i] = \sum_{j=0}^{i-1} L_P[j]$, $L_C[i] = \sum_{j=0}^{i-1} L_C[j]$, $L_{FC}[i] = \sum_{j=0}^{i-1} L_{FC}[j]$ for $0 < i < N_{C0}$;
  $L_{FI}[i] = \sum_{j=0}^{i-1} L_{FI}[j]$ for $0 < i < N_{IF0}$;
  $L_{FB}[i] = \sum_{j=0}^{i-1} L_{FB}[j]$ for $0 < i < N_{BF0}$;

Note: $L_P(0) = 0$, $L_C(0) = 0$, $L_{FC}(0) = 0$, $L_{FI}(0) = 0$, and $L_{FB}(0) = 0$ after using the exclusive scan algorithm.

Algorithm A2
Construction of refinement arrays.

Input: Refinement index for the parent mesh I_0, and total number of inner faces N_{IF0}, cells N_{C0}, and points N_{P0} in the parent mesh.
Output: Cell index (*RC*), interior face index (*RIF*), and boundary face index (*RBF*).

Define an initial value $k = 0$;
1st GPU kernel:
for $i < N_{C0}$ **do**
  if $I_0[i] = 1$ **then**
    Record the refinement array index: $j = \mathrm{atomicAdd}(k)$;
    Record the cell index: $RC[j] = i$;
Initialize the value $k = 0$;
2nd GPU kernel:
for $i < N_{IF0}$ **do**
  Get the owner cell index $\mathrm{ID_o}$ and neighbor cell index $\mathrm{ID_n}$ for inner face i;
  if $I_0[\mathrm{ID_o}] = 1$ or $I_0[\mathrm{ID_n}] = 1$ **then**
    Record the refinement array index: $j = \mathrm{atomicAdd}(k)$;
    Record the interior face index: $RIF[j] = i$;
Initialize the value $k = 0$;
3rd GPU kernel:
for $i < N_{BF0}$ **do**
  Get the owner cell index $\mathrm{ID_o}$ for boundary face i;
  if $I_0[\mathrm{ID_o}] = 1$ **then**
    Record the refinement array index: $j = \mathrm{atomicAdd}(k)$;
    Record the boundary face index: $RBF[j] = i$;

Algorithm A3
Reconstruction of the point array.

Input: Cell index *RC*, point coordinate *P*, numbering lists of points L_P, total number of points in the parent mesh N_{P0}, and total number of added cells after refinement NL_C.
Output: Point coordinate *P*.

1st GPU kernel:
for $i < NL_C$ **do**
  Get the cell index $j = \mathrm{RC[i]}$

(*continued on next page*)

Algorithm A3 (*continued*)

Get the coordinates of points from the parent mesh: $P[0]$ to $P[7]$
Update the coordinates of the added vertices:
$P[N_{P0} + L_P[j]] = 0.5 \times (P[0] + P[1])$;
$P[N_{P0} + L_P[j] + 1] = 0.5 \times (P[0] + P[2])$;
$P[N_{P0} + L_P[j] + 2] = 0.25 \times (P[0] + P[1] + P[2] + P[3])$;
…
$P[N_{P0} + L_P[j] + 18] = 0.5 \times (P[6] + P[7])$;

Note: For clarity, representative steps are emphasized, while redundant operations with similar structure are omitted.

Algorithm A4
Reconstruction of the inner face array.

