A massively parallel CFD/DEM approach for reactive gas-solid flows in complex geometries using unstructured meshes

Yann Dufresne, Vincent Moureau, Ghislain Lartigue, Olivier Simonin

A massively parallel CFD/DEM approach for reactive gas-solid flows in complex geometries using unstructured meshes

Yann Dufresne[a,*], Vincent Moureau[a], Ghislain Lartigue[a], Olivier Simonin[b]

[a] CORIA-UMR6614, Normandie Université, CNRS, INSA and UniRouen, Rouen, 76000, France
[b] Institut de Mécanique des Fluides de Toulouse (IMFT), Université de Toulouse, CNRS, INPT, UPS, Toulouse, France

A B S T R A C T

Despite having been thoroughly described in various simple configurations, the study of gas-fluidized systems in a CFD/DEM (Discrete Element Method) formalism becomes challenging as the computational domain size and complexity rise. For a while, attention has been drawn to the design of physical models for fluid-particles interactions, but a recent challenge for numerical tools has been to take advantage from the increasing power of distributed memory machines, in order to simulate realistic industrial systems. Furthermore, unstructured meshes are appealing for their ability to describe complex geometries and to perform local refinements, but lead to significant coding effort involving sophisticated algorithm. In a attempt to design a numerical tool able to cope with these limitations, the methodology presented here proposes an efficient non-blocking algorithm for massive parallelism management, as well as an exhaustive contact scheme to deal with arbitrarily complex geometries, all to be operated on unstructured meshes. The aim is two-fold: (i) To assist larger scale codes in their endeavor to close the solid stress tensor for example, (ii) to pave the way for complex industrial-scale systems modeling using DEM. The methodology is successfully applied to a pilot-scale fluidized bed gathering 9.6M spherical particles and enables to reach interesting physical times using reasonable computational resources.

1. Introduction

In Fluidized Bed Reactors (FBR), the fluidization regime occurs when the fluid that passes through the granular material exceeds the minimum fluidization velocity. In this regime, the drag force applied to the solid grains counterbalances gravity, which leads to a strong mixing of the fluid and solid phases. This mixing ensures efficient heat and mass transfers across the reactor and minimizes temperature and species concentration gradients in the fluidized region. These properties are particularly beneficial in the field of metallurgy, energy and chemical industry for instance, in large scale operations such as chemical synthesis, coating or drying [1]. Low-temperature combustion with high conversion efficiency and low pollutant emissions such as nitrogen oxides is one of the numerous achievement of FBR. In the past, a lack of understanding of the complex dynamic behavior of such devices has been pointed out [2] as one of the cause of the severe difficulties in their design and scale-up [3]. Thus, much time and resource is spent on the building of preliminary tests on pilot-scale reactors that will lead to the design of the final industrial-scale reactor by the mean of empirical processes [4].

Computational Fluid Dynamics (CFD) has already contributed to the understanding of many elementary physical principle through numerous studies on various system sizes, ranging from the study of heat and mass transfer at the particle scale [5] to the modeling of complete industrial units [6]. The prime difficulty resides in the large spectrum of length and involved time scales. Indeed, even in industrial scale systems where the ratio of the reactor size to the solid particle diameter is very large, the fluidization regime features macroscopic structures such as recirculations, particle clusters and gas bubbles of which dynamics prediction strongly depends on the microscopic description of particle contacts, in particular.

Today, the most promising framework for the modeling of industrial units remains the Two Fluid Model (TFM) also referred to as Euler-Euler method, in which it is assumed that both the gas

Nomenclature for non-obvious or recurrent abbreviations (by order of appearance in the text): DEM, Discrete Element Method; ELGRP, Mesh element Group; PTGRP, Particle Group; INTCOMM, Internal Communicator; EXTCOMM, External Communicator; MPI, Message Passing Interface; PTEXTCOMM, Particle External Communicator; PGTS, Particle Group To Send; $\mathcal{VR}$, Voronoi Region; F, E,V, boundary Face, Edge, Vertex; BFG, Boundary Face Group; BSBFG, Bounding Sphere of Boundary Face Group; BSF, Bounding Sphere of Face.

* Corresponding author.
E-mail addresses: yann.dufresne@insa-rouen.fr, yann.dufresne@coria.fr

and the particle phase are inter-penetrating continua [6]. Its underlying assumption is the existence of a separation of scales: the size of the averaging region is much larger than the particle scale. This class of methods is computationally effective but the establishment of an accurate continuous description of the solid phase is challenging and its formulation requires semi-empirical closures and detailed validations. On the other hand, the Discrete Element Method (DEM) also referred to as discrete particle method allows for a more detailed description of particle-particle and particle-wall interactions. This deterministic approach finds its origins in the molecular dynamics methods initiated by Alder and Wainwright [7] and has been benefiting from its advances ever since. In CFD/DEM, or Euler-Lagrange methods, the gas phase is still considered continuous and its time evolution is obtained from a classical CFD-type Eulerian code, but the particles are described individually assuming that their motion obeys Newton's second law of motion, which is solved using standard schemes for ordinary differential equations. This level of modeling designated as meso-scale still requires closures for drag, collision and other forces as a CFD grid cell typically contains up to a few tens of particles, but its advantages lie in its ability to account for the particle-wall and particle-particle interactions in a more realistic manner than Euler-Euler methods.

For the time being, apart from the closures still needed when using CFD/DEM, two main factors limit its utilization for realistic industrial system study: i) The solving of the momentum balance for each particle gives rise to substantial costs that can only be overcome by the mean of optimized parallelism management and ii) industrial system geometries are often composed of cylindrical and irregular parts that prevent the use of conventional Cartesian meshes and necessitate a proper methodology to treat particle-wall contacts. Reaching sufficient computational performances in CFD/DEM simulations serves two purposes: the first is to develop closure laws which can represent the effective averaged interactions in the larger scale models such as TFM, and the second is to pave the way for pilot and industrial scale system simulations in the long run.

Many open-source or commercial CFD/DEM packages have already shown good capabilities for simulating such systems or more complex ones. Among them, one can cite NGA [8] and MFIX-DEM [9] parallel solvers which are both capable to simulate reactive flows based on Cartesian meshes. Other codes relying on unstructured meshes are built based on the coupling of one solver dedicated to the fluid phase and another to the solid phase, such as OpenFoam®+LIGGGHTS® [10] and Fluent®+EDEM CFD® [11]. This study presents the design of a massively parallel code for simulating both phases on unstructured meshes. Concerning complex geometries, contrary to the algorithm suggested by Lin and Canny [12] implemented in the popular I-Collide [13] collision detection package, the method proposed in this work is able to return the measure of a particle penetration depth into the wall, while being simpler than the Voronoi-clip algorithm [14], which is designed for arbitrary complex 3D polyhedra collisions. This code can also work in reacting conditions.

An approach combining DEM to represent the solid phase with Large-Eddy Simulation (LES) equations solved on an Eulerian unstructured grid for the fluid phase has been implemented in the finite-volume code YALES2 [15], a LES and DNS (Direct Numerical Simulation) solver based on unstructured meshes. This code solves the low-Mach number Navier-Stokes equations for turbulent reactive flows using a time-staggered projection method for constant [16] or variable density flows [17].

There is abundant literature on the subject of the different existing models for drag [18], collision force [19] and other closures that may be used for turbulence or heat transfer modeling. These discussions don't fall within the scope of this work, which focuses on a methodology for performance increase. Thus, only elementary models are used in the present work. Furthermore, as heat transfer neither plays a significant role in code performances nor involves extra specific numerical methodology, our attention turns to the study of an isothermal gas-solid dense fluidized bed experimented at the University of Birmingham [20].

In this context, this paper is organized in seven parts. The Euler–Lagrange formalism is first described for both the gaseous and the particle phase in Section 2. Some noteworthy features of the YALES2 code are then briefly introduced in Section 3. The purpose of Section 4 is to present an efficient algorithm for parallelism management. Then, a viable manner to treat spherical particle contacts with arbitrary complex geometries is presented in Section 5. The main case under study is described in Section 6. Finally, the performances of the code are measured in Section 7. Useful abbreviations can be found in footnote [1].

2. The Euler–Lagrange formalism

This section exposes the main models and numerics used for solving the low-Mach number Navier–Stokes equations derived for granular flows in a LES framework. Then, a description of the closures and numerics for solid phase modeling is presented. The coupling between the phases is provided in the Appendix A, including the interpolation/projection technique and the description of filtering steps suited for unstructured meshes.

2.1. Gas phase modeling

The LES governing equations for granular flows are obtained from the filtering of the unsteady, low-Mach number Navier–Stokes equations, taking the local fluid and solid fractions into account. Further details concerning the volume filtering operations can be found in [21]. The governing equations for mass conservation, momentum transport, sensible enthalpy transport and species transport finally read:

$$\frac{\partial}{\partial t}(\varepsilon\bar{\rho}) + \nabla\cdot(\varepsilon\bar{\rho}\tilde{\mathbf{u}}) = 0, \tag{1}$$

$$\frac{\partial}{\partial t}(\varepsilon\bar{\rho}\tilde{\mathbf{u}}) + \nabla\cdot(\varepsilon\bar{\rho}\tilde{\mathbf{u}}\otimes\tilde{\mathbf{u}}) = -\nabla\bar{P} + \nabla\cdot(\varepsilon\bar{\tau}) + \varepsilon\bar{\rho}\mathbf{g} + \overline{\mathbf{F}}_{p\to f}, \tag{2}$$

$$\begin{aligned}&\frac{\partial}{\partial t}\left(\varepsilon\bar{\rho}\tilde{h}_s\right) + \nabla\cdot\left(\varepsilon\bar{\rho}\tilde{\mathbf{u}}\tilde{h}_s\right)\\ &\quad= \nabla\cdot\left(\frac{\mu_t}{Pr_t}\nabla\tilde{h}_s\right) + \frac{dP_0}{dt} + \nabla\cdot\left(\varepsilon\lambda\nabla\tilde{T}\right) + \varepsilon\overline{\dot{\omega}}_T + \overline{Q}_{p\to f},\end{aligned} \tag{3}$$

$$\begin{aligned}&\frac{\partial}{\partial t}\left(\varepsilon\bar{\rho}\widetilde{Y}_k\right) + \nabla\cdot\left(\varepsilon\bar{\rho}\tilde{\mathbf{u}}\widetilde{Y}_k\right)\\ &\quad= \nabla\cdot\left(\frac{\mu_t}{Sc_{k,t}}\nabla\widetilde{Y}_k\right) + \nabla\cdot\left(\varepsilon\bar{\rho}D_k\nabla\widetilde{Y}_k\right) + \varepsilon\overline{\dot{\omega}}_k.\end{aligned} \tag{4}$$

$\mathbf{u}$, ρ, μ, P, h_s, P_0, T, λ, D_k, Y_k, ε are the gas velocity, density, dynamic viscosity, dynamic pressure, sensible enthalpy, thermodynamic pressure, temperature, thermal conductivity, diffusion coefficient, mass fraction of species k, and fluid fraction, respectively. $\overline{\dot{\omega}}_k$ is the chemical source term and $\overline{\dot{\omega}}_T$ the enthalpy source term. The turbulent variables noted μ_t, Pr_t and $Sc_{k,t}$ are the gas turbulent viscosity, turbulent Prandtl number and turbulent Schmidt number of species k. The viscous strain tensor $\bar{\tau}$ is calculated as:

$$\bar{\tau} = (\mu+\mu_t)\left[\nabla\tilde{\mathbf{u}} + \nabla\tilde{\mathbf{u}}^T - \frac{2}{3}(\nabla\cdot\tilde{\mathbf{u}})\mathcal{I}\right], \tag{5}$$

where $\mathcal{I}$ is the identity tensor. $\overline{\mathbf{F}}_{p\to f}$ and $\overline{Q}_{p\to f}$ are the momentum source term and the heat source due to the coupling with particles,

respectively. There is no species transfer between gas and particles. Details concerning the computation of these terms can be found in the Appendix A. These equations are supplemented by the ideal gas Equation-Of-State (EOS):

$$P_0 = \bar{\rho}\tilde{r}\tilde{T} \quad \text{with} \quad \tilde{r} = \sum_{k\in\mathbb{S}} \frac{R\widetilde{Y}_k}{W_k}, \tag{6}$$

with r being the ideal gas mass constant, R being the ideal gas constant, W_k the molar mass of species k, and $\mathbb{S}$ being the set of species.

For the sake of clarity, the fluid filtered quantities $\tilde{\mathbf{u}}$, $\bar{\rho}$, $\bar{P}$, $\tilde{h}_s$, $\widetilde{T}$, $\widetilde{Y}_k$, $\bar{\tau}$, $\tilde{r}$, $\overline{\mathbf{F}}_{p\to f}$ and $\overline{Q}_{p\to f}$ will be written $\mathbf{u}$, ρ, P, h_s, T, Y_k, $\boldsymbol{\tau}$, r, $\mathbf{F}_{p\to f}$ and $Q_{p\to f}$ in the following sections.

Eqs. (2)–(4) are integrated using an explicit variable density solver providing a fully mass, momentum and enthalpy conserving time advancement.

2.2. Particle phase modeling

The translational motion of a particle's center of gravity and its rotational motion around the center of gravity can be fully described by the following system of equations given by Newton's second law, assuming spherical and constant mass particle with high solid/gas density ratio:

$$m_p \frac{d\mathbf{u}_p}{dt} = \mathbf{F}_D + \mathbf{F}_G + \mathbf{F}_P + \mathbf{F}_C \quad \text{with} \quad \frac{d\mathbf{x}_p}{dt} = \mathbf{u}_p, \tag{7}$$

$$I_p \frac{d\boldsymbol{\omega}_p}{dt} = \mathcal{M}_D + \mathcal{M}_C, \tag{8}$$

where m_p, $\mathbf{u}_p$, $\mathbf{x}_p$, I_p and $\boldsymbol{\omega_p}$ are the particle mass, velocity, position, moment of inertia and angular velocity, $\mathbf{F}_D$ is the drag force, $\mathbf{F}_G = m_p\mathbf{g}$ is the gravity force and $\mathbf{F}_P = -V_p\nabla P_{@p}$ is the pressure gradient force. In the last term, V_p is the particle's volume and $\nabla P_{@p}$ is the local pressure gradient interpolated at the center of the particle. As in many dense gas-fluidized bed cases, a soft-sphere model [22] is employed, in which particles are allowed to overlap other particles or walls in a controlled manner. A resulting contact force $\mathbf{F}_C$ accounting for particle-particle and particle-wall repulsion is thus added in the momentum balance of each particle. $\mathcal{M}_C$ is the torque of the contact force $\mathbf{F}_C$ and $\mathcal{M}_D$ is the torque of fluid drag forces. The particle temperature evolution is given by:

$$m_p C_{p,p} \frac{dT_p}{dt} = Q_F, \tag{9}$$

where $C_{p,p}$ and T_p are the particle mass heat capacity and temperature, and Q_F is the heat flux exchanged with the fluid.

The source terms for particles $\mathbf{F}_D$ and $\mathcal{M}_D$ are calculated using a combination of the Ergun [23] and Wen and Yu [24] drag laws, and a closure from the work of Dennis [25], respectively. The closures used for the computation of Q_F won't be detailed in this study, which focuses on an isothermal application. The relation between $\mathbf{F}_D$, $\mathbf{F}_P$ and $\mathbf{F}_{p\to f}$, between Q_F and $Q_{p\to f}$, as well as details concerning the interpolation kernels are given in the Appendix A.

A second-order explicit Runge-Kutta (RK2) algorithm is used to advance the particles in time. The use of a soft-sphere model demands that $\Delta t_p < T_C$, where Δt_p is the particle timestep and T_C is a contact time described in Section 2.2.1. In this work, $\Delta t_p = T_C/10$ was considered, to ensure a reasonable precision.

2.2.1. Modeling of collisions

The total collision force $\mathbf{F}_C$ acting on particle a is computed as the sum of all pair-wise forces $\mathbf{f}^{col}_{b\to a}$ exerted by the N_p particles and N_w walls in contact. As particles and walls are treated similarly during collisions, the b index refers to both:

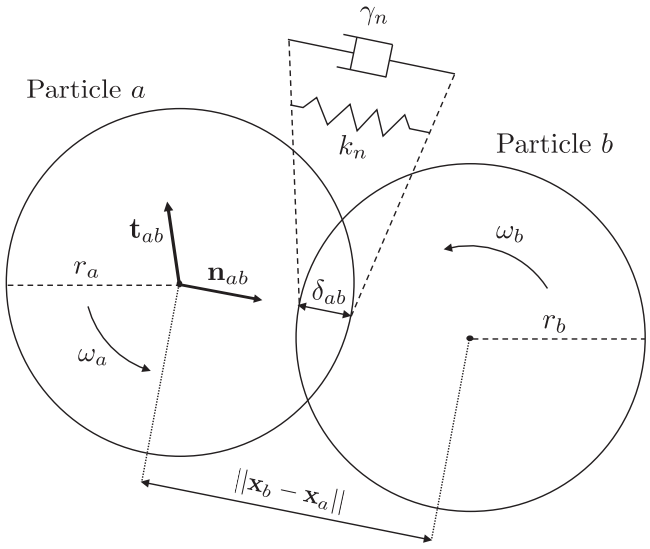

Fig. 1. Soft sphere representation of two particles undergoing collision.

$$\mathbf{F}_C = \sum_{b=1}^{N_p+N_w} \mathbf{f}^{col}_{b\to a} \quad \text{with} \quad \mathbf{f}^{col}_{b\to a} = \mathbf{f}^{col}_{n,b\to a} + \mathbf{f}^{col}_{t,b\to a}. \tag{10}$$

Here a linear-spring/dashpot [22] model is used along with a simple Coulomb sliding model accounting for the normal $\left(\mathbf{f}^{col}_{n,b\to a}\right)$ and tangential $\left(\mathbf{f}^{col}_{t,b\to a}\right)$ components of the contact force, respectively, as in the work of Capecelatro [21]. For one particle (or wall) b acting on a particle a:

$$\mathbf{f}^{col}_{n,b\to a} = \begin{cases} -k_n\delta_{ab}\mathbf{n}_{ab} - 2\gamma_n M_{ab}\mathbf{u}_{ab,n} \\ 0 \end{cases} \quad \text{and}$$

$$\mathbf{f}^{col}_{t,b\to a} = \begin{cases} -\mu_{tan}||\mathbf{f}^{col}_{n,b\to a}||\mathbf{t}_{ab} & \text{if} \quad \delta_{ab} > 0, \\ 0 & \text{else.} \end{cases} \tag{11}$$

Fig. 1 shows a representation of two colliding particles. k_n is the normal spring stiffness, γ_n is the normal damping parameter, and μ_{tan} is the friction coefficient. The term $\delta_{ab} = r_a + r_b - ||\mathbf{x}_b - \mathbf{x}_a||$ is defined as the overlap between the a and b entities expressed using each particle radius r_p and center coordinates $\mathbf{x}_p$. The system effective mass M_{ab} is expressed as $M_{ab} = (1/m_a + 1/m_b)^{-1}$. The unit normal vector $\mathbf{n}_{ab}$ from particle a towards entity b and a unit tangential vector $\mathbf{t}_{ab}$ are defined using particles' relative position and velocity. $\mathbf{n}_{ab}$ and $\mathbf{t}_{ab}$ are calculated as follows:

$$\mathbf{n}_{ab} = \frac{\mathbf{x}_b - \mathbf{x}_a}{||\mathbf{x}_b - \mathbf{x}_a||} \quad \text{and}$$

$$\mathbf{t}_{ab} = \begin{cases} \dfrac{\mathbf{u}_{ab} - \mathbf{u}_{ab,n}}{||\mathbf{u}_{ab} - \mathbf{u}_{ab,n}||} & \text{if} \quad ||\mathbf{u}_{ab} - \mathbf{u}_{ab,n}|| > 0, \\ \mathbf{0} & \text{else.} \end{cases} \tag{12}$$

The relative velocity of the colliding system at the contact point $\mathbf{u}_{ab}$ is written:

$$\mathbf{u}_{ab} = (\mathbf{u}_a - \mathbf{u}_b) + (r_a\boldsymbol{\omega}_a + r_b\boldsymbol{\omega}_b) \wedge \mathbf{n}_{ab}. \tag{13}$$

Its normal component is then given by $\mathbf{u}_{ab,n} = (\mathbf{u}_{ab}\cdot\mathbf{n}_{ab})\mathbf{n}_{ab}$.

