1National Energy Technology Laboratory, Morgantown, WV 26507, USA
2Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
3Leidos Research Support Team, Morgantown, WV 26507, USA
Jordan Musser, National Energy Technology Laboratory, Morgantown, WV 26507, USA
jordan.musser@netl.doe.gov

Abstract

MFIX-Exa is a computational fluid dynamics–discrete element model (CFD-DEM) code designed to run efficiently on current and next-generation supercomputing architectures. MFIX-Exa combines the CFD-DEM expertise embodied in the MFIX code – which was developed at NETL and is used widely in academia and industry – with the modern software framework, AMReX, developed at LBNL. The fundamental physics models follow those of the original MFIX, but the combination of new algorithmic approaches and a new software infrastructure will enable MFIX-Exa to leverage future exascale machines to optimize the modeling and design of multiphase chemical reactors.

Class file, LaTeX 2_ε, *SAGE Publications*
CFD-DEM; MFIX; AMReX; HPC; exascale; ECP; embedded boundaries; multiphase; method-of-lines

MFIX-Exa: A Path Towards Exascale CFD-DEM Simulations

Jordan Musser1, Ann S. Almgren2, William D. Fullmer1,3, Oscar Antepara2, John B. Bell2, Johannes B

1 Introduction

In the United States, electricity production and industrial processes are responsible for approximately half of all greenhouse gas emissions, the bulk of which is attributed to the burning of fossil fuels [United States Environmental Protecti
Carbon capture and storage (CCS) technologies, such as oxy-fuel combustion, chemical looping combustion, and post combustion absorption systems, offer the most promising approaches for decreasing CO_2 emissions from fossil fuel power plants. Large-scale commercial deployment of CO_2 capture technologies requires understanding how to scale laboratory designs to industrial sizes. However, the direct scale up of the design of multiphase reactors, which are often at the core of these devices, is known to be unreliable. Consequently, the current approach is to build and test reactors at increasingly larger scales.

This paper describes the development of a high-fidelity computational tool capable of simulating dense particle-laden flows in systems with complex geometries so that high-performance computing can be used in place of physical testing at one or more intermediate scales. This would decrease both the time and capital investment needed to bring new technologies to market. This work specifically targets the scale-up of chemical looping reactors (CLR) by creating a massively parallel CFD-DEM code called MFIX-Exa, a complete modernization of the legacy NETL code, MFIX (`https://mfix.netl.doe.gov`).

The computational fluid dynamics-discrete element method (CFD-DEM) is a highly accurate way to model multiphase flows. DEM tracks individual particles and their interaction with the surrounding fluid. Unlike models that treat the particles as a continuum, or aggregate particles into parcels, DEM does not require constitutive relations for quantities that result from the averaging process, which are the greatest source of uncertainty. However, DEM is computationally expensive, requiring particle tracking and collision detection, coupling with the fluid phase (e.g., fluid-particle drag force), and evolution of the fluid fields. Additionally, long simulation times are required to accumulate sufficient data for meaningful statistical analysis of these highly transient flows. Consequently, DEM simulations are currently limited to laboratory-scale reactors containing a few million particles, whereas small pilot scale reactors contain billions of particles.

The two primary components of a CLR are the fuel and air reactors. In the fuel reactor, oxygen from particles, typically a metal oxide, is used in place of air to combust fossil fuels (e.g., methane). The spent oxygen carrier then travels to the air reactor where it is regenerated with oxygen from air. The replenished carrier particles return back to the fuel reactor, completing the chemical looping cycle.

The MFIX-Exa Exascale Computing Project (ECP) Challenge Problem is the simulation of NETL's pilot-scale 50kW CLR located in Morgantown, WV. A sketch of the system is given in Figure 1 along with an illustration of its operation. We estimate that the baseline computational load will contain approximately five billion particles and roughly half as many fluid (CFD) cells. A full solution requires calculation of 13 unknowns for every particle and 12 unknowns for every fluid cell, a significant computational footprint.

The computationally challenging aspect of simulating a CLR is the variety of flow regimes encountered as particles traverse the loop. For example, the riser is relatively dilute in terms of solids concentration (roughly 5% by volume), yet is characterized by high-density particle "clusters" that travel laterally and continuously break up and reform. The opposite is true in the bubbling bed fuel reactor, where flow instabilities instead take the form of dilute "bubbles" that travel upward through a dense particle bed (roughly 60% solids volume fraction). Both the clustering and bubbling instabilities are characterized by local particle concentrations that vary greatly in space and time. The cyclone and air reactor operate between these two regimes. Finally, the loop seal is characterized by very dense flow with sustained particle contacts, compared to the more intermittent nature of particle contacts elsewhere. This combination of flow regimes present in the CLR

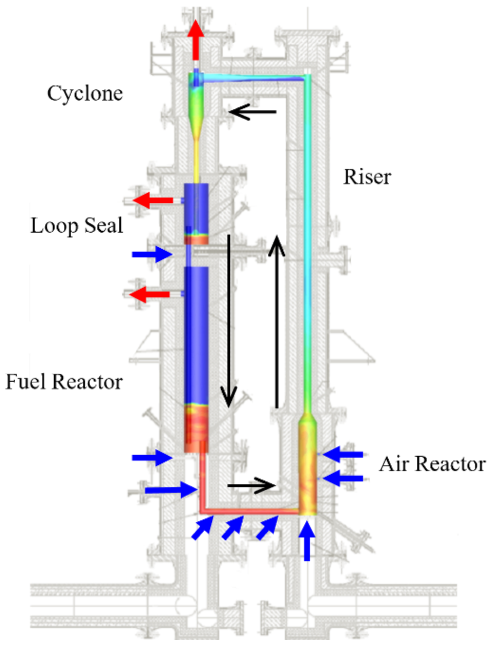


Figure 1: Illustration of the Challenge Problem: NETL's 50kW chemical looping reactor. Blue arrows indicate gas inflows, and red arrows indicate gas outflows. Black arrows show the counter-clockwise solids circulation path. Image adapted from [Bayham et al.(2016)Bayham, Weber, Straub and Breault]

presents the ultimate test for making DEM computationally efficient since (1) particle concentrations range from the very dilute to the very dense between different CLR components, and (2) within a given component, the particle concentration varies greatly in space and time due to flow instabilities.

In the following sections, we present the model equations followed by an introduction to AMReX. We then describe the general algorithmic approach to solving the fluid and particle equations for flows without chemical reactions or heat and mass transfer, followed by more detail for both the fluid algorithm and the particle operations. Finally, we give a short overview of verification / validation results to date.

2 Governing Equations

CFD-DEM is a class of hybrid Euler-Lagrange numerical methods for multiphase flows in which the continuous phase is discretized by a grid and the dispersed phase, e.g., solid particles, are modeled as discrete entities. Here and most commonly, CFD-DEM refers specifically to models in which the discrete phase is not fully resolved by the fluid mesh, as in the case of particle-resolved direct numerical simulation tenneti14, although the mesh spacing is typically on the scale of a few particle diameters in order to achieve mesh-insensitive solutions. For the systems of interest, e.g., as in Figure 1, the discrete particles are treated with a soft-sphere approach [van der Hoef et al.(2008)van der Hoef, van Sint Annaland, Deen and Kuipers, Deen et al.(2007)Deen, Annaland, Van der Hoef and Kuipers] which is well suited for the enduring collisions and multi-particle contacts encountered in dense regions. For the sake of brevity, the details of the collision model have been omitted here [see][]garg12,garg12b. In the remainder of this section, the governing equations for the fluid and particles are summarized, along with the Eulerian-Lagrangian coupling required to close the hybrid scheme.

2.1 Eulerian Gas Phase

CFD-DEM solves the particle-unresolved multiphase Navier-Stokes equations for the fluid (gas) phase. This work concentrates on cold-flow (no heat or mass transfer), low speed (low Mach number) gas-solids flows. The incompressibility assumption, $D\rho_g/Dt = 0$, is used to simplify the fluid mass and momentum conservation equations garg12 to

$$\frac{\partial \varepsilon_g}{\partial t} + \nabla \cdot (\varepsilon_g \boldsymbol{U}_g) = 0 \ , \tag{1}$$

and

$$\begin{aligned}\frac{\partial}{\partial t} (\varepsilon_g \rho_g \boldsymbol{U}_g) + \boldsymbol{\nabla} \cdot (\varepsilon_g \rho_g \boldsymbol{U}_g \otimes \boldsymbol{U}_g) = \\ - \varepsilon_g \boldsymbol{\nabla} p_g + \boldsymbol{\nabla} \cdot \boldsymbol{\tau}_g + \varepsilon_g \rho_g \boldsymbol{g} - \beta (\boldsymbol{U}_g - \boldsymbol{U}_p) \ ,\end{aligned} \tag{2}$$

where ρ_g, $\boldsymbol{U}_g$, and p_g are the gas-phase (thermodynamic) density, velocity vector and pressure. Gravity, $\boldsymbol{g}$, is the only body force acting on the system. The gas-phase viscous stress tensor is given by

$$\boldsymbol{\tau}_g = \mu_g \left[\nabla \boldsymbol{U}_g + (\nabla \boldsymbol{U}_g)^T \right] + \lambda_g (\nabla \cdot \boldsymbol{U}_g) \boldsymbol{I} \ , \tag{3}$$

where μ_g and $\lambda_g = -2\mu_g/3$ are the gas-phase dynamic and bulk viscosities. Unresolved turbulent stresses are neglected here. The phasic volume fraction (i.e. the fraction of space occupied by fluid as opposed to particles), ε_g, and interfacial momentum transfer terms related to β are computed from the Lagrangian field.

2.2 Lagrangian Particle Phase

The particles are governed by

$$\frac{d\boldsymbol{X}_p}{dt} = \boldsymbol{V}_p \ , \tag{4}$$

$$m_p \frac{d\boldsymbol{V}_p}{dt} = m_p \boldsymbol{g} + \boldsymbol{F}_{gp} + \boldsymbol{F}_{wp} + \sum_{q=1}^{N_p} \boldsymbol{F}_{qp} \ , \tag{5}$$

$$I_p \frac{d\boldsymbol{\omega}_p}{dt} = \boldsymbol{T}_{wp} + \sum_{q=1}^{N_p} \boldsymbol{T}_{qp} \ , \tag{6}$$

where $\boldsymbol{X}_p$, $\boldsymbol{V}_p$, and $\boldsymbol{\omega}_p$ are the discrete position, and linear and angular velocity vectors, respectively, of the p-th particle. In this work, all particles are assumed to be spherical so that the volume, mass and moment of inertia of a particle are given by

$$\mathcal{V}_p = \frac{\pi}{6} d_p^3 \ , \ \ m_p = \rho_p \mathcal{V}_p, \ \ \text{and,} \ \ I_p = \frac{1}{10} m_p d_p^2 \ ,$$

where d_p and ρ_p are the p-th particle diameter and material density, respectively. In addition to gravity, other possible forces acting on the p-th particle include transfer with the interstitial gas, $\boldsymbol{F}_{gp}$, and contact forces resulting from collisions with wall boundaries, $\boldsymbol{F}_{wp}$, and other particles, $\boldsymbol{F}_{qp}$. Change in any p-th particle's angular velocity are driven by contact torques resulting from collisions with wall boundaries $\boldsymbol{T}_{wp}$, and other particles, $\boldsymbol{T}_{qp}$. MFIX-Exa uses a soft-sphere type linear-spring dashpot (LSD) model cundall79 for collisions. Details of the LSD contact force models can be found elsewhere, e.g., [see][]garg12,garg12b.