Input: Inner face index *RIF*, face index f, numbering lists of FI face L_{FI}, FC face L_{FC}, and point L_P, owner cell index $\mathrm{ID_o}$, neighbor cell index $\mathrm{ID_n}$, total number of inner faces N_{IF0} and points N_{P0} in the parent mesh, and total number of added interior faces after refinement NL_{FI}.
Output: Face array F.
1st GPU kernel:
for $i < NL_{FI}$ **do**
Get the inner face index $j = RIF[i]$;
Get the owner cell index $\mathrm{ID_o}$ and neighbor cell index $\mathrm{ID_n}$ for inner face j;
Get one refined cell index ID: $\mathrm{ID} = \mathrm{ID_o}$ for $I_0[\mathrm{ID_n}] = 0$ or $\mathrm{ID} = \mathrm{ID_n}$ for $I_0[\mathrm{ID_n}] = 1$;
Get the indices of faces from the parent mesh: $f[\mathrm{ID}][0]$ to $f[\mathrm{ID}][5]$;
Define temporary number: $N'_P = N_{P0} + L_P[\mathrm{ID}]$;
Update the face array:
if $j = f[\mathrm{ID}][0]$ **then**
$F[j] = [p_0,\ N'_P,\ N'_P + 2,\ N'_P + 1]$;
$F[N_{IF0} + L_{FI}[j]] = [N'_P,\ p_1,\ N'_P + 3,\ N'_P + 2]$;
$F[N_{IF0} + L_{FI}[j] + 1] = [N'_P + 1,\ N'_P + 2,\ N'_P + 4, p_2]$;
$F[N_{IF0} + L_{FI}[j] + 2] = [N'_P + 2,\ N'_P + 3,\ p_3, N'_P + 4]$;
…
if $j = f[\mathrm{ID}][5]$ **then**
$F[j] = [p_2,\ N'_P + 4,\ N'_P + 12,\ N'_P + 11]$;
$F[N_{IF0} + L_{FI}[j]] = [N'_P + 4,\ p_3,\ N'_P + 13,\ N'_P + 12]$;
$F[N_{IF0} + L_{FI}[j] + 1] = [N'_P + 11,\ N'_P + 12,\ N'_P + 18, p_6]$;
$F[N_{IF0} + L_{FI}[j] + 2] = [N'_P + 12,\ N'_P + 13,\ p_7, N'_P + 18]$;
2nd GPU kernel:
for $i < NL_C$ **do**
Get the cell index $j = RC[i]$;
Define temporary number: $N'_{IF} = N_{IF0} + NL_{FI} + L_{FC}[j]$;
Define temporary number: $N'_P = N_{P0} + L_P[j]$;
Update the face array:
$F[N'_{IF}] = [N'_P,\ N'_P + 2, N'_P + 9,\ N'_P + 6]$;
$F[N'_{IF} + 1] = [N'_P + 2,\ N'_P + 4,\ N'_P + 12,\ N'_P + 9]$;
$F[N'_{IF} + 2] = [N'_P + 6,\ N'_P + 9, N'_P + 16,\ N'_P + 14]$;
$F[N'_{IF} + 3] = [N'_P + 9,\ N'_P + 12, N'_P + 18,\ N'_P + 16]$;
$F[N'_{IF} + 4] = [N'_P + 1,\ N'_P + 2,\ N'_P + 9,\ N'_P + 8]$;
$F[N'_{IF} + 5] = [N'_P + 2, N'_P + \ 3,\ N'_P + 10,\ N'_P + 9]$;
$F[N'_{IF} + 6] = [N'_P + 8,\ N'_P + 9,\ N'_P + 16,\ N'_P + 15]$;
$F[N'_{IF} + 7] = [N'_P + 9,\ N'_P + 10,\ N'_P + 17,\ N'_P + 16]$;
$F[N'_{IF} + 8] = [N'_P + 5,\ N'_P + 6,\ N'_P + 9,\ N'_P + 8]$;
$F[N'_{IF} + 9] = [N'_P + 6,\ N'_P + 7,\ N'_P + 10,\ N'_P + 9]$;
$F[N'_{IF} + 10] = [N'_P + 8,\ N'_P + 9,\ N'_P + 12,\ N'_P + 11]$;
$F[N'_{IF} + 11] = [N'_P + 9,\ N'_P + 10,\ N'_P + 13,\ N'_P + 12]$;

Note: For clarity, representative steps are emphasized, while redundant operations with similar structure are omitted.

Algorithm A5
Reconstruction of the face array and boundary array.

Input: Boundary face index *RBF*, face index f, numbering lists of FB face L_{FB} and point L_P, owner cell index $\mathrm{ID_o}$, face array F, boundary array B, total number of points N_{P0} and boundary face N_{BF0} in the parent mesh, and total number of added boundary faces NL_{FB} after refinement.
Output: Face array F and boundary array B.
1st GPU kernel:
for $i < N_{BF0}$ **do**
Update the face array:
$F[N_{IF} + i] = F[i]$;
Update the boundary array:
$B[N_{IF} + i] = B[i]$;
2nd GPU kernel:
for $i < NL_{FB}$ **do**
Get the boundary face index $j = RBF[i]$;
Get the owner cell index $\mathrm{ID_o}$ for boundary face j;
Get the indices of faces from the parent mesh: $f[\mathrm{ID_o}][0]$ to $f[\mathrm{ID_o}][5]$;

(*continued on next page*)

Algorithm A5 (*continued*)

