POLITECNICO DI TORINO
Repository ISTITUZIONALE

FluTAS: A GPU-accelerated finite difference code for multiphase flows

Original
FluTAS: A GPU-accelerated finite difference code for multiphase flows / Crialesi-Esposito, Marco; Scapin, Nicolò; Demou, Andreas D.; Rosti, Marco Edoardo; Costa, Pedro; Spiga, Filippo; Brandt, Luca. - In: COMPUTER PHYSICS COMMUNICATIONS. - ISSN 0010-4655. - 284:(2023). [10.1016/j.cpc.2022.108602]

Availability:
This version is available at: 11583/2990233 since: 2024-07-02T14:34:51Z

Publisher:
Elsevier

Published
DOI:10.1016/j.cpc.2022.108602

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
Elsevier postprint/Author's Accepted Manuscript

© 2023. This manuscript version is made available under the CC-BY-NC-ND 4.0 license
http://creativecommons.org/licenses/by-nc-nd/4.0/.The final authenticated version is available online at:
http://dx.doi.org/10.1016/j.cpc.2022.108602

(Article begins on next page)

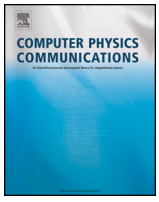
30 May 2026

Computer Physics Communications 284 (2023) 108602

Contents lists available at ScienceDirect

Computer Physics Communications

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



FluTAS: A GPU-accelerated finite difference code for multiphase flows ☆,☆☆

Marco Crialesi-Esposito [a,b,*,1], Nicolò Scapin [a,1], Andreas D. Demou [c], Marco Edoardo Rosti [d], Pedro Costa [e], Filippo Spiga [f], Luca Brandt [a,g]

[a] Department of Engineering Mechanics, Royal Institute of Technology (KTH), Stockholm, Sweden,
[b] Istituto Nazionale di Fisica Nucleare, Sezione di Torino, Italy
[c] Computation-based Science and Technology Research Center, The Cyprus Institute, 20 Konstantinou Kavafi Street, Aglantzia, Nicosia 2121, Cyprus
[d] Complex Fluids and Flows Unit, Okinawa Institute of Science and Technology Graduate University (OIST), 1919-1 Tancha, Onna-son, Okinawa 904-0495, Japan
[e] Faculty of Industrial Engineering, Mechanical Engineering and Computer Science, University of Iceland, Hjardarhagi 2-6, 107 Reykjavík, Iceland
[f] NVIDIA Ltd, Cambridge UK
[g] Department of Energy and Process Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

ARTICLE INFO

Article history:
Received 20 April 2022
Received in revised form 30 September 2022
Accepted 16 November 2022
Available online 24 November 2022

Keywords:
Multiphase flows
Volume-of-fluid method
Turbulence in multiphase flows
High-performance computing
OpenACC directives

ABSTRACT

We present the Fluid Transport Accelerated Solver, FluTAS, a scalable GPU code for multiphase flows with thermal effects. The code solves the incompressible Navier-Stokes equation for two-fluid systems, with a direct FFT-based Poisson solver for the pressure equation. The interface between the two fluids is represented with the Volume of Fluid (VoF) method, which is mass conserving and well suited for complex flows thanks to its capacity of handling topological changes. The energy equation is explicitly solved and coupled with the momentum equation through the Boussinesq approximation. The code is conceived in a modular fashion so that different numerical methods can be used independently, the existing routines can be modified, and new ones can be included in a straightforward and sustainable manner. FluTAS is written in modern Fortran and parallelized using hybrid MPI/OpenMP in the CPU-only version and accelerated with OpenACC directives in the GPU implementation. We present different benchmarks to validate the code, and two large-scale simulations of fundamental interest in turbulent multiphase flows: isothermal emulsions in HIT and two-layer Rayleigh-Bénard convection. FluTAS is distributed through a MIT license and arises from a collaborative effort of several scientists, aiming to become a flexible tool to study complex multiphase flows.

Program summary
Program Title: : Fluid Transport Accelerated Solver, FluTAS.
CPC Library link to program files: https://doi.org/10.17632/tp6k8wky8m.1
Developer's repository link: https://github.com/Multiphysics-Flow-Solvers/FluTAS.git.
Licensing provisions: MIT License.
Programming language: Fortran 90, parallelized using MPI and slab/pencil decomposition, GPU accelerated using OpenACC directives.
External libraries/routines: FFTW, cuFFT.
Nature of problem: FluTAS is a GPU-accelerated numerical code tailored to perform interface resolved simulations of incompressible multiphase flows, optionally with heat transfer. The code combines a standard pressure correction algorithm with an algebraic volume of fluid method, MTHINC [1].

☆ The review of this paper was arranged by Prof. N.S. Scott.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author at: Istituto Nazionale di Fisica Nucleare, Sezione di Torino, Italy.
E-mail addresses: marcoce@kth.se (M. Crialesi-Esposito), nicolos@mech.kth.se (N. Scapin), demou@mech.kth.se (A.D. Demou), marco.rosti@oist.jp (M.E. Rosti), pcosta@hi.is (P. Costa), fspiga@nvidia.com (F. Spiga), luca@mech.kth.se (L. Brandt).
[1] M.C-E. and N.S. contributed equally to this work.

https://doi.org/10.1016/j.cpc.2022.108602
0010-4655/© 2022 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Solution method: the code employs a second-order-finite difference discretization and solves the two-fluid Navier-Stokes equation using a projection method. It can be run both on CPU-architectures and GPU-architectures.

© 2022 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Multiphase flows are ubiquitous in many contexts, ranging from environmental flows to industrial applications. The interaction between phases has a prominent role in the formation and evolution of clouds [2], in sediment transport [3,4], oceanic sprays and bubbles generation [5] and more in general it represents one of the *grand challenges of environmental fluid mechanics* [6]. These flows are also crucial in several industrial applications, such as pharmaceutical, transportation, food processing and power generation [7]. From a theoretical point of view, the main difficulty when analyzing multiphase flows relies on their multiscale nature, since the length-scale of the interface is of the order of the mean-free path while in most applications the typical length-scale is several orders of magnitude larger ($\sim 10^5 - 10^6$). This huge separation of scales is magnified when dealing with turbulent multiphase flows, which further broadens the spectrum of length-scales, thus making unfeasible any attempt to bridge all of them in a single and unique framework. For this reason, all the tools developed so far, both of experimental and numerical nature, have focused on only a portion of the scale spectrum while the remaining part is modelled or neglected.