Using Newton's third law yields an analytical expression for the system's natural frequency $\omega_0 = \sqrt{k_n/M_{ab}}$ and the contact time [26]:

$$T_C = \frac{\pi}{\sqrt{\omega_0^2 - \gamma_n^2}} \quad \left(\propto \sqrt{\frac{m_p}{k_n}}\right). \tag{14}$$

As the particles are all spherical with homogeneous density, the moment of inertia simply is $I_p = m_p d_p^2/10$, and the total torque $\mathcal{M}_C$ applied by all entities b in contact with a particle a only depends on the tangential component of the individual contact forces:

$$\mathcal{M}_C = r_a \sum_{b=1}^{N_p+N_w} \mathbf{n}_{ab} \wedge \mathbf{f}_{t,b\to a}^{col}. \tag{15}$$

In case of a particle-wall collision, the wall is considered as a particle with infinite mass and null radius.

The search for potential collision partners is accelerated by the use of a standard linked-cell data structure [27]. This Cartesian grid, superimposed on the unstructured Eulerian mesh, is dynamically computed. The description of this usual step has been omitted.

3. Specific features of the YALES2 solver

In this section, some properties of the unstructured mesh partitioning used in YALES2 are highlighted. The specific data architecture strongly influences the methodologies that are to be discussed at a later stage, and is thus also presented. Further detail concerning these features can be found in [15].

3.1. Two-level domain decomposition for unstructured grid

As mentioned previously, the low-Mach number Navier–Stokes equations are solved on unstructured meshes in order to fully benefit from high-performance computing on massively parallel machines. A two-level domain decomposition (Double Domain Decomposition, abbreviated DDD) is employed and organized as follows: at a high level, mesh cells are dispatched over processors. It consists in splitting the computational domain into sub-meshes that are affected to each computational core. At the lower level, at the processor scale, mesh cells are gathered in cell groups called ELement GRouPs (ELGRPs) as sketched in Fig. 2. This double domain decomposition allows for easily optimizing the use of processor memory for cache-aware algorithms and may also be exploited by deflation algorithms [28]. In 3D cases, ELGRPs typically contains $\mathcal{O}(10^3)$ cells. Following the same pattern, particles located in an ELGRP are stored in ParTicle GRouPs (PTGRPs) containing up to 500 particles each, here again to improve performances.

3.2. Data structures

Using DDD reinforces the need to work with a specific data structure. Indeed in this context, each ELGRP stands for an individual mesh block, but communications occur between ELGRPs when computing gradients or for residual assembly. Thus, besides classical inter-processor connectivities, some geometrical elements, such as nodes, faces or edges, need to exchange data inside the core during the communication steps. Another data structure is therefore needed to connect the geometrical elements at the border of the ELGRPs. Rather than making use of a ghost cell method, an INTernal COMMunicator (INTCOMM) that contains a copy of all the nodes, faces or edges involved in the communications inside or outside the cores is deployed. EXTernal COMMunicators (EXTCOMMs) also contain their own copy of the nodes, faces and edges that are located at the interface between other computing cores. This architecture is depicted in Fig. 3, where the boundaries are also represented.

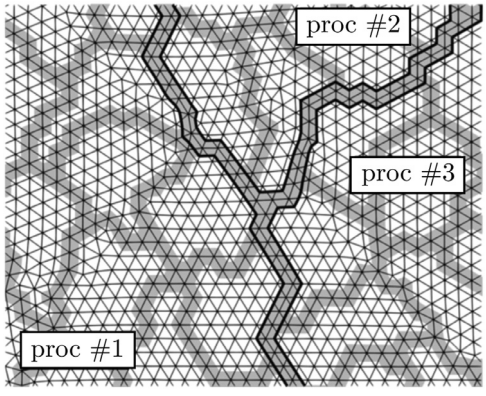

Fig. 2. Double Domain Decomposition (DDD). The highlighted elements are participating in the communications inside and outside each processor and those surrounded in black are participating in the communications between processors. Extracted from [15].

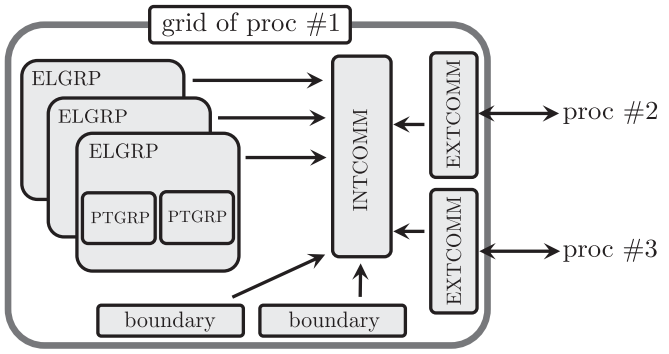

Fig. 3. Internal (INTCOMM) and External (EXTCOMM) communicators corresponding to Fig. 2.

4. Parallelism management

Moving towards pilot-scale CFD/DEM simulations imposes that a satisfactory scalability on massively parallel machines be reached. Parallel simulations require special treatment for particles, as collision might occur between some of them although they are dispatched on different processor domains that have no connection. To cope with this requirement and in accordance with the data structures for unstructured meshes explored in Section 3.2, a ghost particle method is used in a Message Passing Interface (MPI) paradigm. MPI parallel domain decomposition is indeed an attractive option to parallelize CFD/DEM problems, especially with the emergence of massively parallel distributed memory systems and for its high scalability possibilities even for large numbers of processors. Note that a combination of a CFD code executed on CPUs (Central Processing Units) and a DEM code executed on GPUs (Graphics Processing Unit) has been reported as a promising high-performance method for coupled CFD/DEM simulations [29]. This section tackles the design of an efficient parallel strategy using MPI domain decomposition.

The currently implemented global algorithm is sketched in Fig. 4 and can be resumed as follows: first, ghost particles are identified using a cell halo surrounding each processor domain.

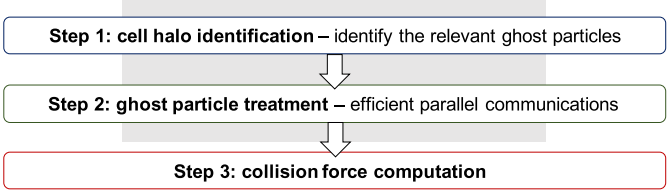

Fig. 4. Global algorithm for parallelism management on unstructured meshes. The two first steps are detailed in this work.

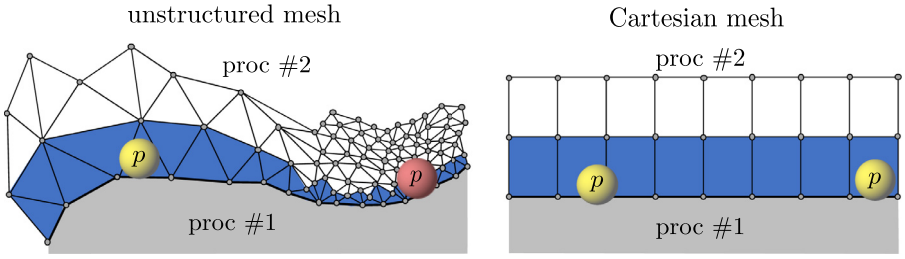

Fig. 5. Flagging of one layer of elements ( ) at the interface of two processor domains. Left: on unstructured mesh, the red particle p don't belong to any flagged element. Right: on Cartesian mesh this instruction is sufficient to identify ghost particles. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

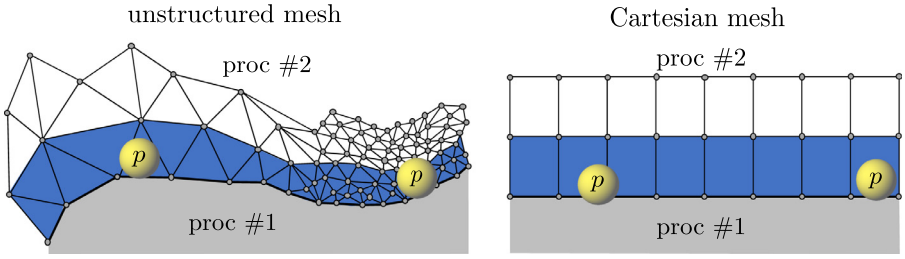

Fig. 6. Flagging of layers of elements ( ) at the interface of two processor domains with a distance instruction. Left : on unstructured mesh, all particles p are now identified. Right : on Cartesian mesh this instruction leads to the same result.

Next, the necessary data of particles belonging to the cell halo are packed, and exchanged between involved processors. Finally, after unpacking on each processor, ghost particles are treated as local particles to treat collisions. The following subsections detail each step of this algorithm. All tests have been run on the Birmingham fluidized bed case (see Section 6), on the Curie supercomputer from CEA in France. Unless otherwise specified, statistics were collected over 1 s of physical time, and started after having initiated fluidization for 2.5 s. It will be seen in Section 6 that these time scales are sufficient to ensure that both the bed height and the pressure loss across the bed are oscillating around their mean value. This leads us to think that this case is relevant enough for performances' measurements.

4.1. Initialization: Cell halo identification

The first step consists in defining a cell halo around each processor domain, in order to identify the closest particles on neighbor processors. A better selection will limit the number of ghost particles to be exchanged, thus facilitating the building of the collision partners' detection grid and increasing the collision force computation speed.

The cell halo identification can be straightforwardly achieved on regular Cartesian meshes, and in most cases one layer of cells is a sufficient criterion for the halo building and no collision can be forgotten. However, when dealing with mesh size heterogeneities encountered on o-grid or unstructured meshes for instance, this criterion can either lead to an excessive halo width sacrificing the code performances, or the forgetting of many contacts impacting the simulation physical meaning, as sketched in Fig. 5. This underlines the need for an adaptive element flagging method which should ideally be based on an exact distance computation. As computing the exact distance between a mesh node and a processor domain border would involve numerous calculation steps that could necessitate inter-processor communications, the implemented approach uses an approximate distance. The algorithm is inspired by fast-marching algorithms developed for Level-Set Methods [30,31]. In particular, it relies on the mapping of the surface of the processor domain border using points called markers, of which coordinates are automatically generated. The general idea is to propagate from node to node these coordinates, from the processors' interface towards neighboring processors. Each step of this algorithm is thoroughly detailed in the Appendix B.

Performing these steps allows, at the beginning of a simulation, to identify the required elements for ghost particles treatment, as any element containing a node closer to the processor domain border than a particle radius is flagged (see Fig. 6). It also works on any mesh element type; tetrahedron, hexahedron, prisms, pyramids, and hybrid meshes.

As a result, Table 1 yields the relative CPU time measured for the main steps of the collision force computation algorithm for the slower processor regarding different methods for the cell halo identification. The reference is the adaptive method studied in this section. The three other methods correspond to simpler ones where the given instruction is to identify the closer, the two closer and the three closer layers of elements for the halo. For this test the Birmingham fluidized bed (see Section 6) was run on 512 processors. It should be noticed that in this case the particle radius that is targeted for the adaptive method is supposed to be smaller than the average mesh cell size, otherwise these results are expected to be different. The obtained values clearly indicate a strong dependency upon the number of identified cells, as it obviously results in different quantities of ghost particles that have to be exchanged between processors, then located in the detection grid and finally treated for collisions. In the tested case it appears that the contact force computation is the most sensitive part of the algorithm. A thorough analysis of the identified cells reveals that the method using one layer of elements may identify additional cells in negative curvature border areas, hence the slightly better results obtained with the adaptive method.

To benefit from this efficient cell halo identification, a new data structure called ParTicle EXTernal COMMunicator (PTEXTCOMM) is created, playing the same role as the previously mentioned EXTCOMM but dedicated to the particles. At this point, the new architecture is sketched in Fig. 7. On each processor, one PTEXTCOMM is allocated for each other processor impacted during the cell halo definition, that will support point-to-point MPI exchanges during the simulation. To make sure that for any

Table 1
Influence of cell halo identification method on the CPU time for various operations.

Method used	Relative CPU time for some key phases measured for slowest processor		
	Ghost particle communications	Potential collision partners identification	Particle-particle contact force computation
adaptive method	1.00	1.00	1.00
one layer of elements	1.04	1.07	1.35
two layers of elements	1.69	1.66	1.98
three layers of elements	2.63	2.55	3.07

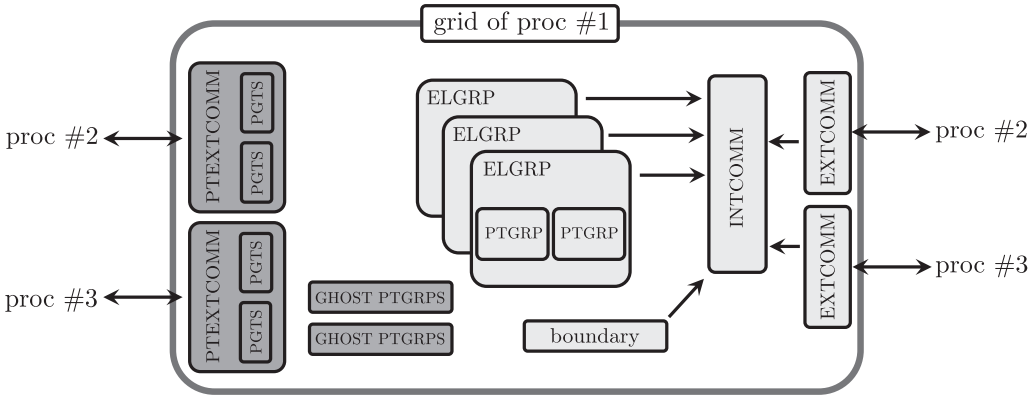

Fig. 7. Improved communicators structure containing Particle EXTernal COMMunicators (PTEXTCOMMs) and ghost ParTicle GRouPS (GHOST PTGRPS).

processor n sharing a PTEXTCOMM with a processor m, the reverse is true as well, a pre-communication step is performed when the last processor exits the cell halo definition loop. As shown in Fig. 7, each PTEXTCOMM can contain several dedicated PTGRPS called PGTS (Particle Group To Send), of which role is detailed in Section 4.2. They also contain information about elements flagged during cell halo building. It is important to notice that the list of PTEXTCOMMS can differ from the list of EXTCOMMS, as the cell halo size can locally exceed the mesh element size and thus impact distant processors.

4.2. Ghost particle treatment

When the collision force computing is needed, *i.e.* at each RK step, the particles' data have been updated and ghost particles have to be exchanged. Ghost particles are identified using the cell halo surrounding each processor domain as shown in Fig. 8 for processor ranked #1 in a cylindrical geometry discretized with an unstructured mesh. This halo has been defined in previous Section 4.1.

Before computing the collision force on a processor #p, the following treatment is applied:

1. the particles of any processor #n located in a mesh element which belongs to processor #p halo are copied into the according PTEXTCOMM of #n and dispatched among its PGTS.
2. Each processor #n sends its PGTS to processor #p.
3. Processor #p stores all the received particles in ghost PTGRPs.

Eventually, ghost particles are treated by proc #p as local particles when computing contact forces, before all ghost particles are discarded to prepare for the next time step. Simulations of the Birmingham fluidized bed performed on 512 processors have shown that a naive coding of the ghost particle exchange could lead it to taking 45% of the total simulation time. This section focuses on the implementation of an efficient parallel strategy to reduce the cost of the 2nd step of the previous algorithm.

A packing strategy, consisting in arranging all the PGTS of a PTEXTCOMM into a unique vector before sending it, is here employed to circumvent the problem of small messages latency (detailed in the Appendix C), as shown in Fig. 9. When the reception of all particle data packet is done, an unpacking step allows to rebuild ghost PTGRPs from it. In order to avoid numerous memory (de)allocation operations, the allocated size of a pack is only enlarged if not sufficient but is never downsized, targeting buffers reuse. Fig. 10 displays the distribution of the number of MPI messages as a function of the message sizes for two strategies: the one without packing of the halo data corresponds to a naive coding in which, for each PGTS of a PTEXTCOMM, each particle data array is sent individually, as well as its size for preliminary memory allocation. The other strategy involves the

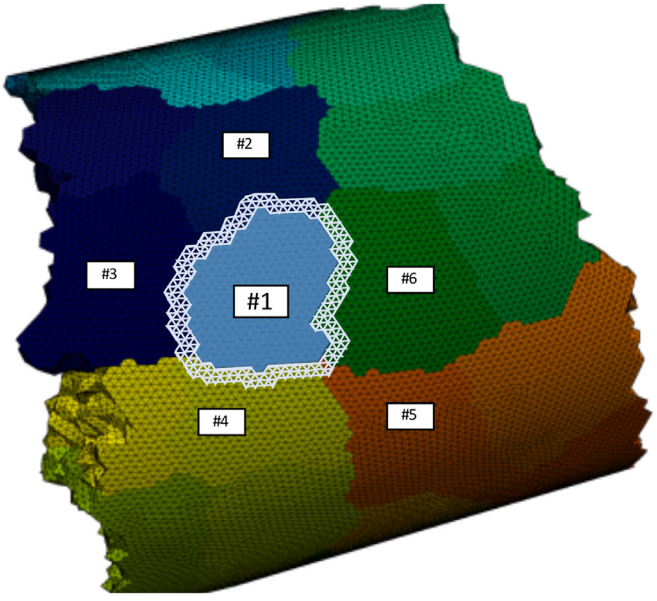

Fig. 8. Ghost particle method principle shown for a part of a cylindrical domain. The different processor domains are colored accordingly. The processor of interest is ranked #1 ( ) and its closest neighbors are ranked #2, #3, #4, #5 and #6. A particle entering the white cell halo around #1 will be sent to #1 as a ghost particle by the processor it belongs to.

