This document is the accepted manuscript version of the following article:
Raphael Egan, Arthur Guittet, Fernando Temprano-Coleto, Tobin Isaac, François J. Peaudecerf, Julien R. Landel, Paolo Luzzatto-Fegiz, Carsten Burstedde, Frederic Gibou, Direct numerical simulation of incompressible flows on parallel Octree grids. *Journal of Computational Physics*, **428**:110084 (2021)
https://doi.org/10.1016/j.jcp.2020.110084.

Originally uploaded to https://n.ethz.ch/~fpeaudec/ 08.01.2021

Direct Numerical Simulation of Incompressible Flows on Parallel Octree Grids

Raphael Egan[a], Arthur Guittet[a], Fernando Temprano-Coleto[a], Tobin Isaac[c], François J. Peaudecerf[e], Julien R. Landel[f], Paolo Luzzatto-Fegiz[a], Carsten Burstedde[d], Frederic Gibou[a,b]

[a]*Department of Mechanical Engineering, University of California, Santa Barbara, CA 93106-5070*
[b]*Department of Computer Science, University of California, Santa Barbara, CA 93106-5110*
[c]*School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0765*
[d]*Institute for Numerical Simulation and Hausdorff Center for Mathematics, University of Bonn, Germany*
[e]*Department of Civil, Environmental, and Geomatic Engineering, ETH Z urich, 8093 Zürich, Switzerland*
[f]*Department of Mathematics, Alan Turing Building, University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom*

Abstract

We introduce an approach for solving the incompressible Navier-Stokes equations on a forest of Octree grids in a parallel environment. The methodology uses the `p4est` library of Burstedde *et al.*, *SIAM J. Sci. Comput.*, 33(3) (2011) [15] for the construction and the handling of forests of Octree meshes on massively parallel distributed machines and the framework of Mirzadeh *et al.*, *J. Comput. Phys.*, 322 (2016) [55] for the discretizations on Octree data structures. We introduce relevant additional parallel algorithms and provide performance analyses for individual building bricks and for the full solver. We demonstrate strong scaling for the solver up to 32,768 cores for a problem involving $\mathcal{O}\left(6.1 \times 10^8\right)$ computational cells. We illustrate the dynamic adaptive capabilities of our approach by simulating flows past a stationary sphere, flows due to an oscillatory sphere in a closed box and transport of a passive scalar. Without sacrificing accuracy nor spatial resolution in regions of interest, our approach successfully reduces the number of computational cells to (at most) a few percents of uniform grids with equivalent resolution. We also perform a numerical simulation of the turbulent flow in a superhydrophobic channel with unparalleled wall grid resolution in the streamwise and spanwise directions.

Keywords: Level-Set, Quad-/Oc-trees, Parallel Adaptive Mesh Refinement, Navier-Stokes, Incompressible, Voronoi tessellation

1. Introduction

In the last decade, the democratization of the access to supercomputers has prompted the development of massively parallel simulation techniques. The previously existing serial codes are progressively being adapted to exploit the hundreds of thousands of cores available through the main computing clusters. We propose a parallel implementation of the solver for the incompressible Navier-Stokes equations introduced in [30], based on the parallel level-set framework presented in [55]. Additional novel algorithms, necessary to solving the Navier-Stokes equations in a forest of Quad-/Oc-trees, are presented.

Numerical simulations at the continuum scale are generally divided into two categories characterized by their meshing techniques. On the one hand, the finite elements community relies on body-fitted unstructured meshes to represent irregular domains. Given a high quality mesh, the resulting solvers are fast and very accurate. This approach has been successfully applied to the simulation of incompressible viscous flows [26, 70, 28]. However, the mesh generation is very costly and impractical when tracking moving interfaces and fluid features requiring high spatial resolution.

*Corresponding author: raphaelegan@ucsb.edu

On the other hand, methods based on structured Cartesian grids render the mesh geometry mainly trivial, but lead to a higher complexity for the implicit representation of irregular interfaces. We focus here on the latter class of methods.

A common approach to represent an irregular interface in a implicit framework is to use Peskin's immersed boundary method [63, 43, 25] or its level-set counterpart [75]. However, these methods introduce a smoothing of the interface through a delta formulation and therefore restrict the accuracy of the solution with $O(1)$ errors near fluid-fluid interfaces[1]. We therefore opt for the sharp interface representation provided by the level-set function [60]. We use the finite-volume/cut-cell approach of Ng *et al.* [59] to impose the boundary condition at the solid-fluid interface for its demonstrated convergence in the L^∞-norm.

Fluid flows are by nature multiscale, thus limiting the scope of uniform Cartesian grids. A range of strategies have been proposed to leverage the spatial locality of the fluid information such as stretched grids [79, 2], nested grids [11, 32, 74, 10, 71], chimera grids [9, 24] or unstructured meshes [22, 38, 49, 50, 51]. Another approach is to use a Quadtree [27] (in two spatial dimensions) or Octree [48] (in three spatial dimensions) data structure to store the mesh information [19, 40]. Popinet applied this idea combined with a non-compact finite volume discretization on the Marker-And-Cell (MAC) configuration [31] to the simulation of incompressible fluid flows [65]. Losasso *et al.* also proposed a compact finite volume solver on Octree for inviscid free surface flows [46], while Min *et al.* presented a node-based second-order accurate viscous solver [52]. The present work is based on the approach presented in Guittet *et al.* [30], which solves the viscous Navier-Stokes equations implicitly on the MAC configuration using a Voronoi partition and where the advection part of the momentum equation is discretized along the characteristic curves with a Backward Differentiation Formula (BDF), semi-Lagrangian scheme [73, 81]. The projection step is solved with the second-order discretization of Losasso *et al.* [45] for the Poisson equation.

The extension of [30] to parallel architectures relies on the existence of an efficient parallel Quad-/Oc-tree structure. Possible ways to implement parallel tree structures include the replication of the entire grid on each process. This approach, however, is not feasible when the grid size exceeds the memory of a single compute node, which must be considered a common scenario nowadays. Using graph partitioners such as `parMETIS` [39] on a tree structure would discard the mathematical relations between neighbor and child elements that are implicit in the tree, and thus result in additional overhead. Another option, which we find preferable, is to exploit the tree's logical structure using space-filling curves [1]. This approach has been shown to lead to load balanced configurations with good information locality for a selection of space-filling curves including the Morton (or Z-ordering) curve and the Hilbert curve [16].

Space-filling curves have been used in several ways, for example augmented by hashing [29], tailored to PDE solvers [13], or focusing on optimized traversals [80]. `Octor` [78] and `Dendro` [68] are two examples of parallel Octree libraries making use of this strategy that have been scaled to 62,000 [14] and 32,000 [69] cores, operating on parent-child pointers and a linearized octant storage, respectively. Extending the linearized storage strategy to a forest of interconnected Octrees [72, 8], the `p4est` library [15] provides a publicly available implementation of the parallel algorithms required to handle the parallel mesh, including an efficient 2:1 balancing algorithm [34]. `p4est` has been shown to scale up to over 458,000 cores [35], with applications using it successfully on 1.57M cores [67] and 3.14M cores [58].

The algorithms pertaining to the second-order accurate level-set method on Quad-/Oc-tree presented in Min and Gibou [54] have been extended in Mirzadeh *et al.* [55] parallel architecture by leveraging the `p4est` library. Starting from this existing basis for the level-set function procedures, we present the implementation of the algorithms pertaining to the simulation of incompressible fluid flows detailed in [30]. The Voronoi tessellation that we construct over the adaptive tree mesh requires (at least) two layers of ghost cells, whose efficient parallel construction we describe in detail. We report on the scalability of the algorithms presented before illustrating the full capabilities of

[1] Although we do not consider fluid-fluid interfaces, the present work is intended as a stepping stone toward that case.

the resulting solver.

2. The computational method

In this section, we present mathematical and computational components pertaining to our numerical method for solving the incompressible Navier-Stokes equations on a forest of Octree grids. The first five subsections are mainly dedicated to the mathematical description of the discretization procedures (the interested reader may find more details in [30]). The implementation of these building bricks in a distributed computing framework reveals two grid-related requirements: access to second-degree (or third-degree) cell neighbors and unambiguous indexing of grid faces. The last two subsections present the computational strategies developed to address challenges related to these requirements. Throughout this section, schematics and illustrations are presented in two dimensions for the sake of clarity. Their extension to three dimensions follows the exact same principles without any loss of generality.

2.1. Representation of the spatial information

2.1.1. The level-set method

A central desired feature of the proposed solver is to be able to handle complex, possibly moving interfaces in a sharp fashion[2]. The level-set framework, first introduced by [60] and extended to Quad-/Oc-trees in [54] is a highly suited tool for such a goal. The level-set representation of an arbitrary contour Γ, separating a domain Ω into two subdomains Ω^- and Ω^+, is achieved by defining a function ϕ, called the level-set function, such that $\Gamma = \{\boldsymbol{x} \in \mathbb{R}^n | \phi(\boldsymbol{x}) = 0\}$, $\Omega^- = \{\boldsymbol{x} \in \mathbb{R}^n | \phi(\boldsymbol{x}) < 0\}$ and $\Omega^+ = \{\boldsymbol{x} \in \mathbb{R}^n | \phi(\boldsymbol{x}) > 0\}$.

Among all the possible candidates that satisfy these criteria, a signed distance function (i.e., $|\nabla\phi| = 1$) is the most convenient one. In order to transform any function $\varphi(\boldsymbol{x})$ into a signed distance function $\phi(\boldsymbol{x})$ that shares the same zero contour, one can solve the reinitialization problem

$$\frac{\partial \phi}{\partial \tau} + \mathrm{sign}(\varphi)\left(|\nabla\phi| - 1\right) = 0, \quad \phi(\boldsymbol{x})|_{\tau=0} = \varphi(\boldsymbol{x})$$

until a steady state in the fictitious time τ is found. The finite difference discretization and its corresponding parallel implementation employed to solve this equation are presented respectively in [54] and [55].

2.1.2. Forests of Quad-/Oc-trees and the `p4est` library

When dealing with physical problems that exhibit a wide range of length scales, uniform Cartesian meshes become impractical since capturing the smallest length scales requires a very high resolution. This is the case for high Reynolds number flows, for which the boundary layers and any wake vortices have a length scale significantly smaller than that of the far-field flow. This observation naturally leads to the use of adaptive Cartesian grids, including Octrees grids.

The `p4est` library [15] is a collection of parallel algorithms that handles a linearized tree data structure and its manipulation methods, which were shown to collectively scale up to 458,752 cores [35], as noted in the previous section. In `p4est` the domain is first divided by a coarse grid, which we will refer to as the "macromesh", common to all the processes. For our purpose we will consider solely uniform Cartesian macromeshes in a brick layout, although a general macromesh is not limited to such a configuration in `p4est`. This layout can be constructed at no cost using predefined and self-contained functions. A collection of trees rooted in each cell of the macromesh is then constructed and partitioned, and their associated (expanded) ghost layers are generated. The refinement and coarsening criteria necessary for the construction of the trees are provided to `p4est` by defining callback functions. We propose to use four criteria based on the physical characteristics at hand. Different combinations of these criteria are used depending on the specific problem considered.

[2] We consider irregular solid objects in this work; multiphase interfaces are left for future work [23].

The first criterion, presented in [54] and [55], captures the location of the interface: coarse cells are allowed locally, provided they are (at least) K cell diagonal(s) away from the interface, where $K \geq 1$ is defined by the user. Specifically, a cell $\mathcal{C}$ is marked for refinement if

$$\min_{v \in V(\mathcal{C})} |\phi(v)| \leq K \, \mathrm{Lip}(\phi) \mathrm{diag}(\mathcal{C}), \tag{1}$$

where $V(\mathcal{C})$ is the set of all the vertices of cell $\mathcal{C}$, $\mathrm{Lip}(\phi)$ is the Lipschitz constant of the level-set function ϕ, and $\mathrm{diag}(\mathcal{C})$ is the length of the diagonal of cell $\mathcal{C}$. Similarly, a cell is marked for coarsening if

$$\min_{v \in V(\mathcal{C})} |\phi(v)| > 2K \, \mathrm{Lip}(\phi) \mathrm{diag}(\mathcal{C}).$$

The second criterion, introduced for Quad-/Oc-trees in [65] and used in [52] and [30], is based on the vorticity of the fluid. High vorticity corresponds to small length scales and therefore necessitates a high mesh resolution. We mark a cell $\mathcal{C}$ for refinement if

$$h_{\max} \frac{\max_{v \in V(\mathcal{C})} \|\nabla \times \boldsymbol{u}(v)\|_2}{\max_{\Omega} \|\boldsymbol{u}\|_2} \geq \gamma, \tag{2}$$

where $h_{\max}$ is the largest edge length of cell $\mathcal{C}$ and γ is a parameter controlling the level of refinement. Analogously, a cell $\mathcal{C}$ is marked for coarsening if

$$2\, h_{\max} \frac{\max_{v \in V(\mathcal{C})} \|\nabla \times \boldsymbol{u}(v)\|_2}{\max_{\Omega} \|\boldsymbol{u}\|_2} < \gamma.$$

Another criterion enforces a band of b grid cells of highest desired resolution around the irregular interface. We mark for refinement every cell such that

$$\min_{v \in V(\mathcal{C})} \mathrm{dist}\,(v, \Gamma) < b \, \max\left(\Delta x_{\text{finest}}, \Delta y_{\text{finest}}, \Delta z_{\text{finest}}\right),$$

where Δx_{finest}, Δy_{finest} and Δz_{finest} are the cell sizes along cartesian directions for the finest cells to be found in the domain.

Finally, the solver was augmented with the optional capability of advecting a passive scalar for visualization purposes (this is illustrated in section 4.4). For enhanced graphical results, we propose to refine the mesh where the density of the marker exceeds a threshold. Given a density $\beta \in [0, 1]$ for the advected passive scalar, a cell $\mathcal{C}$ is marked for refinement (resp. coarsening) if

$$\max_{v \in V(\mathcal{C})} \beta(v) \geq \chi, \qquad (\text{resp. } \max_{v \in V(\mathcal{C})} \beta(v) < \chi), \tag{3}$$

where χ is a parameter controlling the level of refinement.

2.1.3. The Marker-And-Cell layout

The standard data layout used to simulate incompressible viscous flows on uniform grids is the Marker-And-Cell (MAC)[31] layout. The analogous layout for Quadtrees is presented in figure 1 and leads to complications in the discretizations compared to uniform grids. However, second order accuracy is achievable for the elliptic and advection-diffusion problems that appear in our discretization of the Navier-Stokes equations. Two possible corresponding discretizations are presented for the data located at the center of the cells (the leaves of the trees) and at their faces in sections 2.4 and 2.3.2 respectively.

2.2. The projection method

Consider the incompressible Navier-Stokes equations for a fluid with velocity $\boldsymbol{u}$, pressure p, density ρ and dynamic viscosity μ, with a force per unit mass $\boldsymbol{f}$

$$\rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) = -\nabla p + \rho \boldsymbol{f} + \mu \nabla^2 \boldsymbol{u}, \tag{4}$$

$$\nabla \cdot \boldsymbol{u} = 0. \tag{5}$$

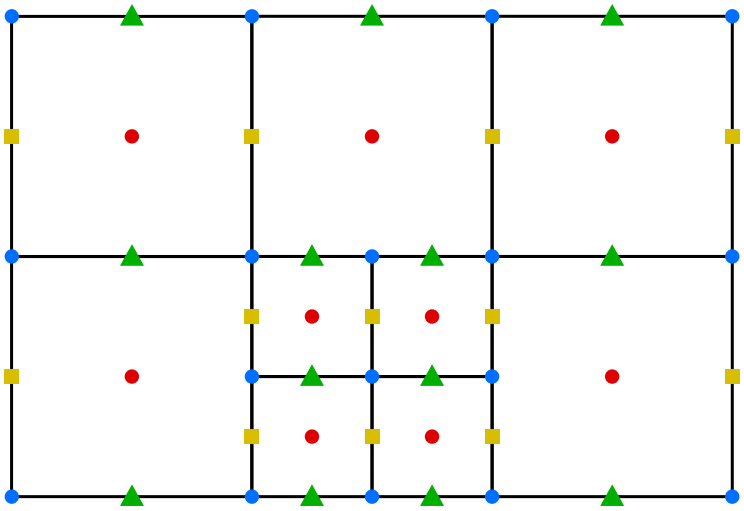

Figure 1: Representation of the Marker-And-Cell (MAC) data layout on a Quadtree structure with the location of the x-velocity (■), the y-velocity (▲), the Hodge variable (●) and the level-set values (●).

The standard approach to solve this system is the projection method introduced by Chorin [18]. We refer the reader to [12] for a review of the variations of the projection method. The system is decomposed into two distinct steps, identified as the viscosity step and the projection step. The first step consists in solving the momentum equation (4) without the pressure term,

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = \rho\boldsymbol{f} + \mu\nabla^2\boldsymbol{u}, \tag{6}$$

to find an intermediate velocity field $\boldsymbol{u}^*$. Since this field does not satisfy the incompressibility condition (5), it is then projected on the divergence-free subspace to obtain $\boldsymbol{u}^{n+1}$, the solution at time t^{n+1}, via

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \nabla\Phi \tag{7}$$

where Φ is referred to as the Hodge variable and satisfies

$$\nabla^2\Phi = \nabla\cdot\boldsymbol{u}^*. \tag{8}$$

The two following sections describe the discretization applied to solve steps (6) and (8) respectively.