As regards multiphase turbulence, where most of our interests and applications are, both experimental investigations and numerical simulations have been extensively used in the last thirty years and have led to important contributions in a variety of problems and configurations: to name a few, particle laden flows and sediment transport discussed in [8,4], bubbly and droplet flows reviewed in [9–11], oceanic sprays and bubbles as highlighted in [5]. Nevertheless, as already discussed in [10] and despite the recent progress in the instantaneous measurement of bubble/droplet shape [12–14], there is still a lack of experimental data for the measurement of the instantaneous velocity fields of both carried and dispersed phase as well as for the turbulent kinetic energy and dissipation near the interface locations. These limitations disappear when dealing with numerical simulations and, therefore, in the last decades interface resolved simulations of multiphase flows have become a central investigation tool. Despite the advantages, numerical simulations are still limited to simple configurations and moderate scale separation, and pose the challenge of the choice of the proper method to fully resolve the two-phase interface. As discussed in [15], there is now consensus that numerical methods suitable to perform interface-resolved simulations of multiphase flow should have the following properties: i) be able to enforce mass, momentum and kinetic energy conservation at discrete level, ii) allow mismatches in the material properties, whose magnitude depends on the application, and iii) handle complex and possibly arbitrary topological changes. Among the four groups of numerical methods for multiphase flows, Front-Tracking (FT) [16], Volume-of-Fluid (VoF) [17], Phase Field (PFM) [18], Level-set (LS) [19], it exists at least a variant of each which possesses the aforementioned numerical properties, giving some freedom to researchers and scientists on the choice of their preferred numerical tool (see [17,20,21] for a review).

Nevertheless, it is becoming ever more clear that another desirable property of any numerical method is its straightforward adaptation to be able to run massively parallel simulations, especially on accelerated architectures. With the increase in the computing power driven by Graphics Processing Units (GPUs) [22], several HPC centers are now shifting towards GPU-only and GPU-accelerated architectures. This trend is making the GPU-parallelization of numerical codes for fluid mechanics a mandatory requirement rather than a simple advantage. This effort has been already taken for single-phase codes, where at least three open-source codes for incompressible and fully compressible simulations are able to run on accelerated architectures: AFiD [23], STREAmS [24] and the accelerated version of CaNS [25]. Conversely, on the multiphase counterpart, despite the large availability of CPU-based open source codes, PARIS Simulator [26], TBFsolver [27], FS3D [28], NGA2 [29], Basilisk [30] and MFC [31] to name few, limited effort has so far been devoted to their adaptation to hybrid architectures.

In this work, we aim to fill this gap and present FluTAS (Fluid Transport Accelerated Solver), a code for massive Direct Numerical Simulations on multi-GPU and multi-CPU architectures targeting incompressible multiphase flows, optionally with heat transfer. The numerical solution of these flows is typically performed using finite-difference methods in a staggered variable arrangement, and it involves the solution of a Poisson equation to enforce the constraints on the velocity divergence. In this context, FluTAS uses as basis the Navier-Stokes solver CaNS [32] and its GPU extension [25], whose key feature is a general implementation incorporating all the possible homogeneous pressure boundary conditions that can benefit of the FFT-based elliptic solvers [33]. The single-phase Navier-Stokes solver is extended to a two-fluid code using the one-domain formulation [17,20] and coupled with the algebraic VoF MTHINC [1] to capture the two-phase interface. This method combines the exact mass conservation properties of certain geometric VoF methods with the reduced number of local operations for the interface reconstruction of the algebraic VoFs, making it a good candidate for properly exploiting hybrid and accelerated architectures. The version available in our group has been validated in [34] and extensively employed in a different variety of multiphase configurations, both for laminar [35–37] and turbulent [38–41] flows. Note that it has been extended to phase changing flows [42] and also to handle weakly compressible multiphase flows (low-Mach approximation) [43,44].

This paper is organized as follows. In §2, we introduce the governing equations for the incompressible two-fluid system. The discretization details of the VoF method, energy equation and Navier-Stokes solver are provided in §3, whereas the standard benchmarks for code validation are discussed in §4. Next, the parallelization for the GPU acceleration is presented together with the scaling tests in §5 and in §6. The code potentialities are shown in two demanding simulations of multiphase turbulence: emulsions in homogeneous isotropic turbulence (HIT) and two-phase thermal convection (see §7). Finally, main conclusions and future perspectives are summarized in §8.

2. Governing equations

We consider a two-phase system of immiscible incompressible Newtonian fluids (e.g., a gas-liquid system). The two phases are bounded by an infinitesimally small interface, through which momentum and energy can be transferred. To describe the system, we define a phase indicator function H distinguishing the two phases at position $\mathbf{x}$ and time t:

$$H(\mathbf{x},t)=\begin{cases}1 & \text{if } \mathbf{x}\in\Omega_1,\\ 0 & \text{if } \mathbf{x}\in\Omega_2,\end{cases} \tag{1}$$

where Ω_1 and Ω_2 are the domains pertaining to phases 1 and 2. We can use H to define the thermophysical properties in the whole domain $\Omega=\Omega_1\cup\Omega_2$ as follows:

$$\xi(\mathbf{x},t)=\xi_1 H(\mathbf{x},t)+\xi_2(1-H(\mathbf{x},t)), \tag{2}$$

where ξ_i $(i=1,2)$ can be the mass density ρ_i, the dynamic viscosity μ_i, the thermal conductivity k_i or the specific heat capacity at constant pressure $c_{p,i}$. Hereafter, unless otherwise stated, thermophysical quantities not specifically referring to one of the phases are defined from eq. (2). The evolution of the indicator function is governed from the following topological equation:

$$\frac{\partial H}{\partial t}+\nabla\cdot(\mathbf{u}_\Gamma H)=H\nabla\cdot\mathbf{u}_\Gamma, \tag{3}$$

where $\mathbf{u}_\Gamma$ is the interface velocity. In absence of phase change, the one-fluid velocity $\mathbf{u}$ is continuous across the interface and therefore, it can be employed as interface velocity in equation (3).