2.3 Phasic Coupling

Closure of the governing equations requires the coupling of the Eulerian and Lagrangian models. Generally, volume filtering capecelatro13 is used to transfer the Lagrangian data onto the Eulerian grid. For example, particle volume is mapped to a continuous volume fraction field by

$$\varepsilon_p(\boldsymbol{x}) = \sum_{p=1}^{N_{tot}} \mathcal{G} (\|\boldsymbol{x} - \boldsymbol{X}_p\|) \mathcal{V}_p \ , \tag{7}$$

where $\mathcal{G}$ is the (unit normal) transfer kernel. Direct application of Eq. (7) is computationally prohibitive for large problems. In practice, $\mathcal{G}$ is compact so that only a small subset of N_{tot} particles local to $\boldsymbol{x}$, N_{loc}, contribute to the transfer operation. The specifics of the deposition algorithm are provided in the Transfer kernel section.

Because the particle interface is not resolved by the CFD-grid, a closure model for interfacial momentum transfer is required. Due to the high density ratios and large Stokes numbers of our target applications, only buoyancy and steady drag forces are considered

$$\boldsymbol{F}_{gp} = -\chi_p(\boldsymbol{\nabla} p_g)\mathcal{V}_p - f_{dp}\left(\boldsymbol{V}_p - \chi_p(\boldsymbol{U}_g)\right) \ . \tag{8}$$

In Eq. (8), χ_p is an interpolation operator that maps a continuous fluid property, here p_g and $\boldsymbol{U}_g$, to the p-th particle position. The numerical details of χ_p are also provided in the Transfer kernel section. The linearized drag coefficient in Eq. (8), f_{dp}, is given by

$$f_{dp} = \frac{1}{2} C_d \rho_g \left\| \boldsymbol{V}_p - \chi_p(\boldsymbol{U}_g) \right\| A_{proj} \ . \tag{9}$$

Due to the assumption of spherical particles, the projected area of the p-th particle is simply $A_{proj} = \pi d_p^2/4$. Three standard multi-particle drag laws have been implemented to close C_d in Eq. (8): `WenYu` wen66, `Gidaspow` ding90, lathouwers01, and `BVK2` beetstra07, tang15. The drag force, when transferred to the grid, becomes

$$\begin{aligned} \beta &= \sum_{p=1}^{N_{loc}} \mathcal{G}\left(\|\boldsymbol{x} - \boldsymbol{X}_p\|\right) f_{dp} \\ \beta \boldsymbol{U}_p &= \sum_{p=1}^{N_{loc}} \mathcal{G}\left(\|\boldsymbol{x} - \boldsymbol{X}_p\|\right) f_{dp} \boldsymbol{V}_p \ . \end{aligned} \tag{10}$$

The equivalent continuum buoyancy force, $(1-\varepsilon_g)\boldsymbol{\nabla} p_g$, is combined with the gas-phase pressure gradient, leading to the $-\varepsilon_g$ coefficient in Eq. (2), i.e., the particle buoyancy force is not explicitly deposited.

It should be noted that the interphase transfer terms are only absolutely conservative in theory. In practice, time marching of the particles and gas-phase is asynchronous and at different time scales, resulting in minor discrepancies in the interfacial forces experienced by the two fields. Finally, the fluid and particles advances are executed sequentially. As outlined in the following sections, the fluid is updated first followed by the particle advance. Consequently, the fluid sees a "frozen" particle field, such that the incompressibility constraint of Eq. (1) may be further reduced to

$$\boldsymbol{\nabla} \cdot (\varepsilon_g \boldsymbol{U}_g) = 0. \tag{11}$$

due to the fixed volume fraction field, i.e., $\partial \varepsilon_g / \partial t = 0$.

3 AMReX

MFIX-Exa is based on AMReX, a software framework that provides the data structures and iterators to enable massively parallel simulation in single- and multi-level domain-decomposed geometries. In addition, AMReX supplies some of the higher-level operations that are used in the time-stepping of the fluid and particles.

In this section we focus primarily on the AMReX functionality for mesh data that is used by the MFIX-Exa code. In particular, while AMReX is designed to support data and operations across multiple levels of refinement, MFIX-Exa is currently run in a single-level mode so we focus on those capabilities. AMReX support for particle data and operations is in the Particle Algorithm section. Additional detail about how AMReX handles memory management, parallel reductions, and multilevel operations can be found in [Zhang et al.(to appear)Zhang, Myers, Gott, Almgren and Bell].

3.1 Mesh Data

AMReX provides a flexible set of data structures that can be used to represent block-structured mesh data in a distributed memory environment. The fundamental AMReX mesh data structure used by MFIX-Exa is the MultiFab which holds array data on a BoxArray, a union of non-intersecting boxes, aka grids, at a single level of refinement. The array container for a single grid is known as a FAB, short for FArrayBox. When running across multiple ranks, the BoxArray at each level is known on all ranks, but the data itself is distributed. A single rank may hold multiple FABs, but a single FAB lives on only one rank.

An FArrayBox contains the Box that defines its valid region, and a pointer to a multidimensional array. For three-dimensional calculations, the data accessed by the pointer in an FArrayBox is a four-dimensional array. The first three dimensions correspond to the spatial dimensions; the fourth dimension is for components. An important feature of an FArrayBox is that extra space is allocated to provide space for ghost cell data, which is often required for efficient stencil operations. Ghost cell data are typically filled by interpolating from coarser data, copying from other FArrayBoxes at the same level, or by imposing boundary conditions outside the domain. The data in an FArrayBox (including ghost cells) are stored in a contiguous chunk of memory with the layout of the so-called struct of arrays (SoA) (i.e., struct of 3D arrays, one for each component).

AMReX provides a class template, Array4<T>, that can be used to access the data in an FArrayBox or other similar objects. This class does not own the data and is not responsible for allocating or freeing memory; it simply provides a reference to the data. This property makes it suitable for being captured by a C++ lambda function without memory management concerns.

The Array4 class has an operator() that allows the user to access it with Fortran multi-dimensional array like syntax. The SoA layout of FArrayBox is typically a good choice for most applications because it provides unit stride memory access for common operations.

Several steps of the MFIX-Exa fluid advance use stencil operations that require access to the data on neighboring and/or nearby cells. When that data lies outside the valid region of the patch being operated on, pre-filling ghost cells is an efficient way to optimize communication. The Boxes stored in MultiFabs determine the "valid" region. The multidimensional array associated with each FArrayBox covers a region larger than the "valid" region by nGrow cells on the low and high sides in each direction, where nGrow is specified when the MultiFab is created. AMReX provides basic communication functions for ghost cell exchanges of data in the same FabArray. It also provides routines for copying data between two different FabArrays at the same level. Data between different MPI ranks are organized into buffers to reduce the number of messages that need to be sent.

3.2 On-node parallelism

MFIX-Exa is designed to run on a variety of platforms, including individual workstations, many-core HPC platforms such as NERSC's Cori, and accelerator-based supercomputers such as Summit and Frontier at OLCF and Aurora at ALCF. AMReX provides various features that enable MFIX-Exa and other codes to obtain high performance on different architectures without substantial recoding.

AMReX uses a hierarchical parallelism model. At a coarse-grained level, the basic AMReX paradigm is based on distribution of one or more patches of data to each node with an owner-computes rule to allocate tasks between nodes. For many use cases, a node is divided into a small number of MPI ranks and the coarse-grained distribution is over MPI ranks. For example, on a system with 6 GPUs per node, the node would typically have 6 MPI ranks.

For code executing on CPUs, AMReX supports logical tiling for cache re-use using OpenMP threading. Tile size can be adjusted at run time to improve cache performance; tile size can also vary between operations. AMReX includes both a standard synchronous strategy for scheduling tiles as well as an asynchronous scheduling methodology. AMReX also provides extensive support for kernel launching on GPU accelerators (using C++ lambda functions) and for the effective use of managed memory, that allows users to control where their data is stored.

MFIX-Exa exploits the abstraction layer provided by AMReX that consists of a number of ParallelFor looping constructs, similar to those provided by Kokkos kokkos or RAJA raja but tailored to the needs of block-structured AMR applications. MFIX-Exa code contains loop bodies written as C++ lambda functions that define the task to be performed over a set of cells or particles. The lambda function is then used in a kernel with a launching mechanism provided by CUDA, HIP or DPC++ for GPU or standard C++ for CPU.

When code using the ParallelFor construct is compiled to execute on CPUs with OpenMP, the MFIter loop includes tiling and the ParallelFor translates to a serial loop over the cells in a tile. However, when compiling for GPU platforms, tiling at the `MFIter` level is switched off, and the `ParallelFor` translates to a GPU kernel launch.

3.3 Linear Solvers

AMReX includes native geometric multigrid and Krylov (CG and BiCG) solvers for nodal or cell-centered data, as well as interfaces to external solvers such as those in hypre hypre-paper and PETSc petsc-web-page. The external solvers can be called at the finest multigrid level, or as a "bottom solve" where the native solver coarsens one or more levels, then calls the external solver to reduce the residual by a specified tolerance at that level.

In each fluid time step, MFIX-Exa uses AMReX multigrid solvers to solve three different types of equations: the variable coefficient elliptic solve in the MAC projection (see Convective update), which solves for a single variable located at cell centers; a viscous tensor solve – which solves the coupled system corresponding to discretizing the full viscous stress tensor implicitly (see Diffusive update); and an approximate projection that is used to update the pressure on nodes and the velocity at cell centroids (see Nodal projection).

The multigrid algorithm itself is encapsulated in the `MLMG` class. Any `MLMG` object can be constructed with a specific linear operator class, depending on the linear system to be solved. The AMReX classes `MacProjector` and `NodalProjector` provide algorithm-specific interfaces to the linear operator and `MLMG` classes.

The AMReX multigrid solvers include aggregation (merging boxes at a level in the multigrid hierarchy to enable additional coarsening within the V-cycle) and consolidation (reducing the number of ranks to reduce communication costs at coarser multigrid levels) strategies to reduce total cost.

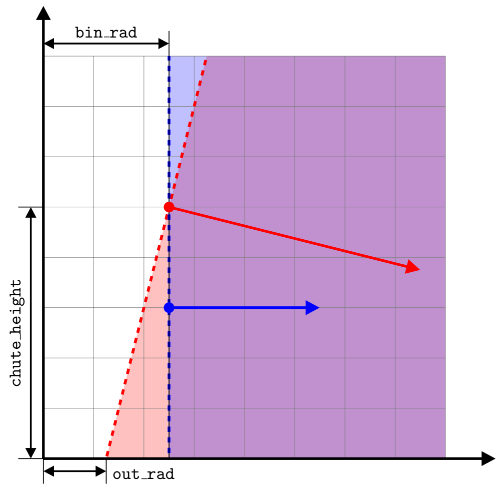
4 Embedded Boundary Method for Complex Geometry

An embedded boundary (also known as "cut cell") formulation is used to represent the non-rectangular bounding geometry in the domain. The irregular boundary is defined by intersecting an analytically-specified boundary with a uniform Cartesian grid, with irregularly shaped cells appearing only adjacent to the boundary. The EB information is precomputed and stored in a distributed database at the beginning of the calculation.