2.3. Implicit discretization of the viscosity step

The viscosity step (6) contains two distinct terms besides the possible bulk force: the advection term on the left-hand side and the viscous term on the right-hand side. In order to prevent stringent time step restrictions due to the latter, we opt for a second order backward differentiation method to advance (6) in time. This integration scheme can address stiff problems without theoretical stability-related constraints on the time step.

2.3.1. Discretization of the advection term with a semi-Lagrangian approach

We discretize the advection part of the viscosity step using a semi-Lagrangian approach [21, 73]. This method relies on the fact that the solution $\boldsymbol{u}(\boldsymbol{x}, t)$ of the advection equation

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = 0 \tag{9}$$

is constant along the characteristics of the equation, i.e., along material trajectories $(\boldsymbol{x}(s),\ t(s))$ such that $\dfrac{\mathrm{d}t}{\mathrm{d}s} = 1$ and $\dfrac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}s} = \boldsymbol{u}\left(\boldsymbol{x}, t\right)$. Using this parameterization, equation (9) is equivalent to

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}s} = 0,$$

which we integrate with respect to s using a second-order BDF [81].

Given the location $\boldsymbol{x}^*$ where the solution $\boldsymbol{u}^*$ is sought at time t_{n+1}, the local material trajectory passing through $\boldsymbol{x}^*$ at time t^{n+1} is traced back in time to find the points $\boldsymbol{x}_d^n$ and $\boldsymbol{x}_d^{n-1}$ through which it passed at times t^n and t^{n-1} respectively. The values $\boldsymbol{u}_d^n = \boldsymbol{u}\left(\boldsymbol{x}_d^n, t_n\right)$ and $\boldsymbol{u}_d^{n-1} = \boldsymbol{u}\left(\boldsymbol{x}_d^{n-1}, t_{n-1}\right)$ are then calculated using quadratic interpolation and the application of the second order BDF (note that $\Delta s = \Delta t$) leads to

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} \approx \frac{\alpha}{\Delta t_n}\boldsymbol{u}^* + \left(\frac{\beta}{\Delta t_{n-1}} - \frac{\alpha}{\Delta t_n}\right)\boldsymbol{u}_d^n - \frac{\beta}{\Delta t_{n-1}}\boldsymbol{u}_d^{n-1},$$

where

$$\Delta t_n = t_{n+1} - t_n, \qquad \Delta t_{n-1} = t_n - t_{n-1}, \qquad \alpha = \frac{2\Delta t_n + \Delta t_{n-1}}{\Delta t_n + \Delta t_{n-1}} \qquad \text{and} \qquad \beta = -\frac{\Delta t_n}{\Delta t_n + \Delta t_{n-1}}.$$

We refer the reader to [30] for further details.

2.3.2. Discretization of face-centered Laplace operators

Since we consider constant-viscosity incompressible flows of Newtonian fluids, the velocity components are effectively decoupled in the viscous terms. This allows us to solve for the individual components of $\boldsymbol{u}^*$ separately when advancing (6). In that context, we require appropriate discretizations for the Laplace operators associated with degrees of freedom sampled at faces of similar orientations, i.e., at faces where similar velocity components are sampled. We obtain these discretized operators by applying a finite volume approach to Voronoi tessellations. We present a summary of the approach and refer the reader to [30] for further details.

Given a set of points in space, called seeds, we define the Voronoi cell of a seed as the region of space that is closer to that seed than to any other seed. The union of all the Voronoi cells forms a tessellation of the domain, i.e., a non-overlapping gap-free tiling of the domain. By placing these seeds at the centers of faces of the computational mesh sharing the same cartesian orientation, one obtains a new computational grid for the corresponding velocity component that is sampled at those faces. ~~A~~ Two-dimensional examples of Voronoi tessellation ~~for the velocity component $u \equiv \boldsymbol{e}_x \cdot \boldsymbol{u}$ is~~ are presented in figure 2 for ~~a~~ Quadtree grids.

Considering a diffusion equation for the unknown u with constant diffusion coefficient μ

$$\mu\nabla^2 u = r,$$

it is discretized on the Voronoi tessellation with a finite volume approach where the control volume for each degree of freedom i is its Voronoi cell $\mathcal{C}_i$. This leads to

$$\int_{\mathcal{C}_i} \mu\nabla^2 u = \int_{\partial\mathcal{C}_i} \mu\ \nabla u\cdot\boldsymbol{n} \approx \sum_{j\in\mathrm{ngbd}(i)} \mu\ s_{ij}\frac{u_j - u_i}{d_{ij}},$$

where $\mathrm{ngbd}(i)$ is the set of neighbors for the degree of freedom i, $\boldsymbol{n}$ is the vector normal to $\partial\mathcal{C}_i$ (pointing outwards), d_{ij} is the length between degrees of freedom i and j, and s_{ij} is the area —or the length, in 2D— of the face between them, as illustrated in figure 2. This discretization provides a second-order accurate solution [56].

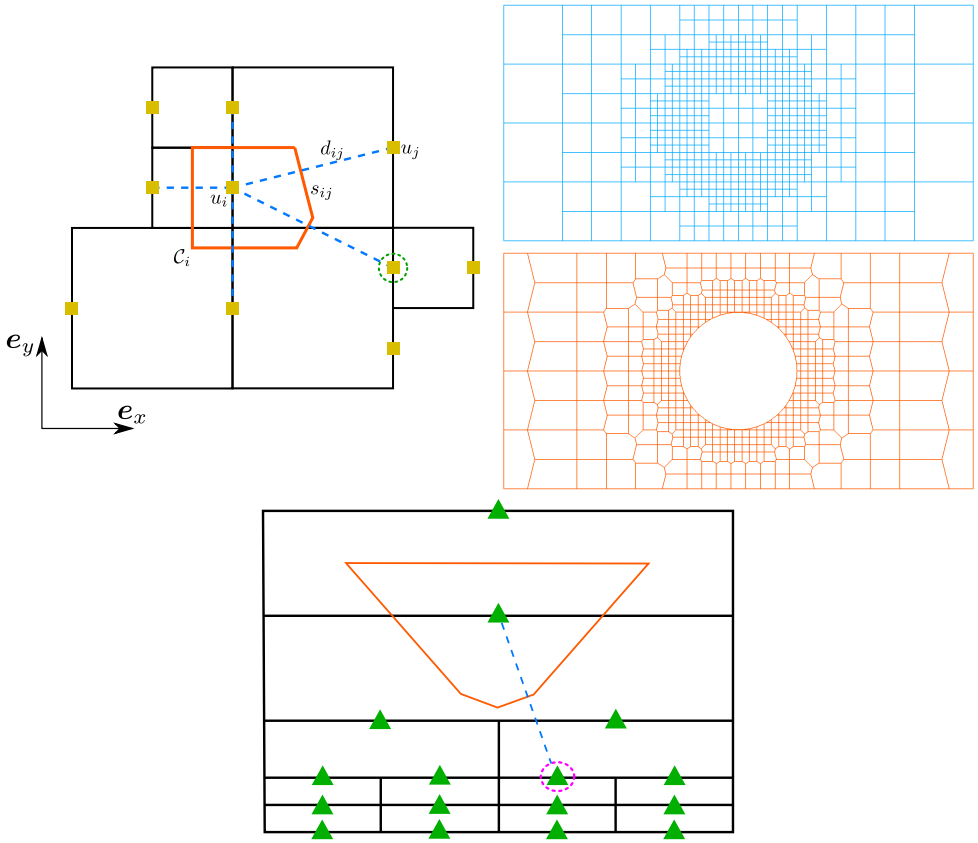

Figure 2: Top left: nomenclature for the discretization of the Laplace operator on a Voronoi diagram (illustrated for a vertical face). The degree of freedom circled in green can potentially belong to a second-degree neighbor cell, i.e., a cell ~~with data needed in the discretization~~ that is not adjacent to the current cell but adjacent to one of the current cell's immediate neighbor. Top right: example of a Quadtree mesh (top) and its Voronoi tessellation for the vertical faces (bottom). Bottom: illustration of a two-dimensional Voronoi cell for a horizontal face which may require knowledge of a face associated with a third-degree neighbor cell, in case of stretched computational grids (aspect ratio much different from 1). The face circled in pink is indexed by a third-degree neighbor quadrant of the top quadrant indexing the center seed.

2.3.3. General discretization for the viscosity step

Combining the discretizations presented in the two previous sections, we obtain the general discretization formula. Considering the x-component of the velocity field u, for the ith face with normal $\boldsymbol{e}_x$ and associated Voronoi cell $\mathcal{C}_i$, we have

$$\mathrm{Vol}(\mathcal{C}_i)\rho\frac{\alpha}{\Delta t_n}u_i^* + \mu\sum_{j\in\mathrm{ngbd}(i)} s_{ij}\frac{u_i^*-u_j^*}{d_{ij}} = \mathrm{Vol}(\mathcal{C}_i)\rho\left[\left(\frac{\alpha}{\Delta t_n}-\frac{\beta}{\Delta t_{n-1}}\right)u_{i,d}^n + \frac{\beta}{\Delta t_{n-1}}u_{i,d}^{n-1} + \boldsymbol{e}_x\cdot\boldsymbol{f}_i\right],$$

where $\mathrm{Vol}(\mathcal{C}_i)$ is the volume of $\mathcal{C}_i$. The very same approach is then used for the y- and z-components of the velocity, i.e., v and w.

This produces a symmetric positive definite linear system that we solve using the BiConjugate Gradient stabilized iterative solver and the successive over-relaxation preconditioner provided by the PETSc library [5, 4, 6].

A note on the boundary conditions. The boundary conditions to consider when solving the viscosity step are to be imposed on $\boldsymbol{u}^*$. As a consequence, the type of boundary condition that is desired for

$\boldsymbol{u}$ is used for $\boldsymbol{u}^*$ as well, but the enforced boundary value is corrected in order to take into account the correction from the projection step (7), given the current best estimate of $\nabla\Phi$.

2.4. A stable projection

The projection step consists in solving the Poisson equation (8) with the data located at the center of the leaves of the tree. Stability and accuracy constraints result in the discretization presented in [45]. The method relies on a finite volume approach with a leaf being the control volume for the degree of freedom located at its center. Using the notations defined in figure 3, we now explain the discretization of the flux of the Hodge variable Φ on the right face of $\mathcal{C}_2$. For the sake of clarity, we assume that all other neighbor cells of $\mathcal{C}_2$ in Cartesian directions are of the same size as $\mathcal{C}_2$; if not, the reasoning presented here below needs to be applied for all variables sampled on faces shared between cells of different sizes.

The first step is to define the weighted average distance Δ between Φ_0 and its neighboring small leaves on the left side,

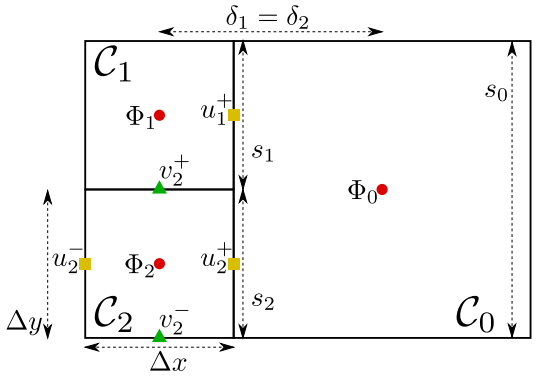

Figure 3: Nomenclature for the discretization of the flux of the Hodge variable at cell faces and the discretization of the divergence of the velocity field.

$$\Delta = \sum_{i\in\mathcal{N}} \frac{s_i}{s_0}\delta_i,$$

where $\mathcal{N}$ is the set of leaves whose right neighbor leaf is $\mathcal{C}_0$. We then define the partial derivative of Φ with respect to x on the right face of $\mathcal{C}_2$ as

$$\frac{\partial\Phi}{\partial x} = \sum_{i\in\mathcal{N}} \frac{s_i}{s_0}\frac{\Phi_0 - \Phi_i}{\Delta}.$$

This discretization collapses to the standard central finite difference discretization in case of (locally) uniform grids. The other components of $\nabla\Phi$ are defined analogously and stored at the corresponding faces. We then define the divergence of $\boldsymbol{u}$ at the center of the leaf containing Φ_2 as

$$\nabla\cdot\boldsymbol{u} = \frac{1}{\Delta x}\left(\sum_{i\in\mathcal{N}} \frac{s_i}{s_0}u_i^+ - u_2^-\right) + \frac{1}{\Delta y}\left(v_2^+ - v_2^-\right).$$

Both the divergence and the gradient operators involve all small leaves having $\mathcal{C}_0$ as a right neighbor. The cell-centered Laplace operator in eq. (8) is obtained by chaining the above divergence and gradient operator, i.e., $\nabla^2\Phi = \nabla\cdot(\nabla\Phi)$. This produces a second-order accurate discretization for cell-centered Poisson equations. The correspondence between the two operators defined here above ensures that the gradient is the negative adjoint of the divergence in a well-defined face-weighted norm, ensuring the stability of the projection step [30]. The linear system resulting from this approach is symmetric positive definite, and it is solved using a (possibly preconditioned) conjugate gradient method.

2.5. Typical flowchart of the solver

As detailed in subsection 2.3, boundary conditions to be enforced on the intermediate velocity field $\boldsymbol{u}^*$ require the knowledge of $\nabla\Phi$. Yet, Φ itself is defined as the solution of an elliptic problem that requires $\nabla \cdot \boldsymbol{u}^*$ (see (8)).

In order to best enforce the desired boundary conditions on $\boldsymbol{u}$, the solver addresses this circular dependency between $\boldsymbol{u}^*$ and $\nabla\Phi$ through a fixed-point iteration. The solver determines a sequence $\boldsymbol{u}^{*,k}$ and Φ^k, with $k \geq 1$: the intermediate velocity field $\boldsymbol{u}^{*,k}$ is the solution of the viscosity step (6) with boundary condition values defined using the known field Φ^{k-1} (see subsection 2.3 - note that Φ^0 is defined as the scalar field Φ obtained at the end of the previous time step, or as 0 for the very first time step). The scalar field Φ^k is determined in turn as the solution of the projection equation (8) using $\nabla \cdot \boldsymbol{u}^{*,k}$ as the right-hand side. Figure 4 illustrates this iterative procedure.

This process is repeated for increasing k until convergence is reached or until a user-defined maximum number of iterations $k_{\max}$ is reached. Note that the standard, approximate projection method corresponds to $k_{\max} = 1$ (the computational cost and the relevance of additional inner-loop iterations are estimated and discussed for some relevant applications within section 4). ~~The convergence criterion we use is $\left\|\Phi^k - \Phi^{k-1}\right\|_\infty < \varepsilon_\Phi$, where ε_Φ is a user-defined threshold.~~ Two different convergence criteria may be used: the user may choose to enforce

- either $\left\|\Phi^k - \Phi^{k-1}\right\|_\infty < \varepsilon_\Phi$, where ε_Φ is a user-defined threshold (most relevant if the pressure is a primary variable of interest and is well-defined everywhere);
- or $\left\| \dfrac{\partial \Phi^k}{\partial \zeta} - \dfrac{\partial \Phi^{k-1}}{\partial \zeta} \right\|_\infty < \varepsilon_{\nabla\Phi}$, where $\varepsilon_{\nabla\Phi}$ is a user-defined threshold and ζ is any (or all) of x, y, z (most relevant to ensure strict wall and/or interface no-slip boundary boundary conditions).

The structure and internal logic of the solver is designed so as to minimize the cost of such extra iterations when $1 < k \leq k_{\max}$: relevant computation-intensive data pertaining to the construction of the discretized linear systems is kept in memory (to avoid re-computing), as well as discretization matrices, possible preconditioners, etc.

2.6. Expansion of the ghost layer

Several building bricks of the solver require second-degree (or even third-degree) neighbor cells to ensure robust behavior and properly defined operators. For instance, the construction of Voronoi cells based on face-collocated seeds requires to connect neighboring face-sampled degrees of freedom. As illustrated in figure 2, such neighboring seeds may lie on a face that is shared between a (large) first-degree neighbor cell and a (small) second-degree neighbor cell. In such a case, only the (small) second-degree neighbor cell indexes the queried face. Therefore, second-degree neighbors need to be accessible from every locally owned face degree of freedom. Besides, when using stretched grids, more remote neighbors may be involved in the construction of a local Voronoi cell (see figure 2). Similarly, the cell-centered operators defined in section 2.4 require second-degree neighbors in case of non-graded grids. As depicted in figure 5, the ability to access second-degree neighbor cells is also desirable regarding the accuracy and the inter-processor smoothness of the moving least-square interpolation procedure used to define the node-sampled velocity fields based on the face-sampled components [30]. The ability to construct deep ghost layers is a recent extension to the `p4est` interface, which we briefly describe here.

The algorithm used by `p4est` to construct a single layer of ghosts ([15, Algorithm 19]) is able to maximize the overlap of computation and communication because each process can determine for itself which other processes are adjacent to it. This is because the "shape" of each process's subdomain (determined by the interval of the space-filling curve assigned to it) is known to every other process. As a consequence, the communication pattern is symmetric and no sender-receiver handshake is required.