The equations governing the momentum and energy transport for the liquid and gas phase are coupled through appropriate interfacial conditions [45], reported below in the so-called one-fluid or whole-domain formulation, where each transport equation is defined in Ω [20].

$$\nabla\cdot\mathbf{u}=0, \tag{4}$$

$$\rho\left[\frac{\partial\mathbf{u}}{\partial t}+\nabla\cdot(\mathbf{u}\mathbf{u})\right]=-\nabla p+\nabla\cdot\left[\mu\left(\nabla\mathbf{u}+\nabla\mathbf{u}^T\right)\right]+\sigma\kappa\delta_\Gamma\mathbf{n}+\hat{\rho}\mathbf{g}, \tag{5}$$

$$\rho c_p\left[\frac{\partial T}{\partial t}+\nabla\cdot(\mathbf{u}T)\right]=\nabla\cdot(k\nabla T). \tag{6}$$

Here, $\mathbf{u}$ is the fluid velocity assumed to be continuous in Ω, p is the hydrodynamic pressure, T the temperature. In equation (5), σ is the surface tension, κ is the local interfacial curvature, δ_Γ is a delta Dirac function, $\mathbf{n}$ is the normal vector, $\mathbf{g}$ is the gravity acceleration and $\hat{\rho}$ is the volumetric density field modified to account for the thermal effects in the gravity forces. Using the Oberbeck–Boussinesq approximation, $\hat{\rho}$ reads as:

$$\hat{\rho}=\rho_{1,r}\left[1-\beta_l(T-T_r)\right]H+\rho_{2,r}\left[1-\beta_g(T-T_r)\right](1-H), \tag{7}$$

where $\rho_{i=1,2,r}$ are the reference phase densities and $\beta_{i=1,2}$ are the liquid and gas thermal expansion coefficients.

3. Numerical methodology

The numerical solution of the governing equations (3), (4), (5) and (6) presented in section 2 is addressed on a fixed regular Cartesian grid with uniform spacing Δx, Δy and Δz along each direction. A marker-and-cell arrangement is employed for velocity and pressure points [46], whereas all scalar fields are defined at the cell centers. Each time-step, the governing equations are advanced in time by $\Delta t^{n+1}=t^{n+1}-t^n$, with the previous time-step indicated with $\Delta t^n=t^n-t^{n-1}$. Hereafter, we present the numerical discretization of the governing equations, following the same order in which they are solved.

3.1. Volume of fluid: the MTHINC method

The first step of the time-marching algorithm consists in the interface reconstruction and its subsequent advection. As previously mentioned, these tasks are addressed within a fully Eulerian framework using a volume-of-fluid (VoF) method. From a numerical point of view, this consists first in defining the volume fraction ϕ in each cell of the computational domain as:

$$\phi=\frac{1}{V_c}\int_{V_c}H(\mathbf{x},t)dV_c, \tag{8}$$

with $V_c=\Delta x\Delta y\Delta z$. Next, equation (3) is written in terms of volume fraction as:

$$\frac{\partial\phi}{\partial t}+\nabla\cdot(H\mathbf{u})=\phi\nabla\cdot\mathbf{u}. \tag{9}$$

The distinct feature of each class of VoF method lies in the way H is approximated. In this work we employ the algebraic volume-of-fluid method based on the Multi-dimensional Tangent Hyperbola reconstruction, MTHINC [1], whose central idea is to approximate H with a hyperbolic tangent:

$$H(\tilde{x},\tilde{y},\tilde{z})=\frac{1}{2}\left[1+\tanh(\beta_{th}\left(\mathcal{T}(\tilde{\mathbf{x}})+d_{th}\right)\right], \tag{10}$$

where β_{th}, d_{th} are the sharpness and the normalization parameter, respectively, and $(\tilde{x},\tilde{y},\tilde{z})$ a local coordinate system $\tilde{\mathbf{x}}=[(x-0.5)/\Delta x,(y-0.5)/\Delta y,(z-0.5)/\Delta z]$. Employing equation (10) has two distinct advantages with respect to a piecewise approximation, commonly employed in the geometric VoF methods. First, the phase indicator H can be approximated with a reconstructing polynomial $\mathcal{T}$ of arbitrary order in a straightforward manner. Next, once $\mathcal{T}$ is known, the resulting interface at the two-phase boundary has smooth but controlled thickness (with the parameter β_{th}), which also allows computing accurately the normal vector $\mathbf{n}$ and curvature tensor $\mathbf{K}$ directly from ϕ. More details about the choice of $\mathcal{T}$ and the calculations of d_{th}, $\mathbf{n}$ and $\mathbf{K}$ are found in the original paper by Ii et al. [1], but for completeness we include them in the appendix A with the numerical implementation details.

After the reconstruction step, the interface is advected using a directional splitting approach [47,48], which consists in evaluating the numerical fluxes sequentially in each direction using, for each split, the latest estimation of VoF field. Accordingly, three provisional fields $\phi^p_{i,j,k}$ (with $p=[x,y,z]$) are first computed:

$$\phi^p_{i,j,k}=\frac{\phi^s_{i,j,k}-\frac{1}{\Delta l^p}\left[f^p_+(\phi^s_{i,j,k})-f^p_-(\phi^s_{i,j,k})\right]}{1-\frac{\Delta t^{n+1}}{\Delta l^p}\left(u^p_+-u^p_-\right)^n}, \tag{11}$$

where $s=[n,x,y]$, $[\Delta l^x,\Delta l^y,\Delta l^z]=[\Delta x,\Delta y,\Delta z]$, $[u^x,u^y,u^z]=[u,v,w]$ with $u^p_\pm$ the p-th velocity component. The calculation of the numerical fluxes $f_\pm$ in equation (11) are evaluated using the hyperbolic tangent approximation of H as detailed in appendix A. Next, the divergence correction step is applied in order to impose the volume conservation of both phases at a discrete level:

$$\phi^{n+1}_{i,j,k}=\phi^z_{i,j,k}-\sum_{p=x,y,z}\frac{\Delta t^{n+1}}{\Delta l^p}\phi^p_{i,j,k}(u^p_+-u^p_-)^n. \tag{12}$$

With the above approach, mass conservation is ensured up to the accuracy with which the divergence free condition (4) is satisfied. Accordingly, if direct methods are employed to solve the Poisson equation, the mass of each phase results to be conserved up to machine precision. Another approach with a similar property has been introduced in [49]. However, in that case the dilatation terms at the denominator of equation (11) are treated in an explicit manner, while here in an implicit strategy is employed. This comes at a cost of the final correction step, given by equation (12), but with the advantage of not introducing additional time-step restrictions (apart the convective one) in the advection of the colour function.