Define temporary number: $N'_{BF} = N_{IF} + N_{BF0}$;
Update the boundary array:
$B\left[N'_{BF} + L_{FB}[j]\right] = B[N_{IF} + j]$;
$B\left[N'_{BF} + L_{FB}[j] + 1\right] = B[N_{IF} + j]$;*ss*
$B\left[N'_{BF} + L_{FB}[j] + 2\right] = B[N_{IF} + j]$;
Update the face array:
if $j = f[\mathrm{ID_o}][0]$ **then**
$F[N_{IF} + j] = \{p_0,\ N_{P0} + L_P[\mathrm{ID}],\ N_{P0} + L_P[\mathrm{ID}] + 2,\ N_{P0} + L_P[\mathrm{ID}] + 1\}$;
$F\left[N'_{BF} + L_{FB}[j]\right] = \{N_{P0} + L_P[\mathrm{ID}],\ p_1,\ N_{P0} + L_P[\mathrm{ID}] + 3,\ N_{P0} + L_P[\mathrm{ID}] + 2\}$;
$F\left[N'_{BF} + L_{FB}[j] + 1\right] = \{N_{P0} + L_P[\mathrm{ID}] + 1,\ N_{P0} + L_P[\mathrm{ID}] + 2,\ N_{P0} + L_P[\mathrm{ID}] + 4, p_2\}$;
$F\left[N'_{BF} + L_{FB}[j] + 2\right] = \{N_{P0} + L_P[\mathrm{ID}] + 2,\ N_{P0} + L_P[\mathrm{ID}] + 3,\ p_3, N_{P0} + L_P[\mathrm{ID}] + 4\}$;
…
if $j = f[\mathrm{ID_o}][5]$ **then**
$F[N_{IF} + j] = \{p_2,\ N_{P0} + L_P[\mathrm{ID}] + 4,\ N_{P0} + L_P[\mathrm{ID}] + 12,\ N_{P0} + L_P[\mathrm{ID}] + 11\}$;
$F\left[N'_{BF} + L_{FB}[j]\right] = \{N_{P0} + L_P[\mathrm{ID}] + 4,\ p_3,\ N_{P0} + L_P[\mathrm{ID}] + 13,\ N_{P0} + L_P[\mathrm{ID}] + 12\}$;
$F\left[N'_{BF} + L_{FB}[j] + 1\right] = \{N_{P0} + L_P[\mathrm{ID}] + 11,\ N_{P0} + L_P[\mathrm{ID}] + 12,\ N_{P0} + L_P[\mathrm{ID}] + 18, p_6\}$;
$F\left[N'_{BF} + L_{FB}[j] + 2\right] = \{N_{P0} + L_P[\mathrm{ID}] + 12,\ N_{P0} + L_P[\mathrm{ID}] + 13,\ p_7, N_{P0} + L_P[\mathrm{ID}] + 18\}$;

Note: For clarity, representative steps are emphasized, while redundant operations with similar structure are omitted.

Algorithm A6
Reconstruction of the owner array and neighbor array for inner faces on the parent mesh.

Input: Refinement index I_0, inner face index *RIF*, face index *f*, numbering lists of FI face L_{FI}, and cell L_C, owner cell index $\mathrm{ID_o}$, neighbor cell index $\mathrm{ID_n}$, total number of inner faces N_{IF0} and cells N_{C0} in the parent mesh, and total number of added interior faces NL_{FI} after refinement.
Output: Owner array *O* and neighbor array *N*.
1st GPU kernel:
for $i < NL_{FI}$ **do**
Get the inner face index $j = RIF[i]$;
Get the owner cell index $\mathrm{ID_o}$ and neighbor cell index $\mathrm{ID_n}$ for inner face *j*;
Define temporary number: $N'_{IF} = N_{IF0} + L_{FI}[j]$;
Define temporary number: $N'_{\mathrm{o}} = N_{C0} + L_C[\mathrm{ID_o}]$, $N'_{\mathrm{n}} = N_{C0} + L_C[\mathrm{ID_n}]$;
if $I_0[\mathrm{ID_o}] = 0$ and $I_0[\mathrm{ID_n}] = 0$ **then**
$O[j] = O_0[j]$, $N[j] = N_0[j]$;
if $I_0[\mathrm{ID_o}] = 0$ and $I_0[\mathrm{ID_n}] = 1$ **then**
Get the indices of faces from the parent mesh: $f[\mathrm{ID_n}][0]$ to $f[\mathrm{ID_n}][5]$;
if $j = f[\mathrm{ID_n}][0]$ **then**
$O[j] = \mathrm{ID_o}$, $N[j] = \mathrm{ID_n}$;
$O\left[N'_{IF}\right] = \mathrm{ID_o}$, $N\left[N'_{IF}\right] = N'_{\mathrm{n}}$;
$O\left[N'_{IF} + 1\right] = \mathrm{ID_o}$, $N\left[N'_{IF} + 1\right] = N'_{\mathrm{n}} + 1$;
$O\left[N'_{IF} + 2\right] = \mathrm{ID_o}$, $N\left[N'_{IF} + 2\right] = N'_{\mathrm{n}} + 2$;
if $j = f[\mathrm{ID_n}][1]$ to $f[\mathrm{ID_n}][5]$ **then**
…
if $I_0[\mathrm{ID_o}] = 1$ and $I_0[\mathrm{ID_n}] = 0$ **then**
… (similar to Step 8)
if $I_0[\mathrm{ID_o}] = 1$ and $I_0[\mathrm{ID_n}] = 1$ **then**
Get the indices of faces from the parent mesh: $f[\mathrm{ID_o}][0]$ to $f[\mathrm{ID_o}][5]$;
Get the indices of faces from the parent mesh: $f[\mathrm{ID_n}][0]$ to $f[\mathrm{ID_n}][5]$;
if $j = f[\mathrm{ID_o}][0]$ **then**
$O[j] = \mathrm{ID_o}$, $O\left[N'_{IF}\right] = N'_{\mathrm{o}}$, $O\left[N'_{IF} + 1\right] = N'_{\mathrm{o}} + 1$, $O\left[N'_{IF} + 2\right] = N'_{\mathrm{o}} + 2$;
if $j = f[\mathrm{ID_o}][1]$ to $f[\mathrm{ID_o}][5]$ **then**
…
if $j = f[\mathrm{ID_n}][0]$ **then**
$N[j] = \mathrm{ID_n}$, $O\left[N'_{IF}\right] = N'_{\mathrm{n}}$, $O\left[N'_{IF} + 1\right] = N'_{\mathrm{n}} + 1$, $O\left[N'_{IF} + 2\right] = N'_{\mathrm{n}} + 2$,
if $j = f[\mathrm{ID_n}][1]$ to $f[\mathrm{ID_n}][5]$ **then**
…

Algorithm A7
Reconstruction of the owner array and neighbor array for inner faces in the sub-mesh.