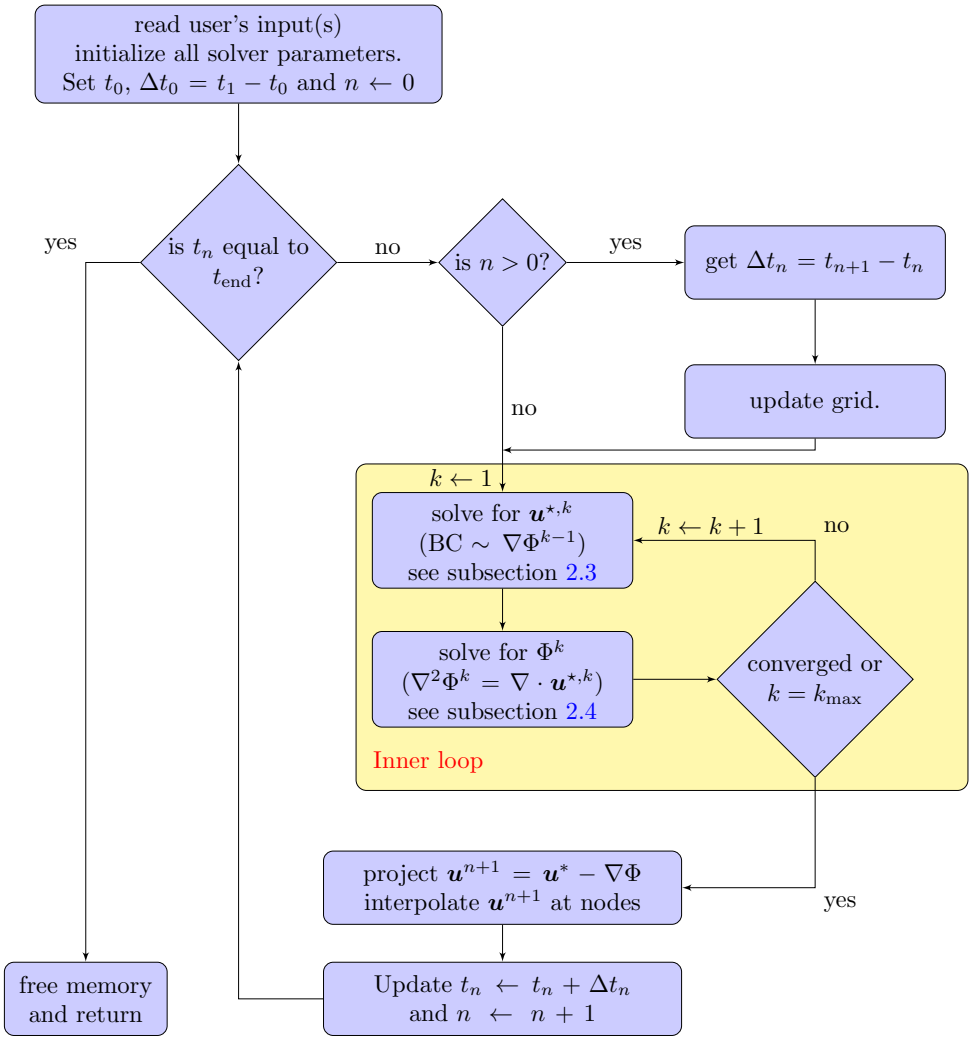

Figure 4: Typical flow chart for the presented incompressible Navier-Stokes solver.

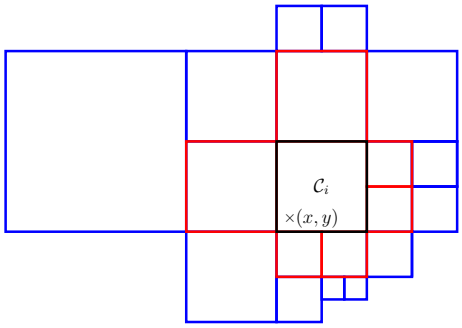

Figure 5: The stencil used to interpolate the velocity at (x, y) in cell $\mathcal{C}_i$ does not only require the data in $\text{ngbd}(\mathcal{C}_i)$ (red), but also in $\text{ngbd}^2(\mathcal{C}_i)$ (blue), a set of cells including second-degree (indirect) neighbors.

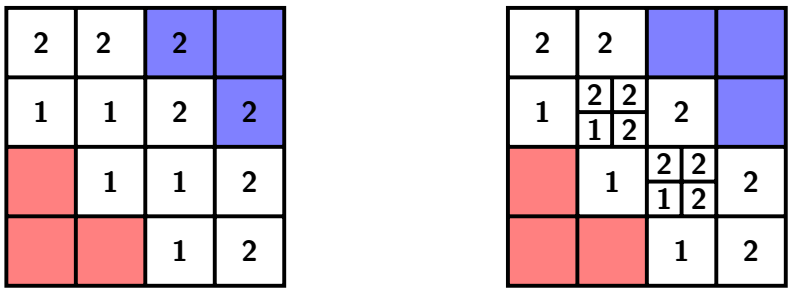

Figure 6: Two meshes with the same partition shapes, but with different two-deep ghost layers. For each mesh we show the first and second layers of the ghost layer of process p (red). In the first mesh, the second layer includes cells from process q (blue), but in the second it does not.

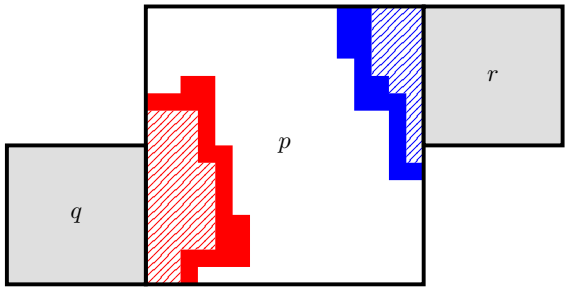

Figure 7: We show the preimages of process q's and process r's ghost layers in the leaves of process p. The solid red area represent the cells at the "front" of the preimage for q (`preimage_front`$[q]$ in algorithm 1), while the solid and dashed together form the whole preimage (`preimage`$[q]$).

As a first extension, when creating the send buffers we remember their entries, since they identify the subset of local cells that are ghosts to one or more remote processes. We store these pre-image cells or "mirrors" in ascending order with respect to the space filling curve, and create one separate index list per remote processor into this array. This data is accommodated inside the ghost layer data structure and proves useful for many purposes, the most common being the local processor needing to iterate through the pre-image to define and fill send buffers with application-dependent numerical data.

The communication pattern of a deeper ghost layer, on the other hand, depends not just on the shapes of the subdomains, but the leaves within them, as illustrated in figure 6. Rather than complicating the existing ghost layer construction algorithm to accommodate deep ghost layers, a function that adds an additional layer to an existing ghost layer has been added to `p4est`. This function is called `p4est_ghost_expand`() and adds to both the ghosts and the pre-images. Thus, as a second extension to the data structure, we also identify those local leaves that are on the inward-facing front of each preimage, in other words the most recently added mirrors. This is illustrated in figure 7.

When process p expands its portion of process q, it loops over the leaves in the front of the pre-image for process q and adds any neighbors that are not already in the ghost layer. Sometimes this will include a leaf from a third process r: process p will also send such leaves to process q, because it may be that r is not yet represented in q's ghost layer, and so communication between q and r is not yet expected. The basic structure of this algorithm is outlined in algorithm 1.

```
1: for q ∈ ghost_neighbors do                          ▷ processes that contribute to ghost layer
2:     initialize empty sets send_forward[q], send_back[q], and new_front[q]
3: end for
4: for q ∈ ghost_neighbors do
5:     for l ∈ preimage_front[q] do
6:         for each neighbor n of l in local_leaves do                          ▷ n found by search
7:             if n ∉ preimage[q] then
8:                 add n to send_forward[q], preimage[q], and new_front[q]
9:             end if
10:        end for
11:        for each neighbor n of l in ghost_layer do                           ▷ n found by search
12:            if n belongs to process r ≠ q then
13:                add n to send_forward[q] and (n, q) to send_back[r]
14:            end if
15:        end for
16:    end for
17:    replace preimage_front[q] with new_front[q]
18: end for
19: for q ∈ ghost_neighbors do
20:    send send_forward[q] and send_back[q] and receive recv_forward[q] and recv_back[q]
21:    add all of recv_forward[q] to ghost_layer
22:    for (l, r) ∈ recv_back[q] do
23:        if r ∉ ghost_neighbors or l ∉ preimage[r] then
24:            add l to preimage[r] and preimage_front[r]    ▷ new lists if r ∉ ghost_neighbors
25:        end if
26:    end for
27: end for
28: recompute ghost_neighbors from leaves in ghost_layer
```

Algorithm 1: Process p's algorithm for expanding other processes' ghost layers, and receiving expansions to its own ghost layer. Note that finding a neighbor of a leaf l entails a fixed number of binary searches through the `local_leaves`, which are sorted by the space-filling curve induced total ordering.

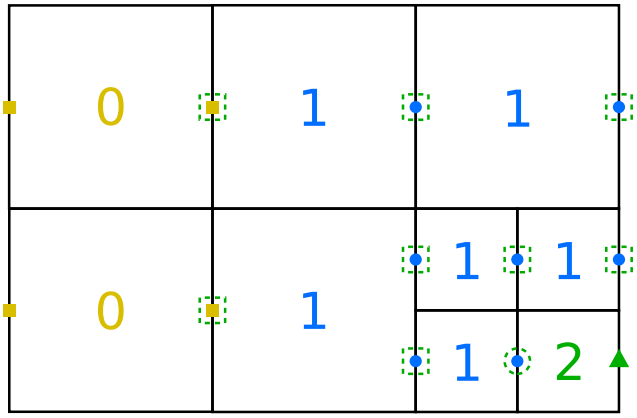

Figure 8: Illustration of the ghost layer of x-faces of depth 2 and of the global indexing procedure for process 2. The numbers in the leaves correspond to the indices of the processes owning them. After the first step, the remote index for the circled face is known to process 2, and after the second step the remote indices for the faces in a square are known to process 2. Note that a single step would not be sufficient for process 2 to gain knowledge of the remote index of the two faces belonging to process 0.

2.7. Indexing the faces

Although the `p4est` library provides a global numbering for the faces of the leaves, its numbering differs from our needs because it does not number the small faces on a coarse-fine interface, where we have degrees of freedom in our MAC scheme. Therefore, we implement a procedure to distribute the faces of the leaves across the processes and to generate a unique global index for each face. Since some faces are shared between two processes, we chose to attribute a shared face to the process with the smaller index. With this rule, each face belongs to a unique process and after broadcasting the local number of faces a global index can be generated for all the local faces. The second step is to update the remote index of the faces located in the ghost layer so that their global index can be constructed easily by simply adding the offset of the process each face belongs to. We do so in two steps, represented in figure 8. First, the indices of the ghost faces of the local leaves are synchronized, then the indices of the faces of the ghost layer of leaves are updated. This has some similarities to the two-pass node numbering from [7], here extended to two layers of ghosts. Algorithm 2 details the steps of our implementation and makes use of the `Notify` collective algorithm described in [34] to reverse the asymmetric communication pattern.

3. Scalability

In this section, we present an analysis of the scaling performance of our implementation. We define the parallel efficiency as $e = s\,(P_0/P)^\sigma$ where $s = t_0/t_p$ is the speed-up, σ is the optimal parallel scaling coefficient ($\sigma = 1$ for linear scaling), P_0 is the smallest considered number of processes with its associated runtime t_0 and P is the number of processes with its associated runtime t_p. All the results were obtained on the "Knights Landing" Intel Xeon Phi 7250 (KNL) compute nodes of the Stampede2 supercomputer at the Texas Advanced Computing Center (TACC), at The University of Texas at Austin, and on the Comet supercomputer at the San Diego Supercomputer Center, at the University of California at San Diego. Those resources are available through the Extreme Science and Engineering Discovery Environment (XSEDE) [77]. The strong scaling performance was analyzed up to 32,768 cores on Stampede2.

```
1: for l ∈ (local|ghost) leaves do
2:     for f ∈ remote_faces(l) do
3:         add proc(f) to receivers
4:         add f to buffer[proc(f)]
5:     end for
6: end for
7: Notify(receivers,senders)                          ▷ reverse communication pattern
8: for p ∈ receivers do                                              ▷ send requests
9:     MPI_Isend(buffer[p])                          ▷ send request to process p
10: end for
11: for p ∈ senders do                                        ▷ process remote requests
12:     MPI_Recv(req)                               ▷ receive request from process p
13:     assemble answer with local indices requested
14:     MPI_Isend(ans)                                ▷ send answer to process p
15: end for
16: for p ∈ receivers do                                             ▷ process answers
17:     MPI_Recv(p)                                 ▷ receive answer from process p
18:     update faces information
19: end for
```

Algorithm 2: Communication algorithm to generate a global indexing of the faces. The `Notify` collective algorithm is used to reverse the communication pattern, described in more detail in [34].

3.1. Expansion of the ghost layer

We present both weak and strong scaling results for the algorithm used to expand the ghost layer of cells for each process in figure 9. The associated efficiency is presented in table 1. The strong scaling consists in choosing a problem and solving it with increasing number of processes. Ideally, for an algorithm with a workload increasing linearly with the problem size (i.e., with parallel scaling coefficient $\sigma = 1$), doubling the amount of resources spent on solving a problem should half the runtime. However, in the case of the ghost layer expansion, the amount of work depends on the size of the ghost layers, as explained in [35]. For a well behaved partition, we expect $O(N^{\frac{(d-1)}{d}})$ of the leaves to be in the ghost layer, where d is the number of spatial dimensions. We therefore consider a parallel scaling coefficient $\sigma = 2/3$ to be optimal for a three dimensional problem, i.e., $O((N/P)^{2/3})$ is the ideal scaling, with P the number of processes and N the problem size. The results presented in figure 9 were obtained on Stampede2 for a mesh of level 9/13, corresponding to 588,548,472 leaves, and on Comet for a mesh of level 10/13, corresponding to 1,595,058,088 leaves. The computed parallel efficiency between the smallest and the largest run is 66% for Stampede2 and 59% for Comet.

The idea behind the weak scaling is to keep the problem size constant for each process while increasing the number of processes. The right graph of figure 9 presents the results obtained on Stampede2 for two problems of sizes 30,248 leaves per process and 473,768 leaves per process, and for a number of processes ranging from 27 to 4,096. The runtime increases by 16% between the smallest and the largest run for the small problem and by 6% for the large problem.

3.2. Indexing the faces

The scaling procedure presented in the previous section is repeated for Algorithm 2 and the results are presented in figure 10. Even though the workload for this procedure increases slightly as the number of processes increases and the number of leaves in the ghost layers increases, we compare our results to an ideal linear scaling $\sigma = 1$. The corresponding efficiency is computed in table 2. The parallel efficiency e computed between the smallest and the largest run from the

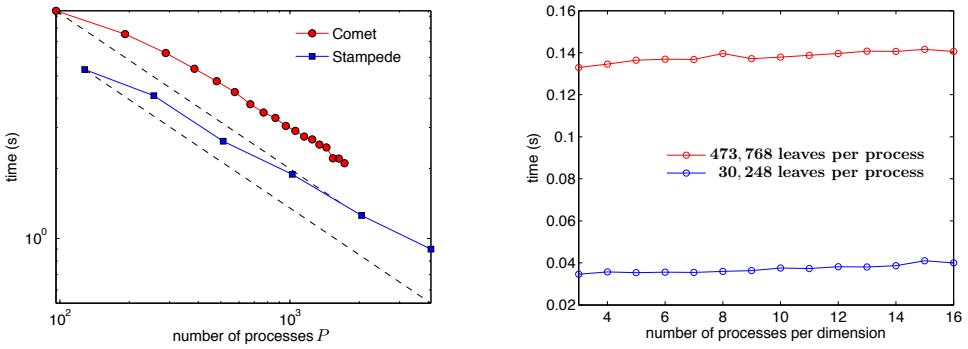

Figure 9: Scaling results for the expansion of the layer of ghost cells (see section 3.1). The strong scaling results are presented in the left figure together with the optimal reference scaling for a parallel scaling coefficient $\sigma = 2/3$ (dashed lines) while the weak scaling results are shown on the right figure. The increases in runtime observed for the weak scaling are of 16% for the small problem and 6% for the large problem.

Stampede2

Number of processes P	128	256	512	1024	2048	4096
Efficiency e	100%	79%	70%	69%	66%	66%

Comet

Number of processes P	96	192	384	672	1152	1728
Efficiency e	100%	82%	81%	71%	67%	59%

Table 1: Efficiency of the procedure for expanding the ghost layer of leaves.

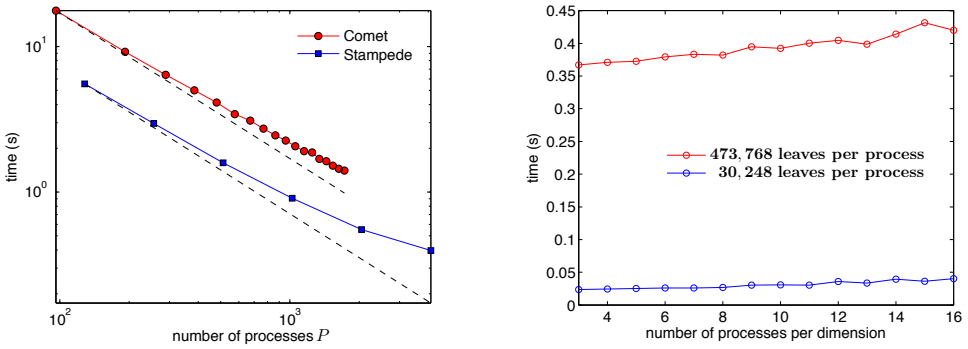

Figure 10: Scaling results for the indexing of the faces with Algorithm 2. The strong scaling results are presented in the left figure together with the reference ideal linear scaling (dash lines) while the weak scaling results are shown on the right figure. The strong scaling problem shown for Comet is three times larger than the one for Stampede2. The increases in runtime observed for the weak scaling are of 71% for the small problem and 14% for the large problem.

strong scaling results is 44% for Stampede2 and 70% for Comet. The weak scaling results show an increase in runtime of 71% for the small problem and of 14% for the large problem.