3.2. Thermal effects

The next step of the time-marching algorithm consists in advancing the temperature field using an explicit second-order Adams-Bashforth method:

$$T^{n+1} = T^n + \Delta t^{n+1}\left(f_{t,1}\mathcal{M}_T^n - f_{t,2}\mathcal{M}_T^{n-1}\right), \tag{13}$$

where $f_{t,1} = (1 + 0.5\Delta t^{n+1}/\Delta t^n)$ and $f_{t,2} = 0.5\Delta t^{n+1}/\Delta t^n$ are the coefficients of the Adams-Bashforth scheme. In equation (13), the operator $\mathcal{M}_T$ accounts for the advection and diffusion contribution and it is provided below in a semi-discrete form:

$$\mathcal{M}_T^n = -\nabla\cdot(\mathbf{u}^n T^n) + \frac{1}{\rho^{n+1}c_p^{n+1}}\nabla\cdot(k^{n+1}\nabla T^n). \tag{14}$$

All the spatial terms in equation (14) are discretized with second-order central schemes, except for the temperature convection term. The discretization of the latter is based on the 5th-order WENO5 scheme, as in reference [50].

3.3. Pressure correction algorithm

Once the energy equation has been advanced, the momentum equation is solved with a second-order pressure correction [51], reported below in a semi-discrete form:

$$\left(\frac{\mathbf{u}^{\star\star} - \mathbf{u}^n}{\Delta t^{n+1}}\right) = f_{t,1}\mathcal{M}_{\mathbf{u}}^n - f_{t,2}\mathcal{M}_{\mathbf{u}}^{n-1} + \frac{\left(\sigma\kappa\delta_\Gamma + \hat{\rho}\mathbf{g}\right)^{n+1}}{\rho^{n+1}}, \tag{15}$$

$$\mathbf{u}^{\star} = \mathbf{u}^{\star\star} - \frac{\Delta t^{n+1}}{\rho_0}\left[\left(1 - \frac{\rho_0}{\rho^{n+1}}\nabla\hat{p}\right) + \nabla p^n\right], \tag{16}$$

$$\nabla^2\psi^{n+1} = \frac{\rho_0}{\Delta t^{n+1}}\nabla\cdot\mathbf{u}^{\star}, \tag{17}$$

$$\mathbf{u}^{n+1} = \mathbf{u}^{\star} - \frac{\Delta t^{n+1}}{\rho_0}\nabla\psi^{n+1}, \tag{18}$$

$$p^{n+1} = p^n + \psi^{n+1}, \tag{19}$$

where the operator $\mathcal{M}_{\mathbf{u}}^n$ and $\mathcal{M}_{\mathbf{u}}^{n-1}$ in equation (15) includes the convective and diffusive terms computed at the current and previous time level, neglecting the surface tension and gravity forces which are then included as source terms. The spatial gradients in $\mathcal{M}_{\mathbf{u}}$ are discretized with central schemes. The intermediate velocity $\mathbf{u}^{\star\star}$ is then updated with the contribution from the terms due to the time-pressure splitting, as in (16). Note that ρ_0 is the minimum value of the density field in the computational domain and $\hat{p}$ represents the time-extrapolated pressure between the current and the old time step, i.e. $\hat{p} = (1 + \Delta t^{n+1}/\Delta t^n)p^n - (\Delta t^{n+1}/\Delta t^n)p^{n-1}$. Following [52] and contrary to [53,54], the terms arising from the pressure splittings are included in the prediction of the velocity field (see eq. (16) before the imposition of the boundary conditions. This approach has two distinct advantages. First, it represents an incremental pressure projection which allows achieving an almost second-order accurate in time pressure field [52]. Next, it ensures the consistency of the pressure field near a solid boundary (i.e., $\mathbf{u}^{n+1} = \mathbf{u}^{\star} = 0$), where the pressure gradient component normal to the boundary (i.e., $\nabla_\perp\psi^{n+1} = 0$) vanishes independently of the local density (see eq. (18)).

Next, the constant coefficients Poisson equation (17) is solved with the method of eigenexpansion technique that can be employed for different combination of homogeneous pressure boundary conditions [33]. Finally, the velocity field is corrected as in equation (18) in order to impose the divergence constrain (i.e., solenoidal velocity field) and the pressure updated as in equation (19).

3.3.1. Poisson solver

The code uses the FFT-based finite-difference direct solver developed and implemented in the DNS code CaNS; see [32,25]. The underlying numerical approach dates back to the late 1970s [55,33], and has regained popularity in recent years, thanks to the improvements of hardware, and of and software frameworks for collective data communications, provided by the MPI standard and higher-level libraries like 2DECOMP&FFT. In a nutshell, the approach uses Fourier-based expansions along two domain directions, which reduce the system of equations resulting from the three-dimensional second-order finite-difference Laplace operator (seven non-zero diagonals) to a simple, tridiagonal system. These Fourier-based expansions depend on the boundary conditions of the system, and can be computed using FFTs, some of them with pre-/post-processing of the FFT input/output (see, e.g., [56]).

The FFT-based expansions are employed along directions x and y, and the resulting tridiagonal system along z is then solved using Gauss elimination. For calculations on CPUs, the method leverages the *guru* interface of the FFTW library [57], which allows for performing all possible combinations of discrete transforms using the same syntax. On GPUs, the fast discrete cosine and sine transforms have been implemented using real-to-complex/complex-to-real FFTs from the cuFFT library, with pre- and post-processing of the input and output signals to calculate the desired series expansion [56,25]. We refer to Refs. [32,25] for more details on this method and its implementation.

Concerning the parallelization of the method in a distributed-memory setting, the FFT-based transforms and Gauss elimination steps require the data along each direction to be local to each MPI task. The domain is decomposed using a 2D pencil decomposition, where collective *all-to-all* communications are required to transpose the orientation of the 2D data decomposition. These transposes are performed using the 2DECOMP&FFT library [58], which was modified to allow for GPU-GPU communication in [23,25].