Following standard notation, we define each grid cell (i, j, k) to be either *covered*, *cut*, or *regular*. We define the geometric volume fraction, Λ, (not to be confused with the phasic volume fraction, ε_g) of each cell to be the fraction of that rectangular cell volume that is inside the fluid/particle region: *covered* cells have $\Lambda = 0$, *regular* cells have $\Lambda = 1$, and for cut cells $0 < \Lambda < 1$. Area fractions are stored on each cell face, again with values in $[0, 1]$ representing the fraction of the face not covered. Finally, the location of the cell centroid (which for regular cells is identical to the cell center), and the locations of the face centroids are stored. Additionally, there is connectivity information between neighboring cells. For additional details on the embedded boundary implementation, we refer to the AMReX documentation `https://amrex-codes.github.io/amrex/docs_html/`. Algorithm-specific adaptations for cut cells are discussed in each algorithm's subsection.

4.1 Complex Geometry Generation

The simulation geometry – i.e., the embedded boundary – is specified using AMReX implicit functions with operations like *union*, *intersect*, and *translate*. A listing of the available shapes and operations is provided in the AMReX documentation. MFIX-Exa makes several basic geometries (e.g., cylinder and box) and advanced geometries accessible to users by specifying the `mfix.geometry` parameter in the project input file. Additional geometries can be incorporated into MFIX-Exa, however this requires modifying the source code to define a geometry using AMReX implicit functions. The following example illustrates how to construct a simple cylindrical hopper using the AMReX embedded boundary capabilities. First, the hopper's chute and bin are defined by planes given by a point and a normal. This is illustrated in Figure 2a where the red dashed line defines the chute wall, and the blue dashed line is the bin wall. The normals point into the solid (away from the fluid) so that taking the union of the two solid regions defines a half-hopper geometry (e.g.,



(a)

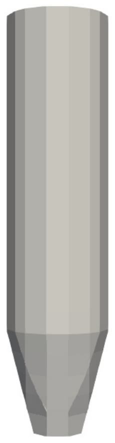

(b)

Figure 2: Example of complex geometry generation. (a) The chute (red) and bin (blue) of a cylindrical hopper are defined as planes. Their union –the red, blue, and purple regions– identify the covered region of the domain. (b) The 2D geometry is lathed about the y-axis to create the 3D hopper geometry.

the white filled region). Finally, the 2D shape is rotated about the y-axis to create the final geometry, shown in Figure 2b. The source code needed to generated the hopper geometry is provided in Listing 1.

```
 // Arrays for plane point and normal
 Array<Real,3> point, normal;

 // Construct plane representing funnel (red)
 point = {outlet_rad, chute_height, 0.0};
 normal = {chute_height, out_rad-bin_rad, 0.0};
EB2::PlaneIF chute(point, normal);

 // Construct plane representing the body (blue)
 point = {bin_rad, 3.0, 0.0};
 normal = {3.0, 0.0, 0.0};
EB2::PlaneIF bin(point, normal);

 // Union body and funnel and hopper planes
 auto hopper2D = EB2::makeUnion(chute, bin);

 // Lathe to create 3D geometry
 auto hopper3D = EB2::lathe(hopper2D);
```

Listing 1: Example of constructing a cylindrical hopper using AMReX implicit functions and geometric operations.

4.2 Mesh Pruning

When the region enclosed by the EB surfaces is such that parts of the domain are completely covered, the flexibility of the AMReX gridding strategy allows the use of "mesh pruning" to reduce the memory required by a simulation. Mesh pruning eliminates fully covered grids at each level, i.e. grids in which all cells are covered (in the EB sense).

The mesh pruning strategy implemented in MFIX-Exa first defines a `BoxArray` on which the geometry is initially defined with all cells defined as regular, cut or covered. Then, using this information, MFIX-Exa creates a second `BoxArray` in which all covered grids are removed from the initial `BoxArray`. The original `BoxArray` covering the full rectangular domain is no longer relevant; all data is now allocated on the smaller region.

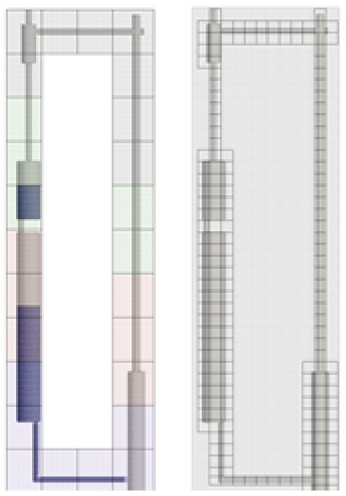


Figure 3: *Left:* Example of Mesh Pruning. Volume rendering of the mesh volume fraction > 0 covered by the (pruned) grids. The figure on the right has a finer mesh resolution; the savings from mesh pruning increase with resolution.

The purpose of this mesh pruning strategy is reduction in memory rather than reduction of computational time. The computational load itself is not reduced since the usual load balancing strategies, and the `MultiFab` iterators, effectively ignore `FABs` in which there is no work. However, without the mesh pruning strategy, the covered grids on which no work is done still require the same amount of memory as if they were regular grids (except for temporary arrays that would be constructed for specific operations). In cases where the simulation is memory-limited, mesh pruning enables us to run larger calculations for the same number of nodes.

Figure 3 shows two examples of pruned meshes; in both cases the fluid and particles are entirely contained within the region enclosed by the EB walls.We note that the mesh reduction happens primarily on the interior of the domain; we can not reduce the memory footprint by shrinking the domain size when solving in geometries in which most of the "empty" space lies in the interior.

5 Fluid Algorithm

In this section we present the predictor-corrector fluid time-stepping algorithm and details on the computation of the convective and diffusive terms. Additionally, the use of specific AMReX solver classes is highlighted throughout.

5.1 Time-stepping algorithm

The fluid velocity is defined at cell centroids, and is advanced from time t^n to t^{n+1} using a single timestep, dt_{CFD}. The dynamic pressure gradient is also defined at cell centroids, but the pressure itself is defined at nodes; both are calculated at half-time steps, e.g. at time $t^{n+1/2}$.

The fluid equations are discretized using a predictor-corrector method-of-lines approach with an approximate projection; details of each step are given below. The reason we use this predictor-corrector approach is to ensure the fluid algorithm is second order accurate in time. For simplicity of notation we use $dt = dt_{CFD}$ below.

5.1.1 Predictor:

The predictor step begins with defining an approximation ${\boldsymbol{U}_g}^{P,*}$ (where the superscript P stands for "predictor") to the new-time velocity $\boldsymbol{U}_g^{n+1,*}$ by solving

$$\begin{aligned}(\rho_g\ \varepsilon_g\ + dt\ \beta^n)\ {\boldsymbol{U}_g}^{P,*}\ &= (\rho_g \varepsilon_g {\boldsymbol{U}_g}^n) \\ &- dt\ \rho_g \mathbf{A}^n_{\boldsymbol{U}_g} + dt\ \mathbf{D}(\boldsymbol{U}_g^n) \\ &- dt\ \varepsilon_g \mathbf{G}{p_g}^{n-1/2} + dt\ \varepsilon_g \rho_g \boldsymbol{g} + dt\ \beta^n \boldsymbol{U}_p^n\end{aligned}$$

Here $\mathbf{A}^n_{\boldsymbol{U}_g}$ is a discretization of $\nabla \cdot (\varepsilon_g \boldsymbol{U}_g^n \otimes \boldsymbol{U}_g^n)$; $\mathbf{G}{p_g}^{n-1/2}$ is an approximation to $\nabla {p_g}^{n-1/2}$, and $\mathbf{D}(\boldsymbol{U}_g^n)$ is an approximation to $(\nabla \cdot \boldsymbol{\tau}_g^n)$. Here the superscript "n" represents the state at time t^n. After solving for $\boldsymbol{U}_g^{P,*}$, we do an approximate nodal projection to enforce the constraint (11). The results of the nodal projection operation are $p_g^{n+1/2,*}$, that is an approximation to the new nodal gas pressure, its nodal gradient approximation $\mathbf{G}p_g^{n+1/2,*}$, and the new velocity $\boldsymbol{U}_g^{n+1,*}$ which is defined as follows and satisfies the divergence constraint (11):

$$\boldsymbol{U}_g^{n+1,*} = \boldsymbol{U}_g^{P,*} - \frac{dt}{\rho_g} \mathbf{G}p_g^{n+1/2,*}.$$

The details of constructing the advective terms in addition to the details of the explicit and implicit construction of the viscous terms, and the approximate nodal projection, are provided in later sections.

5.1.2 Corrector:

In the corrector step, we define a new approximation $\boldsymbol{U}_g^{C,**}$ (where the superscript C stands for "corrector") to the new-time velocity $\boldsymbol{U}_g^{n+1}$ using time-centered advective and viscous updates and pressure gradient

$$\begin{aligned}\rho u^{C,**} &= \rho u^{C,*} \\ &+ \frac{dt}{2}\left(\mathbf{D}(\boldsymbol{U}_g^{C,**}) - \mathbf{D}(\boldsymbol{U}_g^n)\right),\end{aligned} \tag{12}$$

where the intermediate approximation $\boldsymbol{U}_g^{C,*}$ is computed solving the following equation

$$\begin{aligned}(\rho_g {\boldsymbol{U}_g}^{C,*} = (\rho_g \varepsilon_g {\boldsymbol{U}_g}^n) \\ &- \frac{dt\ \rho_g}{2}\ \left(\mathbf{A}^n_{\boldsymbol{U}_g} + \mathbf{A}^{P,*}_{\boldsymbol{U}_g}\right) + dt\ \mathbf{D}(\boldsymbol{U}_g^n) \\ &- dt\ \varepsilon_g \mathbf{G}{p_g}^{n+1/2,*} + dt\ \varepsilon_g\ \rho_g\ \boldsymbol{g} + dt\ \beta^{n,*} \boldsymbol{U}_p^n.\end{aligned}$$

The new-time approximation to the advective term, $\mathbf{A}^{P,*}_{\boldsymbol{U}_g}$, is calculated using $\boldsymbol{U}_g^{n+1,*}$. The drag coefficient $\beta^{n,*}$ is a mixed time level term, evaluated using particle data at time t^n and fluid data at time $t^{n+1,*}$. To enforce the divergence constraint (11) on $\boldsymbol{U}_g^{C,**}$, we proceed similarly to the predictor step. We apply an approximate nodal projection that computes the new-time approximate nodal gas pressure $p_g^{n+1/2}$, its nodal approximate gradient $\mathbf{G}p_g^{n+1/2}$, and the new gas velocity $\boldsymbol{U}_g^{n+1}$ which satisfies the constraint (11) and is defined as follows:

$$\boldsymbol{U}_g^{n+1} = \boldsymbol{U}_g^{C,**} - \frac{dt}{\rho_g}\left(\mathbf{G}p_g^{n+1/2} - \mathbf{G}p_g^{n+1/2,*}\right).$$

The details of constructing the advective and viscous terms, and performing an approximate nodal projection of the solution, are provided in the subsequent sections.

5.2 Convective update

The convective update is a multistep algorithm based on a second-order upwind, method of lines approach. The basic idea is to first compute normal velocities on faces that satisfy the divergence constraint discretely. The resulting velocity field, referred to as U_g^{MAC} is then used to compute the convective update of all three velocity components

$$A_{U_g} = \nabla \cdot (\varepsilon_g U_g^{MAC} \otimes U_g)$$

Details of the procedure are discussed below.