Stampede2

Number of processes P	128	256	512	1024	2048	4096
Efficiency e	100%	94%	87%	76%	63%	44%

Comet

Number of processes P	96	192	384	672	1152	1728
Efficiency e	100%	96%	88%	82%	77%	70%

Table 2: Efficiency of Algorithm 2 producing a global index for the faces.

3.3. Scalability of the full solver

We now analyze the scaling performance of the full incompressible Navier-Stokes solver, breaking its execution time down into its four main fundamental components: the viscosity step (see subsection 2.3), the projection step (see subsection 2.4), the moving least-square interpolation of the velocity components from cell faces to the grid nodes and the re-meshing step (denoted as grid update). We intend to show satisfactory strong scaling on large numbers of processors, so this scaling analysis was conducted on Stampede2 only since we do not have access to the same resources on other supercomputers.

For this purpose, the solver is restarted from a physically relevant and computationally challenging simulation state, defined as the inception of vortex shedding for the flow past a sphere at $Re = 500$, as illustrated in figure 11. A macromesh of size $8 \times 4 \times 4$ is used with two different refinement criteria. In the first case, the Octrees are refined with a minimum level 6 and a maximum level 11 with a vorticity threshold $\gamma = 0.02$ (see (2)), leading to a total of about 270×10^6 grid computational cells. In the second case, the Octrees are refined with a minimum level 7 and a maximum level 11 with a vorticity threshold $\gamma = 0.015$, leading to a total of about 610×10^6 grid cells. The grids for the initial states of the two scenarios are illustrated in figure 12. The three successive linear systems of the viscosity steps are solved using a BiConjugate Gradient Stabilized solver, while a Conjugate Gradient solver is used for the (symmetric positive definite) projection step.

In the first case, the wall-clock execution time is measured and averaged over 10 full time steps, while only 5 time steps are considered for the second larger case (to limit the cost of these runs). A minimum of 64 (resp. 90) KNL nodes were required for the problem to fit in memory in the first (resp. second) case. Therefore, the first two data points in the left (resp. right) graph from figure 13 used less than 68 cores per node (maximum available). In either case, the solver performs two subiterations of the inner loop per time step (see figure 4): for each time step, $\left\|\Phi^k - \Phi^{k-1}\right\|_\infty$ drops by 4 orders of magnitude between $k = 1$ and $k = 2$.

The results are presented in figure 13 and table 3. As expected, the projection step is the most challenging part, thus determining the strong scalability limits of the solver: no significant speed-up is observed when the number of cells per process falls under $10,000$[3]. In comparison, the scaling behavior of the grid update procedure is not impeded yet around that limit, which shows the extremely good performance of all `p4est`'s grid management operations that are in play: grid refinement and/or grid coarsening, grid partitioning, ghost layer creation and ghost layer expansion.

Given that the global solver makes use of various separate routines having different (theoretical) ideal scaling coefficients σ, it is expected that the strong scaling performance of the solver is less than ideal. In fact, some of the operations at play cannot even be attributed such a theoretical scaling coefficients: when considering a very large number of processes, several of them will be associated with regions of the computational domain that are (very) far away from the interface and will end up stalling during the geometric extrapolation tasks of primary face- and cell-sampled fields, for instance. Though these extrapolations represent a small portion of the overall workload when using a small number of processes, they do contribute to less than ideal scaling on large numbers

[3]This is consistent with PETSc scaling performance reported in their documentation.

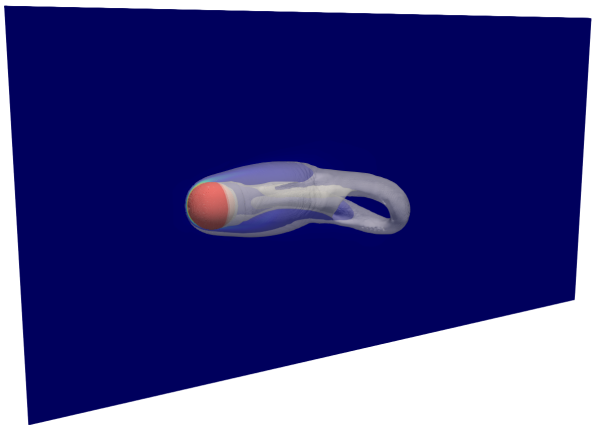

Figure 11: Physically relevant initial state considered for the scaling analysis on large number of cores. This figure illustrates the inception of vortex shedding for the flow past a sphere at $Re = 500$ (the full description of the computational set-up can be found in section 4.2). The vertical slice has equation $z = -0.5$ and it is colored by vorticity. The static sphere is colored in red and the translucent white surface represents the isocontour of vorticity $\|\nabla \times \boldsymbol{u}\| = 0.6\, u_0/r$.

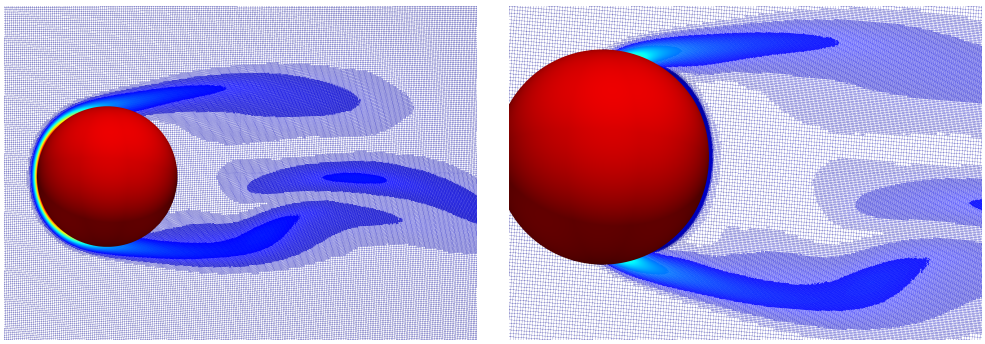

Figure 12: Grid illustrations for the scaling analyses from section 3.3. A grid slice in the computational domain is illustrated and its edges are colored by vorticity intensity. The Octrees are refined with a minimum level 7 and a maximum level 11 with a vorticity threshold $\gamma = 0.015$ (total of about 610×10^6 grid cells).

of processes in such an application. Therefore, regarding several aspects, this analysis may be considered a "worst case" scenario, which aims to produce insightful information when it comes to estimating a lower bound for the (effective) scaling coefficient σ to consider when estimating the computational cost of future large-scale simulations and/or when assessing the limits of accessible simulations for the solver. As illustrated in figure 13, the solver's scaling behavior seems to follow an asymptotic law of $\sigma \simeq 0.78$, in such a symptomatic case; Table 3 also indicates an efficiency above 80% (in the relevant range of P) when considering $\sigma \simeq 0.85$ (almost all calculated efficiencies are 100% or higher when considering $\sigma = 0.78$).

4. Numerical validation and illustrations

In this section, we present a series of numerical examples to validate the implementation as well as to demonstrate the potential of the approach.

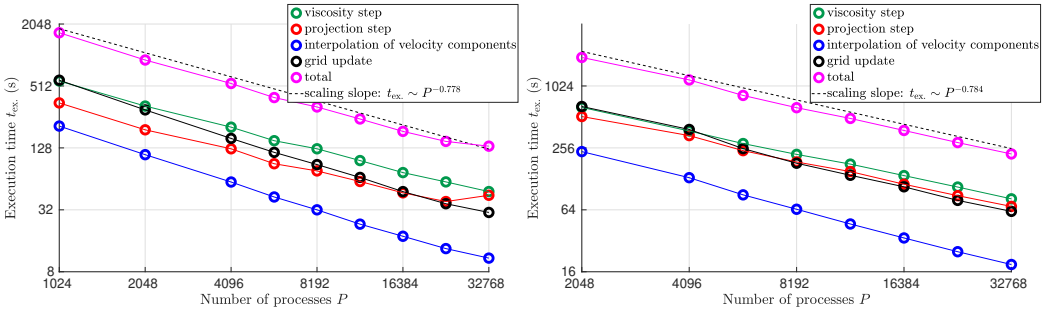

Figure 13: Scaling results on large number of cores on Stampede2 for the two grids considered. Left: scaling results for Octrees refined with a minimum level 6 and a maximum level 11 with a vorticity threshold $\gamma = 0.02$ (total of about 270×10^6 grid cells). Right: scaling results for Octrees refined with a minimum level 7 and a maximum level 11 with a vorticity threshold $\gamma = 0.015$ (total of about 610×10^6 grid cells).

Stampede2 (270×10^6 grid cells)

# of processes P	1,024	2,048	4,096	5,800	8,192	11,590	16,384	23,170	32,768
e ($\sigma = 1$)	100%	92.1%	77.9%	75.4%	65.9%	60.9%	56.9%	50.3%	39.6%
e ($\sigma = 0.85$)	100%	102.2%	95.8%	97.7%	90.0%	87.7%	86.2%	80.36%	66.6%

Stampede2 (610×10^6 grid cells)

# of processes P	2,048	4,096	5,900	8,192	11,590	16,384	23,170	32,768
e ($\sigma = 1$)	100%	82.7 %	82.7 %	77 %	69.2 %	64.1 %	59.4 %	54.2 %
e ($\sigma = 0.85$)	100%	91.8 %	96.6 %	94.8 %	89.8 %	87.6 %	85.5 %	82.2 %

Table 3: Efficiencies e of the full solver proposed for the incompressible Navier-Stokes equations on Stampede2, when considering an ideal linear scaling (i.e., $\sigma = 1$) or a scaling coefficient of 85%.

4.1. Validation with an analytical solution

The first application aims at validating the implementation by monitoring the convergence of the solver using the analytical solution presented in [59]. Consider the irregular domain $\Omega = \{(x, y, z) | -\cos(x)\cos(y)\cos(z) \geq 0.4 \text{ and } \frac{\pi}{2} \leq x, y, z \leq \frac{3\pi}{2}\}$ and the exact solution

$$\begin{aligned} u(x, y, z) &= \cos(x)\sin(y)\sin(z)\cos(t), \\ v(x, y, z) &= \sin(x)\cos(y)\sin(z)\cos(t), \\ w(x, y, z) &= -2\sin(x)\sin(y)\cos(z)\cos(t), \\ p(x, y, z) &= 0. \end{aligned}$$

The exact velocity is prescribed at the domain's boundary and homogeneous Neumann boundary conditions are enforced on the Hodge variable. The corresponding forcing term is applied to the viscosity step. We take a final time of $\frac{\pi}{3}$ and monitor the error on the velocity field and on the Hodge variable as the mesh resolution increases. The computational grid is not dynamically adapted for this accuracy analysis, so the grid-parameter γ (see (2)) is irrelevant in this case. The first computational grid is built to satisfy the distance-based criterion from section 2.1.2 using $\phi(x, y, z) = \cos(x)\cos(y)\cos(z) + .4$, $K = 1.2$ and $b = 5$. The successive resolutions are then obtained by splitting every cell from the previous resolution. The results are presented in table 4 and indicate first-order accuracy for the velocity field and second order accuracy for the Hodge variable in the L^∞ norm.

4.2. Vortex shedding of a flow past a sphere

In order to further assess the performance of the solver, we also address the flow past a sphere. Related properties like drag and lift forces as well as vortex shedding frequency (if applicable) are calculated and compared to available data for this canonical problem.

	u, v		w		Hodge variable	
level (min/max)	L^∞ error	order	L^∞ error	order	L^∞ error	order
4/6	$4.65 \cdot 10^{-3}$	-	$3.50 \cdot 10^{-3}$	-	$8.96 \cdot 10^{-4}$	-
5/7	$3.27 \cdot 10^{-3}$	0.50	$2.11 \cdot 10^{-3}$	0.73	$2.85 \cdot 10^{-4}$	1.65
6/8	$1.67 \cdot 10^{-3}$	0.97	$1.16 \cdot 10^{-3}$	0.86	$8.07 \cdot 10^{-5}$	1.82
7/9	$8.42 \cdot 10^{-4}$	0.99	$5.52 \cdot 10^{-4}$	1.07	$2.27 \cdot 10^{-5}$	1.83

Table 4: Convergence of the solver for the analytical solution presented in section 4.1. First-order accuracy is observed for the velocity field and second order accuracy for the Hodge variable.

We consider a static sphere of radius $r = 1$, located at $(8, 0, 0)$ in the domain $\Omega = [0, 32] \times [-8, 8] \times [-8, 8]$. An inflow velocity $\boldsymbol{u}_0 = u_0 \boldsymbol{e}_x$ is imposed on the $x = 0$ face of the domain as well as on the the side walls, homogeneous Neumann boundary conditions are imposed on the velocity field at the outlet $x = 32$ and no-slip conditions are imposed on the sphere. The pressure is set to zero at the outlet and is subject to homogeneous Neumann boundary conditions on the other walls as well as on the sphere. We set $u_0 = 1$, the density of the fluid to $\rho = 1$ and vary the viscosity μ to match the desired Reynolds number $Re = \dfrac{2\,\rho u_0 r}{\mu}$, set by the user. Eight Reynolds numbers ranging from 50 to 500 are considered.

The Octree mesh is refined around the sphere and according to the vorticity criterion (2) of section 2.1.2: we use $\phi(\boldsymbol{x}) = r - \sqrt{(x-8)^2 + y^2 + z^2}$, $K = 1.2$, $b = 16$, with a vorticity-based threshold of $\gamma = 0.01$. All the results were obtained with a macromesh $8 \times 4 \times 4$ and with trees of levels 4/7, leading to approximately 6 million leaves and corresponding to an equivalent uniform grid resolution of 268,435,456 cells. The time step Δt is set such that $\dfrac{\max_\Omega \|\boldsymbol{u}\|\,\Delta t}{\Delta x_{\min}} \leq 1$ at all times. Figure 14 shows a snapshot of the unsteady flow for $Re = 300$ at $t = 221\, r/u_0$.

The nondimensional force $\boldsymbol{F}$ applied onto the static sphere is monitored over time. The force is evaluated by geometric integration [53] of the stress vector (including viscous and pressure contributions) on the surface of the sphere Γ, i.e.,

$$\boldsymbol{F} = \frac{1}{\frac{1}{2}\rho u_0^2 \pi r^2} \int_\Gamma \left(-p\underline{\underline{I}} + 2\mu \underline{\underline{D}}\right) \cdot \boldsymbol{n}\, \mathrm{d}\Gamma, \tag{10}$$

where $\underline{\underline{I}}$ is the identity tensor, $\underline{\underline{D}}$ is the symmetric strain-rate tensor and $\boldsymbol{n}$ is the outward normal to the sphere. Figure 16 shows the evolution of the streamwise force component. Figure 15 shows the evolution of the pressure and azimuthal vorticity on the sphere surface, at steady state for $Re = 100$. Figure 17 illustrates the transverse force components with respect to time for $Re \geq 250$. Time-averaged drag and lift coefficients C_D and C_L are then calculated as

$$C_D = \frac{1}{(t_{\text{end}} - t_{\text{start}})} \int_{t_{\text{start}}}^{t_{\text{end}}} \boldsymbol{F} \cdot \boldsymbol{e}_x \, \mathrm{d}t, \qquad C_L = \frac{1}{(t_{\text{end}} - t_{\text{start}})} \int_{t_{\text{start}}}^{t_{\text{end}}} \left((\boldsymbol{F} \cdot \boldsymbol{e}_y)^2 + (\boldsymbol{F} \cdot \boldsymbol{e}_z)^2 \right)^{1/2} \mathrm{d}t, \tag{11}$$

where t_{start} is chosen to disregard the initial transient due to the chosen (uniform) initial condition and t_{end} is the final simulation time.

For $Re \geq 275 \pm 5$, the flow is unsteady and vortices shed from the static sphere [57, 3]. Similarly to [3], we evaluate the vortex shedding frequency f and the corresponding Strouhal number $St = \frac{2rf}{u_0}$ by calculating an averaged period between successive peak values in the transverse force components[4]. A main vortex shedding frequency cannot be reliably defined using this methodology for $Re = 500$: as it can be seen from figure 17, the time variations in the transverse force components do not reveal a well-defined periodic pattern for $Re = 500$. In fact, a Fourier decomposition of (pseudoperiodic portions of) the signals actually reveals a broad frequency spectrum with significant contributions up to $St \simeq 0.17$ in that case.

[4]Note that the analysis from [57] is different in that it is based on time variations of the pressure in the wake.

Our results are summarized and presented in tables 5 and 6 along with available data from various publications from the literature.

	$Re = 50$	$Re = 100$	$Re = 150$	$Re = 215$	$Re = 250$	
	C_D	C_D	C_D	C_D	C_D	C_L
Kim *et al.* [42]	-	1.09	-	-	0.70	0.059
Johnson *et al.* [37]	1.57	1.08	0.90	-	0.70	0.062
Constantinescu *et al.* [20]	-	-	-	-	0.70	0.062
Choi *et al.* [17]	-	1.09	-	-	0.70	0.052
Bagchi *et al.* [3]	1.57	1.09	-	-	0.70	-
Marella *et al.* [47]	1.56	1.06	0.85	0.70	-	-
Guittet *et al.* [30]	-	1.11	-	-	~~0.784~~	-
Present	1.61	1.11	0.91	0.76	0.72	0.062

Table 5: Drag coefficient (and lift coefficient, if relevant) for the steady flow past a sphere. The time averages were obtained with $t_{\text{start}} = 50\, r/u_0$ and $t_{\text{end}} = 200\, r/u_0$ for $Re \leq 215$ and with $t_{\text{start}} = 275\, r/u_0$ and $t_{\text{end}} = 400\, r/u_0$ for $Re = 250$ (see (11)).