It is worth noting that, in line with the recent developments of CaNS, the present method uses a default decomposition (i.e., "outside" the Poisson solver) based on a partitioning along y and z, resulting in x-aligned pencils. This reduces the total number of data transposes to be performed during the solution of the Poisson equation from 6 to 4. The approach has been adopted for both CPUs and GPUs, and the required operations to solve the Poisson equation are summarized as follows:

1. perform forward FFT-based transforms along x;
2. transpose x-to-y;
3. perform forward FFT-based transforms along y;
4. transpose y-to-z;
5. solve tridiagonal system using Gauss elimination along z;
6. transpose z-to-y;
7. perform backward FFT-based transforms along y;
8. transpose y-to-x;
9. perform backward FFT-based transforms along x.

Moreover, for the GPU implementation, the solver explicitly reduces the number of *all-to-all* operations when the domain is not decomposed along z (i.e., when a $x-y$ slab decomposition is prescribed). This effectively decreases the number of collective operations from 4 to 2 (steps 2 and 8 above are skipped). This is the approach adopted in the GPU runs presented here – due to the higher memory bandwidth in GPUs, a slab decomposition suffices for distributed-memory calculations with sufficiently small wall-clock time per step. Explicitly skipping these two no-op resulted in a substantial reduction in wall-clock time per step, and in an overall improvement in the parallel scalability of the solver.

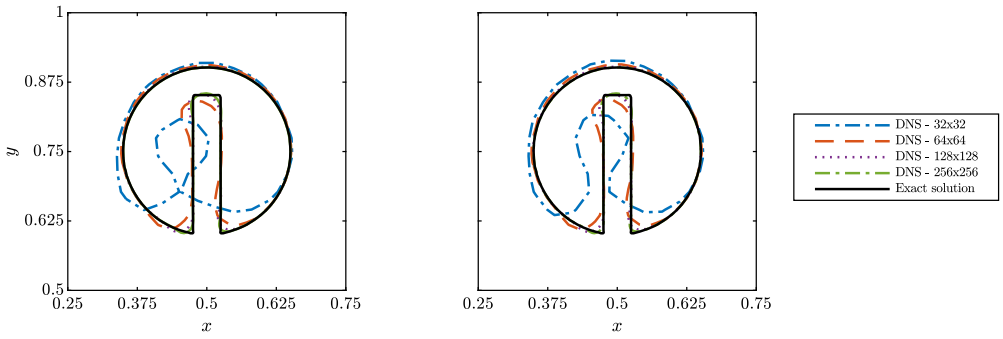


Fig. 1. Deformation of the Zalesak's slotted disk after $t = 2\pi$ for $\beta_{th} = 2$ (left) and $\beta_{th} = 3$ (right).

3.4. Complete solution algorithm

For clarity, a step by step description of the overall solution procedure is presented in Algorithm 1.

Algorithm 1 Overall solution procedure.

ϕ^0, T^0, $\mathbf{u}^0$, p^0 are initialized;
ρ^0, μ^0, k^0 and c_p^0 are calculated using equation (2) from ϕ^0;
$n = 0$ is set,
while $(t < t_{tot} \parallel n < N_{tot})$ **do**
  Set $n = n + 1$ and Δt^{n+1};
  ϕ^{n+1} is calculated from equation (11) and (12);
  $\mathbf{n}^{n+1}$ and κ^{n+1} are evaluated using the procedure described in Appendix A;
  ρ^{n+1}, μ^{n+1}, k^{n+1} and c_p^{n+1} is calculated from equation (2);
  T^{n+1} is calculated from Eq. (13);
  $\mathbf{u}^*$ is calculated from Eq. (15) and Eq. (16);
  ψ^{n+1} is calculated from Eq. (17);
  $\mathbf{u}^{n+1}$ is calculated from Eq. (18);
  p^{n+1} is computed from Eq. (19).
end while
End of simulation.

4. Validation

4.1. Two-dimensional Zalesak's disk

The Zalesak problem represents a classical benchmark to assess the accuracy of the interface capturing/tracking algorithm. It consists in the solid-body rotation of a slotted disk immersed in an imposed two-dimensional velocity field $\mathbf{u} = (0.5 - y, x - 0.5)$. The disk can be easily defined in a Cartesian two-dimensional domain by setting the indicator function $H^0_{i,j,k}$ equal to 1 inside the following domain Ω_H

$$\Omega_H : \left[(x-0.5)^2 + (y-0.75)^2\right] \leq 0.15^2 \cap (|x-0.5| \geq 0.025 \cup y \geq 0.85)\,. \tag{20}$$

The benchmark consists in comparing the deformation of the solid disk with respect to the initial shape after one entire revolution. The VoF equation is solved in a two-dimensional square domain $\Omega = [0,1] \times [0,1]$, discretized with four different grid spacing $[\Delta x, \Delta y] = [1/N_x, 1/N_y]$ with $N_x \times N_y = [32 \times 32, 64 \times 64, 128 \times 128, 256 \times 256]$. Periodic boundary conditions are prescribed in both directions. Simulations are conducted up to $t = 2\pi$ (i.e., one complete revolution of the slotted disk) using a constant time-step $\Delta t = t/3200$. Note that this value has been chosen to ensure a stable time integration for the highest grid resolutions cases (i.e., 256×256) and is employed for the coarser cases. Fig. 1 shows the final disk shape for different grid solutions and for two sharpness parameters $\beta_{th} = 2$ and $\beta_{th} = 3$. Note that the highest deviation from the initial shape is in the corner regions, where the high-curvature regions are located. Moreover, the solution is weakly dependent on the value of β_{th} and deviations between the different employed β_{th} are visible only for the coarser simulations.