For each face with a non-zero area fraction, we first extrapolate the normal velocity component from the centroids of the cells on either side to the face centroid, creating left (L) and right (R) states. When both cells are regular, we compute limited slopes in each cell and define, in the x-direction as an exemplar,

$$\begin{aligned} u^L_{i+1/2,j,k} &= u_{i,j,k} + \frac{dx}{2}\,(u_x)_{i,j,k} \\ u^R_{i+1/2,j,k} &= u_{i+1,j,k} - \frac{dx}{2}\,(u_x)_{i+1,j,k} \end{aligned}$$

where u_x is an approximation to $\partial u/\partial x$.

When one of the cells on either side is a cut cell, we instead use a least squares fit centered on (i,j,k) that uses all regular and cut-cell neighbors, compute slopes in all three coordinate directions. We then define the left and right states by extrapolating from the cell centroid to the face centroid using slopes in all three coordinate directions as necessary.

We then upwind the left and right states to define a unique value of the normal velocity on each face, which we call $\boldsymbol{U}_g^{MAC,*}$. To enforce the constraint (11), we apply a discrete projection by solving the following elliptic equation

$$D^{\mathrm{MAC}}\left(\frac{\varepsilon_g}{\rho_g}\mathbf{G}^{\mathrm{MAC}}\phi^{\mathrm{MAC}}\right) = D^{\mathrm{MAC}}\left(\varepsilon_g \boldsymbol{U}_g^{\mathrm{MAC},*}\right)$$

for ϕ^{MAC}. Here, D^{MAC} represents the divergence at cell centroids of area-weighted velocities defined at face centroids, and $\mathbf{G}^{\mathrm{MAC}}$ represents the gradient at face centroids of data at cell centers. The density ρ_g and volume fraction ε_g are interpolated separately onto face centroids. The solution ϕ^{MAC}, which is defined at cell centers, is then used to compute the constraint-compliant velocity $\boldsymbol{U}_g^{\mathrm{MAC}}$ defined as

$$\boldsymbol{U}_g^{\mathrm{MAC}} := \boldsymbol{U}_g^{\mathrm{MAC},*} - \frac{1}{\rho_g}\mathbf{G}^{\mathrm{MAC}}\phi^{\mathrm{MAC}} \quad .$$

This projection is done using the AMReX EB-aware `MacProjector` class, whose method `MacProjector::project` takes a vector field V and a specified RHS, and returns a vector field that satisfies $\nabla \cdot V = RHS$.

We then extrapolate all three velocity components from cell centroids to face centroids to define $\boldsymbol{U}_g^{adv}$, using $\boldsymbol{U}_g^{MAC}$ in the upwinding procedure to define a unique value. We multiply $\varepsilon_g \boldsymbol{U}_g^{adv}$ by $\boldsymbol{U}_g^{MAC}$, using the interpolated ε_g, to construct fluxes for the momentum equation. The divergence of each cell's area-weighted face-centroid-based fluxes, along with a "redistribution" operation which addresses the classic "small cell problem", defines $\mathbf{A}_{\boldsymbol{U}_g}$ at cell centroids.

5.3 Diffusive update

In the predictor step, we calculate the diffusive term $\mathbf{D}(\boldsymbol{U}_g)$ (approximating $\nabla \cdot \boldsymbol{\tau}_g$) explicitly, followed by a redistribution step for cut cells with small volume fraction analogous to that used to construct the convective update. In the corrector step we solve for the diffusive term implicitly; this does not require redistribution.

The discretization of the tensor operator uses the same divergence operator as in the MAC projection. Unlike the MAC projection, however, in which the contribution from EB faces is identically zero because the boundary condition for the solution variable (an update to pressure) satisfies a homogeneous Neumann condition there, in calculating the diffusive term we must include the contributions from the EB face in each cut cell. The boundary condition on EB faces is Dirichlet (since the velocity, not its gradient, is specified at no-slip boundaries); we currently calculate the normal gradient at the EB face by interpolating the velocity to a point along the normal from the EB face centroid into the fluid region, and computing the difference between the interpolated value at that point and the value on the EB face centroid.

We note here that although the gas itself is assumed to be incompressible, we cannot use the simplified form of the viscous stress tensor (which results from exploiting $\nabla \cdot \boldsymbol{U}_g = 0$) because the divergence constraint satisfied by the fluid is in fact $\nabla \cdot (\varepsilon_g \boldsymbol{U}_g) = 0$.

In conjunction with the `MLMG` class, we use the `MLEBTensorOp` class, which is specifically implemented for solving linear systems $L^D \boldsymbol{V} = \mathbf{f}$ of the following type

$$L^D \boldsymbol{V} := a\boldsymbol{V} - b(\nabla \cdot (\eta \nabla \boldsymbol{V}) + \nabla \cdot (\eta \nabla \boldsymbol{V}^T) \\ + \nabla \cdot ((\kappa - \frac{2}{3}\eta)(\nabla \cdot \boldsymbol{V})\boldsymbol{I}) \ ,$$

where η is the dynamic viscosity, κ is the bulk viscosity, and $\boldsymbol{I}$ is the identity tensor. In our case we set $\eta = \mu_g$ and $\kappa = 0$.

In the predictor we use `MLMG::apply` to compute $\mathbf{D}(\boldsymbol{U}_g^n) = L^D V$ for $a = 0$, $b = -1$ and $V = \boldsymbol{U}_g$. In the corrector, we construct a right-hand-side

$$\mathbf{f} = \rho_g \varepsilon_g {\boldsymbol{U}_g}^{C,*} - \frac{dt}{2}\mathbf{D}(\boldsymbol{U}_g^n)$$

then solve $L^D \boldsymbol{V} = \mathbf{f}$ with $a = \rho_g \varepsilon_g$, $b = dt/2$ using `MLMG::solve` to satisfy Eq. (12).

5.4 Nodal projection

In both the predictor and corrector, we use an approximate nodal projection to enforce the divergence constraint (11) on the predicted velocities at time t^{n+1}. (The approximateness of the projection refers to the fact that after the projection, the divergence of the resultant velocity field is only approximately zero; $\nabla \cdot (\varepsilon_g \boldsymbol{U}_g^{n+1}) \approx 0$ to second order accuracy in the mesh spacing.) Specifically, in the predictor step we solve

$$L^{\varepsilon}_{\rho} \, p_g^{n+1/2,*} = D^{nodal} \left(\frac{\varepsilon_g}{dt} \boldsymbol{U}_g^{P,*} + \frac{\varepsilon_g}{\rho_g} \mathbf{G} {p_g}^{n-1/2} \right) \ ,$$

for $p_g^{n+1/2,*}$, while in the corrector step we solve

$$L^{\varepsilon}_{\rho} \, p_g^{n+1/2} = D^{nodal} \left(\frac{\varepsilon_g}{dt} \boldsymbol{U}_g^{C,**} + \frac{\varepsilon_g}{\rho_g} \mathbf{G} {p_g}^{n+1/2,*} \right) \ .$$

for $p_g^{n+1/2}$. In both equations, D^{nodal} is a divergence operator that returns a nodal divergence of cell-centered vector fields, and L^{ε}_{ρ} is the standard bilinear finite element approximation to the operator $D^{nodal}\left(\varepsilon_g/\rho_g \ \nabla(\bullet)\right)$ (see almgrenBellSzymczak:1996 for a detailed discussion of this approximate projection; see AlmgrenBell-Crutchfield for a discussion of this particular form of the projection operand).

MFIX-Exa uses the AMReX EB-aware `NodalProjector` class to solve the variable coefficient elliptic equations for the pressure. Specifically, the `NodalProjector` class implements functions for solving problems of the following type:

$$D^{nodal}\left(\sigma \nabla \phi\right) = D^{nodal}\left(\boldsymbol{V}^*\right) - S \ ,$$

for the unknown ϕ, and then sets

$$\boldsymbol{V} = \boldsymbol{V}^* - \sigma \nabla \phi \ ,$$

where the vector field $\boldsymbol{V}$ (approximately) satisfies $D^{nodal}\left(\boldsymbol{V}\right) = S$. Setting $S = 0$ enforces the divergence constraint (11). In our case $\sigma = \varepsilon_g/\rho_g$ and $\boldsymbol{V}^* = \varepsilon_g \hat{\boldsymbol{U}}_g + (dt \, \varepsilon_g/\rho_g)\mathbf{G}\hat{p}_g$, where $\hat{\boldsymbol{U}}_g = \boldsymbol{U}_g^{P,*}$ and $\hat{p}_g = p_g^{n-1/2}$ for the predictor, and $\hat{\boldsymbol{U}}_g = \boldsymbol{U}_g^{C,**}$ and $\hat{p}_g = p_g^{n+1/2,*}$ for the corrector.

6 Particle Algorithm

The particles are advanced from t^n to t^{n+1} after the fluid advance. Most often with soft-sphere DEM models, the timestep is taken as a (user-defined) fraction of the collisional time scale, τ_{coll} [e.g. see][]garg12,fullmer19u. For the low Mach number flows of interest, the collisional time scale is typically much smaller than the fluid time step, i.e., the particles are *subcycled* several times per fluid dt. Therefore, we use a simple forward Euler method for the particle subcycling.

Recalling equations Eq. (4-6), for each substep we first sum the forcing terms in the linear and angular momentum equations for each particle. We then update the linear and angular velocity components, and use the new linear velocity to update the particle position. In the rest of this section we describe how MFIX-Exa uses the AMReX particle data structures and iterators to efficiently calculate particle-particle collisions and particle-wall collisions, and to redistribute particles to different MPI ranks with minimal communication cost.

6.1 Particle Data Structures and Iterators

AMReX provides data structures for representing and iterating over particles defined on a patch-based AMR hierarchy. The core particle data structure in AMReX is the `ParticleContainer`, which stores a collection of particles associated with a given `Vector<BoxArray>` and `Vector<DistributionMapping>`. The `BoxArray`s describe the set of patches at each level, and the `DistributionMapping`s describe how each patch is assigned to an MPI rank. Note that, while particles are always associated with a set of patches, this does not necessarily need to be the same set of patches used to store the mesh data in a simulation.

Particles in AMReX always have position coordinates and a unique 64-bit integer identifier. In addition, each `ParticleContainer` can be configured to store additional `Real` (either single- or double- precision floating point) and integer components, laid out in either Array-of-Structs or Struct-of-Arrays format. In MFIX-Exa, particles carry floating point information about their size, mass, velocity, angular momentum, moment of inertia, and information about the drag force between them and fluid, as well as integer components representing their state and phase.

At each level, particles are assigned to a cell index by binning their positions using the level's cell spacing and the global, physical domain offset. They are then assigned to grids based on which `Box` in the `BoxArray` contains that cell. All the particles on a level can then be accessed and iterated over on a grid-by-grid basis, with the guarantee that all the particles associated with a given grid will have positions that map to cells inside it. To reinforce this condition after the particle positions have been updated, we call the `Redistribute()` method in AMReX that puts all particles back on the correct level, grid, and MPI task.

6.2 Particle-Particle interactions

Identifying pairs of particles that are within each other's interaction radii can be computationally expensive. To ameliorate this, MFIX-Exa uses a neighbor list algorithm, in which potential collision partners over the next N substeps are pre-computed and stored for each particle. Once this is done, during the time advance we can quickly loop over only those pairs of particles that could possibly interact with each other.

To construct neighbor lists, we use a standard "cell-list" approach. First, particles are binned into cells, with a cell spacing that is related to the particle interaction distance. Then, for each particle, we perform a direct search only over neighboring cells, looking for particles that are within the appropriate cutoff distance. Particles that pass this check are added to the neighbor list for each particle.