	$Re = 300$			$Re = 350$		$Re = 500$
	C_D	C_L	St	C_D	St	C_D
Kim *et al.* [41]	0.657	0.067	0.134	-	-	-
Johnson *et al.* [37]	0.656	0.069	0.137	-	-	-
Constantinescu *et al.* [20]	0.655	0.065	0.136	-	-	-
Choi *et al.* [17]	0.658	0.068	0.134	-	-	-
Marella *et al.* [47]	0.621	-	0.133	-	-	-
Bagchi *et al.* [3]	-	-	-	0.62	0.135	0.555
Mittal *et al.* [57]	0.64	-	0.135	0.625	0.142	-
Guittet *et al.* [30]	0.659	-	0.137	0.627	0.141	-
Present	0.673	0.068	0.134	0.633	0.132±0.002	0.558

Table 6: Drag coefficients for the unsteady flow past a sphere. If relevant, the lift coefficient and the Strouhal number are presented as well. The time averages were obtained with $t_{\text{start}} = 200\, r/u_0$ and $t_{\text{end}} = 400\, r/u_0$ (see (11)).

For all the simulations from this section, we have used the inner-loop convergence criterion $\left\|\nabla\Phi^k - \nabla\Phi^{k-1}\right\| < 10^{-4}\, u_0$ (see section 2.5) in order to ensure a proper and accurate enforcement of the no-slip boundary condition on the surface of the sphere and to investigate the relevance of such a procedure in this specific case. Figure 18 shows relevant information pertaining to that analysis. As illustrated in that figure, the solver converges most of the time in two iterations and the correction brought by the second iteration is 3 to 4 orders of magnitude smaller than for the approximate projection step, in this case. Although the added computational cost is limited (around 30%), the accuracy of the results would most likely not have suffered from using an approximate projection in this context.

4.3. Oscillating sphere in a viscous fluid

The solver presented in this article is able to handle moving geometries. We illustrate this capacity by computing the drag force developed by the flow of a viscous fluid due to the oscillatory motion of a rigid sphere in a closed box.

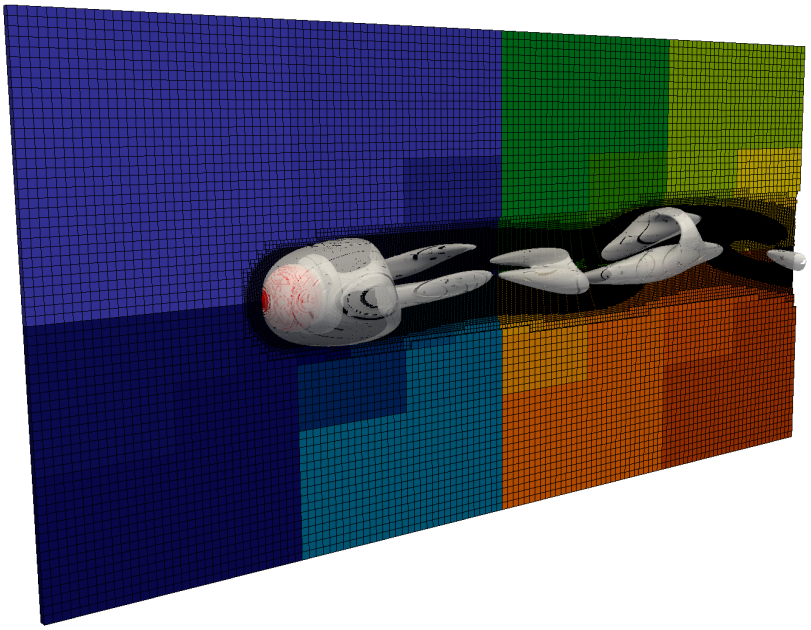

Figure 14: Visualization of the unsteady flow past a sphere for $Re = 300$. The trees are level 4/7 rooted in a $8 \times 4 \times 4$ macromesh, leading to approximately 6 million leaves. The snapshot is taken at time $t = 221\, r/u_0$. The colors correspond to the process ranks and the surface is an isocontour of the Q-criterion [33] for $Q = 0.006$. This simulation was run on the Stampede2 supercomputer with 1024 processes.

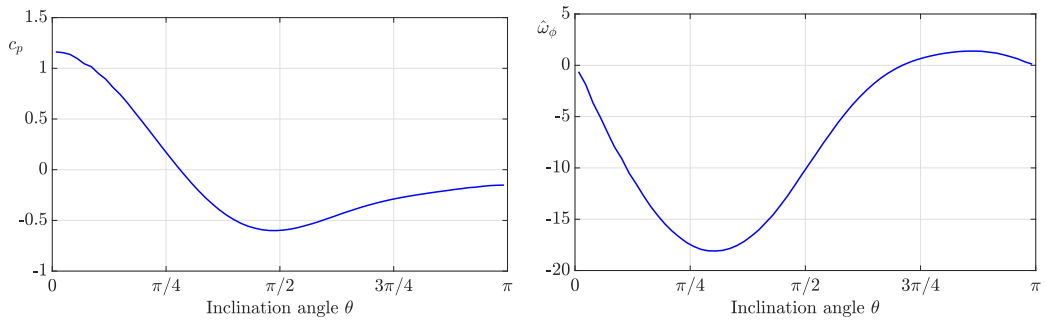

Figure 15: Evolution of relevant surface quantities on the surface of the static sphere for $Re = 100$, as a function of the inclination angle θ measured from the front point $(-r,\, 0,\, 0)$. Left: non-dimensional local pressure $c_p = p/\left(\frac{1}{2}\rho u_0^2\right)$ as a function of the inclination angle. Right: non-dimensional azimuthal vorticity $\hat{\omega}_\phi = \frac{-2r}{u_0}\boldsymbol{e}_\phi \cdot (\nabla \times \boldsymbol{u})$, where $\boldsymbol{e}_\phi = \boldsymbol{e}_r \times \boldsymbol{e}_\theta$, using spherical coordinates centered at the sphere's center along with the defintion of θ given here above. These results are in good agreement with [41].

We consider a sphere of radius $r = 0.1$ in a domain $\Omega = [-1, 1]^3$. The kinematics of the center of the sphere $\boldsymbol{c}\,(t)$ is dictated by

$$\boldsymbol{c}\,(t) = -X_0 \cos\left(2\pi f_0 t\right) \boldsymbol{e}_x, \tag{12}$$

and we use the (time-varying) levelset function $\phi\,(\boldsymbol{x}, t) = r - \|\boldsymbol{x} - \boldsymbol{c}\,(t)\|$ with the corresponding grid-construction parameters $K = 1.2$, $b = 4$ and $\gamma = 0.1$. The motion of the sphere is set to be purely translational (no rotation) so that its kinematics is fully described by (12). The dynamics

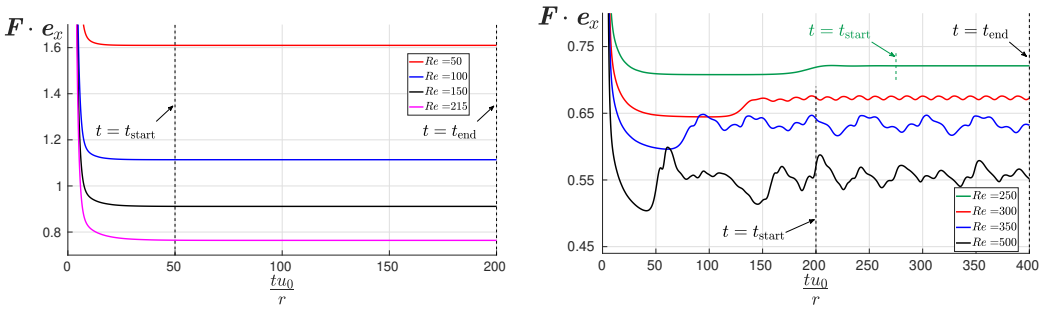

Figure 16: Drag coefficient on a sphere for axisymmetric steady flows (left) and for non-axisymmetric or unsteady flows (right), corresponding to Reynolds numbers ranging from 50 to 500.

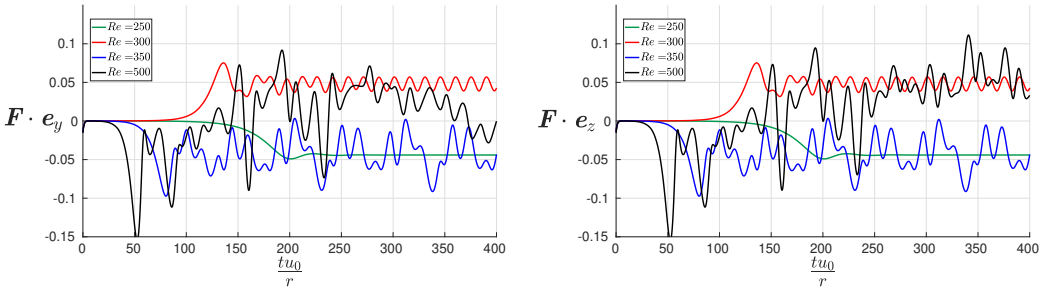

Figure 17: Nondimensional transverse force components for non-axisymmetric and/or unsteady flows.

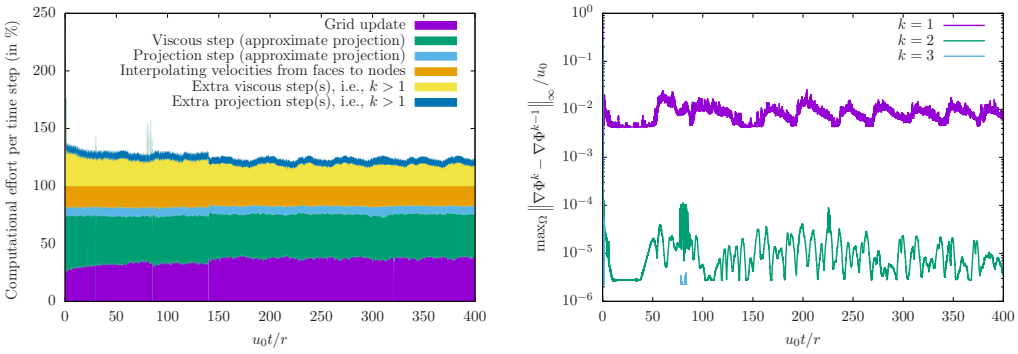

Figure 18: Illustration of the computational cost and convergence rate associated with the fixed point iteration from section 2.5, when ensuring a stringent user-defined control on $\boldsymbol{u}$ at all time steps, for the simulation of the flow past a sphere with $Re = 500$. Left: ratio(s) of the computational costs per time step for the main tasks at play, normalized to the raw computational cost of an approximate projection method (which would correspond to $k_{\max} = 1$). Right: measures of interest considered by the inner loop criterion (only a few time steps required 3 inner iterations hence the partial blue curve).

of the surrounding fluid is dictated by no-slip boundary conditions enforced onto the surface of the oscillatory sphere, i.e.,

$$\boldsymbol{u}(\boldsymbol{x}, t) = 2\pi f_0 X_0 \sin(2\pi f_0 t)\, \boldsymbol{e}_x, \quad \forall \boldsymbol{x} \in \Omega : \|\boldsymbol{x} - \boldsymbol{c}(t)\| = r. \tag{13}$$

No-slip boundary conditions are also enforced on the (static) borders of the computational domain.

This setup naturally defines a characteristic velocity scale $u_0 = 2\pi f_0 X_0$, a characteristic fre-

quency f_0 and a characteristic length scale r leading to two nondimensional numbers

$$\frac{r}{X_0} \quad \text{and} \quad Re = \frac{\rho\, 2\pi f_0 X_0\, 2r}{\mu}. \tag{14}$$

We set the first nondimensional number to 4 by assigning $X_0 = r/4$. The density is set to 1 and the dynamic viscosity μ is determined to match the desired Reynolds number set by the user based on (14).

For this example, we choose the fixed time step $\Delta t = \dfrac{1}{200 f_0}$ and the simulations are run for three full oscillation cycles for 7 different Reynolds numbers ranging from 10 to 300. The iterative procedure explained in section 2.5 is necessary in this case of moving boundary in order to correctly enforce the desired no-slip condition (13). Since we are interested in the overall forces applied onto the sphere as a result of its motion, the inner iterative technique is carried on until $\left\|\Phi^k - \Phi^{k-1}\right\|_\infty < \varepsilon_\Phi = 2\,\varepsilon\,(\pi f_0 X_0)^2\,\Delta t$ where $\varepsilon = 0.1$ for all time steps. This corresponds to limiting the difference in pressure between successive iterates to 10% at most of the maximum dynamic pressure resulting from the motion of the sphere. The fixed point method seems more relevant in this context compared to what was observed in section 4.2: the number of iterations required to ensure this convergence condition grows with Re: for every time step, the solver performs 3 to 4 inner iterations to enforce the desired boundary conditions correctly , as illustrated in figure 21 for $Re = 300$. A mesh of resolution 5/10 rooted in a single macromesh cell is used, resulting in about 500,000 leaves. A uniform grid with similar finest resolution would have $2^{30} \simeq 10^9$ computational cells. Every simulation completed in about 8 hours using 40 MPI tasks on a local workstation running a Dual Intel Xeon Gold 6148 processor with 64 GB of RAM.

As for the flow past a sphere, the nondimensional force applied onto the oscillating sphere is monitored over time, according to equation (10). The results are presented in figure 19. As expected, we observe that the amplitude of the drag coefficient increases as the Reynolds number decreases. Furthermore, we observe a lag in the response as the Reynolds number decreases. Indeed, as the viscous forces become more important, the information takes longer to propagate in the fluid. In contrast, the forces in a system dominated by inertia come mainly from the pressure term and the incompressibility condition enforces instantaneous propagation of the information. Figure 20 shows some visual representations of the computational grid for $Re = 50$ at $t = 2.4/f_0$.

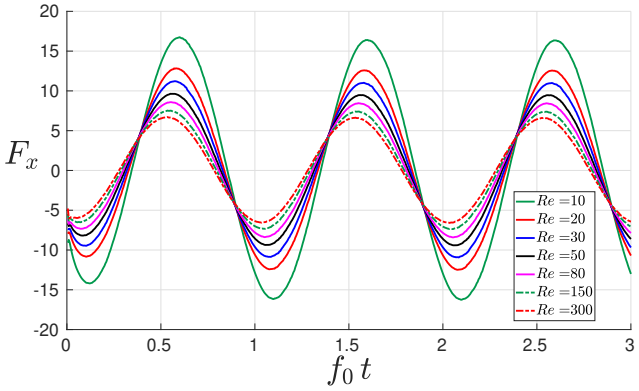

Figure 19: Evolution of the $x-$component of the nondimensional force applied onto the periodically oscillating sphere in a closed box for a range of Reynolds numbers. The magnitude of the force increases and the peaks appear at later times as the Reynolds number decreases and the viscous forces become dominant.

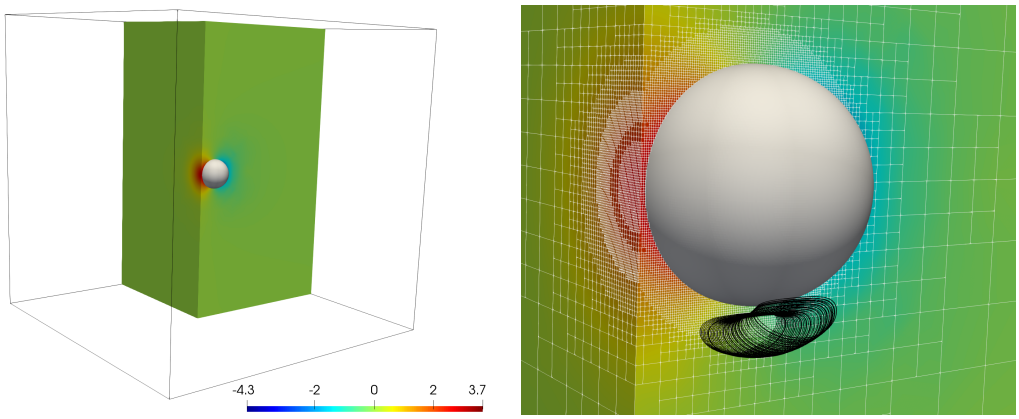

Figure 20: Left: illustration of one fourth of the computational domain ~~and part of the Octree computational grid~~ colored with the pressure (nondimensionalized by $\frac{1}{2}\rho\pi r^2 f_0^2$), along with the solid sphere ($Re = 50$, $t = 2.4/f_0$). Right: zoom-in on the sphere, illustration of the local adaptivity of the computational grid and representation of some streamlines near the moving interface.