Input: cell index *RC*, numbering lists of FC face L_{FC}, and cell L_C, total number of inner faces N_{IF0} and cells N_{C0} in the parent mesh, and total number of added inner faces NL_{FI} and cells NL_C after refinement.
Output: Owner array *O* and neighbor array *N*.
1st GPU kernel:
for $i < NL_C$ **do**
Get the cell index $j = RC[i]$;
Define temporary number: $N'_{IF} = N_{IF0} + NL_{FI} + L_{FC}[j]$;
Define temporary number: $N'_{\mathrm{C}} = N_{C0} + L_C[j]$;
Update the owner array and the neighbor array:
$O\left[N'_{IF}\right] = i$, $N\left[N'_{IF}\right] = N'_{\mathrm{C}}$
$O\left[N'_{IF} + 1\right] = N'_{\mathrm{C}} + 1$, $N\left[N'_{IF} + 1\right] = N'_{\mathrm{C}} + 2$;

(*continued on next page*)

Algorithm A7 (*continued*)

9	$O\left[N'_{IF}+2\right]=N'_{C}+3,\ N\left[N'_{IF}+2\right]=N'_{C}+4;$
10	$O\left[N'_{IF}+3\right]=N'_{C}+5,\ N\left[N'_{IF}+3\right]=N'_{C}+6;$
11	$O\left[N'_{IF}+4\right]=N'_{C}+i,\ N\left[N'_{IF}+4\right]=N'_{C}+1;$
12	$O\left[N'_{IF}+5\right]=N'_{C},\ N\left[N'_{IF}+5\right]=N'_{C}+2;$
13	$O\left[N'_{IF}+6\right]=N'_{C}+3,\ N\left[N'_{IF}+6\right]=N'_{C}+5;$
14	$O\left[N'_{IF}+7\right]=N'_{C}+4,\ N\left[N'_{IF}+7\right]=N'_{C}+6;$
15	$O\left[N'_{IF}+8\right]=i,\ N\left[N'_{IF}+8\right]=N'_{C}+3;$
16	$O\left[N'_{IF}+9\right]=N'_{C},\ N\left[N'_{IF}+9\right]=N'_{C}+4;$
17	$O\left[N'_{IF}+10\right]=N'_{C}+1,\ N\left[N'_{IF}+10\right]=N'_{C}+5;$
18	$O\left[N'_{IF}+11\right]=N'_{C}+2,\ N\left[N'_{IF}+11\right]=N'_{C}+6;$

Algorithm A8
Reconstruction of the owner array for boundary faces.

Input: boundary face index RBF, numbering lists of FB face L_{FB}, and cell L_C, owner array O, total number of boundary faces N_{BF0} and cells N_{C0} in the parent mesh, total number of inner faces N_{IF} after refinement, and total number of added boundary faces NL_{FB} after refinement.
Output: Owner array O.

1st GPU kernel:
for $i < N_{BF0}$ **do**
  Update the owner array:
    $O[N_{IF}+i]=O[i];$
2nd GPU kernel:
for $i < NL_{FB}$ **do**
  Get the boundary face index $j=RBF[i];$
  Get the owner cell index $\mathrm{ID_o}$ for boundary face j;
  Get the indices of faces from the parent mesh: $f[\mathrm{ID_o}][0]$ to $f[\mathrm{ID_o}][5]$;
  Define temporary number: $N'_{BF}=N_{IF}+N_{BF0}+L_{FB}[j];$
  Define temporary number: $N'_{\mathrm{o}}=N_{C0}+L_C[\mathrm{ID_o}];$
  Update the owner array:
    if $j=f[\mathrm{ID_o}][0]$ **then**
      $O[N_{IF}+j]=\mathrm{ID_o},\ O\left[N'_{BF}\right]=N'_{\mathrm{o}},\ O\left[N'_{BF}+1\right]=N'_{\mathrm{o}}+1,\ O\left[N'_{BF}+2\right]=N'_{\mathrm{o}}+2;$
    if $j=f[\mathrm{ID_o}][1]$ to $f[\mathrm{ID_o}][5]$ **then**
      ...

Algorithm A9
Remapping of variable field x.

Input: Refinement index I_0, cell index RC, numbering lists of cell L_C, owner array O, variable field x, and total number of cells N_{C0} in the parent mesh.
Output: Variable field x.