The implementation details differ based on whether we are using OpenMP or CUDA/HIP/DPC++ as the compute backend. For OpenMP, each grid is subdivided into tiles, with each OpenMP thread processing the particles on one or more tiles independently. For the GPU case, we use a parallel prefix sum operation (`amrex::Scan`) to bin the particles into cells. We then make two passes over the bin data structure. In the first pass, we count the number of neighbors for each particle, so that the appropriate amount of memory can be allocated. Then, in the second pass, we actually fill in the neighbor indices for each particle.

For the above process to work, each grid needs to have obtained copies of all the particles that are within some number of cells of its valid region. The AMReX terminology for these copies of particles from other grids is "neighbor particles", which are obtained by calling the AMReX function `FillNeighbors()`. See Figure 4 for an illustration of this operation. As with `Redistribute()`, all required parallel communication is performed automatically. When using neighbor particles, it is common to compute a halo of particles (i.e., identify which particles need to be copied to which other grids) once with some extra padding, and then reuse it for several time steps before computing a new halo. To enable this, AMReX separates computing the halo and filling it with data for the first time (the `FillNeighbors()` operation) from reusing the existing halo but replacing the data with the most recent values from the "valid" particles on other grids (the `UpdateNeighbors()` operation).

6.3 Particle-Wall interactions

The forcing terms due to particle-wall collisions are computed separately from the particle-particle collision terms. In the classic MFIX-DEM implementation, particle-wall collisions were identified by iterating over planar facets (STL elements) and computing the distance between each particle and each facet. By contrast, MFIX-Exa constructs a level set, defined on nodes of the mesh, that captures the distance from each node to the nearest wall. Particle-wall collisions can then be computed with the distance of the particle to the wall

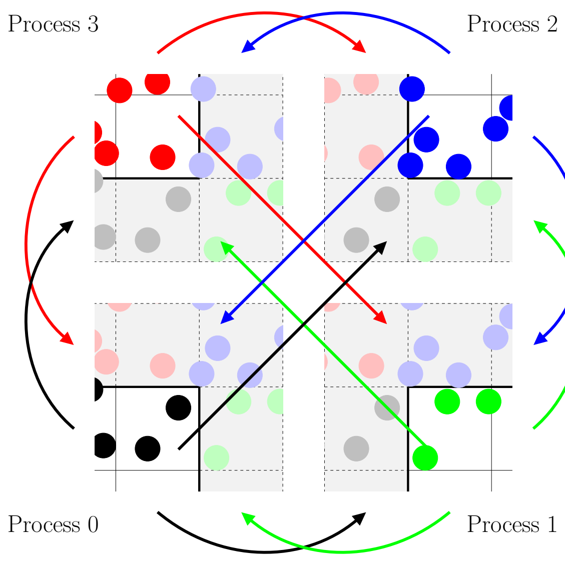


Figure 4: Illustration of the `FillNeighbors()` operation with four grids assigned to four different processes. Particles belonging to each grid are illustrated with different colors. When `FillNeighbors()` is called, particles that reside within a one-cell wide "halo" region of each grid are identified and copies of them are sent to the correct MPI process. Periodic boundary conditions are also applied at this stage, so that a naive distance computation between "valid" and "ghosted" particles on each grid will return the shortest distance between them in the periodic domain. The "ghosted" particles can then be included in neighbor lists and collision calculations.

computed by interpolating the level set values at the nodes of the cell holding the particle, and the normal to the wall at the collision point defined by the gradient of the level set. In practice, for planar boundaries such as those discussed here, the distance calculation is identical except at edges and corners where a slight "rounding" occurs due to the resolution of the level set. When required to adequately represent nonplanar geometry, the level set can be defined at a finer level than the rest of the simulation in order to exploit a finer representation of the geometry for calculating collisions.

7 Particle-Fluid Coupling

7.1 Transfer kernel

MFIX-Exa uses a trilinear transfer kernel with a compact stencil, S_p, of 2^3 fluid cells when interpolating fluid properties (e.g., velocity) to a particle, and when depositing particle information (e.g., particle volume) onto the fluid grid. Using the x-axis as an example, the high side stencil index is

$$i = \lfloor (X_p - D_{low,x})/dx + 0.5 \rfloor \tag{13}$$

where X_p and dx are the the particle position and grid spacing, and $D_{low,x}$ is the physical location of the low side of the domain. The j and k indices are calculated similarly. The cell indexing of the stencil is

$$S_p = \begin{Bmatrix} 0:(i-1, j-1, k-1), & 4:(i-1, j-1, k), \\ 1:(i\quad, j-1, k-1), & 5:(i\quad, j-1, k), \\ 2:(i\quad, j\quad, k-1), & 6:(i\quad, j\quad, k), \\ 3:(i-1, j\quad, k-1), & 7:(i-1, j\quad, k) \end{Bmatrix}$$

The centroids of the cell in S_p can be thought of as the corners (nodes) of a generalized hexahedral (brick); therefore, the isoparametric mappings found in finite element methodologies can be used hughes2012. Specifically, for the eight cells in S_p the weight functions are

$$\begin{aligned}
w_0(\xi,\eta,\zeta) &= 1-\xi-\eta-\zeta+\xi\eta+\xi\zeta+\eta\zeta-\xi\eta\zeta \\
w_1(\xi,\eta,\zeta) &= \xi-\xi\eta-\xi\zeta+\xi\eta\zeta \\
w_2(\xi,\eta,\zeta) &= \xi\eta-\xi\eta\zeta \\
w_3(\xi,\eta,\zeta) &= \eta-\xi\eta-\eta\zeta+\xi\eta\zeta \\
w_4(\xi,\eta,\zeta) &= \zeta-\xi\zeta-\eta\zeta+\xi\eta\zeta \\
w_5(\xi,\eta,\zeta) &= \xi\zeta-\xi\eta\zeta \\
w_6(\xi,\eta,\zeta) &= \xi\eta\zeta \\
w_7(\xi,\eta,\zeta) &= \eta\zeta-\xi\eta\zeta
\end{aligned} \tag{14}$$

where $(x,y,z)\to(\xi,\eta,\zeta)$ is the coordinate transformation from physical space to the unit cube with its origin at $(0,0,0)$. Thus, a fluid property, α_g, interpolated to a particle position is given by

$$\alpha_p \equiv \chi_p(\alpha_g) = \sum_{m\in S_p} w_m\alpha_m \ , \tag{15}$$

and a particle property, β_p, is deposited into the m-th cell in S_p through

$$\beta_m \equiv \mathcal{G}_m\left(\|\boldsymbol{x}-\boldsymbol{X}_p\|\right)\beta_p = w_m\beta_p \tag{16}$$

When all the cells in S_p are regular, the coordinate transformation is easily calculated,

$$\begin{aligned}
\xi\left(x,y,z\right) &= \frac{X_{p,x}-x_0}{dx}, \\
\eta\left(x,y,z\right) &= \frac{X_{p,y}-y_0}{dy}, \\
\zeta\left(x,y,z\right) &= \frac{X_{p,z}-z_0}{dz},
\end{aligned} \tag{17}$$

where (x_0,y_0,z_0) are the physical coordinates of the 0-th indexed cell centroid in S_p, and again, dx, dy, dz are the grid spacings. Additional considerations are needed when interpolating and depositing near the embedded boundary (e.g., one or more cells in S_p is cut or covered). These are addressed below.

7.1.1 Interpolation near EB.

If none of the cells in the interpolation stencil S_p are covered, then there is sufficient information to interpolate fluid properties to the particle. However, the coordinate transformation $(x,y,z)\to(\xi,\eta,\zeta)$ requires solving the non-linear system of equations

$$\begin{aligned}
f_1(\xi,\eta,\zeta) &= x_0-X_{p,x}+x_1\xi+x_2\eta+x_3\zeta+x_4\xi\eta \\
&\qquad +x_5\xi\zeta+x_6\eta\zeta+x_7\xi\eta\zeta \\
f_2(\xi,\eta,\zeta) &= y_0-X_{p,y}+y_1\xi+y_2\eta+y_3\zeta+y_4\xi\eta \\
&\qquad +y_5\xi\zeta+y_6\eta\zeta+y_7\xi\eta\zeta \\
f_3(\xi,\eta,\zeta) &= z_0-X_{p,z}+z_1\xi+z_2\eta+z_3\zeta+z_4\xi\eta \\
&\qquad +z_5\xi\zeta+z_6\eta\zeta+z_7\xi\eta\zeta
\end{aligned} \tag{18}$$

This is done using a Newton-Raphson solver Kincaid2002 with the initial guess given by equation (17). However, if one or more cells in S_p are covered, then there is not sufficient information to use trilinear interpolation, therefore one of the following methods are used. For general fluid properties like volume fraction, the property is assumed constant and the value of cell containing the particle centroid is taken. For fluid velocity, a non-slip boundary condition is imposed and a 1D linear interpolation is performed from the centroid of the cell containing the particle centroid and the embedded boundary.

7.1.2 Deposition near EB.

There are two concerns when depositing particle information near the embedded boundary: (a) particle data should not get deposited into covered cells, and (b) the particle volume deposited into a cut cell should not exceed physical limits. These two cases are illustrated in Figure 5 where the stencil for particle-a (blue) contains a covered cell, and the stencil for particle-b (red) includes a small cut cell. To prevent depositing particle data into covered cells, the weight for each cell is multiplied by a mask, σ, that is zero if the cell is covered, and one otherwise. Then, all weights are normalized by the total weight sum,

$$\hat{w}_m = \frac{w_m \sigma_m}{\sum\limits_{n \in S_p} w_n \sigma_n} .$$

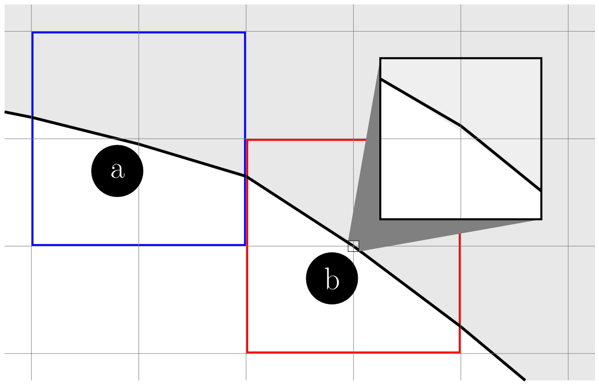


Figure 5: Special consideration is needed for particles near embedded boundaries. Areas occupied by the fluid are shown with a white background, the embedded boundary surface is shown with a black line, and the covered/blocked area of the domain is shaded gray. The general interpolation stencil for particle-a would deposit material outside the domain if the upper-right cell is retained in the interpolation stencil, S_p. Particle-b may result in nonphysical volume fractions and excessive drag forces on the fluid if deposition into the small cell, again the upper-right cell in the stencil, is unbounded.