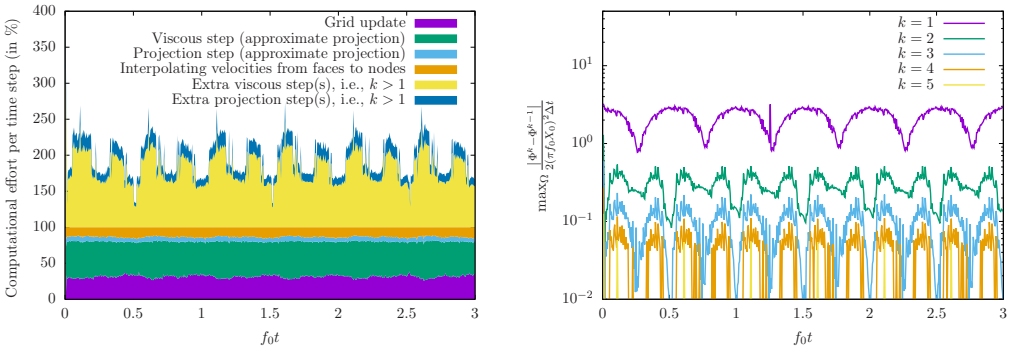

Figure 21: Illustration of the computational cost and convergence rate associated with the fixed point iteration from section 2.5, when ensuring point-wise convergence on the Hodge variable within bounds corresponding to a pressure threshold of 10% of the maximum dynamic pressure resulting from the kinematics of the oscillating sphere, as considered in section 4.3, for $Re = 300$. Left: ratio(s) of the computational costs per time step for the main tasks at play, normalized to the raw computational cost of an approximate projection method (which would correspond to $k_{\max} = 1$). Right: measures of interest considered by the inner loop criterion.

4.4. Transport of a scalar quantity in a flow

In this section, we provide two examples to qualitatively illustrate the ability of the solver to track scalar fields advected by the flow and to capture the resulting complex structures using high spatial resolution in the required regions. The (non-dimensional) node-sampled scalar concentration $\beta(\boldsymbol{x}, t) \in [0, 1]$ is determined by solving the governing equation

$$\frac{\partial \beta}{\partial t} + \boldsymbol{u} \cdot \nabla \beta = 0 \tag{15}$$

using a semi-Lagrangian approach, as described in 2.3.1. Since this numerical example is intended to only illustrate an additional refinement capability of our adaptive-grid strategy, the diffusion terms were discarded in the ~~last~~above equation~~s~~ in order to make the simulation cheaper. In fact, numerical diffusion comes into play through the approximation errors of the semi-Lagrangian approach and the successive interpolation steps, so that $\beta(\boldsymbol{x}, t)$ is numerically smoothened over time.

In this section, the refinement criterion (3) is activated using a refinement threshold set to $\chi = 0.05$ in the mesh construction, along with criteria (1) and (2). We illustrate this capability with the flow past a sphere. The simulation setup is the same as in section 4.2 and the solver's parameters were set to match a Reynolds number of $Re = 5,000$ using an $8 \times 4 \times 4$ macromesh with trees of level 2/8, leading to around 34×10^6 leaves for a fully developed flow. Figure 22 presents a visualization after $48\dfrac{r}{u_0}$ where r and u_0 are defined as in section 4.2, which corresponds to 857 time steps. The entire simulation took 20 hours on 1,024 cores of the Stampede2 supercomputer.

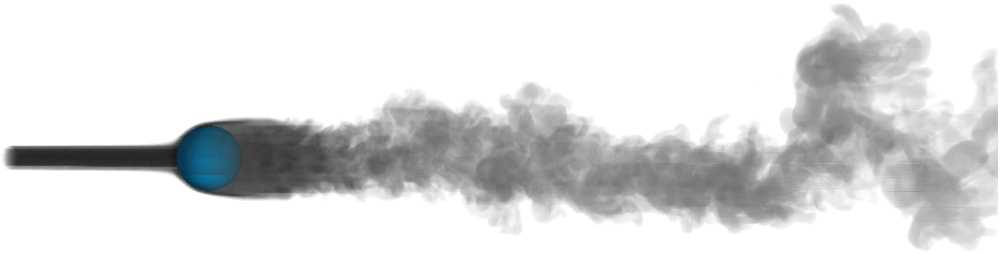

Figure 22: Visualization of a passively advected smoke marker in a flow past a sphere for a Reynolds number $Re = 5,000$.

As a second illustration, we consider a buoyancy-driven flow. In a cubic box of side length L, we consider the motion of a fluid of dynamic viscosity μ due to buoyancy with gravity acceleration $\boldsymbol{g} = -g\boldsymbol{e}_z$. A relative scalar concentration $\beta \in [0, 1]$ is passively advected by the flow according to (15) and the local fluid density depends linearly on β as $\rho_0 + \beta\Delta\rho$ with $\dfrac{\Delta\rho}{\rho_0} \ll 1$. Using Boussinesq approximation, the momentum equation reads

$$\rho_0 \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) = -\nabla p - \Delta\rho g \beta \boldsymbol{e}_z + \mu \nabla^2 \boldsymbol{u}. \tag{16}$$

The simulation is initialized with a flow at rest, $\beta = 1$ in a ball of radius $r = 0.1L$ centered at $(L/2, L/2, 3L/4)$ and $\beta = 0$ outside that ball. Using L as a characteristic length scale and $\dfrac{gr^2\Delta\rho}{\mu}$ as a characteristic velocity scale, (16) takes the non-dimensional form

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p - \frac{L^2}{r^2 Ar} \beta \boldsymbol{e}_z + \frac{1}{Ar} \nabla^2 \boldsymbol{u} \tag{17}$$

where the Archimedes number is defined as $Ar = \dfrac{\rho_0 g r^2 \Delta\rho L}{\mu^2}$. In figure 23, we show three snapshots of a simulation obtained with $Ar = 2.4525 \times 10^6$, using a $2 \times 2 \times 2$ macromesh with trees of resolution 4/8, leading to about 20 million leaves. Here again, this simulation intends to show the capabilities of our parallel adaptive refinement approach, accurate results would require much finer and much more computationally intensive runs. The simulation took 20 hours on 256 cores of the Stampede2 supercomputer for 603 time ~~iterations~~ steps.

4.5. Turbulent superhydrophobic channel

As detailed and illustrated above, the solver is designed to allow dynamic grid adaptation over time. This feature is especially relevant and appealing when the complex flow dynamics to be captured are bounded to a (small) evolving portion of the computational domain, as it dramatically reduces the global number of computational cells compared to a regular uniform grid. Indeed, in such cases, the computational overhead associated with dynamically adapting the

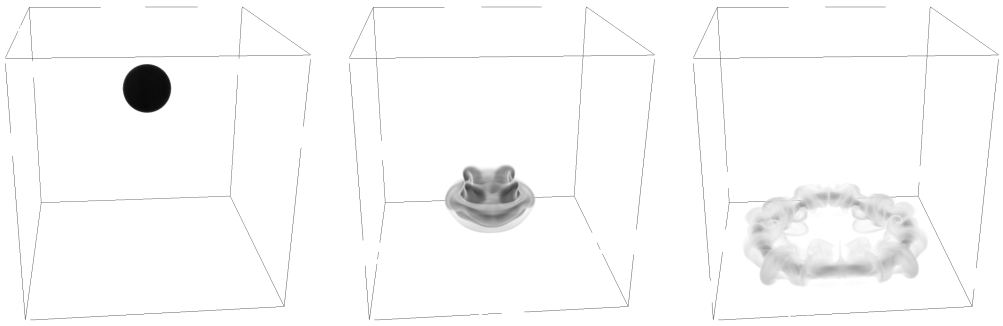

Figure 23: Visualization of a drop of high density smoke falling under gravity at the initial state (left), after nondimensional times of 367.88 (center) and 760.28 (right), where the nondimensional time is defined as $\frac{gr^2\Delta\rho t}{\mu L}$.

computational grid, with setting new operators and new linear solvers is small compared to the prohibitive computational cost of using a uniform grid of equivalent finest resolution throughout the domain.

However, while local mesh refinement remains valuable, dynamic re-meshing may not be most desirable in applications requiring (almost) static, dense regions of fine grid cells, as extra operations associated with dynamic grid adaptation would significantly increase the overall computational cost, without any significant reduction of the overall number of computational cells to consider. We have alleviated this issue by allowing the solver to store all possible data structures[5] (linear solvers, possible preconditioners, interpolation operators, etc.) in memory and use them as long as they are valid, i.e., as long as the grid is not modified.

Statistically steady physical problems align perfectly with the above solver features, since they require fixed regions of specified spatial resolution by nature, and the results need to be accumulated over a (very) large number of time steps to ensure their statistical convergence. In order to illustrate the capability of the solver to address such problems, we consider the fully developed turbulent flow in a superhydrophobic channel as previously simulated in [61, 62]. We use a computational domain of dimensions $6\delta \times 2\delta \times 3\delta$, where δ is half of the channel height, with periodic boundary conditions along the streamwise and spanwise directions, as illustrated in figure 24. The coordinates are chosen such that x points downstream, y is normal to the walls and z is in the spanwise direction. The flow is driven in the positive streamwise direction by a spatially uniform and constant force per unit mass $\boldsymbol{f} = f_x \boldsymbol{e}_x$. By analogy with canonical channel flows, we define the friction velocity $u_\tau = \sqrt{f_x \delta}$.

We consider gratings oriented parallel to the flow on both walls $y = \pm\delta$. The superhydrophobic nature of these surfaces enables them to entrap pockets of air, such that parts of he walls are replaced by a liquid-air interface, which is assumed to be shear-free. We assume the liquid-air interface to be and remain flat at all times (deflections of the interface are neglected). We therefore model the air-liquid interface regions using no-penetration, free-slip boundary conditions[6]

$$v|_{y=\pm\delta} = 0, \qquad \left.\frac{\partial u}{\partial y}\right|_{y=\pm\delta} = 0 \quad \text{and} \quad \left.\frac{\partial w}{\partial y}\right|_{y=\pm\delta} = 0, \tag{18}$$

while the rest of wall surfaces use no-slip boundary conditions,

$$\boldsymbol{u}|_{y=\pm\delta} = \boldsymbol{0}.$$

[5] For the linear solvers associated with the viscosity step, only diagonal terms are affected by a new value of Δt (see subsection 2.3). Therefore, only diagonal terms are updated when the computational grid is not modified.

[6] Note that $\left.\frac{\partial v}{\partial x}\right|_{y=\pm\delta} = \left.\frac{\partial v}{\partial z}\right|_{y=\pm\delta} = 0$ since $v|_{y=\pm\delta} = 0$ on the entire wall surfaces.

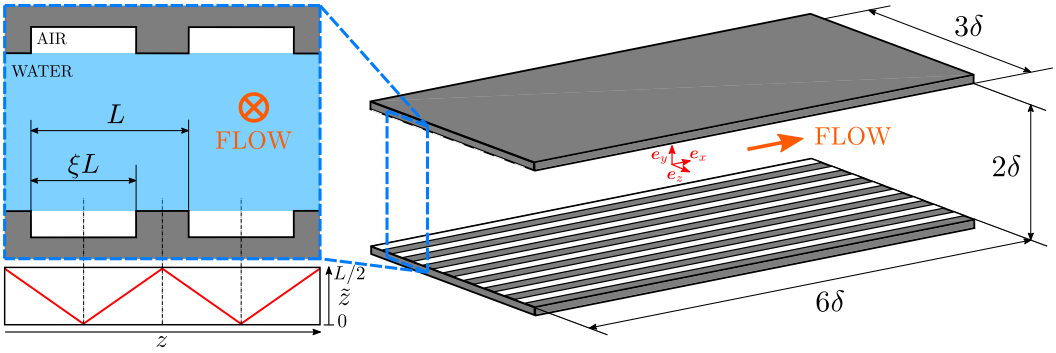

Figure 24: Schematic of the superhydrophobic surface and corresponding notations considered in section 4.5: we denote the pitch by L and the gas fraction by ξ. The variable $\tilde{z}$ is used to average over corresponding spanwise locations.

Eight longitudinal grates giving a gas fraction of 50% are used, i.e., the pitch length L is set to $3\delta/8$ and the gas fraction ξ is set to 0.5 (see figure 24). The fluid properties and other control parameters are set such that the canonical friction Reynolds number $Re_\tau = \dfrac{\rho u_\tau \delta}{\mu}$ is equal to 143. The computational domain is meshed with one single Octree of minimum level 7 and maximum level 9. This choice of macromesh results in computational cells with an aspect ratio so different from 1 that third-degree neighbor cells are required for the reliable construction of face-seeded Voronoi cells (see figure 2 for a two-dimensional illustration). Therefore, this simulation setup makes an extensive use of the capability to fetch third-degree ghost neighbor cells, which is enabled by the algorithms from section 2.6. The capability of the solver to address such problems even with stretched computational cells is discussed in Appendix A using a known analytical solution in the laminar case.

In order to ensure sufficient grid resolution for regions close to the no-slip parts of the walls, we define the level-set function[7]

$$\phi(\boldsymbol{x}) = -\text{dist}(\boldsymbol{x}, \text{no-slip region of the wall})$$

to be used with the refinement criterion (1) setting $K = 10$. This choice of K and maximum refinement level ensures that the walls are entirely covered by the finest computational cells over a thickness of 0.1δ. Except for the thickness of four grid cells layering the walls, the local grid resolution is equivalent to, or finer than, the resolution from [61, 62] everywhere. In [61, 62], a stretched grid was used with constant mesh size in the streamwise and spanwise directions while the cell thickness was distributed using a hyperbolic tangent profile in the wall-normal direction. The main difference between such a stretched grid and our Octree approach lies in the fact that the aspect ratio of our computational cells is constant. As cells get thinner when approaching the wall regions, they also get shorter and narrower: the spatial resolution close to the walls for the Octree grid is four times finer than for the stretched grid in the spanwise and streamwise directions, hence producing more accurate results in those directions than stretched grids do. This however significantly increases the total number of computational cells to be used in our approach, since we need more than 21.8×10^6 cells, as opposed to about 2.1×10^6 in the case of a stretched grid. Dynamic grid adaptation based on local vorticity is less useful in this example, since the background grid already captures enough details (γ is thus irrelevant in this case), which enables us to reduce time execution. The conjugate gradient method is used for solving the projection step along with an algebraic multigrid preconditioner (from the HYPRE distribution).

A thorough analysis of the analytical solution known in the laminar cases shows that the viscous

[7] Note that ϕ is negative everywhere so no interface is defined within the domain.

stress is singular at the edges of the walls transitioning between free-slip and no-slip boundary conditions (see Appendix A and references therein for more details). Early numerical tests revealed that setting the inner loop convergence criterion (see section 2.5) to ensure $\left\|\Phi^k - \Phi^{k-1}\right\|_\infty < \varepsilon_\Phi$ would fail because the (floating-value) Hodge variable Φ would grow unbounded in the cells layering these transition edges. However, the velocity field must be bounded everywhere, and therefore, so must be $\|\nabla\Phi\|_\infty$. As a matter of fact, setting the inner loop convergence criterion (see section 2.5) to ensure $\max_\Omega \left\|\nabla\Phi^k - \nabla\Phi^{k-1}\right\|_\infty < 10^{-6}U_\mathrm{b}$, wherein U_b is the mean, bulk velocity in the streamwise direction through the channel, resulted in fully controlled simulations. For most time steps, the solver required three inner iterations to converge (the value of the convergence measure for the first iterate, i.e., $\max_\Omega \left\|\nabla\Phi^1 - \nabla\Phi^0\right\|_\infty$, was observed to be of the order of $10^{-4}U_\mathrm{b}$).

The simulation is initialized to the known laminar solution and executes until flow instabilities amplify and a fully-developed turbulent state is eventually reached. The bulk Reynolds number

$$Re_\mathrm{b} = \frac{\rho U_\mathrm{b}\delta}{\mu}, \text{ where } U_\mathrm{b} = \frac{1}{6\delta^2}\int_{-\delta}^{\delta}\int_{-1.5\delta}^{1.5\delta} \boldsymbol{u}\cdot\boldsymbol{e}_x \,\mathrm{d}z\,\mathrm{d}y$$

and the nondimensional viscous forces from the no-slip regions of the walls

$$\boldsymbol{F}_{\text{wall, visc.}} = \frac{1}{\rho f_x\, 36\delta^3}\sum_{k_y=\{-1,1\}}\sum_{k_z=-4}^{3}\int_{-3\delta}^{3\delta}\int_{(k_z+\xi)L}^{(k_z+1)L} -k_y\left[\mu\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^\mathrm{T}\right)\cdot\boldsymbol{e}_y\right]\Big|_{y=k_y\delta}\mathrm{d}z\,\mathrm{d}x$$

are monitored over time. Their evolution is illustrated in figure 25.

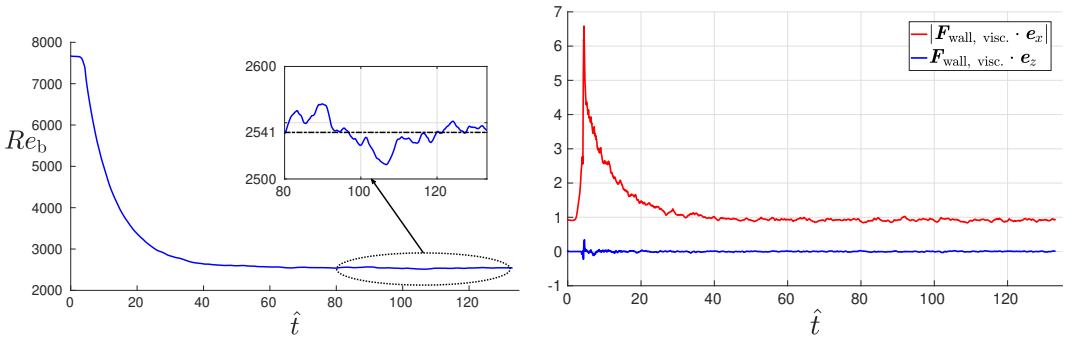

Figure 25: Macroscopic variables monitored over the course of the simulation of a turbulent flow through a superhydrophobic channel. Left: evolution of $Re_\mathrm{b} = \rho U_\mathrm{b}\delta/\mu$; right: evolution of the nondimensional viscous forces from the no-slip regions of the walls. In these graphs, the nondimensional time $\hat{t}$ is defined as $\hat{t} = \dfrac{u_\tau t}{\delta}$.