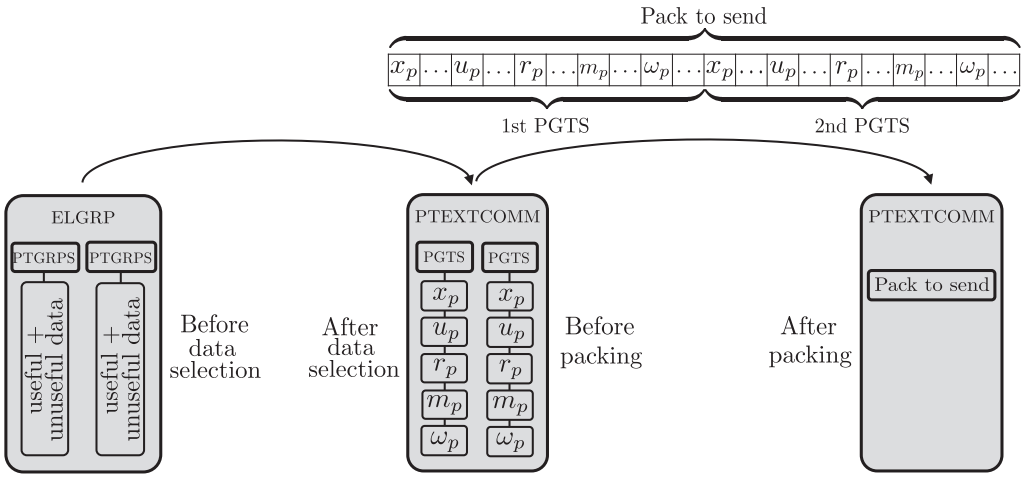

Fig. 9. Before packing, a PTEXTCOMM contains two PGTS. Each PGTS is composed of as many arrays as particle data. The packing consists in arranging all these arrays in one unique vector which is the pack to send, thus simplifying MPI communications.

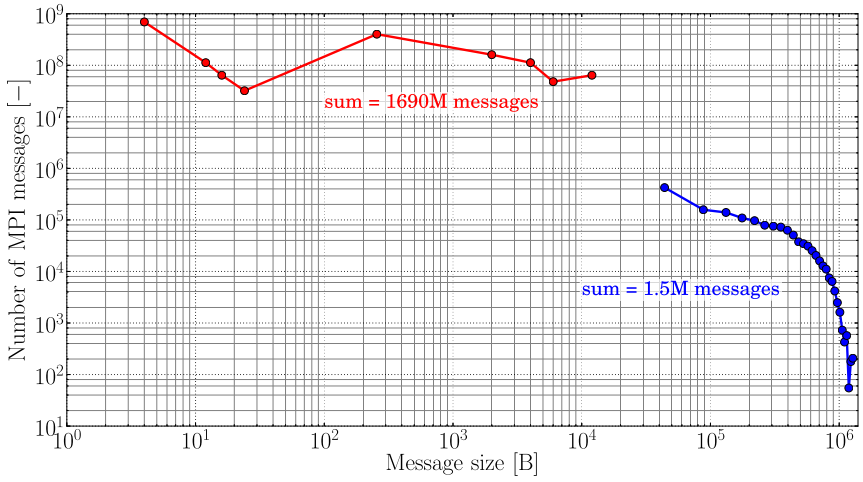

Fig. 10. Distribution of the number of MPI messages as a function of the message sizes for two strategies; : without packing of the halo data, : with packing of the halo data. Records come from runs performed on the Birmingham configuration on 512 processors over 30 solver iterations. The sums under each curve indicates the total amount of messages exchanged.

aforementioned packing/unpacking of the halo data. Records come from runs performed on the Birmingham configuration on 512 processors over 30 solver iterations. It is clear that the naive coding leads to very large amounts of messages: approximately one thousand times more than the packing/unpacking strategy. These messages are also much smaller in the first strategy, roughly one thousand time smaller, and more than 40% of the total amount are 4B messages, whereas the largest ones are 12kB messages. As regards the packing strategy, the observed message sizes corresponds to the packing of discrete numbers of PGTS, here ranging from one PGTS (representing 45kB messages containing up to 500 particles) which are the more represented, up to twenty-nine PGTS (representing 1.28MB messages containing up to 14500 particles).

The total time spent in the communications is estimated as a function of the message size in Fig. 11. The total time is calculated as follows for the strategy without packing, k denotes a message size:

$$\text{total time}(k) = \tau_{latency}(k) \times \text{number of messages}(k), \tag{16}$$

where $\tau_{latency}$ is the global latency, and as follows for the strategy with packing:

$$\text{total time}(k) = \left[\tau_{latency}(k) + \tau_{pack}(k) + \tau_{unpack}(k)\right] \times \text{number of messages}(k), \tag{17}$$

where τ_{pack} and τ_{unpack} stand for the CPU cost of the packing and unpacking operations, respectively. The performances of the Curie supercomputer's network are assessed in the Appendix C in order to quantify $\tau_{latency}(k)$, as well as $\tau_{pack}(k)$ and $\tau_{unpack}(k)$. Here the global latency is taken from extra node communications (see Fig. C.35), and the packing/unpacking costs are taken with preliminary particle data selection (see Fig. C.36).

It can be observed that even when accounting for the cost of the packing and unpacking steps of each message, the second strategy is still 12 times quicker than the naive coding without data packing, approximately. It should be noted that these calculations only give maximum times because the underlying hypothesis is that exchanges only occur one at a time, while in a real simulation some are done simultaneously. It can also be argued that even on an ideal network with null latency and infinite bandwidth, messages cannot be treated concurrently at the time of their reception, hence additional contention that should be avoided. Eventually, these results are all in favor of an MPI strategy involving fewer data packets to exchange. Simulations of the Birmingham fluidized bed running on 512 processors show that MPI communications could represent up to 45% of the total simulation cost without special treatment of the data exchanges. Results using the presented packing/unpacking strategy exhibit a communication cost divided by 3, allowing the overall simulation to run 30% faster.

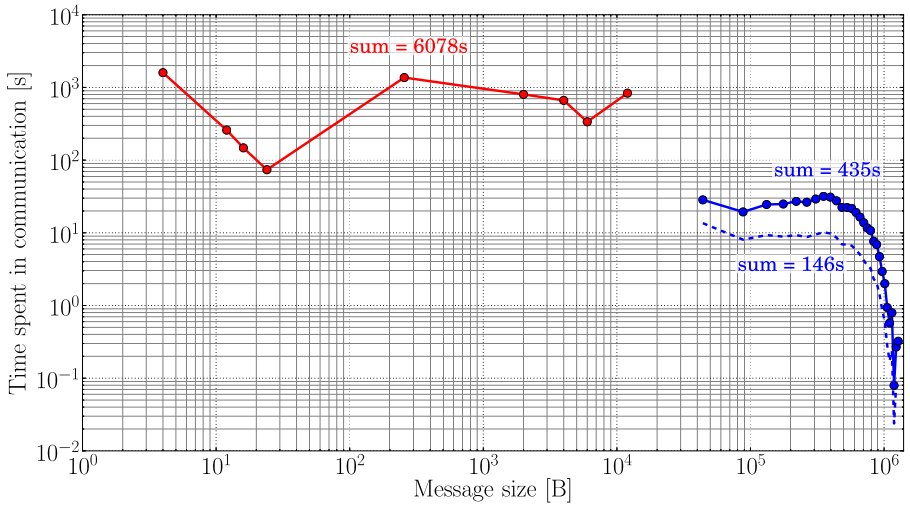

Fig. 11. Theoretical maximum time spent in communications as a function of the message sizes for two strategies; : without packing of the halo data, : with packing of the halo data. As an indication, stands for the case with packing but only accounting for the cost of communications. Records come from runs performed on the Birmingham configuration on 512 processors over 30 solver iterations. The sums close to each curve indicates the total amount of time.

To accelerate the treatment of data packets, a fully asynchronous algorithm featuring computation/communication overlap is implemented, as sketched in Fig. 12. The objective of such a method is to try to perform on-the-fly packing and unpacking operations in order to overlap with communication times due to global latency. It is divided into two nested parts, the first one being the exchange of pack sizes to allocate the necessary memory on the receiver's side, and the second one being the exchange of the actual packs. The main idea is to probe the non-blocking receive requests in order to perform the packing and unpacking operations as soon as some data are available, while waiting for the next ones to be completed. By checking the size of a pack to send and the one of the pack to receive, any PTEXTCOMM empty of particles is discarded from the second part of the algorithm, as well as all the PTEXTCOMM that would have to exchange particles with it.

Theoretically, this algorithm should reveal its full potential in cases where the packing and unpacking computational costs are close to the global latency. Indeed in this configuration, the packing and unpacking operations can occur between two reception completions without any time loss. The capacity of this algorithm to provide computation/communication overlap can be assessed by comparing its performances with the ones of a blocking coding also including the packing and unpacking features. In the latter case, pack sizes are all exchanged in an orderly manner before packing is carried out, then actual packs are treated the same before unpacking is performed. Run on 512 processors, the Birmingham fluidized bed case demonstrates that the computational cost for ghost particles treatment decreased by a factor 2.6 when using the asynchronous method along with packing/unpacking, compared to packing/unpacking with blocking communications, therefore providing a gain of 11.5% on the overall simulation time.

5. Complex boundaries management

As industrial systems often contain non-planar boundaries, such as cylindrical parts or more complex elements like pipe junctions, a special treatment is required to treat particle-wall contacts. Several options have been proposed by different authors to address this problem. Among them, the most simple method is the one of the glued particles to approximate geometric surfaces and thus treat particle-wall interactions the same fashion as particle-particle interactions ([32],[33]). However, this simplification suffers from a lack of accuracy as it doesn't represent complex shapes exactly, especially in the vicinity of convex parts. It can also result in uncontrolled wall roughness and larger computational overheads associated with the use of additional particles [34]. Further coding effort can also be needed for surface particles generation [35].

The explicit methods for the treatment of the contact between complex objects can be of two types : "simplex based" algorithms treat a polyhedron as the convex hull of a point set and perform operations on simplices defined by subsets of these points [14]. Among these, the *Discrete Function Representation* algorithm proposed by Williams [36] allows the treatment of numerous varieties of shapes but may imply fine discretization with consequent set of points for edgier bodies. The iterative algorithm originating from the work of Gilbert, Johnson & Keerthi [37], which has served as a basis for several other methods, may be the most famous representative among this type of methods. On the contrary, "feature-based" algorithms treat a polyhedron as a set of points, segments and faces. The *finite wall method* studied by Kremmer [38] is a good candidate, but starts with the assumption that the boundary surfaces can be discretized into triangular elements, the positions and dimensions of which are known and controllable, which is not the case in general CFD simulations. It also requires an empirical "shrink factor" to be defined. The popular algorithm suggested by Lin and Canny (Lin–Canny algorithm) [12], implemented in the I-Collide [13] collision detection package, is a "feature-based" algorithm designed for arbitrary complex 3D polyhedra collisions. It is based on the existence of a unique decomposition into Voronoi regions of the wall geometry. The Lin–Canny algorithm raises problems due to its lack of robustness, and is not able to return the measure of the penetration depth, therefore it is not suited to a soft-sphere model implementation. It has been improved by Mirtich (Voronoi-clip algorithm) [14] in order to overcome these limitations, however it can still only treat spheres by tessellating them. Note that analytical contacts can be elegantly resolved for some particular shapes [39], but to the author's knowledge, this option offers few prospects for general 3D applications.

Here is thus proposed an algorithm for detecting the interactions between a spherical particle and an arbitrarily complex geometric surface and mesh in the framework of the DEM, and consistent with massive parallelism. This last point is of particular importance, as this aspect is seldomly addressed in the literature. It relies on the fact that a particle can collide with only three types of geometrical entities: either a vertex (V), or an edge (E), or a

Fig. 12. Flow chart of the final asynchronous algorithm for ghost particle treatment featuring packing/unpacking of the halo data and communication/computation overlap. : pack sizes communication parts. : pack communication parts.

face (F), or with any combination of these objects simultaneously in any fashion. It thus belongs to the "feature-based" algorithms. It doesn't require any input parameter nor preprocessing of the geometry, and doesn't use any iterative process. It is also only based on the present state of the contact configuration (it's an "exhaustive" scheme [36]), and also relies on a Voronoi decomposition. The global algorithm is sketched in Fig. 13. As for most of the above mentioned methods, a first phase of spatial sorting seeks to avoid an all-to-all body comparison by culling the number of objects which are potential contactors of a given particle. In a further stage, all possible contact conditions including contact with Fs, Es and Vs (Faces, Edges and Vertices) are explicitly determined. The following subsections detail these steps the other way around, as the last ones are actually at the core of the method. All tests have been run on the Curie super-computer from CEA in France. Unless otherwise specified, statistics were collected over 1 s of physical time, and started after having initiated fluidization for 2.5 s.

5.1. Use of Voronoi regions

Voronoi regions are used for their ability to yield an object's closest boundary feature(s) and then the shortest distance between this object and the boundary. The definition of the Voronoi regions for several geometrical features is given: for a feature $X \in [F, E, V]$ on a polyhedron, the Voronoi Region $\mathcal{VR}(X)$ is the set of points outside the polyhedron that are closer to X than to any other feature on the polyhedron. The Voronoi regions collectively cover the entire space outside the polyhedron. Examples of $\mathcal{VR}$ are shown in Fig. 14.

It stems from the building of the boundary features' $\mathcal{VR}s$ that:

- The number of planes delimiting $\mathcal{VR}s(F)$ is equal to the number of edges of F, say three for a triangle, and the normals to each of these planes are given by the normals of each edge contained in the plane defined by F.

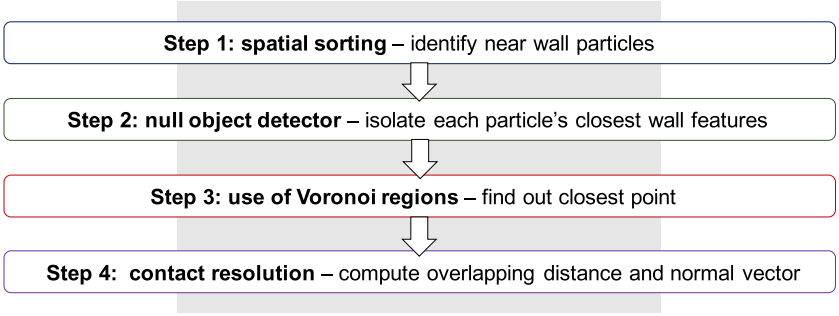

Fig. 13. Global algorithm for arbitrarily complex geometries management.

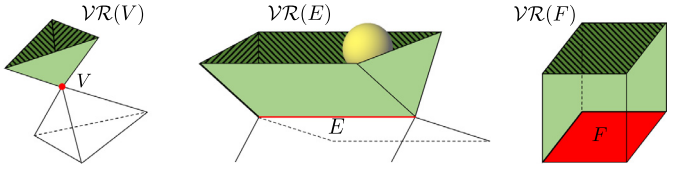

Fig. 14. Voronoi regions of convex node ($\mathcal{VR}(V)$), convex edge ($\mathcal{VR}(E)$) and face ($\mathcal{VR}(F)$).

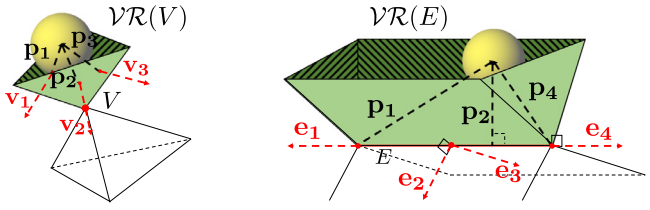

Fig. 15. On the left, $\mathcal{VR}(V)$ normals $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$ are elucidated along with the approaching particle's corresponding vectors $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ for dot products calculations. On the right, $\mathcal{VR}(E)$ normals $\mathbf{e_1}$, $\mathbf{e_2}$, $\mathbf{e_3}$ and $\mathbf{e_4}$ are elucidated along with the approaching particle's corresponding vectors $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_4}$ for dot product calculations. $\mathbf{p_3}$ is equal to $\mathbf{p_2}$.

- The number of planes delimiting $\mathcal{VRs}(V)$ is equal to the number of edges connected to V, and the normals to each of these planes are simply equal to the direction vectors carried by each edge. This number can *a priori* reach any value.
- $\mathcal{VRs}(E)$ are all limited by four planes. The normals to two of them are given by the normals of E contained in the planes defined by each faces connected by E. The two others are obtained by taking the direction vector of E and its opposite.

It appears that the knowledge of the $\mathcal{VRs}$ normals of each boundary feature should be sufficient to identify an object's closest boundary feature in convex parts, and eventually, all $\mathcal{VRs}(E)$ and $\mathcal{VRs}(V)$ normals are built and stored. The $\mathcal{VRs}$ normals of the features that are common to several processors are entirely known to each of these processors. To identify an object's closest point on the boundary, the following methodology is retained. Here the example of a particle of radius r_p of center P approaching a boundary composed of several triangular faces is taken:

1. Projection P' of the point P onto the plane defined by the first boundary face F.
2. Computation of the distance d_{pF} between P and P'.
3. In case of overlap ($d_{pF} < r_p$), to determine whether P' belongs to F or not. To this end, the coordinates of P' are expressed in the face's barycentric coordinates. The full description of the operation is in the Appendix D.
4. In case P' belongs to F, then the particle is actually colliding with F, its shortest distance to the boundary is d_{pF} and the contact treatment can be applied (see Section 5.2). In this case the algorithm moves on to the next boundary face. Otherwise the contact between P and any E $\in$ F has to be checked.
5. To check a particle-edge contact, the distance d_{pE} between the particle and the line defined by the direction vector of E is calculated first.
6. In case of overlap ($d_{pE} < r_p$), the belonging of P to $\mathcal{VR}(E)$ is checked by performing dot products between each $\mathcal{VR}(E)$ normal and the appropriate vector for P, as sketched in Fig. 15, so that:

$$P \in \mathcal{VR}(E) \text{ if } \forall \mathrm{i} \in [\![1;4]\!],\ \mathrm{p_i} \cdot \mathrm{e_i} < 0. \tag{18}$$

For an edge, $\mathbf{p_3}$ is equal to $\mathbf{p_2}$. To quicken these operations, P is first assumed to belong to $\mathcal{VR}(E)$, then each dot product is consecutively checked and the test ends as soon as one gives a positive result.

7. In case $P \in \mathcal{VR}(E)$, the particle is actually colliding with E, its shortest distance to the boundary is d_{pE} and the contact treatment can be applied (see Section 5.2). In this case the algorithm moves on to the next boundary face. Otherwise the contact between P and any V $\in$ F has to be checked.
8. To check a particle-vertex contact, the distance d_{pV} between the particle and V is calculated first.
9. In case of overlap ($d_{pV} < r_p$), the belonging of P to $\mathcal{VR}(V)$ is checked by performing dot products between each $\mathcal{VR}(V)$ normal and the appropriate vector for P, as sketched in Fig. 15, so that:

$$P \in \mathcal{VR}(V) \text{ if } \forall \mathrm{i} \in [\![1;\text{number of edges connected to V}]\!],\ \mathrm{p_i} \cdot \mathrm{e_i} < 0 \tag{19}$$

To quicken these operations, $P \in \mathcal{VR}(V)$ is first assumed to be true, then each dot product is consecutively checked and the test ends as soon as one gives a positive result.

10. In case $P \in \mathcal{VR}(V)$, the particle is actually colliding with V, its shortest distance to the boundary is d_{pV} and the contact treatment can be applied (see Section 5.2). In this case the algorithm moves on to the next boundary face.