For cut cells that have a particle volume fraction – the ratio of deposited particle volume to actual cell volume – that exceeds a user-defined threshold, $\varepsilon_p > \varepsilon_{p,\max}$, excess particle volume is transferred to the 3^3 neighborhood of cells, S_ϕ. Again, a masking function, σ, is used to exclude covered cells and cut cells exceeding the maximum particle volume fraction, to prevent transferring material from one over-packed small cell to another. First, a local averaged particle volume fraction is calculated,

$$\bar{\varepsilon}_p = \frac{\sum\limits_{m \in S_\phi} \sigma_m \Lambda_m \varepsilon_{p,m}}{\sum\limits_{m \in S_\phi} \sigma_m \Lambda_m}$$

where $\varepsilon_{p,m}$ is the initial deposited particle volume fraction and, as before, Λ_m is the geometric volume fraction for the m-th cell. To minimize the local gradient, the new volume fraction for the cut cell is taken as the smaller of the neighborhood average and user-defined maximum, $\varepsilon_p^* = \min\{\bar{\varepsilon}_p, \varepsilon_{p,\max}\}$. Next, the amount of excess particle volume is computed,

$$\delta\varepsilon_p = \frac{\max\left\{\varepsilon_p - \varepsilon_p^*, 0\right\}}{\sum\limits_{m \in S_\phi} \hat{w}_m} ,$$

transferred to the neighborhood cells,

$$\varepsilon_{p,m} = \varepsilon_{p,m} + \sigma_m \delta\varepsilon_p \qquad \forall m \in S_\phi ,$$

and the new value of the cut-cell is imposed, $\varepsilon_p = \varepsilon_p^*$. This approach is done in the context of particle volume because a maximum value can be reasonably approximated (e.g., close-packing of spheres can be used to guide the selection of $\varepsilon_{p,\max}$). To maintain consistency, particle volume is included in the deposition of all particle properties (e.g., drag force) to guide redistribution.

7.2 Implementation in AMReX

The above particle-mesh operations are handled by the AMReX framework. A deposition operation is used to represent the volume fraction of the solid phase on the mesh (Equation 7). Likewise, drag forces between the fluid and solid phases are calculated via an analogous mesh-to-particle interpolation (Equation 8). To parallelize these operations, there are two aspects to consider; the “on-node” parallelization, which is implemented using either OpenMP or CUDA/HIP/DPC++ depending on whether MFIX-Exa is compiled with GPU support, and the “off-node” component, in which information must be exchanged between MPI ranks. We consider each of these in turn.

In the “on-node” operation, we are concerned with interpolating between the particles and the mesh on a single grid. In AMReX, these operations are expressed by having multiple threads (either on the CPU or the GPU) loop over the particles simultaneously, accessing the appropriate mesh cells independently. In the case of deposition, a potential complication is that race conditions can occur when multiple particles interact with the same cell. Our strategy for dealing with these race conditions differs depending on whether we are using GPUs. For the CPU case, we divide each grid into multiple tiles, and pre-sort the particles onto them so that the particles on each tile can be accessed separately. Each CPU thread is then responsible for processing a set of tiles. To handle deposition, each thread allocates its own private, tile-sized deposition buffer into which it can deposit particles without needing to worry about race conditions. Then, once all the particles on the tile have been deposited, the deposition buffer is added back to the main `FArrayBox` using atomic writes. Thus, atomics are only needed on a per-cell basis, not a per-particle basis. When running on GPUs, on the other hand, we do not use any deposition buffers, instead writing directly to global memory with atomic operations. This approach works surprisingly well on V100 GPUs, particularly if periodic sorting onto tiles is employed to exploit the GPU’s memory hierarchy. Evaluating and, if necessary, modifying this approach for AMD and Intel hardware is ongoing work.

Once each grid is processed, we also must handle ghost cells between adjacent cells. For mesh-to-particle interpolation, prior to interpolation, we simply call `FillBoundary()` with the appropriate number of ghost cells to fully capture the support of all the particles with positions inside the grid. For deposition, we call the analogous `SumBoundary()` operation in AMReX after performing the deposition on each grid. This function takes the values in the ghost cells of each grid and adds them to corresponding valid cells. By calling this method after deposition, we ensure that particles partially deposited into ghost cells have their full contribution counted.

When the particles and the fluid data are defined on the same set of grids, the above `FillBoundary` and `SumBoundary` operations are the only MPI communications required. However, in situations where the particles are non-uniformly distributed, it can be advantageous to use a different domain decomposition for the fluid and the particles. In this case, temporary `MultiFab`s are defined to store the fluid velocity and volume fraction data on the particle grids, and an additional `ParallelCopy` must be performed to copy this information back and forth between the two decompositions.

8 Performance Assessment

Performance assessments of the MFIX-Exa code have been limited thus far; however, a pair of weak scaling studies have been conducted to establish a baseline scalabililty metric for the fully GPU-enabled code. This weak scaling study uses one of the simplest multi-particle gas-solid systems of relevance: the homogeneous cooling system (HCS). The HCS is defined entirely by its initial state: particle locations are statistically uniform, the particle velocity magnitude is characterized by a Maxwellian distribution (i.e. a thermal speed but no mean motion), the fluid is at rest, and there are no external forces nor boundary conditions acting on the system (i.e. triply periodic and $\boldsymbol{g} = 0$). As time evolves, particle kinetic energy dissipates (i.e., ”cools”) due to inelastic collisions and fluid-particle drag.

The single-grid serial problem size considers a 64^3 fluid mesh with mesh spacing $2d_p$. The particle and fluid properties are: $\rho_p = 1000$ kg/m^3, $d_p = 100$ microns, $\rho_g = 1$ kg/m^3 and $\mu_g = 2 \times 10^{-5}$ kg/m-s. Two variants are considered which differ only the number of particles: a dilute case with 40K particles per grid (1% solids concentration by volume) and a dense case with 1,200K particles per grid (30% solids concentration by volume). We expect that the ideal particle work load per GPU will be somewhere in this range, although GPU computation is still somewhat in its infancy as applied to CFD-DEM simulation.

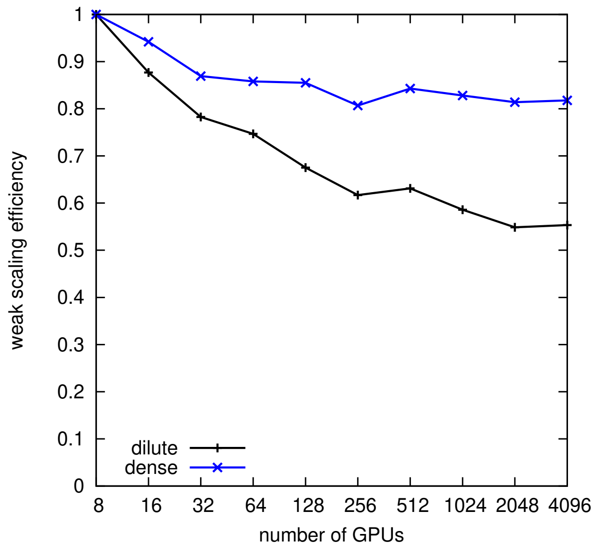


Figure 6: Weak scaling efficiency of MFIX-Exa for dense and dilute HCS simulations.

We define a sequence of problem sizes, starting with 2^3 grids, and replicate the initial data from a checkpoint file to define the initial conditions for these runs. Weak scaling performance data is collected starting from the problem with 2^3 grids run on 8 ranks; this is successively doubled in the x-, then y-, then z-directions up to a 4096-rank problem with 16^3 grids. Each problem size is simulated for 100 (fluid) steps on OLCF's Summit supercomputer at Oak Ridge National Laboratory. Each simulation is repeated five times, the best of which (smallest average time per step) is reported in Figure 6. It is evident that the dense HCS scales better than the dilute HCS, indicating that the particle work has been offloaded to the GPU more efficiently than the fluid work. The average percentage of fluid and particle work per step is approximately 73.7% and 23.2% in the 8-rank dilute case and in the dense case shifts to 11.5% and 88.0%, respectively. The remaining time is in coupling (drag force calculation and deposition), which is small in both cases. The baseline results indicate that performance gains can be realized; however, the ideal nature of the HCS does not require load balancing or mesh pruning, which are essential for the challenge problem of Figure 1 and other complex reactor configurations.

9 Software Quality Assurance Program

In this work we use the term software quality assurance (SQA) as a blanket term to cover all code testing practices that touch on performance, regression tests, algorithm verification and model validation. Similar to the AIAA's validation tiers aiaa98, we recognize that SQA is layered and multifaceted and perhaps best tackled with a hierarchical approach. The SQA program we have developed, and indeed are still developing, spans a range of problem complexity and computational overhead which vary inversely with the frequency with which the tests are preformed. In the following, the SQA hierarchy is broken down into levels of testing frequency, "continuous," daily, weekly, annually, singularly, with an example problem for each level.

9.1 Continuous integration

The MFIX-Exa source code and case files for more frequent tests are hosted in NETL's GitLab repository. MFIX-Exa has a single branch for code development, `develop`, and bi-monthly tags pointing to specific instances of `develop`. All contributors fork a personal copy of the main repository. Active development takes place on the personal forks and changes to the code base are introduced by means of merge requests. Each merge request initiates a pipeline job that is run through a series of Continuous Integration (CI) tests before merging. The CI tests first build CPU and NVIDIA GPU (CUDA enabled) solvers, checking for compiler warnings or errors. After successful builds, a test suite is then carried out that includes five particle-only tests, three fluid-only tests, and two coupled tests; all of the fluid tests contain single-grid serial, multi-grid serial and multi-grid parallel variants. Furthermore, all tests are setup to run in each coordinate direction to verify solution symmetry. All tests are small and quick-running, as the CI pipeline may be run several times a day. The CI test suite is built and run in the cloud, using a docker image on Amazon Web Service servers.

Example: Planar Poiseuille Flow. This is a simple and classic problem. We consider (laminar) pressure-driven fluid flow between horizontal plane walls. The length and width of the domain are $L_x = L_y = 0.01\,\mathrm{m}$, and the depth is $L_z = L_x/2$. The flow is periodic in the x- and z-directions with pressure drop of $DP = 12\,\mathrm{Pa}$. No-slip walls are placed at elevations of $y = 1.25 \times 10^{-6}\,\mathrm{m}$ and $y = L_y - 1.25 \times 10^{-6}\,\mathrm{m}$. The fluid viscosity is $\mu_g = 0.001\,\mathrm{Pa\,s}$. The domain is resolved by a coarse $8 \times 8 \times 4$ fluid grid and the result of the MFIX-Exa numerical solution is compared to the analytical solution,

$$u(y) = \frac{DP}{L_x} \frac{y}{2\mu_g} \left(L_y - y\right) \ ,$$

in Fig. 7

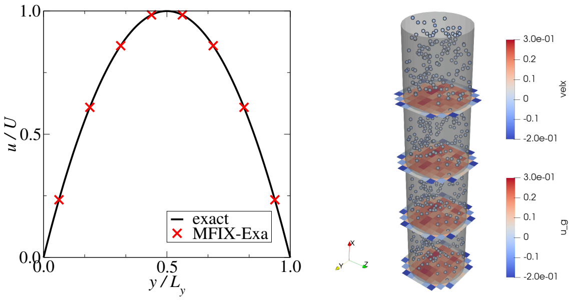


Figure 7: *Left*: Simple Poiseuille flow of the CI tests compared to the analytical solution. *Right*: Sketch of the cylindrical riser of the nightly regression tests, particles and fluid slices colored by streamwise velocity.

9.2 Nightly regression tests

Regardless of the level of development activity, a regression test harness is executed nightly. The nightly test problems are typically larger in size than the CI tests, therefore the additional computational overhead is mitigated by running the tests for only a limited number of timesteps, typically less than 100. There are two test harnesses, one for CPU and one for NVIDIA GPU, each consisting of benchmark problems with particles and fluid. Additionally, four of the particle-only CI tests are also run. Plot files containing fluid and particle states are written at the end of the brief simulations and a bit-wise comparison is made with archived benchmark results.