From the evolution of the monitored macroscopic quantities of interest illustrated in figure 25, we consider the time window from $\hat{t}_\mathrm{start} = \dfrac{u_\tau t}{\delta} = 80$ until the end of the simulation, $\hat{t}_\mathrm{end} = 133.1$, which we use for time-averaging results associated with the fully-developed regime (which corresponds to about 300,000 time steps). An illustrative snapshot of the simulation in this time window is presented in figure 27.

We obtain an average bulk Reynolds number of 2541; we also consider the time-and-slice-averaged velocity profile, i.e.,

$$\left\langle\frac{u}{u_\tau}\right\rangle_{x,z,t} = \frac{1}{18\delta^2\left(\hat{t}_\mathrm{end} - \hat{t}_\mathrm{start}\right)}\int_{\hat{t}_\mathrm{start}}^{\hat{t}_\mathrm{end}}\int_{-3\delta}^{3\delta}\int_{-1.5\delta}^{1.5\delta}\frac{u}{u_\tau}\,\mathrm{d}z\,\mathrm{d}x\,\mathrm{d}\hat{t}$$

as a function of y/δ, as well as the time-and-line-averaged velocity profile, i.e.,

$$\left\langle\frac{u}{u_\tau}\right\rangle_{x,t} = \frac{1}{6\delta\left(\hat{t}_\mathrm{end} - \hat{t}_\mathrm{start}\right)}\int_{\hat{t}_\mathrm{start}}^{\hat{t}_\mathrm{end}}\int_{-3\delta}^{3\delta}\frac{u}{u_\tau}\,\mathrm{d}x\,\mathrm{d}\hat{t},$$

which is built by also averaging corresponding locations over the air-interface and over the ridges in the spanwise direction. Formally, this results in a function of y/δ and of the grate-normalized spanwise coordinate $\tilde{z}$ defined as

$$\tilde{z} = \begin{cases} \left| (z \pmod L) - \dfrac{L\xi}{2} \right| & \text{if } z \pmod L \leq L\xi, \\ \dfrac{L}{2} - \left| (z \pmod L) - \dfrac{L(1+\xi)}{2} \right| & \text{otherwise,} \end{cases} \tag{19}$$

(see illustration in figure 24). These time-averaged velocity profiles are illustrated in figure 26.

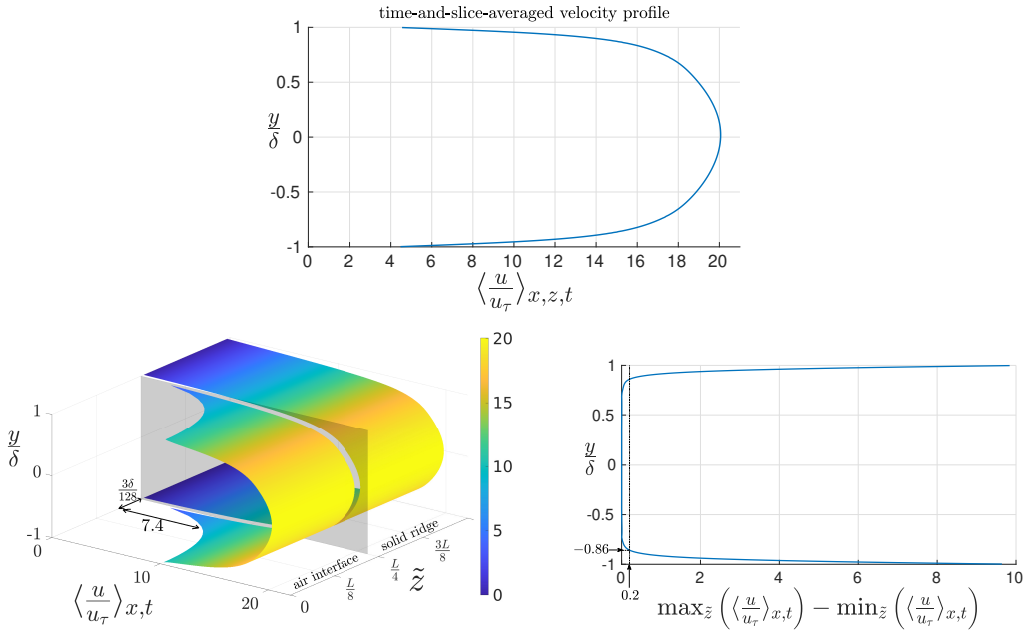

Figure 26: Top: time-and-slice-averaged velocity profile. Bottom left: time-and-line-averaged velocity profiles. Notice the sharpness of the transition between no-slip and free-slip line-averaged profiles. A significant portion of the transition (74%) takes place over a distance of $\frac{3\delta}{128}$ in the spanwise direction. While this distance corresponds to the width of one single computational cell in the stretched grid approach of [62], our octree grid uses 4 narrower computational cells over that region. Bottom right: illustration of how the time-and-line-averaged profiles become $\tilde{z}$-independent far enough from the walls; in this case, time-and-line-averaged velocity profiles are essentially all equivalent (within 1% of the mean velocity to be found in the center of the channel) farther than 0.14δ from the walls.

As illustrated in figure 26, the mean fluid velocity at the air interface can reach up to 50% of the maximum mean velocity (found at the center of the channel). This figure also illustrates how sharp the change is: 74% of the variation from the no-slip ridge to the maximum air interface velocity (found above the center line of the interface) occurs over a distance of $3\delta/128$ in the spanwise direction, which corresponds to 4 computational cells in our setup (versus 1 cell in [62]).

When averaging velocity profiles across entire planar sections of the channel, the existence of such free-slip regions results in a nonzero slip velocity U_s at the wall. This slip velocity is a quantity of primary relevance in the context of Navier's slip model, along with the slip length b which relates the slip velocity to the mean wall shear via $U_s = \pm b \left. \dfrac{\partial}{\partial y} \langle u \rangle_{x,z,t} \right|_{y=\mp\delta}$. The slip parameters U_s and b were evaluated by least-square fitting the linear profiles $\dfrac{U_s}{u_\tau}\left(1 + \dfrac{\delta \pm y}{b}\right)$ (+ and − correspond

to bottom and top walls, respectively) to our results for $\left\langle \frac{u}{u_\tau} \right\rangle_{x,z,t}$ over the 5 finest grid cells layering the walls, which corresponds to a thickness of about $2.8\delta_\tau$, where $\delta_\tau = \frac{\mu}{\rho u_\tau}$ is the viscous lengthscale. The two sets of fitting parameters yield $b = (0.0302 \pm 0.0002)\,\delta = (4.320 \pm 0.025)\,\delta_\tau$ and $U_s = (4.255 \pm 0.040)\,u_\tau$.

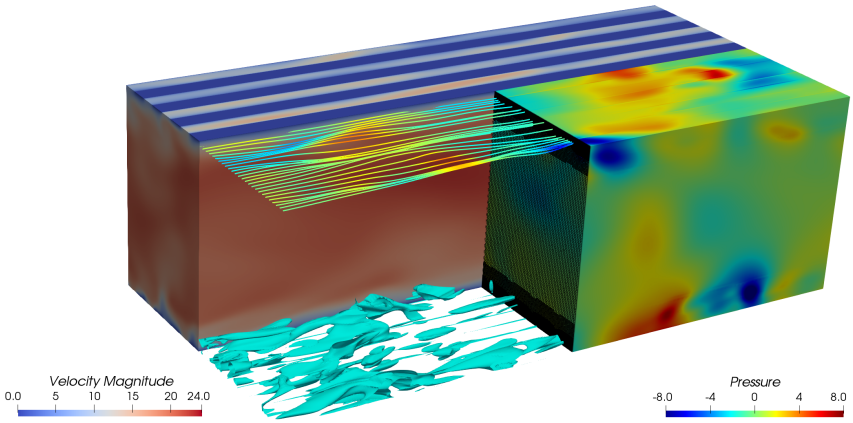

Figure 27: Visualization of a snapshot for the simulation of the turbulent superhydrophobic channel flow. The half of the domain corresponding to $z < 0$ is colored by $\frac{\|\boldsymbol{u}\|}{u_\tau}$; the quarter of the domain corresponding to $z > 0$ and $x > 0$ is colored by $\frac{p}{\rho u_\tau^2}$. A slice of the computational grid, streamlines and isocontours of $\lambda_2 = -0.3\left(\frac{U_\mathrm{b}}{\delta}\right)^2$ (using the λ_2-criterion from [36]) are also shown.

Our setup value of Re_τ was chosen for comparison purposes with one of the simulations from [62, 61], which reports a (mean) value of $Re_\tau = 143$ using a simulation setup enforcing a (constant) mass flow corresponding to $Re_\mathrm{b} = 2,800$ for the same channel geometry. Under these conditions, [62, 61] reports $b = 0.0366\,\delta = 5.17\,\delta_\tau$ and $U_s = 5.26\,u_\tau$. Therefore, when compared to those results, our simulation leads to a reduced flow rate for a comparable driving force (about 10% less) and to a smaller slip velocity as well as a smaller slip length. Besides the difference in simulation setup (constant driving force as opposed to constant mass flow rate), such deviations may also originate from numerical and/or modeling differences to be found between the two approaches. To assess this, we also compare our results to the simulations of [66], who used a lattice-Boltzmann method. We select their simulation whose value of L/δ_τ is closest to ours, since [62] established that L/δ_τ is the single most important parameter that determines slip length (at fixed gas fraction). We therefore compare our simulation, which has $L/\delta_\tau = 53.6$, to the $L/\delta_\tau = 56.2$ case of [66], who found $b/\delta_\tau = 4.23$. This value is appreciably smaller than the result of 5.17 of [62], but matches closely our $b/\delta_\tau = 4.32$.

In terms of modeling, [62, 61] opted for a simplified wall-treatment for the spanwise velocity component w, by setting $w|_{y=\pm\delta} = 0$ instead of the stress-free condition (18) above the free-slip wall regions. Enforcing $w|_{y=\pm\delta} = 0$ above the air pockets does not rely on physical grounds, and may result in simplified near-wall flow structures that artificially promote streamwise velocity. Indeed, this simplified condition results in $\frac{\partial w}{\partial z} = 0$, which in turn simplifies the incompressibility condition, at the walls, into $\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$. At the wall-located air interfaces, this latter equation stands as an artificial constraint, within z-orthogonal planes, enabling transfer of kinetic energy only between wall-normal and streamwise velocity components. This constraint could lead, in turn, to an overestimation of the slip velocity and/or of the total mass flow across the channel, which

could help explain the difference between the results of [62] and those of subsequent simulations.

In terms of numerical methods, we emphasize that the use of a semi-Lagrangian scheme for the advection terms comes with a significant amount of numerical dissipation, which may in turn lead to overestimated viscous dissipation. While this cannot be excluded, we also want to point out that our simulation setup makes use of a spatial resolution that is 4 times finer than the resolution from [62], in both the streamwise and spanwise directions. Firstly, such a fine resolution in the streamwise direction is expected to alleviate the numerical dissipation associated with our advection scheme. Secondly, such a resolution in the spanwise direction may actually stand as a requirement in order to capture the sharp variation in velocity profiles between free-slip and no-slip wall regions. Indeed, figure 26 illustrates that $\frac{\partial}{\partial z}\langle u\rangle_{x,t}\Big|_{y=\pm\delta}$ is the largest near the boundaries transitioning from free-slip to no-slip regions; therefore, using a coarse spanwise resolution in that area may lead to an underestimation of the overall viscous dissipation.

In order to quantify the relative importance of this term, we estimate its contribution to the viscous dissipation taking place in near-wall layers and compare it to the contribution of the mean, slice-averaged streamwise shear term (i.e., as if we were dealing with a regular channel). Assuming that $\frac{\partial}{\partial z}\langle u\rangle_{x,t}\Big|_{y=\pm\delta}$ is not negligible within bands of $3\delta/128$ around solid ridges only (in our case, we have 16 such bands on either wall), our comparative estimate is

$$\frac{6\delta\,\frac{16\times 3\delta}{128}\mu\left(\frac{\partial}{\partial z}\langle u\rangle_{x,t}\Big|_{y=-\delta}\right)^2}{6\delta\,3\delta\,\mu\left(\frac{\partial}{\partial y}\langle u\rangle_{x,z,t}\Big|_{y=-\delta}\right)^2}\approx\frac{16}{128}\frac{\left(\frac{7.4u_\tau}{3\delta/128}\right)^2}{\left(\frac{U_s}{b}\right)^2}=62.8\%,$$

where we estimated $\frac{\partial}{\partial z}\langle u\rangle_{x,t}\Big|_{y=-\delta}=\frac{7.4u_\tau}{3\delta/128}$ based on the results illustrated in figure 26 to produce a fair measure in comparison with the grid resolution from [62], although $\frac{\partial}{\partial z}\langle u\rangle_{x,t}\Big|_{y=-\delta}$ becomes almost twice as large as we approach the edge in our computational setting. Although wall viscous shear dissipation may be smaller than the overall (bulk) turbulent dissipation, we expect it to be non-negligible nonetheless, in particular when considering a relatively low friction Reynolds number as it is the case here[8]; in this context, the above comparative estimation indicates that spanwise wall shear, though not evenly distributed on the walls, is not a negligible factor to the overall viscous dissipation.

[8] In fact, if we consider an equivalent canonical channel with a mean, streamwise wall shear of U_s/b that we assume constant over layers of at least $3\frac{\mu}{\rho u_\tau}$, the viscous dissipation in these layers amounts for

$$2\times 3\frac{\mu}{\rho u_\tau}\times 6\delta\times 3\delta\times\mu\left(\frac{U_s}{b}\right)^2\simeq 105\,\rho u_\tau^3\delta^2$$

which represents about 1/6 of the energy injection rate $36\delta^3\rho f_x U_\mathrm{b}$ in our simulation setup.

5. Conclusion

We have described a Navier-Stokes solver for simulating incompressible flows in irregular domains on a forest of Octrees in a distributed computing framework. The parallel implementation of the solver requires the ability to access second- (or third-)degree cell neighbors, which led to the need for an expanded ghost layer of cells. We have introduced an algorithm to address that computational challenge on distributed forest of octree grids. We also have introduced parallel algorithms for the unambiguous definition and synchronization of global faces indices as required in a standard MAC arrangement. The performance of these individual algorithms has been assessed in terms of strong and weak scaling analyses. The strong scaling behavior of the entire solver has been verified up to more than 32,000 cores using a problem on a grid of 6.1×10^8 grid cells. The performance of the solver has been assessed on several large-scale three-dimensional problems: accurate results for the flow past sphere at various Reynolds numbers have been shown, several illustrations of the capabilities of our adaptive refinement approach have been provided and a simulation of the turbulent flow through a superhydrophobic channel has been performed with unparalleled spanwise (and streamwise) spatial resolution over regions of interest. When flow structures have a limited lifetime and/or are bounded to relatively small regions in the computational domain, our adaptive grid refinement approach was shown to successfully simulate the problems of interest using only a few percent of the number of grid cells that a uniform grid of equivalent finest resolution would require. The encapsulation of such a feature in a distributed computing framework allows for very-large scale simulations to be considered tractable from a computational standpoint and, therefore, to address increasingly complex multiscale and/or multiphysics problems.

Acknowledgments

This research was funded by ONR N00014-11-1-0027. In addition, F. Temprano-Coleto and P. Luzzatto-Fegiz were partially supported by ARO MURI W911NF-17-1-0306. F. J. Peaudecerf acknowledges funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 798411. This work was also partially supported by the California NanoSystems Institute at UC Santa Barbara through a Challenge Grant. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575. The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC and visualization resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu. In addition, use was made of computational facilities purchased with funds from the National Science Foundation (CNS-1725797) and administered by the Center for Scientific Computing (CSC). The CSC is supported by the California NanoSystems Institute and the Materials Research Science and Engineering Center (MRSEC; NSF DMR 1720256) at UC Santa Barbara.

References

[1] S. Aluru and F. E. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In Proceedings of the Fourth International Conference on High-Performance Computing, HIPC '97, pages 230–, Washington, DC, USA, 1997. IEEE Computer Society.

[2] E. J. Avital, N. D. Sandham, and K. H. Luo. Stretched cartesian grids for solution of the incompressible Navier-Stokes equations. International Journal for Numerical Methods in Fluids, 33(6):897–918, 2000.

[3] P. Bagchi, M. Y. Ha, and S. Balachandar. Direct numerical simulation of flow and heat transfer from a sphere in a uniform cross-flow. Journal of Fluids Engineering, 123(2):347–358, 2001.

[4] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Users Manual. Argonne National Laboratory, 2012.

[5] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc web page, 2014.

[6] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In B. user Press, editor, Modern Software Tools in Scientific Computing, pages 163—202, 2012.

[7] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. ACM Transactions on Mathematical Software, 38(2):14:1–14:28, 2011.

[8] W. Bangerth, R. Hartmann, and G. Kanschat. Deal.II - a general-purpose object-oriented finite element library. ACM Trans. Math. Software, 33, 2007.

[9] J. A. Benek, J. Steger, and F. C. Dougherty. A flexible grid embedding technique with applications to the Euler equations. 6th Computational Fluid Dynamics Conference, AIAA, 373–382., 1983.

[10] M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. J. Comput. Phys., 82:64–84, 1989.

[11] M. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. J. Comput. Phys., 53:484–512, 1984.

[12] D. Brown, R. Cortez, and M. Minion. Accurate projection methods for the incompressible Navier-Stokes equations. J. Comput. Phys., 168:464–499, 2001.