This algorithm allows several simultaneous contacts with any kind of boundary feature in convex geometrical parts. Furthermore, in concave areas such as the one depicted in Fig. 16, vectors orientation invariably prevents the particle from accounting the concave feature E for collision, while allowing both the side faces, which has a physical meaning. Fig. 16 also reveals a good behavior of the algorithm in more complex cases that can occur for nodes which have more than three connected edges. By providing suitable exit conditions it also prevents contacts from being detected with several entities belonging to the same face: indeed when a contact is going to be treated between a particle and the face F, no further tests are performed for Es and $Vs \in F$. Also, as soon as a particle is found overlapping an $E \in F$, the remaining Es and Vs are discarded

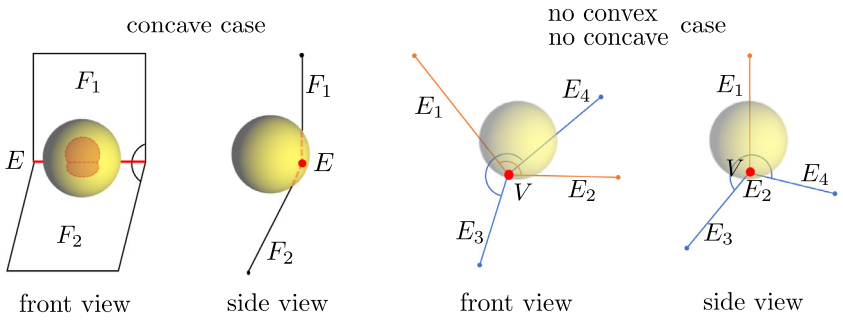

Fig. 16. Left part: classical concave case. The particle is overlapping faces F_1, F_2 and edge E in pale red areas. The algorithm for the use of $\mathcal{VR}s$ will detect that $P \in \mathcal{VR}(F_1)$, $P \in \mathcal{VR}(F_2)$ but $P \notin \mathcal{VR}(E)$. Hence, repulsion forces on the particle will be calculated for F_1 and F_2 even if the particle actually overlaps E. Right part: neither convex nor concave case featuring an angle $> 180°$ (—) and an angle $< 180°$(—). The particle is overlapping all edges and vertex V. The algorithm will detect that $P \in \mathcal{VR}(E_1)$, $P \in \mathcal{VR}(E_2)$ but $P \notin \mathcal{VR}(V)$, $P \notin \mathcal{VR}(E_3)$ and $P \notin \mathcal{VR}(E_4)$. Hence, repulsion forces on the particle will be calculated for E_1 and E_2 only. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and the algorithm repeats for the next face. In a last case, as soon as a particle is found overlapping a $V \in F$, the remaining Vs are discarded and the algorithm is repeated for the next face. These exits are thus essential for computational cost saving. Lastly, it appears that for contacts with nodes and edges, each connected face can detect the contact, resulting in a repulsion force accounted several times instead of one. To cope with this limitation, all the contact forces exerted by edges are divided by two, while contact forces exerted by nodes are divided by the node's number of connected faces. This multiplicity is computed in a parallel fashion.

5.2. Contact resolution

Having identified the type of contact point for a particle, the algorithm finishes with the calculation of the effective repulsive forces and torques exerted on the particle. As shown in Eq. 10, the total collision force exerted on a particle in contact with a wall is taken as the sum of the forces exerted by each colliding feature of the wall. These particle-wall forces are composed of normal and tangential components written the same fashion as for particle-particle contacts (see Eq. 11 and Eq. 15), by treating the wall parts as a particle with null radius and infinite mass. The parameters k_n, γ_n and μ_{tan} can be set to different values regarding the type of contact, either particle-particle or particle-wall. To describe the repulsion force exerted on a particle by a boundary feature, a unit normal vector and a measure of the interpenetration distance (overlap) between the sphere and wall element are to be yielded. As in a majority of works, contacts are here treated considering a unique contact point, despite the actual overlapping parts may reveal an extensive set of possibilities.

Referring to the different contact cases sketched in Fig. 17, the treatment of particle-face contact is trivial and consists in building a unit vector $\mathbf{n}_{pF}$ normal to the face plane and an overlapping distance δ_{pF}. Let $\mathcal{N}_E$ be the number of edges of a face, $\mathbf{n}_{pF}$ and δ_{pF} are obtained by:

$$\mathbf{n}_{pF} = \frac{\mathbf{n}^*_{pF}}{||\mathbf{n}^*_{pF}||} \quad \text{with} \quad \mathbf{n}^*_{\text{pF}} = \frac{1}{\mathcal{N}_{\text{E}}} \sum_{\text{i}\in\mathcal{N}_{\text{E}}} \mathbf{e}_{\text{i}} \wedge \mathbf{e}_{\text{i+1}}, \tag{20}$$

$$\delta_{pF} = r_p - (\mathbf{x}_F - \mathbf{x}_p) \cdot \mathbf{n}_{pF}, \tag{21}$$

where $\mathbf{e}_i$ is the direction vector of edge E_i oriented in such a manner that $\mathbf{e}_i \wedge \mathbf{e}_{i+1}$ yields a vector oriented towards the outside of the domain, and $\mathbf{x}_F$ is the face center coordinates. As $\mathbf{n}_{pF}$ is unique, its value is stored in the appropriate data structure. The treatment of particle-edge contact consists in building a unit vector $\mathbf{n}_{pE}$ normal to the edge oriented from the particle center to the edge and an overlapping distance δ_{pE}. Let E_i be this edge, composed of points A and B:

$$\mathbf{n}_{pE} = \frac{\mathbf{n}^*_{pE}}{||\mathbf{n}^*_{pE}||} \quad \text{with} \quad \mathbf{n}^*_{\text{pE}} = [(\mathbf{x}_{\text{p}} - \mathbf{x}_{\text{A}}) \cdot \mathbf{e}_{\text{i}}]\mathbf{e}_{\text{i}} - (\mathbf{x}_{\text{p}} - \mathbf{x}_{\text{A}}), \tag{22}$$

$$\delta_{pE} = r_p - [((\mathbf{x}_p - \mathbf{x}_A) \cdot \mathbf{e}_i)\mathbf{e}_i - (\mathbf{x}_p - \mathbf{x}_A)] \cdot \mathbf{n}_{pE}, \tag{23}$$

where $\mathbf{e}_i = \mathbf{x}_B - \mathbf{x}_A$ is the direction vector of E_i. The treatment of particle-vertex contact consists in building a unit vector $\mathbf{n}_{pV}$ oriented from the particle center to the vertex and an overlapping distance δ_{pV} such as:

$$\mathbf{n}_{pV} = \frac{\mathbf{n}^*_{pV}}{||\mathbf{n}^*_{pV}||} \quad \text{with} \quad \mathbf{n}^*_{\text{pV}} = \mathbf{x}_{\text{V}} - \mathbf{x}_{\text{p}}, \tag{24}$$

$$\delta_{pV} = r_p - (\mathbf{x}_V - \mathbf{x}_p) \cdot \mathbf{n}_{pV}, \tag{25}$$

where $\mathbf{x}_V$ are the vertex coordinates. In this formalism, it can be noticed that the resolution of a particle-edge contact is tantamount to a particle-face contact of which face plane would be orthogonal

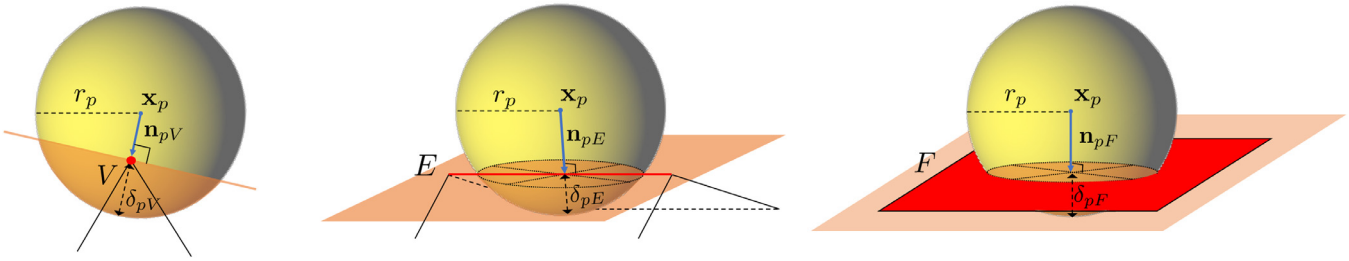

Fig. 17. A particle of radius r_p with center coordinates $\mathbf{x}_p$ overlapping (here unreasonably) a vertex (left), an edge (center) and a face (right). Unit normal vectors considered for collision force computation are indicated by $\mathbf{n}_{pV}$, $\mathbf{n}_{pE}$ and $\mathbf{n}_{pF}$, respectively. Interpenetration distances are indicated by δ_{pV}, δ_{pE} and δ_{pF}, respectively. For these three features, the method used considers the contact the same manner as the one with the pale red plane, which is orthogonal to the normal vector and contains the boundary feature. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

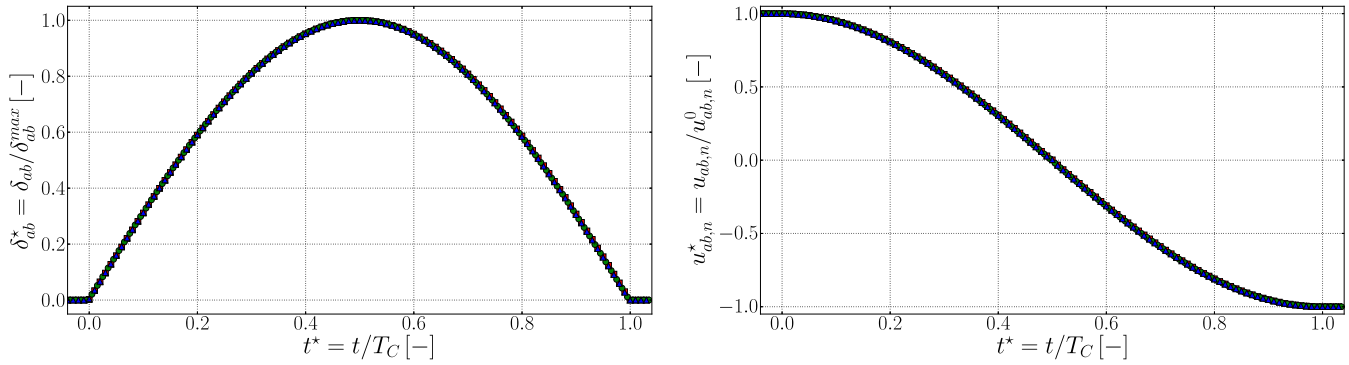

Fig. 18. Non-dimensional overlap (left) and non-dimensional normal velocity (right) as a function of the non-dimensional time during a particle-face (▲), a particle-edge (■) and a particle-vertex (●) contact. RK2 method with $\Delta t_p = T_C/100$ is used. Comparison with analytical method (—).

to $\mathbf{n}_{pE}$. Equally, the resolution of a particle-vertex contact is tantamount to a particle-face contact of which face plane would be orthogonal to $\mathbf{n}_{pV}$. The time evolution of a particle's overlap and translational velocity during the contact with a face, an edge and a vertex have been plotted in Fig. 18. Results are tested against the following non-dimensional analytical solution of the contact equation (see [4]) and exhibit the expected behaviors:

$$\delta^\star_{ab}(t) = \frac{\delta_{ab}(t)}{\delta^{max}_{ab}} = \frac{\omega_0}{\Omega_0}\exp\left(\gamma_n\left[\frac{1}{\Omega_0}\arcsin\left(\frac{\Omega_0}{\omega_0}\right) - t\right]\right)\sin(\Omega_0 t), \tag{26}$$

$$\mathbf{u}^\star_{ab,n}(t) = \frac{\mathbf{u}_{ab,n}(t)}{\mathbf{u}^0_{ab,n}} = \frac{1}{\Omega_0}e^{-\gamma_n t}[-\gamma_n\sin(\Omega_0 t) + \Omega_0\cos(\Omega_0 t)], \tag{27}$$

with

$$\Omega_0 = \sqrt{\omega_0^2 - \gamma_n^2}, \tag{28}$$

and $\mathbf{u}^0_{ab,n}$ being the particle's initial normal velocity.

5.3. Spatial sorting and null object detector

Numerous works interested in collisions involving complex shapes report drastic costs, and it clearly appears that setting a list of potential contacts between objects is of paramount importance to prevent a vast majority of unuseful operations from being performed. In this regards, various methods of spatial sorting such as the grid method, the octree technique, and the body-base approach have been reported in the literature [36]. This overcost is particularly verified in the case of so-called exhaustive contact schemes, which make no *a priori* assumptions about the problem evolution and reason based only on the present state of the geometry, such as the presented approach. As an example, the case of the Birmingham fluidized-bed run on 512 processors shows that more than 99.9% of the total CPU time would be dedicated to the treatment of boundaries in a brute-force approach for which all particles have to check collisions with every boundary faces. However, a deeper analysis of this case indicates that only 1% of the particles are actually colliding a boundary feature at a given instant, thus promising improvement prospects if a spatial sorting step is used to discard particles distant from the walls. In this case, it also reveals that each colliding particle hardly hit more than one object at a given instant, say barely 0.0005% of the total amount of boundary features, approximately. A part of the algorithm referred to as *null object detector* is thus mandatory in order to quickly discard a particle's farthest objects. First, the *null object detector* is described.

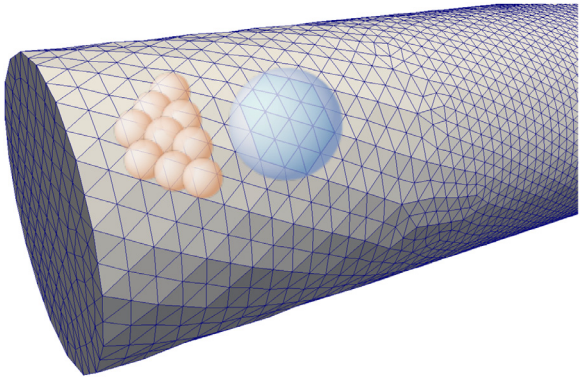

Fig. 19. Cylindrical domain meshed with tetrahedra. A Boundary Face Group (BFG) with its Bounding Sphere (BSBFG) is represented in blueish colors. Some orange Bounding Sphere of Faces (BSFs) are also visible.

A priori, the $\mathcal{VRS}$ belonging tests introduced in Section 5.1 for each particle have to be performed and for each V, E and F of all boundaries. To minimize the cost of this search, a new data structure is created from groups of adjacent wall faces belonging to the same boundary, called Boundary Face Groups (BFG, see Fig. 19). This additional coloring is obtained thanks to the METIS library [40]. The associated improved data structure is sketched in Fig. 20. These BFGs contain all the necessary boundary metrics and connectivities along with the $\mathcal{VRS}$ normals, computed in a parallel fashion. They also support other relevant geometrical data that are used for quick distance checking:

- the center $\mathbf{x}_{BSBFG}$ and radius r_{BSBFG} of each BFG Bounding Sphere (BSBFG) are computed using the BFG nodes mean coordinates and the distance between the center and the most remote BFG node, respectively, and stored.
- The center $\mathbf{x}_{BSF}$ and radius r_{BSF} of each Bounding Sphere of Face (BSF) are also computed using the face barycenter and its distance to the farther node of the face, respectively, and stored.

These preliminary operations find their justification in the fact that checking the intersection between two spheres is simple, but also among the quickest tests. In the literature, this is often referred to as the "sphere-tree" technique [41]. It consists in prioritizing the tests by using sets of spheres that describe the three-dimensional surface of an object at different levels of detail. In this study, a two-level hierarchy is employed : The BSBFGs stand for the first level, each one composed of several BSFs, which is the second level.

Because of the various geometrical and mesh configurations that can occur in complex cases, an object may be found very close

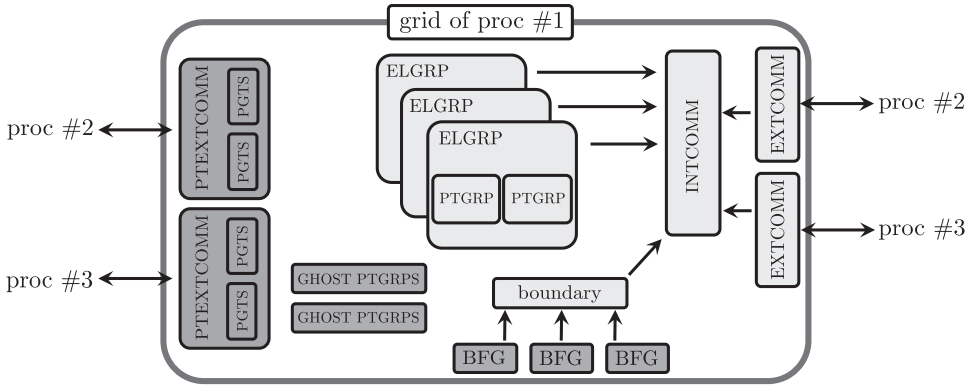

Fig. 20. Improved communicators structure containing Particle External communicators (PTEXTCOMMs) and Boundary Face Groups (BFGs).

to a boundary that doesn't share elements with the processor it is located in. To cope with this fact, it is conceivable to dispatch all BFGs on all processors, so that no omission is allowed. Implementing this solution could however result in unnecessary tests, all the more since most particles reside in the bottom of the reactor in the majority of fluidized-bed systems. The option selected is to rely on the list of processors sharing PTEXTCOMMs. By using an additional method looking alike the one explored in Section 4.1, this makes it possible to identify the closest BFGs of closest processors. The identified BFGs are then exchanged to constitute the ghost BFGs of each processor. In case of static mesh, these calculations are performed once during the solver initialization. As a preliminary analysis to the algorithm introduced in Section 5.1, the following method referred to as *null object detector* allows to identify a particle's closest boundary faces relying on the local and ghost BFGs:

1. The particle of interest p of center coordinates $\mathbf{x}_p$ and radius r_p loops over local and ghost BFGs. Distant BFGs are discarded if $||\mathbf{x}_{BSBFG} - \mathbf{x}_p||^2 > (r_{BSBFG} + r_p)^2$.
2. For each intersected BFG, p loops over all its faces' BSF. Distant faces are discarded if $||\mathbf{x}_{BSF} - \mathbf{x}_p||^2 > (r_{BSF} + r_p)^2$.
3. Eventually, only the faces of which bounding sphere is intersecting p are treated by the algorithm presented in Section 5.1.

All $\mathbf{x}_{BSBFG}$, r_{BSBFG}, $\mathbf{x}_{BSF}$ and r_{BSF} having been precalculated, only square distances have to be quantified during the run, thus avoiding costly square roots.