1st GPU kernel:
for $i < N_{C0}$ **do**
  Get the cell index $j=RC[i];$
  if $I_0^n[i]=0$ and $I_0^{n+1}[i]=0$ **then**
    $x^{n+1}[i]=x^n[i];$
  if $I_0^n[i]=1$ and $I_0^{n+1}[i]=1$ **then**
    $x^{n+1}[i]=x^n[i],\ x^{n+1}\left[L_C^{n+1}[i]\right]=x^n\left[L_C^n[i]\right];$
    $x^{n+1}\left[L_C^{n+1}[i]+1\right]=x^n\left[L_C^n[i]+1\right],\ x^{n+1}\left[L_C^{n+1}[i]+2\right]=x^n\left[L_C^n[i]+2\right];$
    $x^{n+1}\left[L_C^{n+1}[i]+3\right]=x^n\left[L_C^n[i]+3\right],\ x^{n+1}\left[L_C^{n+1}[i]+4\right]=x^n\left[L_C^n[i]+4\right];$
    $x^{n+1}\left[L_C^{n+1}[i]+5\right]=x^n\left[L_C^n[i]+5\right],\ x^{n+1}\left[L_C^{n+1}[i]+6\right]=x^n\left[L_C^n[i]+6\right];$
  if $I_0^n[i]=1$ and $I_0^{n+1}[i]=0$ **then**
    $x^{n+1}[i]=\left(x^n[i]+x^n\left[L_C^n[i]\right]+x^n\left[L_C^n[i]+1\right]+\cdots+x^n\left[L_C^n[i]+6\right]\right)/8;$
  if $I_0^n[i]=0$ and $I_0^{n+1}[i]=1$ **then**
    $x^{n+1}[i]=x^n[i]/8,\ x^{n+1}\left[L_C^{n+1}[i]\right]=x^n[i]/8;$
    $x^{n+1}\left[L_C^{n+1}[i]+1\right]=x^n[i]/8,\ x^{n+1}\left[L_C^{n+1}[i]+2\right]=x^n[i]/8;$
    $x^{n+1}\left[L_C^{n+1}[i]+3\right]=x^n[i]/8,\ x^{n+1}\left[L_C^{n+1}[i]+4\right]=x^n[i]/8;$
    $x^{n+1}\left[L_C^{n+1}[i]+5\right]=x^n[i]/8,\ x^{n+1}\left[L_C^{n+1}[i]+6\right]=x^n[i]/8;$

Data availability

Data will be made available on request.