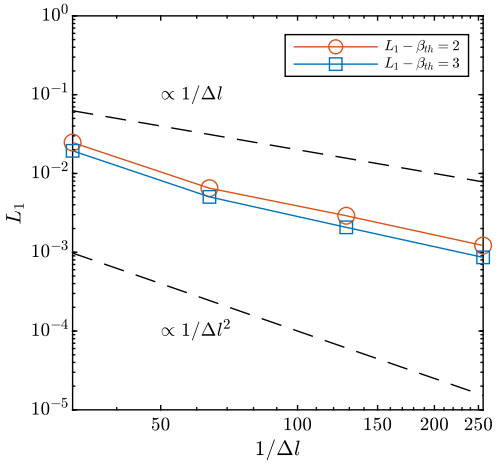


Fig. 2. L_1 error of the Zalesak disk problem for $\beta_{th} = 2 - 3$.

Finally, to assess the accuracy of the solution, we compute the L_1 norm and the order of convergence as:

$$L_1 = \frac{1}{N_x N_y} \sum_{i=1}^{N_x} \sum_{y=1}^{N_y} |\phi(i,j) - \phi^0(i,j)|, \tag{21}$$

$$n_{L1} = \frac{\log\left(\dfrac{L_{2,N}}{L_{1,N}}\right)}{\log(2)}, \tag{22}$$

where $L_{1,N}$ is the L_1-error using $N_x \times N_y$ grid points and $L_{1,2N}$ is the L_1-error evaluated with $2N_x \times 2N_y$ grid points. Results are reported in Fig. 2, where an order of convergence between the first and the second-order is achieved for ϕ, almost independent of the employed value of β_{th}.

4.2. Three-dimensional rising bubble

The rising bubble test case is a well-established numerical benchmark for multiphase flows [59]. This test is presented here to showcase the ability of the numerical tool to accurately capture the topological changes of a moving interface. The flow is driven by the density difference between the two phases, and is influenced by the viscosity difference and the surface tension. The relevant dimensionless groups for this flow are the Reynolds number $\mathrm{Re} = \rho_g u_r l_r / \mu_g$, the Weber number $\mathrm{We} = \rho_g u_r^2 l_r / \sigma$, the Froude

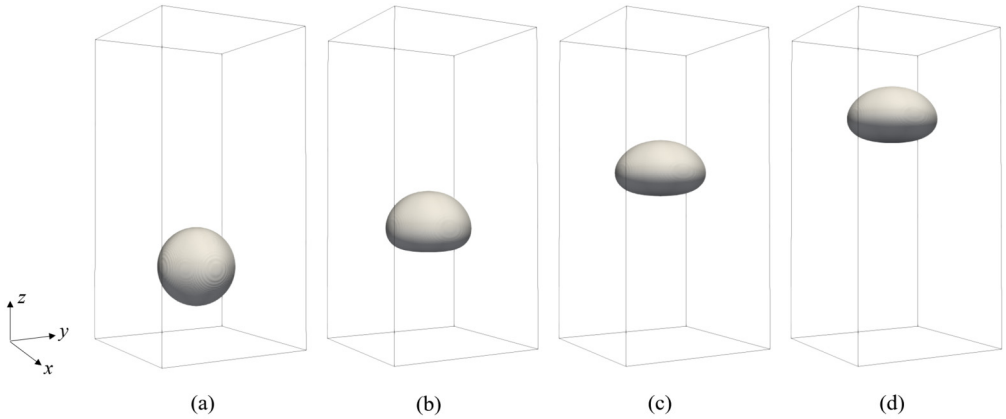


Fig. 3. Isosurfaces of $\phi = 0.5$ at dimensionless times $t\sqrt{|\mathbf{g}|/d_0}$ (a) 0, (b) 1.4, (c) 2.8 and (d) 4.2.

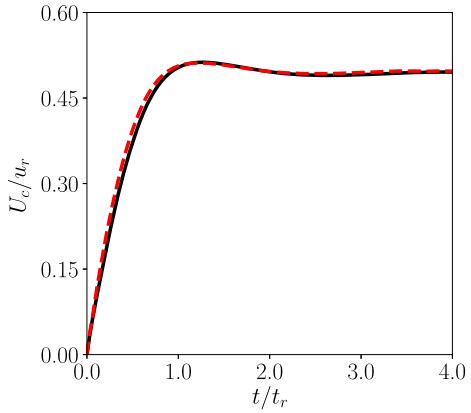


Fig. 4. Bubble rise velocity as a function of time. Black solid line, present results; red dashed line, reference results from [59]. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

number Fr$= u_r/\sqrt{|\mathbf{g}|d_0}$, the density ratio $\lambda_\rho = \rho_l/\rho_g$ and the viscosity ratio $\lambda_\mu = \mu_l/\mu_g$. In these definitions, l_r is a reference length and u_r the reference velocity. Moreover, σ is the surface tension coefficient, ρ_g and ρ_l the reference gas and liquid densities, and μ_g and μ_l the reference gas and liquid dynamic viscosity. Finally, $\mathbf{g}$ is the acceleration of gravity and d_0 is the initial diameter of the spherical bubble.

Following the benchmark study [59], the values adopted for the dimensionless groups are Re$=$ 35, We$=$ 1, Fr$=$ 1, $\lambda_\rho = 10$ and $\lambda_\mu = 10$, setting $l_r = d_0$, $u_r = \sqrt{|\mathbf{g}|d_0}$ and the reference time $t_r = \sqrt{d_0/|\mathbf{g}|}$. The dimensions of the computational domain are $l_x = l_y = 2d_0$ and $l_z = 4d_0$. The acceleration of gravity acts along the z-direction. No-slip and no-penetration boundary conditions are prescribed at the horizontal top and bottom boundaries of the domain (z-normal) and periodic conditions are prescribed at the vertical boundaries (x- or y-normal). A uniform Cartesian grid of $128\times128\times256$ cells is used. Initially, stagnant flow conditions are applied and the position of the center of mass of the spherical bubble, denoted as $(x_c(t), y_c(t), z_c(t))$, is located at (d_0, d_0, d_0). A constant time-step $\Delta t/t_r = 2.8 \times 10^{-4}$ is used to advance the solution in time. Fig. 3 shows the isosurfaces of $\phi = 0.5$ at various time instances. It is evident that as the initially spherical bubble rises, its surface topology changes. To compare against the reference results from [59], Fig. 4 shows the evolution of the bubble rising velocity U_c in time. The bubble velocity is defined as,

Table 1
Comparison of the dimensionless rise velocity U_c/u_r and bubble sphericity A_0/A between reference and present results.

	t/t_r	U_c/u_r	A_0/A
ref. [59]		0.51013	0.97418
present	1.4	0.51144	0.97892
dev. %		0.26	0.49
ref. [59]		0.49823	0.95925
present	4.2	0.49512	0.96057
dev. %		0.62	0.14

$$U_c = \frac{\int_\Omega \phi\, w\, d\Omega}{\int_\Omega \phi\, d\Omega}, \tag{23}$$

where w is the vertical velocity component and Ω the volume of the entire domain. After an initial period where the bubble accelerates, the rise velocity reaches a maximum and then stabilizes. Fig. 4 demonstrates an excellent agreement between present and reference results. To further quantify this agreement, Table 1 presents benchmark quantities for comparison at specific time instances. Besides the bubble velocity, the table shows the bubble sphericity A_0/A, defined as the initial value of the bubble surface area over the value at a later time. The deviation between reference and present values is less than 1%.