A spatial sorting step is added in order to prevent unnecessary bounding sphere intersection tests. Indeed, an optimal sorting would be able to discard all the particles of which distance to the wall exceeds their radius. In the same fashion as the cell halo identification dealt with in Section 4.1, this very first step focuses on flagging layers of mesh elements covering physical boundaries during the simulation initialization, so that only the particles located in these elements will be treated by the previous *null object detector* during the run. The problem can thus be formulated in the same terms: this close-boundary element flagging can be easily operated on Cartesian meshes, but requires further coding effort to deal with unstructured meshes, as mesh size heterogeneities are frequently encountered. Here again, the first option consists in flagging successive layers of cells in order to ensure sufficient identification, but without yielding certainty on the distance criterion, this method can result in the flagging of numerous unwanted cells in addition. On the contrary, the implemented approach focuses on local exact wall distance calculation, allowing the flagging of more elements in refined area and fewer in coarse ones. The objective is to compute the distance between some interior mesh nodes and the wall features to deduce whether a mesh element has to be flagged or not, relying on the previous *null object detector*, the $\mathcal{VR}s$ introduced in Section 5.1 and the contact resolution presented in Section 5.2. Each of these steps is detailed in the Appendix E. In case of a static mesh, these steps are performed once during the solver initialization. Mean results obtained from simulations of 1s physical time of the Birmingham fluidized-bed run on various number of processors show that 91% of the particles are eliminated by the spatial sorting test. Then, each near wall particle intersects 2.5 BSF in average thanks to the *null object detector*, thus drastically reducing the number of costly $\mathcal{VR}$ tests to perform. Eventually, these gains in selectivity enable the slowest processor to spend approximately 4% of its computation time in the treatment of boundary contacts.

The definitive procedure for particle-boundary contact treatment involving boundaries' closest element flagging, *null object detector*, use of Voronoi regions and contact resolution is displayed in Fig. 21.

Note that the parts concerning the Voronoi regions management and the contact resolution presented here could be considered as particular cases of the collision of two complex shape object (two non-spherical particles for instance). In this latter case, Voronoi regions are required on both colliding objects to find out the pair of closest features, then compute their overlapping distance and normal vector [14]. One of these objects being a sphere in our case, some simplifications arise. Many already existing parts in the current algorithm could be useful and directly applicable in more complex cases, for instance when considering the bounding sphere of non-spherical particles for quick discarding tests purposes.

As an illustration, numerical simulations were performed to measure the solid mass flow rate W across the orifice of diameter D_0 of an hourglass meshed with tetrahedron for six values of particle diameter d_p ranging from 7.5% to 15%D_0. No fluid phase was accounted in these simulations. Results were compared to the empirical law of Beverloo [42], frequently encountered in silos or hopper discharge studies, that can be written:

$$W = C\rho_p\sqrt{g}(D_0 - kd_p)^{5/2}, \tag{29}$$

where C and k are empirical discharge and shape coefficients respectively. Comparison is shown in Fig. 22, for which the constant C was set to a classical value of 0.55 [42]. In order to extract a value for k in this configuration, the following form of the law of Beverloo is plotted :

$$\frac{1}{D_0}\left(\frac{W}{C\rho_p\sqrt{g}}\right)^{2/5} = 1 - k\frac{d_p}{D_0}, \tag{30}$$

with which the numerical results exhibit a good agreement for $k \approx 1.18$. This value seems consistant with the literature [43].

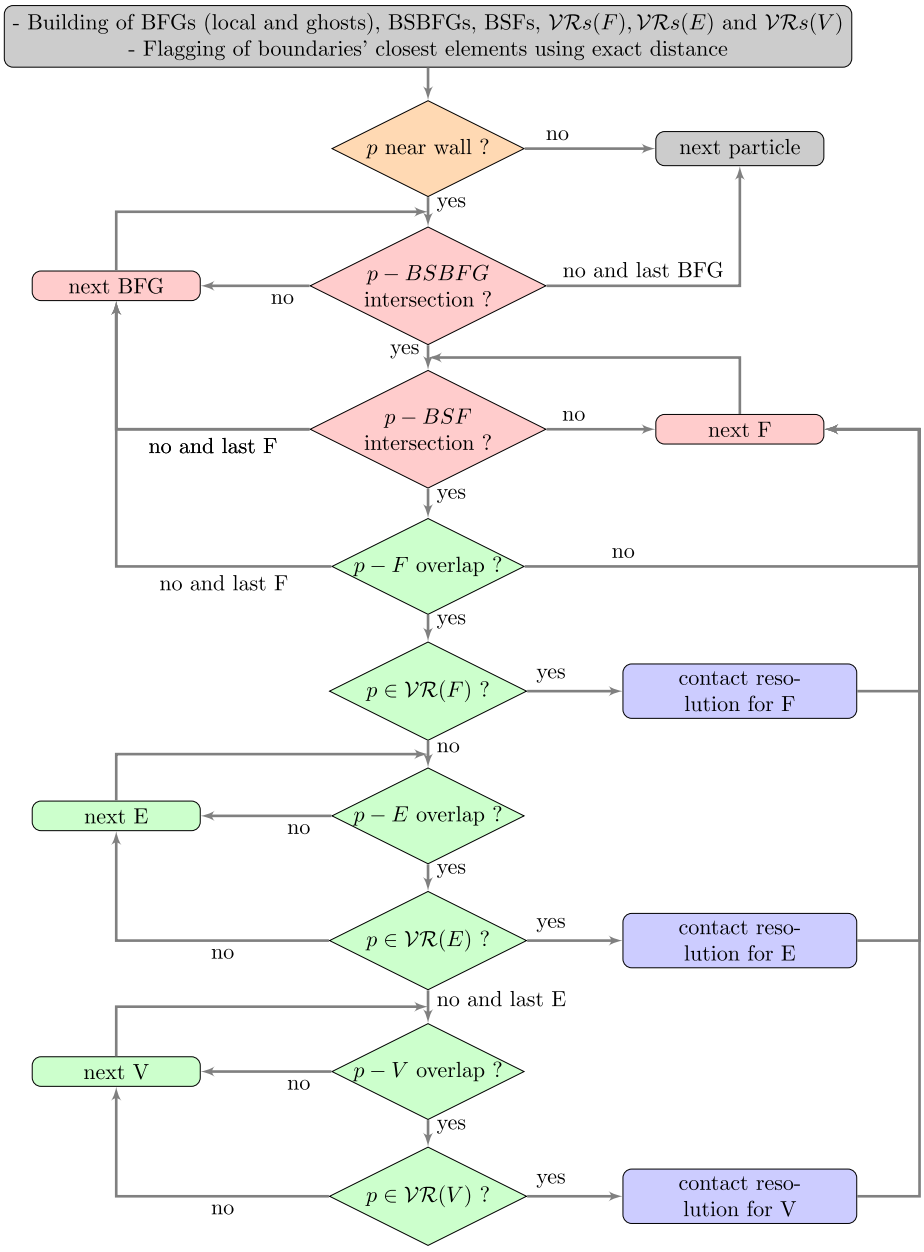

Fig. 21. Flow chart of the definitive procedure followed by each particle for boundary contacts treatment, adapted for arbitrary complex walls. : initializing part and output. : spatial sorting. : *null object detector*. : contact resolution parts. : $\mathcal{VR}$ tests.

6. The Birmingham gas-fluidized bed

6.1. Configuration

All the performance measurements assessed in the previous sections were performed for an isothermal dense gas-fluidized bed experimented at the University of Birmingham, which was previously already simulated by using TFM approach [20]. This pressurized lab-scale reactor is axisymmetric and composed of a cylindrical column of internal radius $R = 77$ mm widening to a internal radius of 127 mm. The vertical distance between the horizontal gas fluidization distributor plate and the top of the exhaust, corresponding to the computational domain, is 1.75 m. Nitrogen enters the distribution plate with a fluidization velocity u_{inlet} of 0.32 m/s and the pressure in the fluidized bed is 12 bar. The particle phase is almost monodisperse with a median diameter of 875 µm and a material density of 740 kg/m^3. The reactor is filled with approximately 9.6M particles (2.5 kg of solid material). Details can be found in [20]. The experimental setup and the computational domain are sketched in Fig. 23. The employed mesh is composed of 3.7M tetrahedra and divided in a refined zone in the smaller section part, with an average mesh element size of 1.85 mm, and a coarser zone in the upper part, with an average mesh element size of 3.9 mm. The tests were carried out on the Curie supercomputer of the TGCC center (Très Grand Centre de Calcul, France), featuring an InfiniBand QDR Full Fat Tree interconnect. The nodes used comprise two Intel Sandy Bridge octo-core processors running at 2.7GHz with 64GB RAM (about 4GB per core). The numerical parameters used for the simulations are summarized in Table 2.

6.2. Statistics

The numerical simulations are performed during 20 s of physical time. A first period of 10 s is employed to establish converged

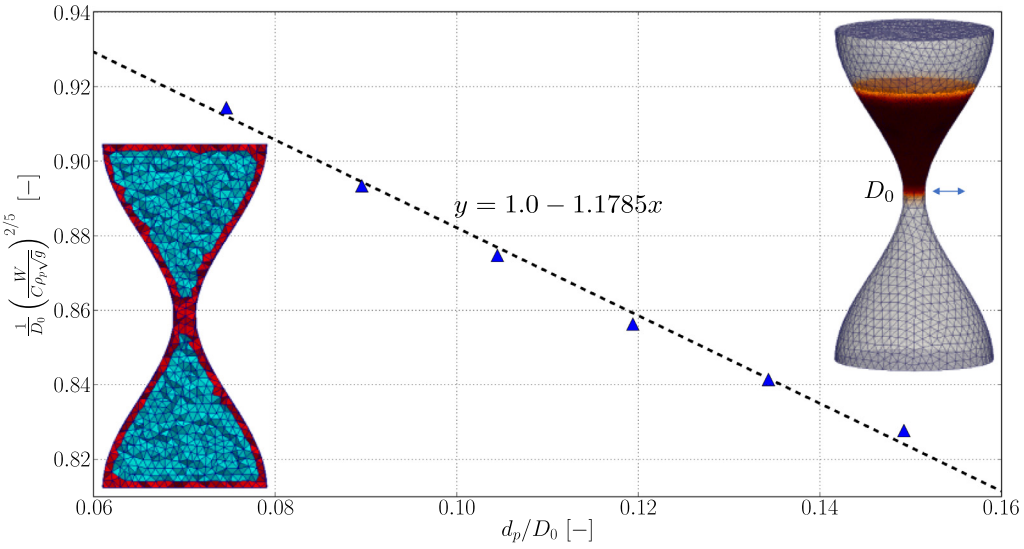

Fig. 22. Non-dimensional solid mass flow rate obtained for six different values of particle diameter (▲) compared with Beverloo law with $C = 0.55$ (- - -). The value of $k \approx 1.18$ is extracted from the slope (see Eq. 30). On the left, the mesh is displayed and the cells used for the spatial sorting are colored in red. On the right, the particles are shown at $t = 0$s and colored by the fluid fraction. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

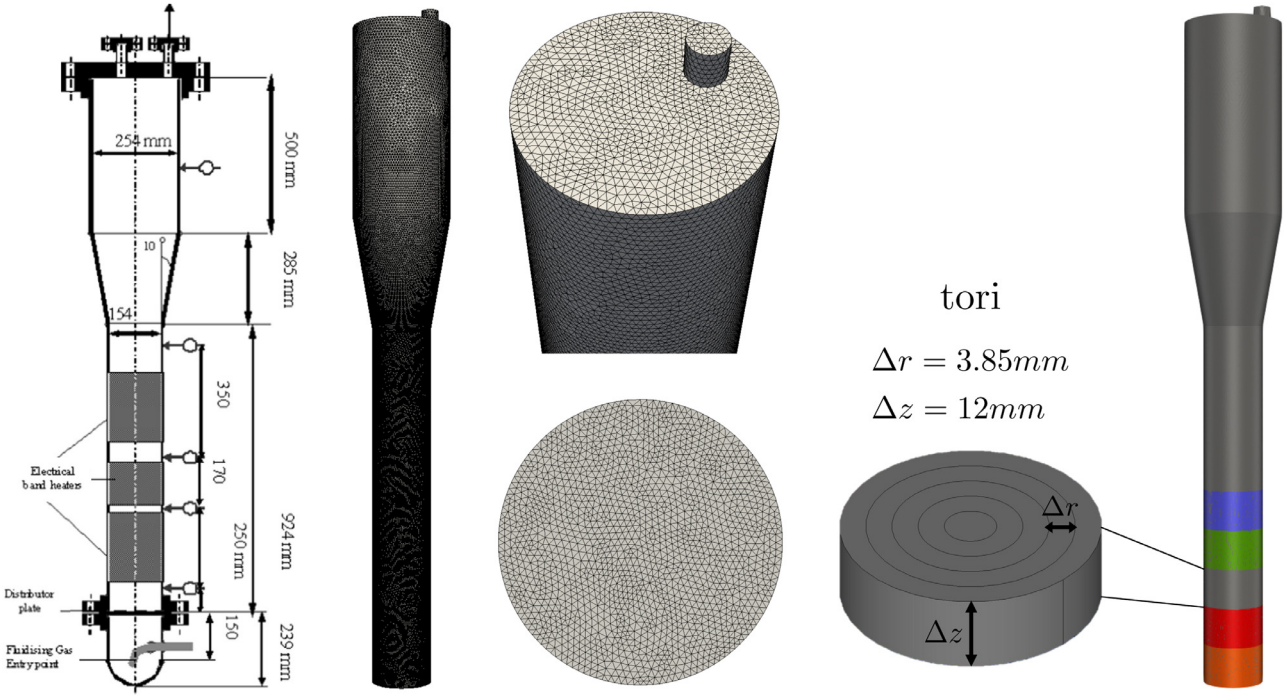

Fig. 23. Lab-scale Birmingham gas-fluidized bed. On the left, experimental setup extracted from [20]. In the center, front view of the computational domain meshed with 3.7M tetrahedra, accounting for the parts beyond the gas distribution plate, for a total height of 1.75 m. On the top, top-view featuring the chimney. On the bottom, distribution plate. On the right, domain decomposition into tori for statistics computation.

Table 2
Numerical parameters of the particle-particle and particle-wall soft-sphere collision model.

PARTICLE PHASE	
Spring stiffness k_n [N/m]	75
Normal restitution coefficient e_n [-]	0.9
Dynamic friction coefficient $\mu_{\tan}$ [-]	0.3

state regarding bed expansion (see Fig. 24) and pressure loss (see Fig. 25) through the bed, then time-averaged statistics are computed during the remaining 10s. It should be noted that the original simulations involving TFM [20] were carried out during 360s, the last 240s being used to compute statistics. Even these durations couldn't ensure complete statistical convergence.

The profile of the time-averaged pressure across the reactor is visible in Fig. 26. As expected, the profile displays two distinct slopes: for lower altitudes the slope corresponds to the pressure evolution across a particle bed with a given fluid fraction described by Ergun for fixed particle beds [23], while only gas is present in the higher region.

In order to assess possible comparisons with Euler-Euler formalism, particle physical quantities have to be translated into solid

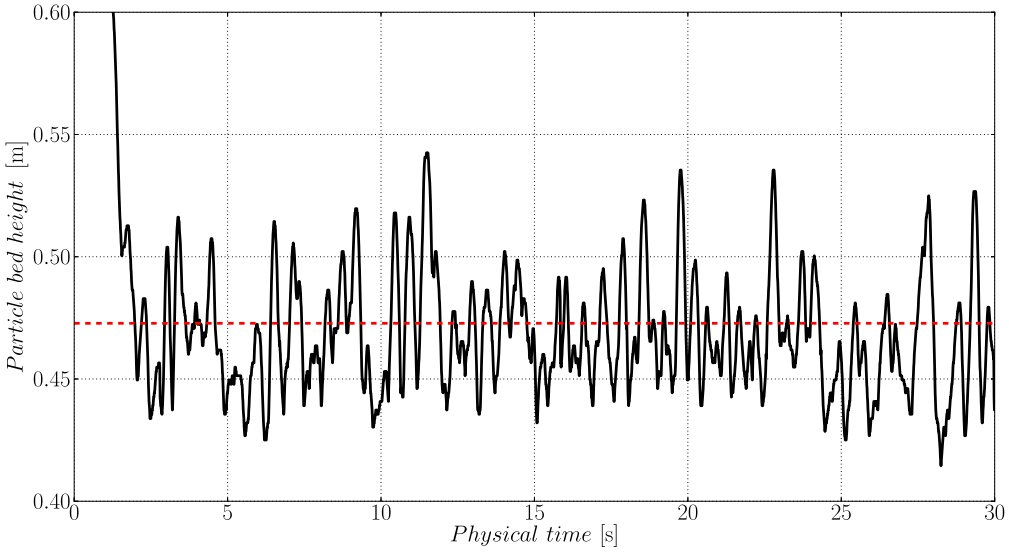

Fig. 24. Profile of the fluidized bed expansion (—) and its time-averaged value (- - -) measured in the simulation. The instantaneous bed expansion is extracted as the height of the 99th percentile of the particles. The time-averaged value is ≈ 0.47 m. At $t = 0$ s, the particles were seeded in a regular packing.

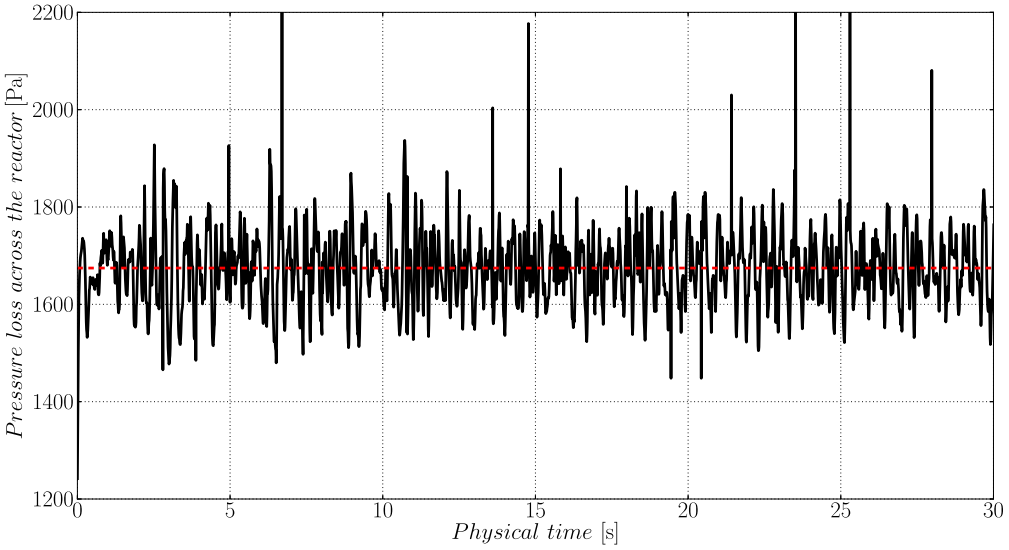

Fig. 25. Profile of the pressure loss across the fluidized bed (—) and its time-averaged value (- - -) measured between the inlet and the outlet in the simulation which is ≈ 1677 Pa. The value of the mean pressure loss neglecting walls' influence $\Delta P \approx (m_s + \varepsilon \rho V_{total}) \frac{g}{S_{inlet}}$ predicts a value of 1575 Pa. At $t = 0$ s, the particles were seeded in a regular packing.

quantities onto the Eulerian grid. $N_p(C,\ t)$ is the instantaneous amount of particles of which center is located inside the mesh cell C of volume V_C. The instantaneous solid velocity in C is:

$$\mathbf{u}_p(C,t) = \frac{1}{N_p(C,t)} \sum_{p \in C} \mathbf{u}_p. \tag{31}$$

In case $N_p(C,\ t)$ is null, $\mathbf{u}_p(C,\ t)$ is set to zero. The time-averaged solid velocity on a time T in the cell C is then defined by:

$$< \mathbf{u}_p(C) >_T = \frac{\sum_T \mathbf{u}_p(C,t) N_p(C,t) \Delta t}{\sum_T N_p(C,t) \Delta t}. \tag{32}$$