References

[1] T. Yu, J. Zhao, Semi-coupled resolved CFD–DEM simulation of powder-based selective laser melting for additive manufacturing, Comput. Methods Appl. Mech. Eng. 377 (2021) 113707.
[2] G. Hu, B. Zhou, Z. Shen, H. Wang, W. Zheng, A resolved CFD–DEM investigation on granular sand sedimentation considering realistic particle shapes, Géotechnique (2024) 1–13.
[3] H. Aziz, S.N. Ahsan, G. De Simone, Y. Gao, B. Chaudhuri, Computational modeling of drying of pharmaceutical wet granules in a fluidized bed dryer using coupled CFD-DEM approach, AAPS PharmSciTech 23 (2022) 59.
[4] D. Gerogiorgis, B. Ydstie, Multiphysics CFD modelling for design and simulation of a multiphase chemical reactor, Chem. Eng. Res. Des. 83 (2005) 603–610.
[5] O. Hennigh, S. Narasimhan, M.A. Nabian, A. Subramaniam, K. Tangsali, Z. Fang, M. Rietmann, W. Byeon, S. Choudhry, NVIDIA SimNet™: an AI-accelerated multi-physics simulation framework, in: Proceedings of the International Conference on Computational science, Springer, 2021, pp. 447–461.
[6] E. Buber, D. Banu, Performance analysis and CPU vs GPU comparison for deep learning, in: Proceedings of the 6th International Conference on Control Engineering & Information Technology (CEIT), IEEE, 2018, pp. 1–6.
[7] C. Peng, S. Wang, W. Wu, H.S. Yu, C. Wang, J.Y. Chen, LOQUAT: an open-source GPU-accelerated SPH solver for geotechnical modeling, Acta Geotech. 14 (2019) 1269–1287.
[8] X. Xia, Q. Liang, A GPU-accelerated smoothed particle hydrodynamics (SPH) model for the shallow water equations, Environ. Model. Softw. 75 (2016) 28–43.
[9] Q. Xiong, B. Li, J. Xu, GPU-accelerated adaptive particle splitting and merging in SPH, Comput. Phys. Commun. 184 (2013) 1701–1707.
[10] K.R. Tubbs, F.T.C. Tsai, GPU accelerated lattice Boltzmann model for shallow water flow and mass transport, Int. J. Numer. Methods Eng. 86 (2011) 316–334.
[11] S. Shu, J. Zhang, N. Yang, GPU-accelerated transient lattice Boltzmann simulation of bubble column reactors, Chem. Eng. Sci. 214 (2020) 115436.
[12] C. Nită, L.M. Itu, C. Suciu, GPU accelerated blood flow computation using the lattice Boltzmann method, in: Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC), IEEE, 2013, pp. 1–6.
[13] F. Jiang, K. Matsumura, J. Ohgi, X. Chen, A GPU-accelerated fluid–structure-interaction solver developed by coupling finite element and lattice Boltzmann methods, Comput. Phys. Commun. 259 (2021) 107661.
[14] P. Bailey, J. Myre, S.D. Walsh, D.J. Lilja, M.O. Saar, Accelerating lattice Boltzmann fluid flow simulations using graphics processors, in: Proceedings of the International Conference on Parallel Processing, IEEE, 2009, pp. 550–557.
[15] S. Zhao, Z. Lai, J. Zhao, Leveraging ray tracing cores for particle-based simulations on GPUs, Int. J. Numer. Methods Eng. 124 (2023) 696–713.
[16] S. Zhao, J. Zhao, SudoDEM: unleashing the predictive power of the discrete element method on simulation for non-spherical granular particles, Comput. Phys. Commun. 259 (2021) 107670.
[17] J. Gan, Z. Zhou, A. Yu, A GPU-based DEM approach for modelling of particulate systems, Powder Technol. 301 (2016) 1172–1182.
[18] M. Gao, X. Wang, K. Wu, A. Pradhana, E. Sifakis, C. Yuksel, C. Jiang, GPU optimization of material point methods, ACM Trans. Graph. (TOG) 37 (2018) 1–12.
[19] X. Wang, Y. Qiu, S.R. Slattery, Y. Fang, M. Li, S.C. Zhu, Y. Zhu, M. Tang, D. Manocha, C. Jiang, A massively parallel and scalable multi-GPU material point method, ACM Trans. Graph. (TOG) 39 (2020) 30, 31-30.
[20] A. Lani, M.S. Yalim, S. Poedts, A GPU-enabled finite volume solver for global magnetospheric simulations on unstructured grids, Comput. Phys. Commun. 185 (2014) 2538–2557.
[21] S. Kestur, J.D. Davis, O. Williams, Blas comparison on fpga, cpu and gpu, in: Proceedings of the IEEE Computer Society Annual Symposium on VLSI, IEEE, 2010, pp. 288–293.
[22] H.Y. Schive, Y.C. Tsai, T. Chiueh, GAMER: a graphic processing unit accelerated adaptive-mesh-refinement code for astrophysics, Astrophys. J. Suppl. Ser. 186 (2010) 457.
[23] F. Kuo, C. Chiang, M. Lo, J. Wu, Development of a parallel explicit finite-volume Euler equation solver using the immersed boundary method with hybrid MPI-CUDA paradigm, J. Mech. 36 (2020) 87–102.
[24] M. Aissa, T. Verstraete, C. Vuik, Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes, Comput. Math. Appl. 74 (2017) 201–217.
[25] P. Wang, T. Abel, R. Kaehler, Adaptive mesh fluid simulations on GPU, New Astron. 15 (2010) 581–589.
[26] J. Xu, H. Fu, W. Luk, L. Gan, W. Shi, W. Xue, C. Yang, Y. Jiang, C. He, G. Yang, Optimizing finite volume method solvers on Nvidia GPUs, IEEE Trans. Parallel Distrib. Syst. 30 (2019) 2790–2805.
[27] I. Kampolis, X. Trompoukis, V. Asouti, K. Giannakoglou, CFD-based analysis and two-level aerodynamic optimization on graphics processing units, Comput. Methods Appl. Mech. Eng. 199 (2010) 712–722.
[28] X. Trompoukis, V. Asouti, I. Kampolis, K. Giannakoglou, CUDA implementation of vertex-centered, finite volume CFD methods on unstructured grids with flow control applications. GPU Computing Gems Jade Edition, Elsevier, 2012, pp. 207–223.
[29] V.G. Asouti, X.S. Trompoukis, I.C. Kampolis, K.C. Giannakoglou, Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on graphics processing units, Int. J. Numer. Methods Fluids 67 (2011) 232–246.