[13] H.-J. Bungartz, M. Mehl, and T. Weinzierl. A parallel adaptive Cartesian PDE solver using space–filling curves. Euro-Par 2006 Parallel Processing, pages 1064–1074, 2006.

[14] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong. Scalable adaptive mantle convection simulation on petascale supercomputers. In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC '08, pages 62:1–62:15, Piscataway, NJ, USA, 2008. IEEE Press.

[15] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. SIAM Journal on Scientific Computing, 33(3):1103–1133, 2011.

[16] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.

[17] J.-I. Choi, R. C. Oberoi, J. R. Edwards, and J. A. Rosati. An immersed boundary method for complex incompressible flows. Journal of Computational Physics, 224(2):757 – 784, 2007.

[18] A. J. Chorin. Numerical solution of the Navier-Stokes equations. Mathematics of computation, 22(104):745–762, 1968.

[19] W. J. Coirier. An adaptively-refined, cartesian, cell-based scheme for the Euler and Navier-Stokes equations. Technical report, NASA, 1994.

[20] G. Constantinescu and K. Squires. LES and DES investigations of turbulent flow over a sphere at Re = 10,000. Flow, Turbulence and Combustion, 70(1-4):267–298, 2003.

[21] R. Courant, E. Isaacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. Comm. Pure Appl. Math., 5:243–255, 1952.

[22] D. DeZeeuw and K. G. Powell. An adaptively refined cartesian mesh solver for the Euler equations. Journal of Computational Physics, 104(1):56 – 68, 1993.

[23] R. Egan and F. Gibou. Numerical simulations of incompressible two-phase flows on distributed quad-/oc-tree grids with fully sharp interface treatment. In preparation.

[24] R. E. English, L. Qiu, Y. Yu, and R. Fedkiw. An adaptive discretization of incompressible flow using a multitude of moving cartesian grids. Journal of Computational Physics, 254:107 – 154, 2013.

[25] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof. Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. Journal of Computational Physics, 161(1):35 – 60, 2000.

[26] J. H. Ferziger and M. Peric. Computational Methods for Fluid Dynamics. Springer-Verlag Berlin Heidelberg, 2002.

[27] R. Finkel and J. Bentley. Quad trees a data structure for retrieval on composite keys. Acta Informatica, 4(1):1–9, 1974.

[28] V. Girault, B. Riviere, and M. F. Wheeler. A discontinuous Galerkin method with nonoverlapping domain decomposition for the Stokes and Navier-Stokes problems. Mathematics of Computation, 74:53–84, 2005.

[29] M. Griebel and G. W. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. Parallel Computing, 25:827–843, 1999.

[30] A. Guittet, M. Theillard, and F. Gibou. A stable projection method for the incompressible Navier-Stokes equations on arbitrary geometries and adaptive quad/octrees. Journal of Computational Physics, 2015.

[31] F. Harlow and J. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. Phys. Fluids, 8:2182–2189, 1965.

[32] L. H. Howell and J. B. Bell. An adaptive mesh projection method for viscous incompressible flow. SIAM Journal on Scientific Computing, 18(4):996–1013, 1997.

[33] J. C. Hunt, A. Wray, and P. Moin. Eddies, stream, and convergence zones in turbulent flows. Center for Turbulence Research Report, CTR-S88, 1988.

[34] T. Isaac, C. Burstedde, and O. Ghattas. Low-cost parallel algorithms for 2:1 octree balance. In Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, pages 426–437, 2012.

[35] T. Isaac, C. Burstedde, L. C. Wilcox, and O. Ghattas. Recursive algorithms for distributed forests of octrees. SIAM Journal on Scientific Computing, 37(5):C497–C531, 2015.

[36] J. Jeong and F. Hussain. On the identification of a vortex. Journal of Fluid Mechanics, 285:69–94, 1995.

[37] T. A. Johnson and V. C. Patel. Flow past a sphere up to a Reynolds number of 300. Journal of Fluid Mechanics, 378:19–70, 1 1999.

[38] S. Karman. SPLITFLOW: A 3D unstructured cartesian/prismatic grid CFD code for complex geometries. In 33rd Aerospace Sciences Meeting and Exhibit, 1995.

[39] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. Journal of Parallel Distributed Computing, 48(1):71–95, Jan. 1998.

[40] A. Khokhlov. Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations. Journal of Computational Physics, 143(2):519 – 543, 1998.

[41] J. Kim and H. Choi. An immersed-boundary finite-volume method for simulation of heat transfer in complex geometries. KSME International Journal, 18(6):1026–1035, 2004.

[42] J. Kim, D. Kim, and H. Choi. An immersed-boundary finite-volume method for simulations of flow in complex geometries. Journal of Computational Physics, 171(1):132 – 150, 2001.

[43] M.-C. Lai and C. S. Peskin. An immersed boundary method with formal second-order accuracy and reduced numerical viscosity. Journal of Computational Physics, 160(2):705 – 719, 2000.

[44] Z. Li and T. Lu. Singularities and treatments of elliptic boundary value problems. Mathematical and Computer Modelling, 31(8):97 – 145, 2000.

[45] F. Losasso, R. Fedkiw, and S. Osher. Spatially Adaptive Techniques for Level Set Methods and Incompressible Flow. Computers and Fluids, 35:995–1010, 2006.

[46] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. ACM Trans. Graph. (SIGGRAPH Proc.), pages 457–462, 2004.

[47] S. Marella, S. Krishnan, H. Liu, and H. Udaykumar. Sharp interface Cartesian grid method I: An easily implemented technique for 3D moving boundary computations. Journal of Computational Physics, 210:1–31, Nov. 2005.

[48] D. Meagher. Geometric modeling using octree encoding. Computer Graphics and Image Processing, 19(2):129 – 147, 1982.

[49] J. Melton. Automated three-dimensional cartesian grid generation and euler flow solutions for arbitrary geometries. PhD thesis, Univerty of California, Davis, 1996.

[50] J. Melton, M. Berger, M. Aftosmis, and M. Wong. 3d applications of a cartesian grid Euler method. In 33rd Aerospace Sciences Meeting and Exhibit, 1995.

[51] J. Melton, F. Enomoto, and M. Berger. 3D automatic cartesian grid generation for Euler flows. In 11th Computational Fluid Dynamics Conference, AIAA-93-3386-CP, 1993.

[52] C. Min and F. Gibou. A second order accurate projection method for the incompressible Navier-Stokes equation on non-graded adaptive grids. J. Comput. Phys., 219:912–929, 2006.

[53] C. Min and F. Gibou. Geometric integration over irregular domains with application to level set methods. J. Comput. Phys., 226:1432–1443, 2007.

[54] C. Min and F. Gibou. A second order accurate level set method on non-graded adaptive Cartesian grids. J. Comput. Phys., 225:300–321, 2007.

[55] M. Mirzadeh, A. Guittet, C. Burstedde, and F. Gibou. Parallel level-set methods on adaptive tree-based grids. Journal of Computational Physics, 322:345–364, 2016.

[56] I. D. Mishev. Finite volume methods on Voronoi meshes. Numerical Methods for Partial Differential Equations, 14(2):193–212, 1998.

[57] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, and A. von Loebbecke. A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. Journal of Computational Physics, 227(10):4825 – 4852, 2008.

[58] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo. Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA. http://arxiv.org/abs/1511.01561, 2015.

[59] Y. T. Ng, C. Min, and F. Gibou. An efficient fluid–solid coupling algorithm for single-phase flows. Journal of Computational Physics, 228(23):8807–8829, Dec. 2009.

[60] S. Osher and J. Sethian. Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. Journal of Computational Physics, 79:12–49, 1988.

[61] H. Park. A numerical study of the effects of superhydrophobic surfaces on skin-friction drag reduction in wall-bounded shear flows. PhD thesis, 2015.

[62] H. Park, H. Park, and J. Kim. A numerical study of the effects of superhydrophobic surface on skin-friction drag in turbulent channel flow. Physics of Fluids, 25(11):110815, 2013.

[63] C. Peskin. Flow patterns around heart valves: A numerical method. J. Comput. Phys., 10:252–271, 1972.

[64] J. R. Philip. Flows satisfying mixed no-slip and no-shear conditions. Zeitschrift für angewandte Mathematik und Physik ZAMP, 23(3):353–372, 1972.

[65] S. Popinet. Gerris: A tree-based adaptive solver for the incompressible Euler equations in complex geometries. J. Comput. Phys., 190:572–600, 2003.

[66] A. Rastegari and R. Akhavan. On the mechanism of turbulent drag reduction with super-hydrophobic surfaces. Journal of Fluid Mechanics, 773:R4, 2015.

[67] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas. An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, page 5. ACM, 2015.

[68] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, and G. Biros. Dendro: Parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees. In International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008., 2008.

[69] R. S. Sampath and G. Biros. A parallel geometric multigrid method for finite elements on octree meshes. SIAM Journal on Scientific Computing, 32(3):1361–1392, May 2010.

[70] K. Shahbazi, P. F. Fischer, and C. R. Ethier. A high-order discontinuous Galerkin method for the unsteady incompressible Navier-Stokes equations. Journal of Computational Physics, 222(1):391 – 407, 2007.

[71] F. Sousa, C. Lages, J. Ansoni, A. Castelo, and A. Simao. A finite difference method with meshless interpolation for incompressible flows in non-graded tree-based grids. Journal of Computational Physics, 396:848 – 866, 2019.

[72] J. R. Stewart and H. C. Edwards. A framework approach for developing parallel adaptive multiphysics applications. Finite Elements in Analysis and Design, 40(12):1599–1617, 2004.

[73] J. Strain. Semi-Lagrangian methods for level set equations. Journal of Computational Physics, 151:498–533, 1999.

[74] M. Sussman, A. Almgren, J. Bell, P. Colella, L. Howell, and M. Welcome. An adaptive level set approach for incompressible two-phase flows. J. Comput. Phys., 148:81–124, 1999.

[75] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. J. Comput. Phys., 114:146–159, 1994.

[76] C. J. Teo and B. C. Khoo. Analysis of Stokes flow in microchannels with superhydrophobic surfaces containing a periodic array of micro-grooves. Microfluidics and nanofluidics, 7(3):353, 2009.

[77] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkens-Diehr. Xsede: Accelerating scientific discovery. Computing in Science and Engineering, 16(5):62–74, 2014.

[78] T. Tu, D. R. O'Hallaron, and O. Ghattas. Scalable parallel octree meshing for terascale applications. In Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, SC '05, pages 4–, Washington, DC, USA, 2005. IEEE Computer Society.

[79] M. Vinokur. On one-dimensional stretching functions for finite difference calculations. Journal of Computational Physics, 50(2):215–234, 1983.

[80] T. Weinzierl and M. Mehl. Peano—a traversal and storage scheme for octree-like adaptive Cartesian multiscale grids. SIAM Journal on Scientific Computing, 33(5):2732–2760, Oct. 2011.

[81] D. Xiu and G. Karniadakis. A semi-Lagrangian high-order method for Navier-Stokes equations. J. Comput. Phys, 172:658–684, 2001.

Appendix A. Validation test in laminar cases superhydrophobic channel flows

Given the numerical difficulties typically associated with this kind of mixed boundary-value problems [44], we assess the performance of the solver for the simpler *laminar* steady case, in which the flow is unidirectional and exact solutions exist (see [64, 76]). In this context, we consider longitudinal gratings, i.e., aligned with the flow direction, on both the upper and lower walls. The analytical solution can be expressed as the following series

$$\boldsymbol{u}_{\text{exact}}(y,\tilde{z}) = \frac{\mu Re_\tau^2}{\rho\delta}\left\{\frac{1}{2}\left[1-\left(\frac{y}{\delta}\right)^2\right] + c_0 + \sum_{n=1}^{\infty} c_n \frac{\cosh(2\pi n y/L)}{\cosh(2\pi n\delta/L)}\cos\left(\frac{2\pi n\tilde{z}}{L}\right)\right\}\boldsymbol{e}_x,$$

where $Re_\tau = \dfrac{\rho\delta\sqrt{f_x\delta}}{\mu}$ is the canonical friction Reynolds number, L is the pitch (that is the distance between consecutive gratings), and $\tilde{z}\in[0, L/2]$ is the normalized spanwise coordinate that parametrizes each periodic unit as defined in (19) (see figure 24 for a detailed illustration of all parameters).

Due to the mixed boundary conditions, the coefficients c_n must be determined numerically by enforcing the dual cosine series conditions [76]

$$g_{\text{no slip}}\left(\tilde{z}; c_0, c_1, \ldots\right) = c_0 + \sum_{n=1}^{\infty} c_n \cos\left(\frac{2\pi n\tilde{z}}{L}\right) = 0, \qquad \forall\tilde{z}\in\left[\frac{\xi L}{2}, \frac{L}{2}\right]$$

$$g_{\text{free slip}}\left(\tilde{z}; c_0, c_1, \ldots\right) = \sum_{n=1}^{\infty} c_n n \tanh\left(\frac{2\pi n\delta}{L}\right)\cos\left(\frac{2\pi n\tilde{z}}{L}\right) - \frac{L}{2\pi\delta} = 0, \qquad \forall\tilde{z}\in\left[0, \frac{\xi L}{2}\right[.$$

where $0 < \xi < 1$ is the gas fraction. We truncate the above series up to N terms and solve the (dense) $N \times N$ linear system derived from the resulting algebraic conditions

$$\int_0^{\frac{\xi L}{2}} g_{\text{free slip}}\left(\tilde{z}; c_0, c_1, \ldots, c_{N-1}\right) \cos\left(\frac{2\pi k \tilde{z}}{L}\right) \mathrm{d}\tilde{z} + \int_{\frac{\xi L}{2}}^{\frac{L}{2}} g_{\text{no slip}}\left(\tilde{z}; c_0, c_1, \ldots, c_{N-1}\right) \cos\left(\frac{2\pi k \tilde{z}}{L}\right) \mathrm{d}\tilde{z} = 0, \qquad \forall k \in \{0, 1, \ldots, N-1\}\,.$$

Even in this simplified laminar case, the velocity gradient components $\frac{\partial u}{\partial y}$ and $\frac{\partial u}{\partial z}$ display singularities at the edges between the no-slip and free-slip wall regions that may compromise the point-wise convergence of the numerical solution. We thus explore the performance of the solver, setting simulations of increasing spatial resolution for $Re_\tau = 10$, using 2 grates with pitch $L = \delta$ and gas fraction $\xi = 0.875$. We use $N = 2500$ terms in the above series, which ensures that all the coefficients in the exact solution are computed to machine precision for this particular setup.

In order to assess the validity of the face-seeded Voronoi diagrams associated with stretched computational cells, we also investigate how the accuracy is affected by the aspect ratio of the computational cells by replicating the same simulation runs in two macromeshes of different aspect ratios. This is especially relevant to anticipate the validity of our results in the turbulent simulations, since in those cases the cells must be stretched in the streamwise direction to result in a feasible computational cost. The solver is then executed until a steady state is reached, and the resulting numerical errors for the streamwise velocity component are measured. We present them in Table A.7.

	u/u_τ							
	Domain dimensions: $\delta \times 2\delta \times 2\delta$ Macromesh: $1 \times 2 \times 2$ root trees Cubic cells ($\Delta x/\Delta y = \Delta x/\Delta z = 1$)				Domain dimensions: $6\delta \times 2\delta \times 2\delta$ Macromesh: $1 \times 1 \times 1$ root tree Stretched cells ($\Delta x/\Delta y = \Delta x/\Delta z = 3$)			
levels	L^1 error	order	L^∞ error	order	L^1 error	order	L^∞ error	order
4/6	$9.63 \cdot 10^{-2}$	-	$3.09 \cdot 10^{-1}$	-	$2.02 \cdot 10^{-1}$	-	$4.67 \cdot 10^{-1}$	-
5/7	$2.67 \cdot 10^{-2}$	1.85	$1.69 \cdot 10^{-1}$	0.87	$9.75 \cdot 10^{-2}$	1.05	$2.99 \cdot 10^{-1}$	0.64
6/8	$6.20 \cdot 10^{-3}$	2.11	$1.39 \cdot 10^{-1}$	0.29	$3.84 \cdot 10^{-2}$	1.34	$1.67 \cdot 10^{-1}$	0.84

Table A.7: Convergence of the solver for the case of laminar flow over a superhydrophobic surface, with $Re_\tau = 10$. The pitch is $L = \delta$ and the gas fraction is $\xi = 0.875$.

As expected from Section 4.1, the accuracy is close to, or even exceeds, first order in the L^1 norm[9] for all cases. Moreover, the errors are comparable for similar Δy and Δz between cubic cells and stretched cells: we set $\Delta y = \Delta z$ in both cases, but we use twice as many root trees along y and z in the case of cubic cells, which effectively doubles the wall-normal and spanwise resolutions compared to stretched cells.

On the other hand, the L^∞ error does not display a clean first-order convergence. Indeed, a closer analysis of the spatial distribution of the error reveals that, unsurprisingly, its maximum is located at the transitions between no-slip and free-slip regions at the walls, and that the convergence in these areas is slower than in the rest of the domain. This drop in the convergence rate is expected in mixed boundary-value problems [44], although we still observe that in all cases the maximum error is monotonically decreasing with refinement.

[9] Here, the L^1 error is already normalized with the domain volume for a straightforward comparison with the L^∞ error. Specifically, we define the L^1 error as the discrete equivalent of $\int_\Omega |u_{\text{num.}} - u_{\text{exact}}| \,\mathrm{d}\Omega \,/\, \mathrm{vol}(\Omega)$.