In case $\sum_T N_p(C,t)\Delta t$ is null, $< \mathbf{u}_p(C) >_T$ is set to zero. Owing to the axisymmetry of the reactor geometry, the spatial averaging of the time-averaged variables can be performed in the azimuthal direction. To this end, once these time-averaged fields in each mesh cell are computed, spatial means are performed by summing the contributions of all mesh cells of which center belongs to a given torus. Each torus has a height $\Delta z = 12$ mm and a difference between its exterior and interior radius $\Delta r = 3.85$ mm (see Fig. 23), as advocated by [20] to prevent too few accounted events while keeping relevant spatial information. In each torus is computed:

$$< N_p(torus) >_T = \sum_{C \in torus} \sum_T N_p(C,t) \Delta t. \tag{33}$$

Then, the spatial average of the time-averaged solid velocity in the torus is:

$$< \mathbf{u}_p(torus) >_T = \frac{\sum_{C \in torus} < \mathbf{u}_p(C) >_T \sum_T N_p(C,t) \Delta t}{< N_p(torus) >_T}. \tag{34}$$

In case $< N_p(torus) >_T$ is null, $< \mathbf{u}_p(torus) >_T$ is set to zero. The necessary passing in cylindrical coordinates to compute the spatial

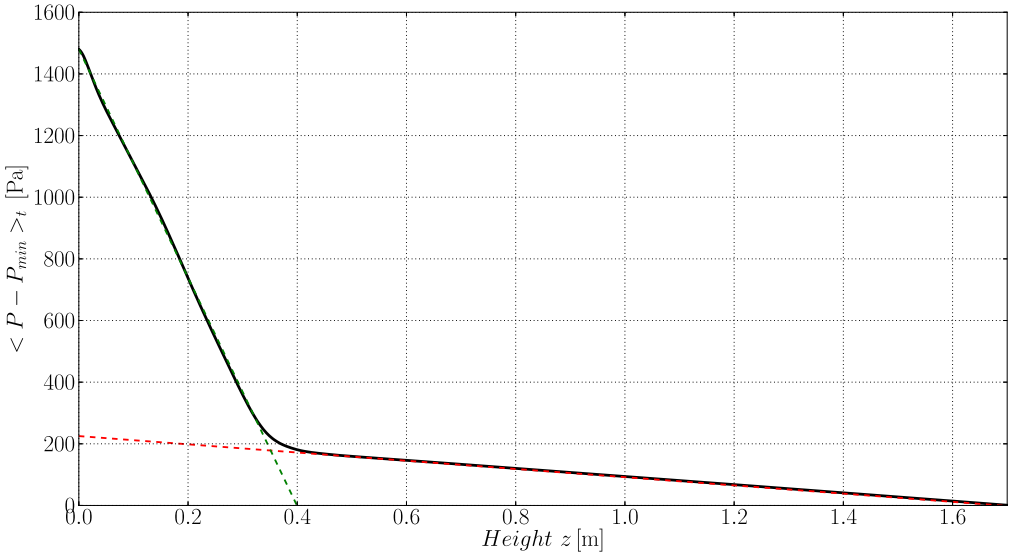

Fig. 26. Profile of the time-averaged pressure minus the minimum pressure both taken at the central axis of the cylinder (▬). The pressure loss across a particle bed given by Ergun empirical correlation [23] calculated for $\varepsilon = 0.66$ (- - -) and the pressure loss given by hydrostatic law (- - -), both shifted to ease comparison, are indicated. In the simulation, the mean fluid fraction measured inside the bed is ≈ 0.62.

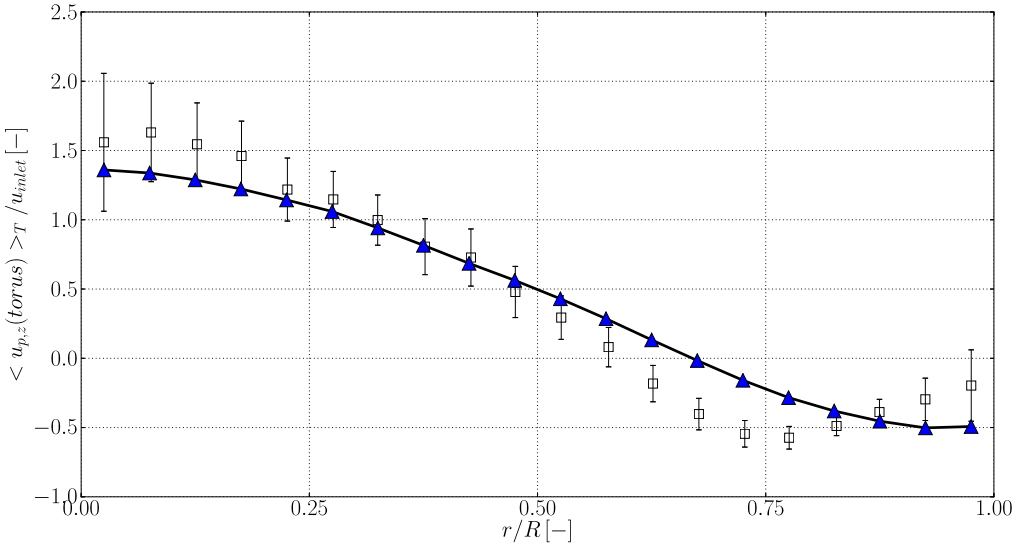

Fig. 27. Radial profile of time-averaged solid vertical velocity normalized by the fluidization velocity measured in the tori at $z = 3.45R$ (▲) compared with experimental values with error bars (□).

averages of quantities related to velocity is not detailed here, as only vertical components are shown. These statistics are assessed once every 10 fluid timesteps.

The radial profile of time-averaged solid vertical velocity normalized by the fluidization velocity measured at $z = 3.45R$ are visible in Fig. 27, for which experimental values are available [20]. In the center of the reactor the experiment exhibits an upward mean solid velocity for $0 \leq r \leq 0.6R$, of which maximum reaches $\approx 1.6u_{inlet}$. Beyond $0.6R$ a downward solid flow is observed. The maximum of the downward solid velocity is found at $r \approx 0.75R$ and reaches $\approx 0.6u_{inlet}$. At $r \approx 0.75R$, the slope changes and the measured mean solid velocity at the wall is nearly equal to zero. The predictions are in good accordance with the experiments at the center of the reactor but in the near wall region the downward solid velocity is overestimated by the numerical simulation, that fails in reproducing the location of the slope twist. This problem has already been reported for TFM simulations [20] and was interpreted as an underestimation of the particle-wall friction. It has been addressed by accounting for a wall roughness effect through the particle velocity boundary conditions. But the increase of the effective particle-wall friction may also be due to the non-spherical shape of the real particles. The CFD/DEM approach looks very promising to clarify such a crucial effect and to support the development of more satisfactory TFM particle wall boundary conditions.

7. Performances of the solver

The global performances of the solver are quantified in this section, by investigating its speed-up (strong scaling) and its scale-up (weak scaling). A canonical case containing homogeneously dispatched particles is first studied, then the Birmingham fluidized bed from Section 6 in a second step.

7.1. Measurements on a canonical case

Some measurements are first extracted from a canonical isothermal case, disregarding its relevancy concerning physics. This case consists of a cubic box meshed with tetrahedra. Particules are randomly seeded in the box, with a mean porosity about 0.54, and each tetrahedron contains roughly 11 particles. A fluid phase is present, to account for the cost of the interpolation and projection steps. The particle timestep is chosen so that $\Delta t = 10\Delta t_p$. Thus, ten particle timesteps are performed for each fluid timestep, which corresponds to a usual substepping configuration. Measurements are obtained from a single fluid timestep. All the tests were carried out on the regional supercomputer Myria of the CRIANN center (Centre Régional Informatique et d'Applications Numériques de Normandie, France), featuring a Intel Omni-Path interconnect. The processors used are two-sockets Intel Broadwell with 14 cores running at 2.4GHz with 128GB RAM (about 4 GB memory per core) for total peak power of 400TFlop/s.

The speed-up is first obtained by running the same simulation on different numbers of cores, ranging from 532 (reference case) to 4144. Each simulation roughly gathers 38M tetrahedra and 410M particles. The reference CPU time t_{CPU}^{ref} being associated with the temporal loop of the solver on $Nprocs^{ref} = 532$ cores, the speed-up for a CPU time t_{CPU} on $Nprocs$ is calculated by:

$$\text{speed-up}(Nprocs) = Nprocs^{ref} \times \frac{t_{CPU}^{ref}}{t_{CPU}}, \tag{35}$$

and is illustrated on Fig. 28. The solver exhibits a good scalability up to 4144 cores, with a speed-up reaching 80% of the ideal scaling.

Secondly, the scale-up of the solver is quantified by measuring the performances at constant load per core on different numbers of cores, ranging from 252 (reference case) to 4144. The number of particles per core is about 99k, and the number of tetrahedra is about 9.1k per core. The scale-up is given by:

$$\text{scale-up}(Nprocs) = Nprocs^{ref} \times \frac{t_{CPU}^{ref}}{N_c^{ref}} \frac{N_c}{t_{CPU}}, \tag{36}$$

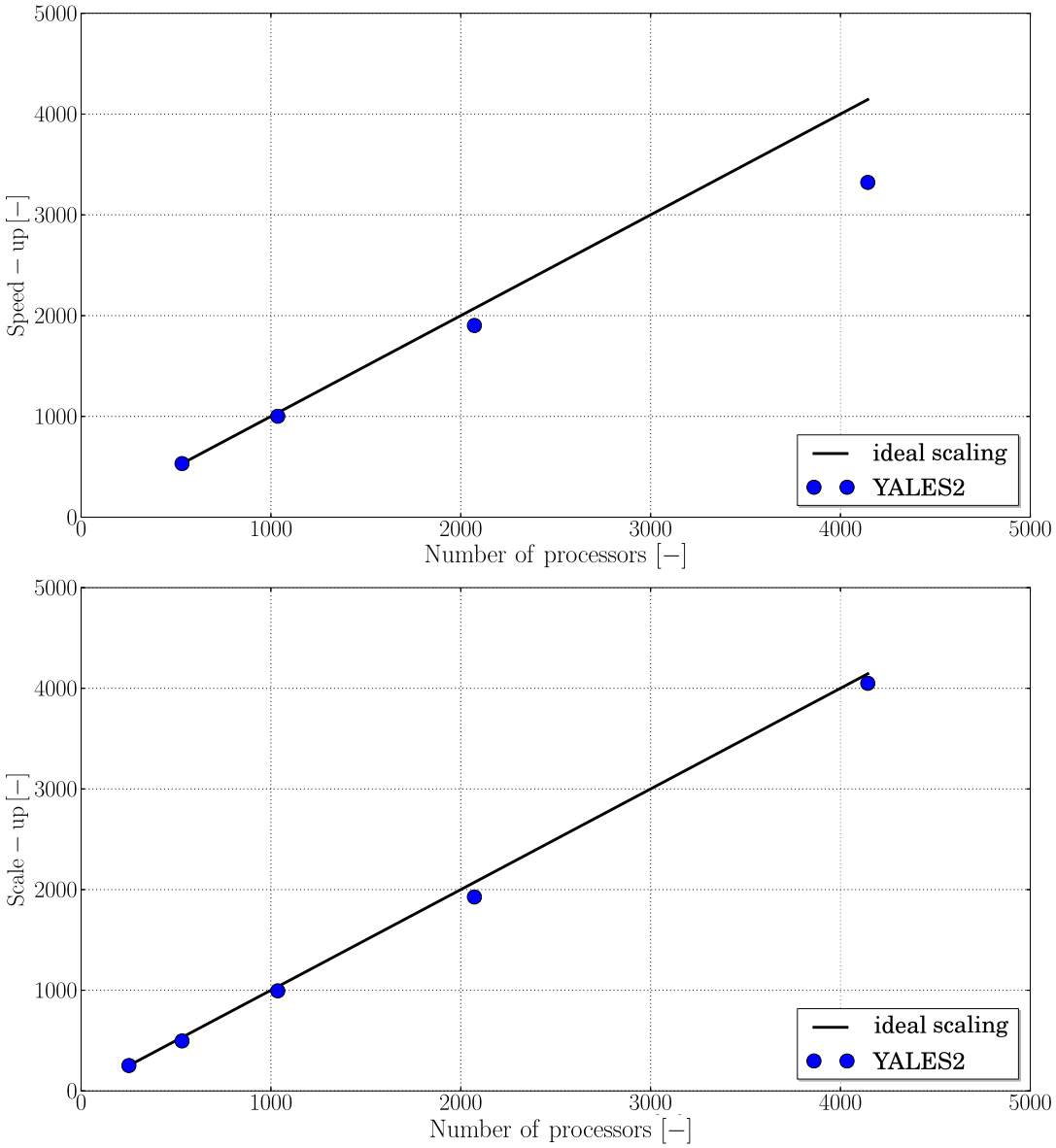

Fig. 28. Speed-up (up) and scale-up (bottom) curves extracted from the canonical case.

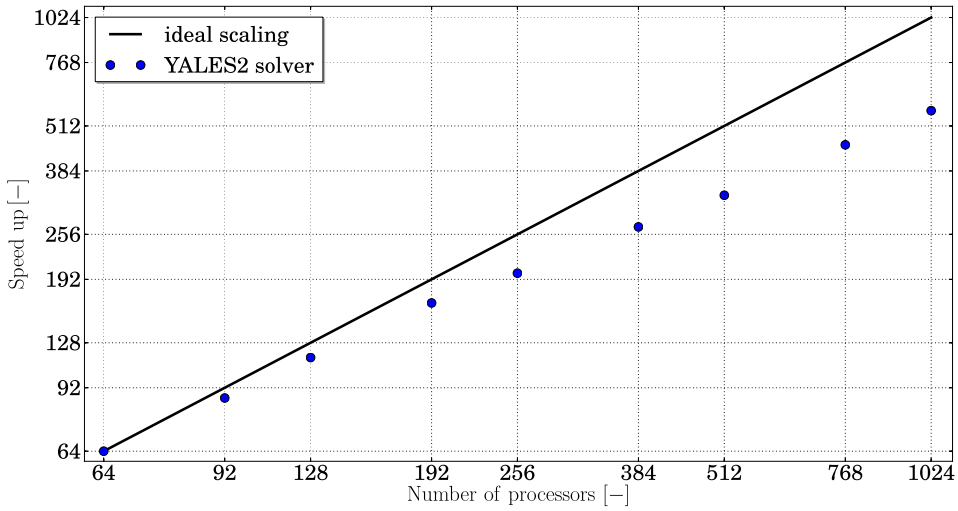

Fig. 29. Speed-up curve obtained on the Birmingham fluidized bed.

where N_c^{ref} and N_c are the number of cells in the reference case and in the current case, respectively. The scale-up curve is represented in Fig. 28 and shows an excellent scaling.

7.2. Measurements from the Birmingham case

Finally, the analysis is completed by measuring the performances on the fluidized bed experimented at the university of Birmingham and described in Section 6. This case gathers 10M particles in a cylindrical domain meshed with 3.7M tetrahedra. Here, the performances are extracted over 1s of physical time after the stabilization of the bed fluidization has been assessed by monitoring both the bed height and the pressure loss across the system. Thus, this case deals with realistic conditions where the execution speed highly depends on the local physics (presence of dense or void zones), and the fluid and particle timesteps are recomputed throughout the simulation. The tests were carried out on the Curie supercomputer.

The speed-up is obtained by running the simulation on various numbers of cores, ranging from 64 (reference case) to 1024 cores, and is illustrated in Fig. 29. The solver reaches 55% of its ideal scaling value, which is acceptable considering the high dispersion of the particles amongst the cores, causing their de-synchronizing.

7.3. Dynamic load balancing algorithm

In simulations such as the Birmingham case, a convenient scalability is hardly obtained because the particles mainly reside at the bottom of the reactor (see Fig. 24), which alternates between void regions (gas bubbles) and dense regions (clusters). Some tests carried out on the Birmingham fluidized bed case show that more than 50% of the processors don't contain any particle when using regular partitionning, and in this case the computational cost of the fluid phase remains negligible compared to the solid phase's. This problem is all the more visible since particles are usually more numerous than the mesh elements. A load balancing algorithm provided by the METIS library [40] is then used to help cope with this mismatch. Provided that some weights are attributed to each ELGRP, it is able to build a double-constraint partitioning on the ELGRPs graph, yielding a better repartition of the weight per processor while guaranteeing contiguous partitions. Finally, ELGRPS are transferred between processors according to the graph. The way the ELGRP weights are calculated to be supplied to METIS is of paramount importance to ensure both the convergence and the

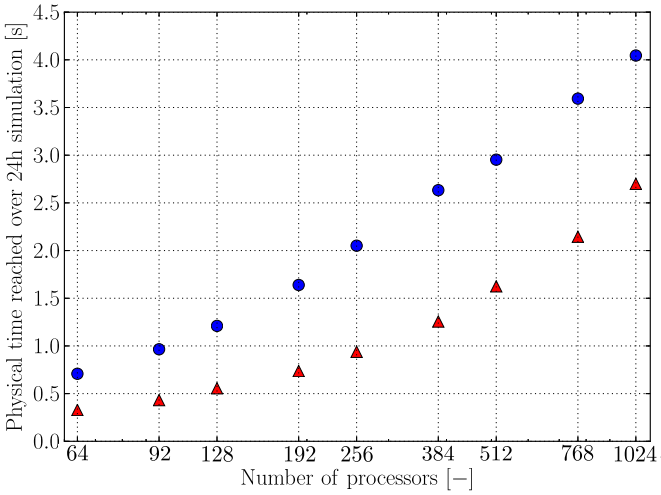

Fig. 30. Physical time reached over 24h simulation of the Birmingham fluidized bed without (▲), and with dynamic load balancing (●). For this test the maximum particle time step was set to $\Delta t_p = T_C/6$.

efficiency of the load balancing algorithm. Indeed, imposing additional constraint on the regular mesh coloring may result in a high cost overhead regarding the Eulerian phase, because enhancing the particles' balance is likely to deteriorate the fluid's. In this study, the ELGRP weights are computed so that $\approx 85\%$ of processors are located under the fluidized bed height (see Fig. 24). Moreover for the ELGRP under the bed height, higher weight is given to ELGRP containing more particles. The algorithm is designed so that it loosens this latter constraint in case the graph building leads to too many empty processors. The load balancing is dynamically updated if the bed height shift exceeds 10% of its previous value, or if too many processors inside the bed have a weight too far from the ideal theoretical weight, which should be obtained if all particles are equally split between each inside bed processor. Depending on the number of processors and the second constraint intensity, analysis reveals that an average of $\approx 10-15$ partitionning steps are necessary to maintain a sufficient load balance quality for 1 second of simulation, according to the previously mentioned criteria. A scaling of the code obtained on the Birmingham fluidized bed simulation is displayed in Fig. 30. The time saving provided by the dynamic load balancing algorithm exceeds 2 up to 384 processors.

Starting from 512 processors the gain decreases, until it reaches 1.5 for 1024 processors. This is due to the fact that the second constraint has automatically been loosened by the code to help build the graph: indeed for large numbers of processors, there are fewer elements per processor and the number of degrees or freedom for load balancing is reduced. It can be noticed that the simulation on 384 processors with load balancing nearly achieves the same performances as the simulation on 1024 processors without load balancing. Among these simulations, the maximum cost observed for the load balancing algorithm was 5% of the solver temporal loop.