[30] R. Löhner, Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods, John Wiley & Sons, 2008.
[31] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484–512.
[32] M. Berger, R. Leveque, An adaptive Cartesian mesh algorithm for the Euler equations in arbitrary geometries, in: Proceedings of the 9th Computational Fluid Dynamics Conference, 1989, p. 1930.
[33] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. Comput. Phys. 82 (1989) 64–84.
[34] H. Ji, F.S. Lien, F. Zhang, A GPU-accelerated adaptive mesh refinement for immersed boundary methods, Comput. Fluids 118 (2015) 131–147.
[35] S. Zaghi, F. Salvadore, A. Di Mascio, G. Rossi, Efficient GPU parallelization of adaptive mesh refinement technique for high-order compressible solver with immersed boundary, Comput. Fluids 266 (2023) 106040.
[36] H. Tong, E. Halilaj, Y.J. Zhang, HybridOctree_Hex: hybrid octree-based adaptive all-hexahedral mesh generation with Jacobian control, J. Comput. Sci. 78 (2024) 102278.
[37] J. Lei, D.L. Li, Y.L. Zhou, W. Liu, Optimization and acceleration of flow simulations for CFD on CPU/GPU architecture, J. Braz. Soc. Mech. Sci. Eng. 41 (2019) 290.
[38] X. Luo, L. Wang, W. Ran, F. Qin, GPU accelerated cell-based adaptive mesh refinement on unstructured quadrilateral grid, Comput. Phys. Commun. 207 (2016) 114–122.
[39] M. Biazewicz, K. Kurowski, B. Ludwiczak, K. Napieraia, Problems related to parallelization of CFD algorithms on GPU, multi-GPU and hybrid architectures, in: Proceedings of the AIP Conference, American Institute of Physics, 2010, pp. 1301–1304.
[40] V. Moureau, P. Domingo, L. Vervisch, Design of a massively parallel CFD code for complex geometries, C. R. Méc. 339 (2011) 141–148.
[41] A. Hager, C. Kloss, S. Pirker, C. Goniva, Parallel resolved open source CFD-DEM: method, validation and application, J. Comput. Multiph. Flows 6 (2014) 13–27.
[42] Z. Wang, Y. Teng, M. Liu, A semi-resolved CFD–DEM approach for particulate flows with kernel based approximation and Hilbert curve based searching strategy, J. Comput. Phys. 384 (2019) 151–169.
[43] Z. Lai, J. Zhao, S. Zhao, L. Huang, Signed distance field enhanced fully resolved CFD-DEM for simulation of granular flows involving multiphase fluids and irregularly shaped particles, Comput. Methods Appl. Mech. Eng. 414 (2023) 116195.
[44] Z. Wang, W. Yan, W.K. Liu, M. Liu, Powder-scale multi-physics modeling of multi-layer multi-track selective laser melting with sharp interface capturing method, Comput. Mech. 63 (2019) 649–661.
[45] R.I. Issa, Solution of the implicitly discretised fluid flow equations by operator-splitting, J. Comput. Phys. 62 (1986) 40–65.
[46] V. Vuorinen, J.P. Keskinen, C. Duwig, B. Boersma, On the implementation of low-dissipative Runge–Kutta projection methods for time dependent flows using OpenFOAM®, Comput. Fluids 93 (2014) 153–163.
[47] A.A. Shirgaonkar, M.A. MacIver, N.A. Patankar, A new mathematical formulation and fast algorithm for fully resolved simulation of self-propulsion, J. Comput. Phys. 228 (2009) 2366–2390.
[48] G.T. Nguyen, E.L. Chan, T. Tsuji, T. Tanaka, K. Washino, Interface control for resolved CFD-DEM with capillary interactions, Adv. Powder Technol. 32 (2021) 1410–1425.
[49] G.T. Nguyen, E.L. Chan, T. Tsuji, T. Tanaka, K. Washino, Resolved CFD–DEM coupling simulation using Volume Penalisation method, Adv. Powder Technol. 32 (2021) 225–236.
[50] Z.S. Saldi, Marangoni Driven Free Surface Flows in Liquid Weld Pools, Delft University of Technology, The Netherlands, 2012.
[51] J.U. Brackbill, D.B. Kothe, C. Zemach, A continuum method for modeling surface tension, J. Comput. Phys. 100 (1992) 335–354.
[52] C. Panwisawas, C. Qiu, M.J. Anderson, Y. Sovani, R.P. Turner, M.M. Attallah, J. W. Brooks, H.C. Basoalto, Mesoscale modelling of selective laser melting: thermal fluid dynamics and microstructural evolution, Comput. Mater. Sci. 126 (2017) 479–490.
[53] Z. Wang, W. Yan, W.K. Liu, M. Liu, Powder-scale multi-physics modeling of multi-layer multi-track selective laser melting with sharp interface capturing method, Comput. Mech. 63 (2019) 649–661.
[54] C.J. Greenshields, OpenFOAM User Guide, 3, OpenFOAM Foundation Ltd, version, 2015, p. 47.
[55] M. Rudman, Volume-tracking methods for interfacial flow calculations, Int. J. Numer. Methods Fluids 24 (1997) 671–691.
[56] H. Rusche, Computational fluid dynamics of dispersed two-phase flow at high phase fractions, Ph.D. Thesis, University of London, (2002).
[57] A. Di Renzo, F.P. Di Maio, Comparison of contact-force models for the simulation of collisions in DEM-based granular flow codes, Chem. Eng. Sci. 59 (2004) 525–541.
[58] A.B. Stevens, C. Hrenya, Comparison of soft-sphere models to measurements of collision properties during normal impacts, Powder Technol. 154 (2005) 99–109.
[59] H. Zhu, Z. Zhou, R. Yang, A. Yu, Discrete particle simulation of particulate systems: theoretical developments, Chem. Eng. Sci. 62 (2007) 3378–3396.
[60] R.D. Mindlin, H. Deresiewicz, Elastic spheres in contact under varying oblique forces, (1953).
[61] T. Yu, J. Zhao, Quantitative simulation of selective laser melting of metals enabled by new high-fidelity multiphase, multiphysics computational tool, Comput. Methods Appl. Mech. Eng. 399 (2022) 115422.