4.3. Differentially heated cavity

To demonstrate the accuracy of the code in the presence of thermal effects, this section considers the flow of air in a closed two-dimensional square heated cavity. The cavity is heated and cooled by the vertical side walls (y-normal), while the horizontal walls are adiabatic (z-normal). Within this configuration, a circulation is formed and maintained by the ascending hot fluid next to the heated wall and the descending cold fluid next to the cooled wall.

The flow is therefore purely thermally-driven and is characterized by the Rayleigh number Ra$= |\mathbf{g}|\beta\Delta T l_r^3/(\nu\alpha)$ and the Prandtl number Pr$= \nu/\alpha$. In these definitions, β is the fluid thermal expansion coefficient, ν is the fluid viscosity, α is the fluid thermal diffusivity and $\Delta T = (T_h - T_c)$ is the temperature difference between the heated (T_h) and cooled (T_c) walls. Typically, the height of the cavity is taken as the reference length ($l_r = L_z$), while the reference velocity and time are defined as $u_r = \alpha/l_r$ and $t_r = l_r^2/\alpha$. The case simulated here follows the setup presented in several

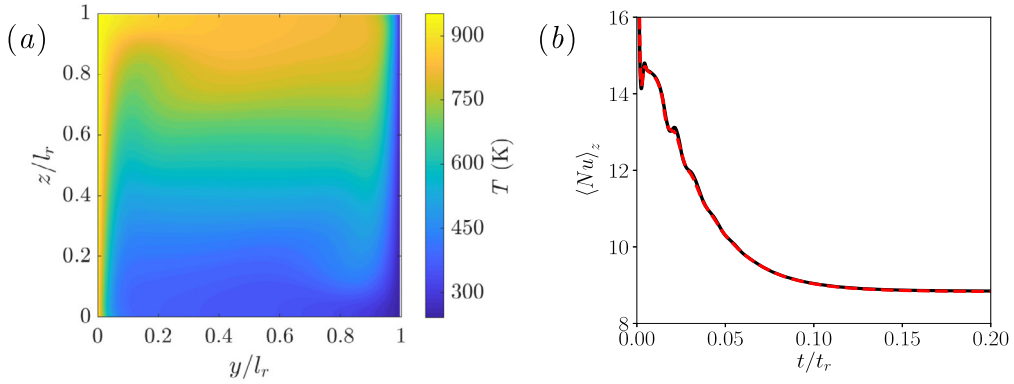


Fig. 5. (*a*) Contour plot of the temperature field at $t/t_r = 0.5$ (steady state) for the differentially heated cavity test case, (*b*) Temporal evolution of the wall-averaged Nusselt number on the heated wall. Black solid line, present results; red dashed line, reference results from [60].

Table 2
Comparison of key benchmark quantities at steady state for the differentially heated cavity test case. V_{max} is the maximum horizontal velocity along the vertical mid-plane ($y = 0.5l_r$), W_{max} is the maximum vertical velocity along the horizontal mid-plane ($z = 0.5l_r$), Nu_{max} and Nu_{min} are the maximum and minimum values of the Nusselt number on the heated wall, and $\langle Nu \rangle_z$ is the averaged Nusselt number value on the heated wall.

	V_{max}/u_r	W_{max}/u_r	Nu_{max}	Nu_{min}	$\langle Nu \rangle_z$
Ref. [60]	64.85	220.6	17.58	0.9794	8.830
Present	64.86	220.3	17.67	0.9773	8.843
% dev.	0.02	0.14	0.51	0.21	0.14

studies [61,62,60] with Ra$= 10^6$ and Pr$= 0.71$. The domain boundaries are solid walls, and no-slip boundary conditions are applied. With respect to the temperature field, constant temperature boundary conditions are applied on the vertical walls and a zero temperature gradient along the normal direction is applied on the horizontal walls. The domain is discretized in space using a uniform Cartesian grid with 256×256 cells. Initially, the air in the cavity is stagnant and isothermal at a temperature $T_0 = T_c$. A constant time-step Δt is used to advance the solution in time, given by $\Delta t/t_r = 5.0 \times 10^{-7}$. Fig. 5*a*) shows the contour of the temperature field at $t/t_r = 0.5$, at which point a steady state has been reached. The temperature field is characterized by thin and spatially developing thermal boundary layers next to the thermally active vertical walls, and a stratified region at the central area of the cavity. The heat transfer rate inside the cavity is expressed through the Nusselt number, defined as,

$$\mathrm{Nu} = \frac{hl_r}{k} = \frac{l_r}{\Delta T} \nabla T\big|_w \cdot \mathbf{n}_w, \tag{24}$$

where h is the heat transfer coefficient, k is the fluid thermal conductivity, $\nabla T\big|_w$ is the temperature gradient on any of the thermally active vertical walls and $\mathbf{n}_w$ is the corresponding unit normal vector on the wall. Fig. 5*b*) shows the comparison of the temporal evolution of the wall-averaged Nusselt number $\langle Nu \rangle_z$ on the heated wall between the present and reference results from [60]. It is evident that the present results are in excellent agreement with the reference solution for the entire duration of the simulation. Furthermore, Table 2 presents the comparison of key benchmark quantities at steady state, confirming the agreement between present and reference results.