8. Conclusion

In this paper, a methodology for the massively parallel 3D simulations of gas-fluidized bed in complex geometry on unstructured meshes in the framework of CFD/DEM is proposed, along with its implementation in the YALES2 code. This approach enables to run simulations involving several million particles in realistic geometries. It features an efficient fully non-blocking MPI algorithm for extra-processor communications with a packing/unpacking of the necessary halo data, and allows to solve particle/complex wall contacts explicitly relying on the Voronoi regions decomposition of the walls' features. The code shows a good scalability. A dynamic load balancing algorithm is implemented to limit the overcost due to the location of the particles by processing a better coloring of the processors computational domains.

This methodology has been successfully applied on the pilot-scale gas-fluidized bed that was experimentally studied at the University of Birmingham gathering 9.6M particles. Here, a first analysis of the results shows a satisfactory agreement with the available experimental data. A more detailed study is currently under progress to characterize, in particular, the influence of particle-particle and particle-wall interaction parameters, involving a full comparison with an Euler-Euler code. Many other types of simulations involving DEM could benefit from the implemented approach.

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.

CRediT authorship contribution statement

Yann Dufresne: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization, Supervision. **Vincent Moureau:** Conceptualization, Methodology, Software, Resources, Writing - review & editing, Project administration, Funding acquisition. **Ghislain Lartigue:** Conceptualization, Methodology, Software, Resources, Writing - review & editing, Project administration, Funding acquisition. **Olivier Simonin:** Methodology, Software, Writing - review & editing.

Acknowledgements

This work was granted access to the HPC resources from TGCC-CEA (Très Grand Centre de calcul) under the allocations x20162b7345 made by GENCI (Grand Equipement National de Calcul Intensif), and from IDRIS (Institut du Developpement et des Ressources en Informatique Scientifique) through the MORE4LESS project (Modelling of reactive particulate flows for low energy sustainable processes) coordinated by the IFP-EN laboratory (France).

Appendix A. Phase coupling

The coupling between the particle and fluid phases is a key point for the modeling of particle-laden flows, especially when the particle size approaches the Eulerian cell size. Many Eulerian fields have to be interpolated at the center of the particles for the numerous closures, as shown in Section 2.2. In the YALES2 solver, particles are located in a unique mesh cell using the position of their center. For any Eulerian scalar or vector field $\Phi(\mathbf{x}, t)$, its value taken at the particle p center $\Phi_{@p}(t)$ obeys:

$$\Phi_{@p}(t) = \sum_{i \in C} \omega^C_{p,i} \Phi(\mathbf{x}, t) \quad \text{with} \quad \sum_{i \in C} \omega^C_{p,i} = 1. \tag{A.1}$$

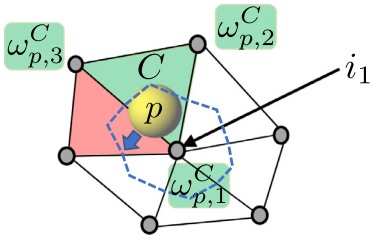

Fig. A.31. 2D representation of the six surrounding cells (SC_1) of a central node i_1. A particle p located in cell C ( ) is moving towards a neighboring cell ( ). : mesh nodes. - - - : contour of node i_1 control volume (ΔV_1). $\omega^C_{p,i}$:interpolation weight of particle p on node i.

Referring to Fig. A.31, i is a node index so that '$i \in C$' means 'all nodes i composing the mesh cell C in which the particle p is located'. $\omega^C_{p,i}$ is the interpolation weight of the particle p on cell node i and is calculated using a trilinear interpolation on hexahedra and a linear interpolation on tetrahedra. The same interpolation weights are used for data transfer from grid to particles (interpolation step) and from particles onto the grid (projection step).

The conservative projection operator needed to compute $\mathbf{F}_{p \to f}$ and $Q_{p \to f}$ (see Eqs. (2) and (3)) is thus written on each node i as:

$$\mathbf{F}_{p\to f,i} = -\frac{1}{\Delta V_i} \sum_{p \in SC_i} \omega^C_{p,i} (F_D + F_P) \quad \text{and} \quad Q_{p\to f,i} = -\frac{1}{\Delta V_i} \sum_{p \in SC_i} \omega^C_{p,i} Q_F, \tag{A.2}$$

where ΔV_i denotes the control volume of node i. The fluid fraction at node i is written:

$$\varepsilon_i = 1 - \frac{1}{\Delta V_i} \sum_{p \in SC_i} \omega^C_{p,i} V_p. \tag{A.3}$$

'$p \in SC_i$' means 'all particles belonging to any surrounding cell SC_i of node i'. Still referring to Fig. A.31, any particle belonging to one of the SC_1 cells will be accounted for when computing ε (as well as $\mathbf{F}_{p \to f}$ and $Q_{p \to f}$) at node i_1. It should be noticed that the particles' rotation doesn't affect the fluid.

This method consisting in distributing particles' quantities only in the cell where its center resides, referred to as particle centroid method, can lead to large calculation errors in particular regarding the fluid fraction, as pointed out in [44]. This is partly due to the fact that many CFD/DEM codes feature a staggered grid where the fluid fraction is defined at cell centers, causing dramatic discontinuities in time and space derivatives when a particle enters or leaves a cell. Here on the contrary, the fluid fraction and all the Eulerian fields are computed at the grid nodes. As depicted in Fig. A.31, any particle crossing the interface from the green cell to the red one won't cause any discontinuity on the computation of neither $\omega_{p,1}$ nor $\omega_{p,3}$. Indeed, the linear interpolation ensures that the interpolation weights are piecewise linear in each cell, but also continuous at cells' interface. Moreover $\omega_{p,2}$ won't be affected neither during the crossing, because as the particle approaches the cell's interface, $\omega_{p,2}$ tends towards 0. This is still true in 3D cases and on Cartesian meshes.

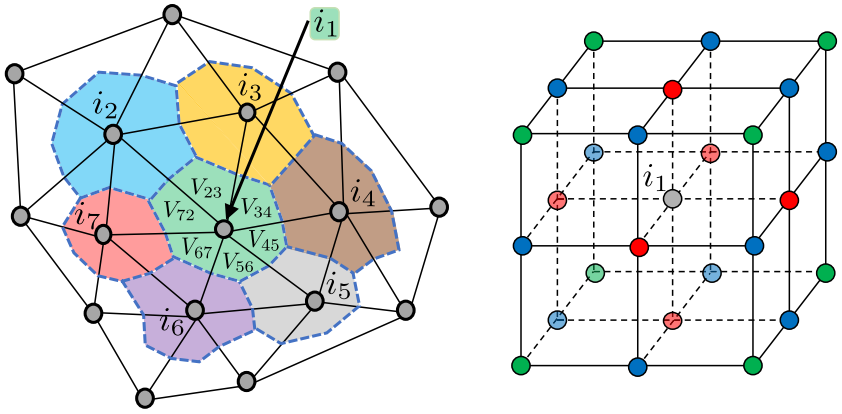

Fig. A.32. On the left: 2D representation of an unstructured mesh. ● :mesh nodes. The control volumes of nodes i_1 (■), i_2 (■), i_3 (■), i_4 (■), i_5 (■), i_6 (■), i_7 (■) are shown. The control volume ΔV_{i_1} of node i_1 is composed of volumes V_{23}, V_{34}, V_{45}, V_{56}, V_{67} and V_{72}. On the right: 3D representation of a regular Cartesian mesh part. ● : node i_1. ● : nodes at Faces' Center (FCN). ● : nodes at Edges' Center (ECN). ● : nodes at Corners (CN).

Nevertheless, it is well known that the particle centroid method can induce inaccuracies and lead to numerical instabilities because it cannot prevent the fluid fraction from reaching unrealizable values, in particular when dealing with close to unity particle diameter/mesh cell size ratios. In such cases, the fluid fraction value can locally decrease below the theoretical packing limit. To cope with this limitation, a filtering operator well suited for distributed memory machines is used. Here is explicited a 2D example case as shown in Fig. A.32 rather than a lesser clear mathematical description: this filtering operator is built for any Eulerian field noted Φ_i, its filtered value being $\hat{\Phi}_i$. At node i_1, $\hat{\Phi}_i$ reads:

$$\hat{\Phi}_{i_1} = \frac{1}{3}\Phi_{i_1} + \frac{1}{3\Delta V_{i_1}}\Big[(V_{23}+V_{34})\Phi_{i_3} + (V_{34}+V_{45})\Phi_{i_4} + \cdots + (V_{72}+V_{23})\Phi_{i_2}\Big], \tag{A.4}$$

where ΔV_{i_1} is the control volume associated to node i_1 and S_{mn} is the part of ΔV_{i_1} contained in the face delimited by nodes i_1, i_m and i_n, as shown on the left in Fig. A.32. If all the control volumes are equal, Eq. A.4 becomes:

$$\hat{\Phi}_{i_1} = \frac{1}{3}\Phi_{i_1} + \frac{1}{9}\sum_{m\in[\![2;7]\!]}\Phi_{i_m}. \tag{A.5}$$

The same type of filter can be derived in 3D. The following equation gives the value of $\hat{\Phi}_{i_1}$ in a 3D structured case with all equal control volumes as shown on the right in Fig. A.32:

$$\hat{\Phi}_{i_1} = \frac{1}{8}\Phi_{i_1} + \frac{1}{64}\left(4\sum_{m\in FCN}\Phi_{i_m} + 2\sum_{m\in ECN}\Phi_{i_m} + \sum_{m\in CN}\Phi_{i_m}\right). \tag{A.6}$$

This fully conservative operation being performed on all volumes at the same instant provides a filtered field, and can be repeated several times to increase the filter width. It should be underlined that for the computation of $\hat{\varepsilon}$, the filtering step is applied before dividing by the local control volume in order to conserve the total solid mass over the whole computational domain volume Ω:

$$\text{Total solid mass} = \rho_p\int_\Omega(1-\varepsilon)dV = \rho_p\int_\Omega(1-\hat{\varepsilon})dV. \tag{A.7}$$

In order to filter intensive quantities such as $\mathbf{F}_{p\to f,i}$ and $Q_{p\to f,i}$, these are multiplied by the local control volume value beforehand. The resulting extensive field can therefore be filtered conservatively, and after dividing by the local control volume value, allows to recover the desired filtering. Finally, for an intensive field:

$$\hat{\Phi} = \frac{\widehat{\Phi\cdot\Delta V}}{\Delta V}. \tag{A.8}$$

The properties of such a filtering operator, *i.e.* its moments, are not straightforward to determine on unstructured meshes but it can be noticed that it is based on direct neighbors and thus doesn't need distant nodes, hence its attractiveness regarding parallelism. The main drawback is that the filter width can't be directly obtained because it depends on the local mesh size. Thus, when using this filtering operator, the filter width cannot be prescribed.

Appendix B. Fast-marching like method for building cell halo around processors' domain

As computing the exact distance between a mesh node and a processor domain border would involve numerous calculation steps that could necessitate inter-processor communications, the implemented approach uses an approximate distance. The algorithm is inspired by fast-marching algorithms developed for Level-Set Methods [30,31]. In particular, it relies on the mapping of the surface of the processor domain border using points called markers, of which coordinates are automatically generated. The general idea is to propagate from node to node these coordinates, from the processors' interface towards neighboring processors.

The algorithm is conducted through the following points, as sketched in Fig. B.33, with $d_{\min}$ being the instructed distance, *e.g.* the desired halo thickness:

1. flagging the nodes belonging to the border (level 0 nodes) and each mesh element with a level 0 node (level 0 elements). Thus, level 0 elements share at least one node, one edge, or one face with the interface. Unflagged nodes of these level 0 elements are flagged (level 1 nodes). Thus, level 1 nodes belong to cells contacting the interface.
2. creating a marker's list on each mesh node. This list can contain the coordinates of several markers, as well as their distance d from the current node. At this moment, level 1 nodes' lists are filled, which consists in:
   (a) Looping over level 0 cells,
   (b) For each of them, looping over level 1 nodes,
   (c) For each of these nodes, generating some markers on the node(s), edge(s) or face(s) that the current level 0 cell is sharing with the interface,
   (d) Computing all the distances between these markers and the current level 1 node,
   (e) Filling the current level 1 node's list with the coordinates and the distance of the closest markers, and sorting them in descending order accordingly.

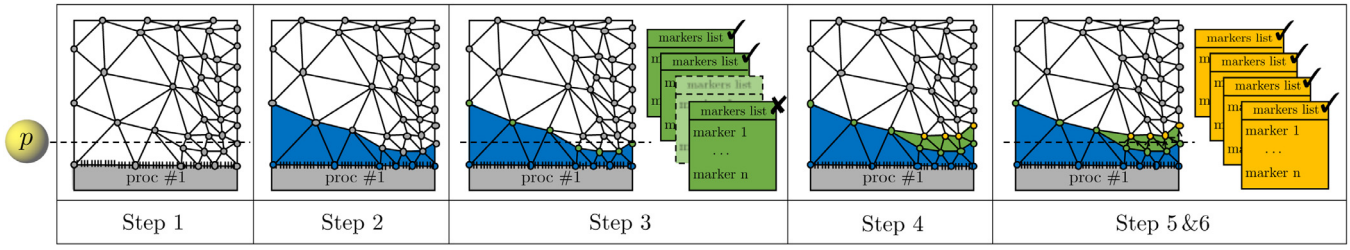

Fig. B.33. Fast-marching-like algorithm. The distance instruction $d_{\min}$ corresponds to the p particle radius and a colored element or node means it is flagged. Markers are generated on the processor #1 domain border. : level 0 elements and nodes. : first layer of level 1 elements, nodes and their markers list. Here some markers are closer to the node than the particle radius. : second layer of level 1 nodes and their markers list. Now all markers are at a sufficient distance. All colored elements belong to the cell halo around processor #1.

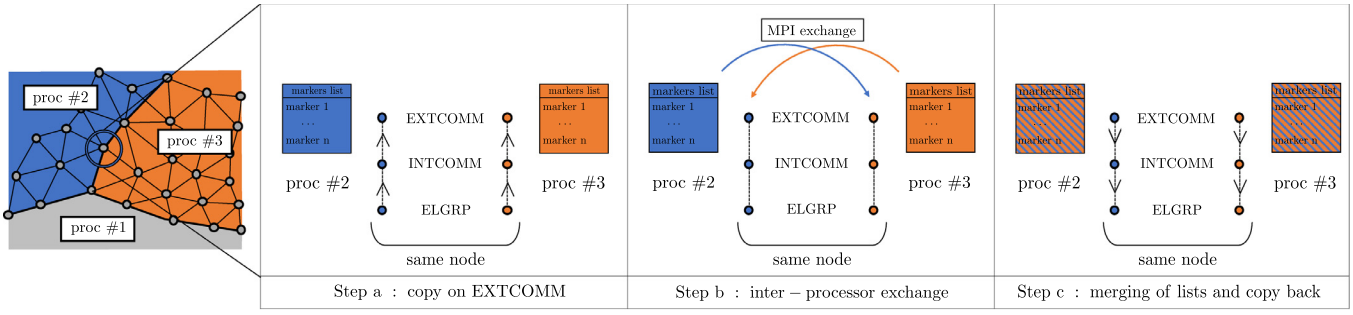

Fig. B.34. Parallel update performed during fast-marching algorithm.

3. Once treated all level 1 nodes, markers lists are propagated: each level 1 node updates its list by merging successively its actual list with the one of the other level 1 nodes sharing a connection. During the merging, only the closest markers are kept, and sorted.
4. The markers list of each level 1 node are checked to verify if the closest marker (the first one) is at a distance d_{approx}, which represents the best distance approximation, such that $d_{approx} \leq d_{\min}$. If so, all unflagged elements containing the node are flagged (level 1 elements), as well as their nodes (level 1 nodes).
5. repeating items 3 and 4 until there is no more flagged nodes such that $d_{approx} \leq d_{\min}$.

Note that during the markers lists' update step (step 3), unicity tests are performed to prevent several copies of a same marker to swarm in a list. Moreover, at step 2(c), markers are successively generated during the treatment of each level 0 cell, thus allowing buffering, hence the memory saving even in case a large quantity of markers is desired.

The previous algorithm is designed to give the same results if more than two processors are involved during the markers list propagation, *i.e.* when the list propagates across another processor domain border. To this end, at step 3, a parallel update is performed involving the data structures introduced in Section 3.2, as depicted in Fig. B.34: Each duplicated node (one copy on proc #2, one copy on proc #3) updates its own markers list from local neighboring nodes. Next, these local lists are copied to the INTCOMM, then towards the EXTCOMM where an MPI operation allows to share the two lists. Finally, in the INTCOMM of each processor, the update step presented in step 3 can be performed between the local node's markers list and its copy's, so that each duplicated node ends up with the same updated markers list.

It can be easily proved that the difference between the exact node-interface distance d_{exact} and the approximated distance given by the fast-marching-like method d_{approx} is negative (or null at its best), meaning that a node may appear farther than what it really is. This can cause problematic situations in case $d_{exact} \leq d_{\min} \leq d_{approx}$, leading to a cell misidentification. The gap between d_{exact} and d_{approx}, which is all the more wider that the closer the nodes are, can be statistically reduced by increasing the number of markers used to map the interface. However, to avoid errors occurring during the propagation of the markers lists (such as the forgetting of some markers), it is fundamental that the size of the lists, *e.g.* the number of markers they can store, is increased accordingly with the number of markers. Indeed, if some generated markers are missing in the list of the first layer of level 1 nodes, these can never be retrieved at a later stage, *e.g.* for farther nodes. Some tests performed on a very realistic unstructured case have shown that the choosing of 30 markers per face of the interface (along with a few ones per edge and one per node), with lists containing up to 60 markers, allows the maximum absolute error among all halo nodes to lie below $\Delta x/20$, Δx being the characteristic mesh size. In these conditions, an additional safety margin of $\Delta x/20$ should be sufficient to ensure the identification of the desired halo cells.

Appendix C. Characterization of the performances of the Curie supercomputer

Several material limitations can deteriorate the efficiency of MPI exchanges, among which the bandwidth, that is the amount of data that can be transferred from one processor to another during a time unit, and the network latency $\tau_{network}$ which is the time that a null size data packet takes to travel between the processors. The global latency $\tau_{latency}$ can thus be expressed as follows:

$$\tau_{latency}(\text{in s}) = \tau_{network}(\text{in s}) + \text{Bandwidth (in s/B)} \times \text{Message size (in B)}. \quad \text{(C.1)}$$

The term in the right-hand side of Eq. C.1 that is more likely to limit the global latency has to be foreseen in order to design the appropriate coding. To this end, a preliminary test is performed to measure the global latency as a function of the message size. Two processors interconnected by an InfiniBand QDR Full Fat Tree network of the Curie supercomputer from CEA in France performing blocking MPI communications ("ping pong" test [45]) were used

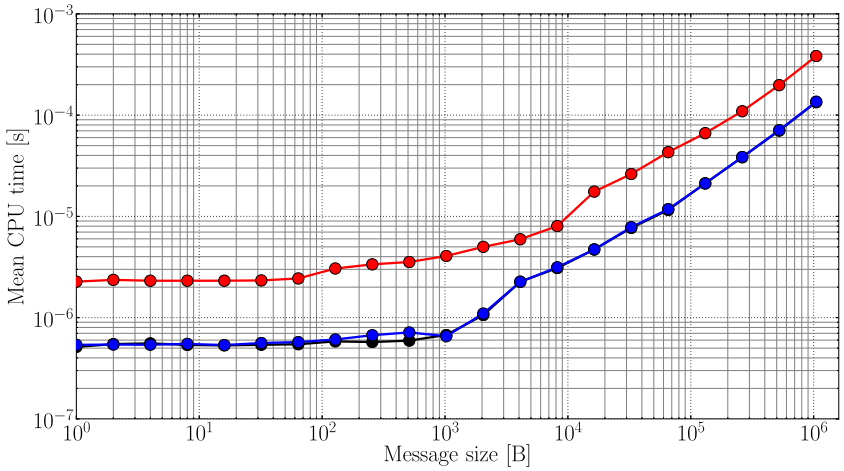

Fig. C.35. Measurement of the global latency of the Curie supercomputer using a "ping pong" test for three network configurations: ●—● : intra node intra socket, ●—● : intra node extra socket, ●—● : extra node.