Example: Cylindrical riser. As shown on the right of Figure 7, 620 particles having diameter of 0.1 mm are pseudo-randomly initialized in a 2 mm diameter cylinder. The streamwise boundary is periodic with an imposed pressure drop of 2 Pa over the riser length of 8 mm. Both fluid and particles are initially at rest and the simulation evolves over 100 steps. The solution time is also collected and compared to a running average of previous tests.

9.3 Weekly regression tests

Often in the development process, changes to the code base are anticipated to produce changes in the solution and the benchmarks simply need to be updated. However, it is often difficult to determine if the changes have adversely affected the prediction. Approximately once a week, a subset of the nightly regression tests are run for a longer period of time, typically a few seconds of physical time are modeled, which takes up to a few hours of wall clock time. The results are compared to benchmark MFIX-Classic solutions of the same system. In this case, differences exist and the difficulty lies in ascertaining just how much code-to-code discrepancy is acceptable. However, it does provide a quick check that the code is predicting behavior similar to the legacy code for some of the most fundamental problems of interest. The quantity of interest for each problem is time averaged into a scalar and stored. The most recent twelve results are compared to the averaged MFIX-Classic result and a table is populated linking the index to the code's specific Git SHA-1 hash and the date the regression test was run.

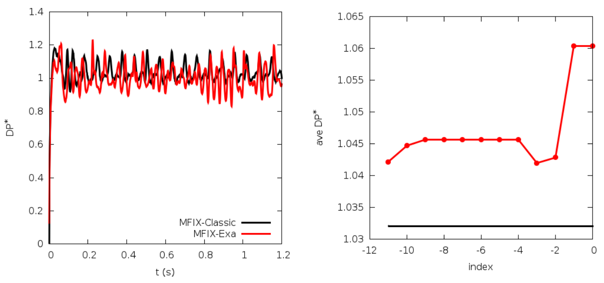


Figure 8: The dimensionless bed pressure drop of the square fluidized bed in the weekly regression test compared to a benchmark MFIX-Classic result of the same system. Instantaneous results of a single test (left) and the time averaged values of the most recent twelve tests (right).

Example: Rectangular Fluidized Bed A simple fluidized bed in a rectangular domain of width and depth $L_y = L_z = 2\,\text{mm}$, and length $L_x = 8\,\text{mm}$ is discretized by a $40 \times 10 \times 10$ fluid mesh. The lower 5.6 mm section of the bed is filled with approximately 12000 particles of $d_p = 0.1\,\text{mm}$ diameter. The inflow velocity at $x = 0\,\text{m}$ is 15 mm/s. The outflow is zero constant pressure and the remaining boundary conditions are bounding no-slip walls. The quantity of interest is the cross-sectional averaged pressure drop between the inflow and exit planes normalized by the weight of the bed over the area, provided on the left of Figure 8. The last 1.0 s of simulation is time averaged and compared to previous results on the right of Figure 8, showing generally good agreement between the two codes.

9.4 Validation problems

As a further step up on complexity, specifically in simulation time and data processing, we consider validation problems: real, physical systems for which experimental data has been collected. Validation experiments are smaller in size, "bench-scale," so that accurate measurements may be taken. In addition to experimental data, MFIX-Exa predictions are also code-to-code benchmarked against MFIX-Classic predictions. The validation suite fullmer19u currently consists of three rectangular fluidized beds goldschmidt03, mueller09, gopalan16 and one spout-fluid bed link08. The validation problem suite has been run once a year and takes roughly one week to compile results. All required inputs, data processing scripts and supporting text is maintained in a separate repository.

Example: Müller's Bed. The fluidized bed experiments of [Müller et al.(2008)Müller, Holland, Sederman, Scott, Dennis Müller et al.(2009)Müller, Scott, Holland, Clarke, Sederman, Dennis and Gladden] have been widely used for CFD-DEM validation, including the original MFIX-Classic implementation [Li et al.(2012)Li, Garg, Galvin and Pannala]. The experiments consist of a thin, "pseudo-2D" clear bed of width $L_x = 44\,\text{mm}$, height $L_y = 176\,\text{mm}$, and depth $L_z = 10\,\text{mm}$, which is discretized by a uniform $16 \times 64 \times 4$ fluid mesh. The bed is filled with 9240 poppy seeds approximately $d_p = 1.2\,\text{mm}$ in diameter and a density of $\rho_p = 1000\,\text{kg/m}^3$. Moisture in the seeds allows for magnetic resonance imaging to measure high-frequency particle concentration and velocity contours of the 2-D (depth averaged) field. The bed is fluidized by ambient air at superficial velocities of $U/U_{mf} = 2$ and 3. Only the latter is considered here. Constant zero pressure is specified at the outlet and all vertical walls are no-slip. MFIX-Exa monitors are used to collect particle concentration and velocity in spatially-averaged zones for a simulation time of 55 s, the first 5 s being neglected and the remaining 50 s used to compute time averages and confidence intervals from ten non-overlapping batch means.

Figure 9 compares the MFIX-Exa predictions to two MFIX-Classic models and the experimental data at $U = 3\,U_{mf}$. In general, we observe good code-to-code agreement and reasonable experimental agreement. The most obvious deficiency being the over-estimation of the near-wall void fraction at the upper elevation. This modeling error is consistent with both the original simulation results mueller09 and the MFIX-Classic

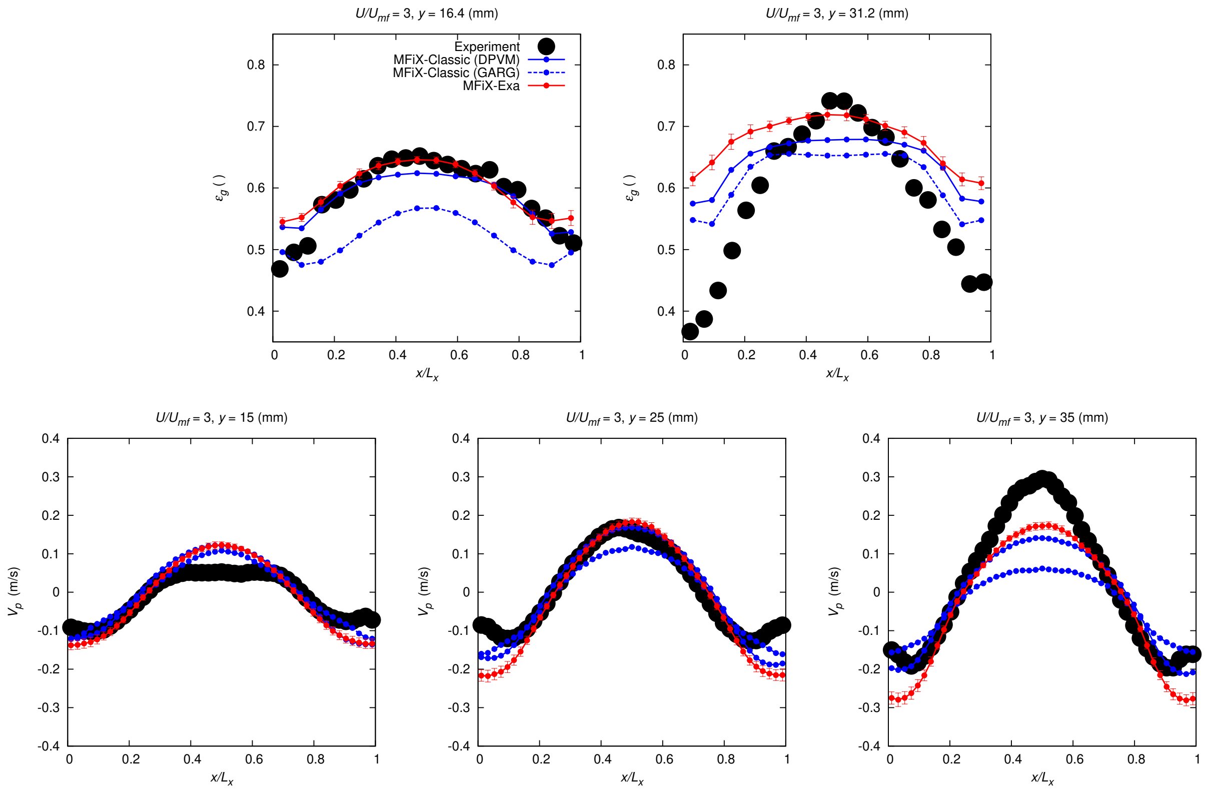


Figure 9: Time-averaged mean void fraction profiles (top row) and streamwise particle velocity at $U = 3U_{mf}$ in fluidized bed of mueller09.

validation study [Li et al.(2012)Li, Garg, Galvin and Pannala]. The velocity profiles show a positive velocity in the center as particles move up with slugs and a negative velocity near the walls as particles move laterally as the bubbles rupture at top of the bed and then fall back down along the walls. The simulated profiles tend to under-predict the measured centerline velocity at the highest elevation, which is also consistent with previous CFD-DEM studies.

9.5 Demonstration problems

Finally, at the top of the hierarchy, are large-scale, computationally intensive problems that more fully demonstrate the code's capabilities. Associated experimental measurements may be sparse and coarse (e.g., pressure signals at a few locations) or not available at all, i.e., true prediction. Due to the computational expense and time commitment, demonstration problems tend to be one-off exercises and are not regularly repeated. Unlike regression testing, application specific problems have the potential for contributions from an extended user-base beyond the core code development team, as is the case of the predecessor code. While MFIX-Exa has reached a level of maturity for undertaking demonstration problems, development remains very active and the code has not yet been publicly released. Hence, large scale applications have been limited thus far.

Example: Spouted bed. A spouted bed was recently constructed and operated at NETL that spanned a range of flow conditions, bed weights and two different materials [Monazam et al.(2018)Monazam, Breault, Weber and Layfiel Here, we are concerned with the pure spouted bed (i.e., no additional fluidizing gas) of high-density polyethylene (HDPE) particles of diameter $d_p = 870\,\mu\text{m}$ and density $\rho_p = 855\,\text{kg/m}^3$. The spout velocity is increased and the quantity of interest is U_{ms}, the minimum spout velocity (converted into a bed superficial velocity) that forms an external spout through the bed. The bed is simulated with MFIX-Exa with a $40 \times 512 \times 40$ fluid mesh with mesh spacing $dx = 2.814\,\text{mm}$. The fluid grid is chosen so that the cross-sectional area of one cell is equal to the spout area, i.e., $dx = \sqrt{\pi} d_j/2$, where d_j is the jet diameter in the experimental unit. The

Table 1: Comparison of the minimum spout velocity between experimental observations, MFIX-Exa simulation and the correlation of mathur55.

Experiment	15.42 cm/s
MFIX-Exa	12.65 cm/s
Eq. (19)	12.56 cm/s

$D = 0.1009\,\mathrm{m}$ diameter bed is off-set in the domain; centered at $x = z = 0.054\,873\,\mathrm{m}$, which corresponds to the center of a grid cell that is assigned the spout inlet boundary condition. The spout velocity increases linearly in time at a rate of $10\,\mathrm{m/s^2}$. The bed is initialized by fluidizing a larger than required bed mass through a uniform mass inlet for 1 s, then settling with no flow for 1 s, and finally removing all particles with centroids above the reported static bed height of $h_0 = 0.1524\,\mathrm{m}$, approximately 2.2 million particles.