[62] M. Bayat, A. Thanki, S. Mohanty, A. Witvrouw, S. Yang, J. Thorborg, N.S. Tiedje, J. H. Hattel, Keyhole-induced porosities in Laser-based Powder Bed Fusion (L-PBF) of Ti6Al4V: high-fidelity modelling and experimental validation, Addit. Manuf. 30 (2019) 100835.
[63] M. Gharbi, P. Peyre, C. Gorny, M. Carin, S. Morville, P. Le Masson, D. Carron, R. Fabbro, Influence of various process conditions on surface finishes induced by the direct metal deposition laser technique on a Ti–6Al–4V alloy, J. Mater. Process. Technol. 213 (2013) 791–800.
[64] J.H. Cho, S.J. Na, Implementation of real-time multiple reflection and Fresnel absorption of laser beam in keyhole, J. Phys. D Appl. Phys. 39 (2006) 5372.
[65] R. McVey, R. Melnychuk, J. Todd, R. Martukanitz, Absorption of laser irradiation in a porous powder layer, J. Laser Appl. 19 (2007) 214–224.
[66] L.C. Dutto, C.Y. Lepage, W.G. Habashi, Effect of the storage format of sparse linear systems on parallel CFD computations, Comput. Methods Appl. Mech. Eng. 188 (2000) 441–453.
[67] Y. Lu, S. Rane, A. Kovacevic, Evaluation of cut cell cartesian method for simulation of a hook and claw type hydrogen pump, Int. J. Hydrog. Energy 47 (2022) 23006–23018.
[68] P.G. Tucker, Z. Pan, A Cartesian cut cell method for incompressible viscous flow, Appl. Math. Model. 24 (2000) 591–606.
[69] A. Hager, C.D.L. für partikuläre Strömungen, D.-I.D.C. Goniva, CFD–DEM on Multiple Scales, An Extensive Investigation of Parti-cle–Fluid Interactions. Johannes Kepler University Linz, Linz, (2014).
[70] J. Mao, L. Zhao, Y. Di, X. Liu, W. Xu, A resolved CFD–DEM approach for the simulation of landslides and impulse waves, Comput. Methods Appl. Mech. Eng. 359 (2020) 112750.
[71] C. Li, Y. Zhang, J. Shen, W. Zhang, Coupled simulation of fluid-particle interaction for large complex granules: a resolved CFD-DEM method for modelling the airflow in a vertical fixed bed of irregular sinter particles, Particuology 90 (2024) 292–306.
[72] W. Schroeder, K.M. Martin, W.E. Lorensen, The Visualization Toolkit an Object-Oriented Approach to 3D Graphics, Prentice-Hall, Inc, 1998.
[73] S. Koshizuka, A particle method for incompressible viscous flow with fluid fragmentation, Comput. Fluid Dyn. J. 4 (1995) 29–46.
[74] A. Ten Cate, C. Nieuwstad, J.J. Derksen, H. Van den Akker, Particle imaging velocimetry experiments and lattice-Boltzmann simulations on a single sphere settling under gravity, Phys. Fluids 14 (2002) 4012–4025.
[75] M. Fan, D. Su, L. Yang, Development of a benchmark for drag correlations of nonspherical particles based on settling experiments of super-ellipsoidal particles, Powder Technol. 409 (2022) 117811.
[76] C. Yang, F. Zhu, J. Zhao, Coupled total-and semi-Lagrangian peridynamics for modelling fluid-driven fracturing in solids, Comput. Methods Appl. Mech. Eng. 419 (2024) 116580.
[77] H. Jasak, A. Jemcov, Z. Tukovic, OpenFOAM: a C++ library for complex physics simulations, in: Proceedings of the International Workshop on Coupled Methods in Numerical Dynamics, Dubrovnik, Croatia, 2007, pp. 1–20.
[78] T. Yu, J. Zhao, Quantifying the mechanisms of keyhole pore evolutions and the role of metal-vapor condensation in laser powder bed fusion, Addit. Manuf. 72 (2023) 103642.
[79] P. Bidare, I. Bitharas, R. Ward, M. Attallah, A.J. Moore, Fluid and particle dynamics in laser powder bed fusion, Acta Mater. 142 (2018) 107–120.
[80] R. Cunningham, C. Zhao, N. Parab, C. Kantzos, J. Pauza, K. Fezzaa, T. Sun, A. D. Rollett, Keyhole threshold and morphology in laser melting revealed by ultrahigh-speed x-ray imaging, Science 363 (2019) 849–852.
[81] C. Kloss, C. Goniva, A. Hager, S. Amberger, S. Pirker, Models, algorithms and validation for opensource DEM and CFD–DEM, Prog. Comput. Fluid Dyn. Int. J. 12 (2012) 140–152.