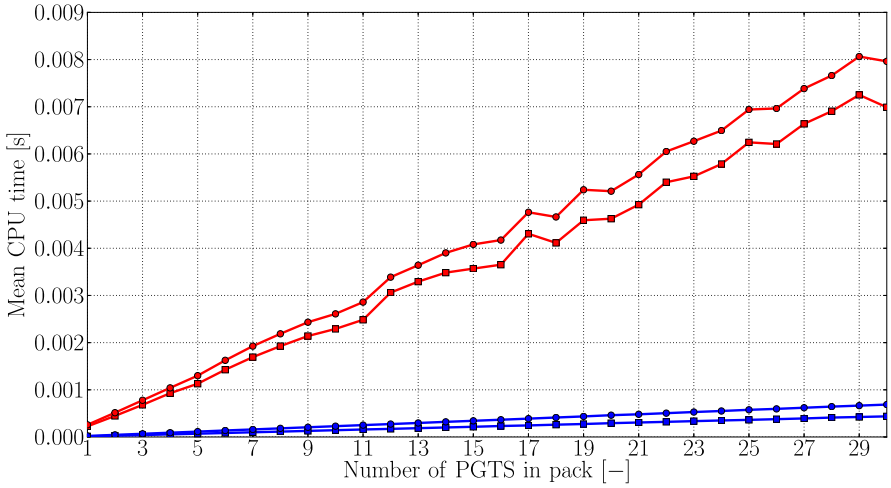

Fig. C.36. Computational cost of the packing (■—■) and unpacking (●—●) operations with preliminary particle data selection, and packing (■—■) and unpacking (●—●) operations without any selection, as a function of the number of PGTSs to pack/unpack.

for this test, of which results are shown in Fig. C.35. The mean CPU times shown were obtained from 10′000 sample tests for each case in order to hide the timer's own trigger time and other side effects, for instance when processors "heat up" their caches, allocate memory for internal buffers during the first communication, and to discard outliers due to kernel calls.

Two distinct behaviors can be identified in Fig. C.35, depending upon the message size. The fact that the exchange time remains stable for message sizes ranging from 0B to 1kB approximately indicates that the network latency imposes its limit in this zone. Its value can be estimated to $2\mu s$ for an extra node communication and $0.5\mu s$ for an intra node communication. For message sizes larger than 10kB all curves exhibit a linear behavior attributable to the bandwidth. Its value can be estimated to $0.4\mu s$/MB for an extra node communication and $0.15\mu s$/MB for an intra node communication. As a consequence it appears that depending on the quantity of exchanges, small messages may not always be the best option. Despite a similar tendency, each network configuration exposes different quality in terms of time efficiency, as shown by network latency and bandwidth estimated values. As expected, extra node communications are more costly and limit the global performances of the code as soon as several nodes are involved. As nodes are part of bigger structures (racks), it should be noted that performances could also vary when a larger number of processors are used.

The computational cost of the packing and the unpacking of PGTSs is investigated in Fig. C.36 and shows a linear dependency upon the number of PGTSs concerned. In the case where a preliminary particle data selection allows to pack the data necessary for the collision force computation only, instead of packing every particle data arrays, the total cost is reduced by a factor 14 approximately. Indeed, in a multiphysics context, miscellaneous data are stored on particles, hence the significance of such a step.

Appendix D. Use of barycentric coordinates

In order to determine whether a point P' belongs to a face F or not, the coordinates of P' are expressed in the face's barycentric coordinates. Let's assume F is a triangle defined by three points O, U and V and two vectors $\mathbf{u}$ and $\mathbf{v}$ as sketched in Fig. D.37. The coordinates of P' in F basis are noted $\mathbf{w}$ and are given by:

$$\mathbf{w} = h\mathbf{u} + k\mathbf{v}, \tag{D.1}$$

where h and $k \in \mathbb{R}$. If the following conditions are fulfilled, then P' is considered inside $\mathcal{VR}(F)$:

$$P' \in \mathcal{VR}(F) \text{ if: } \begin{cases} h \geq 0, \\ k \geq 0, \quad \text{and} \\ h + k \leq 1. \end{cases} \tag{D.2}$$

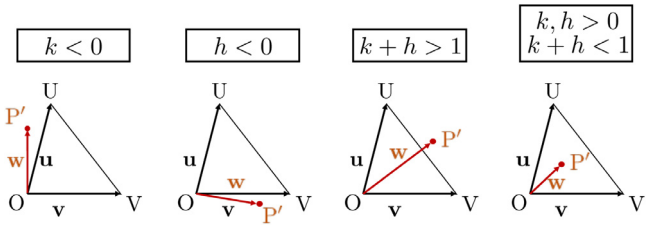

Fig. D.37. The coordinates of point P', $\mathbf{w}$, are expressed in the barycentric coordinates of the triangular face F composed of points O, U and V using $\mathbf{w} = h\mathbf{u} + k\mathbf{v}$. $\mathbf{u}$ and $\mathbf{v}$ are the vector of the basis, directed from O to U and from O to V, respectively. Four cases are represented. On the far right, the case where P' belongs to $\mathcal{VR}(F)$.

Taking the scalar product of Eq. D.1 by $\mathbf{v}$ then by $\mathbf{u}$ gives:

$$h(\mathbf{u}\cdot\mathbf{v}) + k(\mathbf{v}\cdot\mathbf{v}) = \mathbf{v}\cdot\mathbf{w} \tag{D.3}$$

and

$$h(\mathbf{u}\cdot\mathbf{u}) + k(\mathbf{u}\cdot\mathbf{v}) = \mathbf{u}\cdot\mathbf{w}. \tag{D.4}$$

Multiplying Eq. D.3 by $(\mathbf{u}\cdot\mathbf{u})$ and Eq. D.4 by $(\mathbf{v}\cdot\mathbf{u})$ and subtracting the results yields:

$$k\Big[(\mathbf{v}\cdot\mathbf{v})(\mathbf{u}\cdot\mathbf{u}) - (\mathbf{u}\cdot\mathbf{v})^2\Big] = \mathbf{w}\cdot[(\mathbf{u}\cdot\mathbf{u})\mathbf{v} - (\mathbf{v}\cdot\mathbf{u})\mathbf{u}]. \tag{D.5}$$

Then, noticing that:

$$det = (\mathbf{v}\cdot\mathbf{v})(\mathbf{u}\cdot\mathbf{u}) - (\mathbf{u}\cdot\mathbf{v})^2 \geq 0, \tag{D.6}$$

it finally follows that h and k can be calculated and that:

$$\mathrm{sgn(k)} = \mathrm{sgn}\{\mathbf{w}\cdot[(\mathbf{u}\cdot\mathbf{u})\mathbf{v} - (\mathbf{v}\cdot\mathbf{u})\mathbf{u}]\} \tag{D.7}$$

gives the sign of k, while the sign of h is given by:

$$\mathrm{sgn(h)} = \mathrm{sgn}\{\mathbf{w}\cdot[(\mathbf{v}\cdot\mathbf{v})\mathbf{u} - (\mathbf{v}\cdot\mathbf{u})\mathbf{v}]\}. \tag{D.8}$$

As $\mathbf{u}$, $\mathbf{v}$, $(\mathbf{u}\cdot\mathbf{u})$, $(\mathbf{v}\cdot\mathbf{v})$ and $(\mathbf{u}\cdot\mathbf{v})$ can be stored on the grid during the solver initialization, only $\mathbf{w}$, $(\mathbf{w}\cdot\mathbf{u})$, $(\mathbf{w}\cdot\mathbf{v})$ and det need to be computed during the run. To quicken the necessary operations, P' is first assumed to belong to $\mathcal{VR}(F)$, then each condition introduced in Eq. D.2 is consecutively checked and the test ends as soon as one is not met.

Appendix E. Exact method for building cell halo near boundaries

The implemented approach focuses on local exact wall distance calculation, allowing the flagging of more elements in refined area and fewer in coarse ones. The objective is to compute the distance between some interior mesh nodes and the wall features to deduce whether a mesh element has to be flagged or not, relying on the previous *null object detector*, the $\mathcal{VR}s$ introduced in Section 5.1 and the contact resolution presented in Section 5.2. Let $d_{\min} = r_p$ be the exact distance instruction defined by user, the followed steps are:

1. Flagging the nodes belonging to the wall boundaries (level 0 nodes) and each mesh element with at least a level 0 node (level 0 elements). Unflagged nodes of level 0 elements, the boundaries' immediate neighboring nodes, are also flagged (level 1 nodes).
2. Each level 1 node enters the null object detector to isolate its closest boundary faces.
3. Each level 1 node performs belonging tests on according $\mathcal{VR}s(F), \mathcal{VR}s(E)$ and $\mathcal{VR}s(V)$.
4. Each level 1 node operates the contact resolution algorithm to determine its exact distance d_{exact} to the boundary, however without computing forces. If $d_{exact} \leq d_{\min}$, the element is flagged (level 1 elements) as well as its remaining unflagged nodes (level 2 nodes).
5. Repeating from step 2 with node and element level incrementation until there is no more interior nodes closer than $d_{\min}$ to the wall.

References

[1] Kunii D, Levenspiel O. Fluidization engineering. Butterworth-Heinemann series in chemical engineering. 2nd Edition. Boston: Butterworth-Heinemann; 1991.
[2] van Swaaij W. Fluidization; 1985.
[3] Rüdisüli M, Schildhauer TJ, Biollaz SM, van Ommen JR. Scale-up of bubbling fluidized bed reactors-a review. Powder Technol 2012;217:pp.21–38. doi:10.1016/j.powtec.2011.10.004.
[4] van der Hoef M, Ye M, van Sint Annaland M, Andrews A, Sundaresan S, Kuipers J. Multiscale modeling of gas-fluidized beds. In: Adv. in Chem. Eng., 31. Elsevier; 2006. p. 65–149. doi:10.1016/S0065-2377(06)31002-2.
[5] Hammouti A, Euzenat F, Rakotonirina D, Wachs A. Direct numerical simulation of reactive flow through a fixed bed of catalyst particles. In: Proc. of Sixth Int. Conf. on Porous Media and Its Appl. in Sci., Eng. and Ind.. ECI Symposium Series; 2016.
[6] Gobin A, Neau H, Simonin O, Llinas J, Reiling V, Sélo J. Fluid dynamic numerical simulation of a gas phase polymerization reactor. Int J Numer MethodsFluids 2003;43:1199–220. doi:10.1002/fld.542.
[7] Alder BJ, Wainwright TE. Phase transition for a hard sphere system. J Chem Phys 1957;27(5):pp.1208–1209. doi:10.1063/1.1743957.
[8] Desjardins O, Blanquart G, Balarac G, Pitsch H. High order conservative finite difference scheme for variable density low Mach number turbulent flows. J Comput Phys 2008;227(15):pp.7125–7159. doi:10.1016/j.jcp.2008.03.027.
[9] Gopalakrishnan P, Tafti D. Development of parallel DEM for the open source code MFIX. Powder technol 2013;235:pp.33–41. doi:10.1016/j.powtec.2012.09.006.
[10] Goniva C, Kloss C, Hager A, Pirker S. An Open Source CFD-DEM Perspective; 2010. p. 1–10. Göteborg.
[11] Cole S, Bharadwaj R. The simulation of pneumatic transport within a pipe using a coupled DEM-CFD numerical method; 2007.
[12] Lin M, Canny J. A fast algorithm for incremental distance calculation. IEEE Comput. Soc. Press; 1991. p. 1008–14. doi:10.1109/ROBOT.1991.131723.
[13] Cohen JD, Lin MC, Manocha D, Ponamgi M. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. ACM Press; 1995. p. 189–ff.. doi:10.1145/199404.199437.
[14] Mirtich B. Polyhedral collision detection. Submitted to ACM Trans Graph 1997.
[15] Moureau V, Domingo P, Vervisch L. Design of a massively parallel CFD code for complex geometries. Comptes Rendus Mécanique 2011;339(2–3):pp.141–148. doi:10.1016/j.crme.2010.12.001.
[16] Chorin AJ. Numerical solution of the Navier-Stokes equations. Math Comput 1968;22(104):pp.745–745. doi:10.1090/S0025-5718-1968-0242392-2.
[17] Pierce CD, Moin P. Progress-variable approach for large-eddy simulation of non-premixed turbulent combustion. J Fluid Mech 2004;504:pp.73–97. doi:10.1017/S0022112004008213.
[18] Du W, Bao X, Xu J, Wei W. Comput. fluid dynamics (CFD) modeling of spouted bed: assessment of drag coefficient correlations. Chem Eng Sci 2006;61(5):pp.1401–1420. doi:10.1016/j.ces.2005.08.013.
[19] Dziugys A, Peters B. An approach to simulate the motion of spherical and non-spherical fuel particles in combustion chambers. Granul matter 2001;3(4):pp.231–266. doi:10.1007/PL00010918.
[20] Fede P, Simonin O, Ingram A. 3D numerical simulation of a lab-scale pressurized dense fluidized bed focussing on the effect of the particle-particle restitution coefficient and particle-wall boundary conditions. Chem Eng Sci 2016;142:215–35. doi:10.1016/j.ces.2015.11.016.
[21] Capecelatro J, Desjardins O. An Euler–Lagrange strategy for simulating particle-laden flows. J Comput Phys 2013;238:pp.1–31. doi:10.1016/j.jcp.2012.12.015.
[22] Cundall PA, Strack ODL. A discrete numerical model for granular assemblies. Géotech 1979;29(1):pp.47–65. doi:10.1680/geot.1979.29.1.47.
[23] Ergun S. Fluid flow through packed columns. J Chem Eng Prog 1952;48(2):pp.89–94. doi:10.1021/ie50474a011.
[24] Wen CY, Yu YH. A generalized method for predicting the minimum fluidization velocity. AIChE J 1966;12(3):pp.610–612. doi:10.1002/aic.690120343.
[25] Dennis SCR, Ingham DB, Singh SN. The steady flow of a viscous fluid due to a rotating sphere. Q J Mech App Math 1981;34(3):pp.361–381. doi:10.1093/qjmam/34.3.361.
[26] Wachs A, Girolami L, Vinay G, Ferrer G. Grains3d, a flexible DEM approach for particles of arbitrary convex shape-part i: numerical model and validations. Powder Technol 2012;224:374–89. doi:10.1016/j.powtec.2012.03.023.
[27] Komiwes V, Mege P, Meimon Y, Herrmann H. Simulation of granular flow in a fluid applied to sedimentation. Granul Matter 2006;8(1):pp.41–54. doi:10.1007/s10035-005-0220-3.
[28] Malandain M, Maheu N, Moureau V. Optimization of the deflated conjugate gradient algorithm for the solving of elliptic equations on massively parallel machines. J Comput Phys 2013;238:32–47. doi:10.1016/j.jcp.2012.11.046.
[29] Jajcevic D, Siegmann E, Radeke C, Khinast JG. Large-scale CFDDEM simulations of fluidized granular systems. Chem Eng Sci 2013;98:pp.298–310. doi:10.1016/j.ces.2013.05.014.
[30] Sethian JA. A fast marching level set method for monotonically advancing fronts. Proc Natl Acad Sci 1996;93(4):pp.1591–1595. doi:10.1073/pnas.93.4.1591.
[31] Tsitsiklis J. Efficient algorithms for globally optimal trajectories. IEEE Trans Autom Control 1995;40(9):pp.1528–1538. doi:10.1109/9.412624.
[32] Kodam M, Bharadwaj R, Curtis J, Hancock B, Wassgren C. Force model considerations for glued-sphere discrete element method simulations. Chem Eng Sci 2009;64(15):pp.3466–3475. doi:10.1016/j.ces.2009.04.025.
[33] Ren B, Zhong W, Chen Y, Chen X, Jin B, Yuan Z, et al. CFD-DEM simulation of

spouting of corn-shaped particles. Particuology 2012;10(5):pp.562–572. doi:10.1016/j.partic.2012.03.011.
[34] Norouzi HR, Zarghami R, Sotudeh-Gharebagh R, Mostoufi N. Coupled CFD-DEM modeling: formulation, implementation and application to multiphase flows. first edition. Chichester, West Sussex: Wiley; 2016.
[35] Bell N, Yu Y, Mucha PJ. Particle-based simulation of granular materials. ACM Press; 2005. p. 77. doi:10.1145/1073368.1073379.
[36] Williams J, O'Connor R. A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries. Eng Comput 1995;12(2):pp.185–201. doi:10.1108/02644409510799550.
[37] Gilbert E, Johnson D, Keerthi S. A fast procedure for computing the distance between complex objects in three-dimensional space. IEEE J Robot Autom 1988;4(2):pp.193–203. doi:10.1109/56.2083.
[38] Kremmer M, Favier JF. A method for representing boundaries in discrete element modelling-part i: geometry and contact detection. Int J Numer Methods Eng 2001;51(12):pp.1407–1421. doi:10.1002/nme.184.
[39] Chittawadigi RG, Saha SK. An analytical method to detect collision between cylinders using dual number algebra. In: 2013 IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS). IEEE; 2013. p. 5353–8. doi:10.1109/IROS.2013.6697131.
[40] Karypis G, Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J Sci Comput 1998;20(1):pp.359–392. doi:10.1137/S1064827595287997.
[41] Hubbard P. Collision detection for interactive graphics applications. IEEE Trans Vis Comput Graph 1995;1(3):218–30. doi:10.1109/2945.466717.
[42] Beverloo W, Leniger H, van de Velde J. The flow of granular solids through orifices. Chem Eng Sci 1961;15(3–4):260–9. doi:10.1016/0009-2509(61)85030-6.
[43] Mills AA, Day S, Parkes S. Mech. of the sandglass. Eur J Phys 1996;17(3):97. doi:10.1088/0143-0807/17/3/001.
[44] Sun R, Xiao H. Diffusion-based coarse graining in hybrid continuum–discrete solvers: applications in CFD-DEM. Int J of Multiph Flow 2015;72:pp.233–247. doi:10.1016/j.ijmultiphaseflow.2015.02.014.
[45] Snell QO, Mikler AR, Gustafson JL. Netpipe: a network protocol independent performance evaluator. In: IASTED international conference on intelligent information management and systems, 6. Washington, DC, (USA); 1996. p. 49.