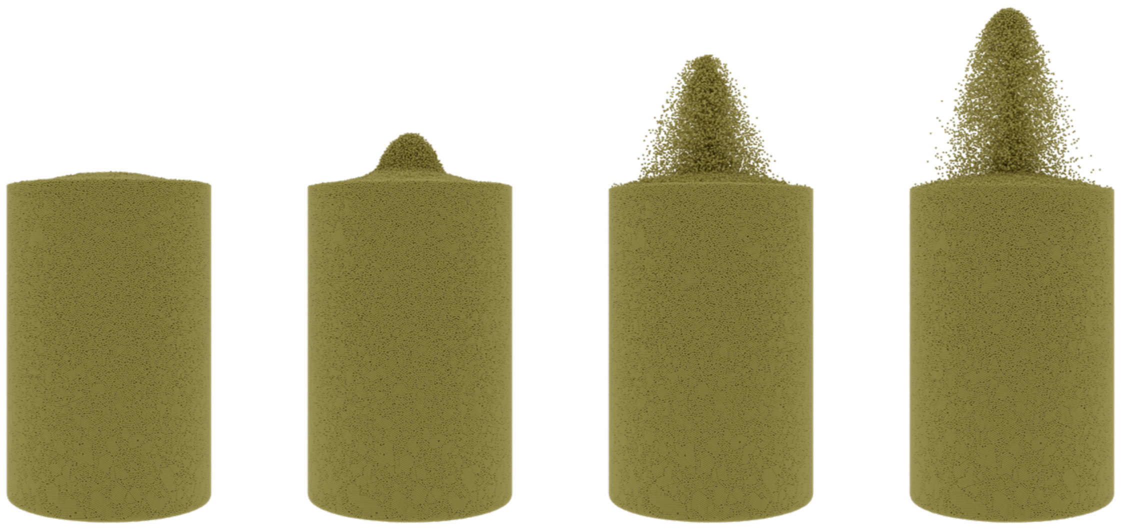


Figure 10: Visualization of the spouted bed just before (far left), just after (second from left) spout break-through and the rising external spout (second and far right). Blender (`blender.org`) renderings courtesy of Justin Weber (NETL).

During the simulation, the bed pressure is monitored near the inlet, which, consistent with experimental observations, increases nearly linearly in time (proportional to the spout velocity) until it reaches a maximum and then drops off sharply to a minimum and then remains nearly constant as the spout breaks through the surface of the bed. Using the pressure signal to identify the key time period, the particle data was saved and visualised to identify the exact moment of break-through, as depicted in Figure 10. The identified break-through time is converted into U_{ms} and reported in Table 1. We find that the predicted U_{ms} value is approximately 18% lower than the experimentally measured value. However, it should be noted that this discrepancy is on the order of the uncertainty of the drag law, where here we have used used the model of wen66 without calibration. Further, the empirical correlation of mathur55,

$$U_{ms} = \left(\frac{d_p}{D}\right)\left(\frac{d_j}{D}\right)^{1/3}\sqrt{\frac{2\,|\boldsymbol{g}|\,h_0\,(\rho_p - \rho_g)}{\rho_g}}, \tag{19}$$

was found to agree well with the larger data set. Equation (19) gives a U_{ms} in very good agreement with the numerical prediction, also provided in Table 1.

10 Future Work

Presently, work is underway to extend the fluid and particle models presented here to include heat and mass transfer with interphase exchanges. This requires advecting density, species concentrations, and enthalpy for the fluid, while tracking a unique temperature and composition for each particle. The fluid will be treated as a multicomponent ideal gas and a new constraint will be used to ensure the evolution of the density and temperature are consistent with the equation of state.

Additionally, a new solids model is being implemented that combines several particles into a single parcel and replaces the particle collision model with a continuum stress formulation. Although less accurate, the dense-multiphase particle-in-cell (PIC) approach is far more computationally efficient as there is less overall work (fewer parcels) and the communication heavy neighbor-search routines are not needed. The goal is to execute coarse, fast running simulations with this lower fidelity PIC model, and use the result to 'bootstrap' the initial conditions for CFD-DEM simulations.

Finally, several algorithmic advances for the fluid solve are being explored to reduce the number of linear solves needed per fluid advance while retaining second order accuracy in time.

11 Acknowledgements

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. Research performed by Leidos Research Support Team staff was conducted under the RSS contract 89243318CFE000003. This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[Bayham et al.(2016)Bayham, Weber, Straub and Breault] Bayham S, Weber JM, Straub D and Breault RW (2016) Performance of a raw hematite and a manufactured copper-iron oxygen carrier in a 50-kw natural gas chemical looping system. Technical Report NETL-PUB-20850, NETL. URL https://www.osti.gov/servlets/purl/1599158.

[Deen et al.(2007)Deen, Annaland, Van der Hoef and Kuipers] Deen N, Annaland MVS, Van der Hoef MA and Kuipers J (2007) Review of discrete particle modeling of fluidized beds. *Chemical Engineering Science* 62(1-2): 28–44.

[Li et al.(2012)Li, Garg, Galvin and Pannala] Li T, Garg R, Galvin J and Pannala S (2012) Open-source mfix-dem software for gas-solids flows: Part ii–validation studies. *Powder Technology* 220: 138–150.

[Monazam et al.(2018)Monazam, Breault, Weber and Layfield] Monazam ER, Breault RW, Weber J and Layfield K (2018) Minimum spouting velocity of flat-base spouted fluid bed. *Particuology* 36: 27–36.

[Müller et al.(2008)Müller, Holland, Sederman, Scott, Dennis and Gladden] Müller CR, Holland DJ, Sederman AJ, Scott SA, Dennis JS and Gladden LF (2008) Granular temperature: comparison of magnetic resonance measurements with discrete element model simulations. *Powder Technology* 184(2): 241–253.

[Müller et al.(2009)Müller, Scott, Holland, Clarke, Sederman, Dennis and Gladden] Müller CR, Scott SA, Holland DJ, Clarke BC, Sederman AJ, Dennis JS and Gladden LF (2009) Validation of a discrete element model using magnetic resonance measurements. *Particuology* 7(4): 297–306.

[United States Environmental Protection Agency(2017)] United States Environmental Protection Agency (2017) Sources of greenhouse gas emissions. URL https://19january2017snapshot.epa.gov/ghgemissions/sources-greenhouse-gas-emissions_.html.

[van der Hoef et al.(2008)van der Hoef, van Sint Annaland, Deen and Kuipers] van der Hoef M, van Sint Annaland M, Deen N and Kuipers J (2008) Numerical simulation of dense gas-solid fluidized beds: A multiscale modeling strategy. *Annual Review of Fluid Mechanics* 40: 47–70.

[Zhang et al.(to appear)Zhang, Myers, Gott, Almgren and Bell] Zhang W, Myers A, Gott K, Almgren A and Bell J (to appear) Amrex: Block-structured adaptive mesh refinement for multiphysics applications. *IJHPCA* .

Jordan Musser is a Research Scientist at the National Energy Technology Laboratory in Morgantown. His primary research interest is in computational methods for dense particle laden flows, and their application to large industrial systems.

Ann Almgren is a Senior Scientist at Lawrence Berkeley National Laboratory, and the Group Lead of LBL's Center for Computational Sciences and Engineering. Her primary research interest is in computational algorithms for solving PDE's in a variety of application areas. Her current projects include the development and implementation of new multiphysics algorithms in high-resolution adaptive mesh codes that are designed for the latest hybrid architectures. She is a Fellow of the Society of Industrial and Applied Mathematics and the Deputy Director of the AMReX Co-Design Center.

William Fullmer is a Research Scientist with the Leidos Research Support Team under the Research Support Services contract within the Research and Innovation Center at the National Energy Technology Laboratory, Morgantown. His research interests span a wide range of applied computational multiphase flow. He is an alumnus of Purdue University's School of Nuclear Engineering and the University of Colorado Boulder's Hrenya Research Group.

Oscar Antepara is a Postdoctoral Scholar in the Center for Computational Science and Engineering at Lawrence Berkeley National Laboratory. His expertise lies in the development and analysis of numerical methods related to computational fluid dynamics. His current research interests are in computational algorithms for multiphase flows using high-performance computing (HPC) systems. Oscar holds a Ph.D. in Engineering from the Technical University of Catalonia-BarcelonaTech (UPC), where his research focused on developing adaptive mesh refinement algorithms for turbulent and multiphase flows.

John Bell is a Senior Scientist at Lawrence Berkeley National Laboratory. His research focuses on the development and analysis of numerical methods for partial differential equations arising in science and engineering. He is a Fellow of the Society of Industrial and Applied Mathematics, a member of the National Academy of Sciences, and the Director of the AMReX Co-Design Center.

Johannes Blaschke is an application performance specialist at the National Energy Research Scientific Computing Center; engaging with scientists to optimize their software for large-scale HPC environments. He is passionate about helping science teams tackle the problem of deploying their existing algorithms on emergent, highly heterogeneous hardware.

Kevin Gott is an application performance specialist at the National Energy Research Scientific Computing Center. His primary focus is supporting the AMReX Co-Design Center as it develops performance portable solutions for next generation Exascale supercomputers. He also coordinates training sessions and Hackathon events to improve the coding knowledge and expertise of the scientific community. His interests include C++ programming, portable GPU code design, MPI performance and advanced parallel programming techniques.

Andrew Myers is a Computer Systems Engineer in the Center for Computational Sciences and Engineering at Lawrence Berkeley National Laboratory. His work focuses on scalable particle methods for emerging architectures in the context of adaptive mesh refinement. He is an active developer of the AMReX framework and contributes to many ECP application development projects.

Roberto Porcu is a Research Scientist with the Leidos Research Support Team under the Research Support Services contract within the Research and Innovation Center at the National Energy Technology Laboratory, Morgantown. He received a Master's degree in Mathematical Engineering from the Polytechnic University of Milan in 2013 and a PhD in Mathematical Models and Methods in Engineering from the Polytechnic University of Milan in 2017. His research concentrates on combining high-performance computing techniques with the mathematical and numerical modeling of multi-phase fluid dynamics and continuum mechanics problems.

Deepak Rangarajan is a Research Scientist with the Leidos Research Support Team under the Research Support Services contract within the Research and Innovation Center at the National Energy Technology

Laboratory, Morgantown. He has a varied background that spans Laser Doppler Velocimetry (LDV) experiments, continuum modeling of bubbling fluidized beds, and development of software for highly distributed systems. His current focus is on generalizing complex geometry generation.

Michele Rosso is a Project Scientist in the Computational Center for Science and Engineering (CCSE) at the Lawrence Berkeley National Laboratory (LBNL). He has a Ph.D in Mechanical and Aerospace engineering from the University of California, Irvine.He is currently involved in the development and implementation of new multiphysics algorithms in high-resolution adaptive mesh codes that are designed for the latest multicore architectures. His main research interests include numerical modeling of multiphase-flows and high performance computing.

Weiqun Zhang is a Computer Systems Engineer at Lawrence Berkeley National Laboratory. He is interested in high-performance computing, computational physics, and programming in general. Currently, he works primarily on the AMReX software framework and WarpX, an advanced electromagnetic Particle-in-Cell code.

Madhava Syamlal is Senior Fellow for Computational Engineering, overseeing the development of computational science and engineering capabilities at National Energy Technology Laboratory. His primary research interest is in multiphase computational fluid dynamics; he has made numerous contributions to the theory and numerical techniques and the open-source code MFIX. He led the effort to develop Aspen Plus® and FLUENT® co-simulations and served as the founding Technical Director of Carbon Capture Simulation Initiative. He is a fellow of American Institute of Chemical Engineers (AIChE) and the recipient of numerous awards, including R&D100 awards, DOE Secretary's Achievement Honor Award, and AIChE's Fluidization Process Recognition Award.