5. Code parallelization and GPU acceleration

5.1. Domain decomposition

The code is designed to run both in multi-CPU and multi-GPU architectures. For domain decomposition, both slab (1D) and pencil (2D) are allowed [32] through the library 2DECOMP [58]. The type of decomposition can be implicitly set via the 2-component array `dims` (e.g. $[1, n]$ for slabs and $[n, m]$ for pencils) in one input file `dns.in`. The pencil/slab orientation can be arbitrarily chosen as in CaNS, via the preprocessor flags `-D_DECOMP_X`, `-D_DECOMP_Y` and `-D_DECOMP_Z` which set the direction over which the domain is not decomposed. This flexibility allows improving the efficiency of both the CPU and the GPU implementations. For CPU, using pencils allows increasing the number of processes used per execution (i.e. up to N^2 for $n_x = n_y = n_z = N$), hence reducing the time to solution. In the GPU implementation, only the *z-pencil* and *x-slabs* decompositions are allowed. It is recommendable to use *x-slabs* on GPU (i.e. compiling with `-D_DECOMP_X` and using `dims`$= [1, n]$) as this implementation reduces the number of *all-to-all* calls to the minimum, hence reducing GPU-GPU communication and improving performances on multi-nodes runs.

5.2. Code parallelization

The parallelization is performed using MPI. When GPU acceleration is enabled, MPI allocates one rank for each GPU. The code assumes the chosen MPI library is "CUDA-aware", meaning GPU data is directly passed to MPI function call and the MPI implementation takes care of moving the data in the most efficient way. If available, GPU-to-GPU communication can leverage NVIDIA NVLink which is a physical GPU-to-GPU interconnection known to have higher bandwidth (at least one order of magnitude) than Infiniband. Throughout the code, all nested for-loops, i.e. iterations over all the domain points, are accelerated on GPUs using OpenACC [63], a portable standard directive-based programming model that can execute code on multi-core CPUs as well as accelerators like NVIDIA GPU. Such offload is not used for CPU-only compilation and execution. To execute FluTAS, the platform needs to support NVIDIA Unified Memory, which has two main advantages:

- the ability of allocating and managing more GPU allocated memory than what is physically present on the device;
- the ability to avoid explicitly to handle data movements Host-to-Device and Device-to-Host, leaving the runtime do the work for the developers.

Both features are used in the code and proved crucial for an efficient GPU acceleration.

6. Code performance

We now present an analysis of the code performances on standard CPU-based and accelerated GPU-based architectures. Tests on

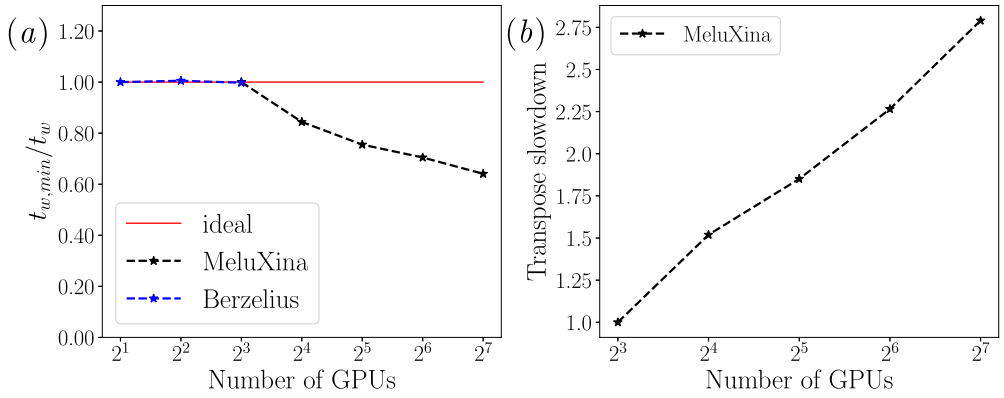


Fig. 6. For the two-layer Rayleigh-Bénard convection problem discussed in §7.2: a) code performance on MeluXina and Berzelius, b) slowdown due to transpose operation. For each set of data, we compute $t_{w,min}$ as the time per-timestep at the minimum number of GPUs tested over t_w, i.e. the time-per-timestep at the specific number of GPUs.

GPUs were performed on MeluXina at *LuxProvide* (LXP, Luxembourg) [64] and Berzelius at *National Supercomputer Centre* (NSC, Sweden) [65], while tests on CPUs were performed on Tetralith also managed by NSC.

6.1. Weak and strong scaling

We first discuss the weak-scaling tests for a Rayleigh-Bénard problem with the same set-up as it will be discussed in §7.2. For this test, we start with a "base" computational grid of $N_x \times N_y \times N_z = 1024 \times 512 \times 256$ grid points on 2 GPUs. Then, while keeping fixed N_x and N_y, we increase N_z proportionally to the number of GPUs, resembling a procedure of spatial ensemble-average (i.e. more structures simulated to improve the convergence of the large-scale statistics). As discussed in §5, we adopt a slab parallelization along the z direction using the `-D_DECOMP_X` compilation option, which reduces to 2 the number of *all-to-all* operations. It is worth noticing that, although both HPC machines are equipped with A100-40GB NVIDIA cards, Berzelius has 8 GPUs/node while MeluXina has 4 GPUs/node. Moreover, the interconnection between GPUs is handled through NVLink, whereas node-to-node connection is performed through Infiniband (IB), known for having lower bandwidth and for operating on a different protocol. Hence, IB is less straightforward to handle as it requires a more careful configuration from a hardware and software perspective. This implies choosing the right MPI configurations and selecting compatible communication libraries, resulting in an efficiency that may vary significantly among different HPC centers. For these reasons, to prove the non IB-dependent scaling and maximize the GPU-to-GPU communication throughput, Berzelius was used to perform weak-scaling tests within a node. On the other hand, MeluXina was used for multiple-node tests to assess the IB-dependent scaling.[2] Fig. 6a) shows that weak-scaling is linear when bounded by NVLink communications (i.e. no IB communications), as clearly supported by tests on Berzelius. When IB communications are required (i.e. node-to-node data transfer) the code performances decrease. It is worth noticing that, while an increasing communication overhead is provided by node-to-node communication on IB network, additional slowdown is caused by the slab parallelization. By increasing the number of elements along z, more data need to be transferred during the x-to-z transposes, further increasing the communication load. This is clearly shown in Fig. 6b), where the slowdown is found to increase proportionally to the number of GPUs.

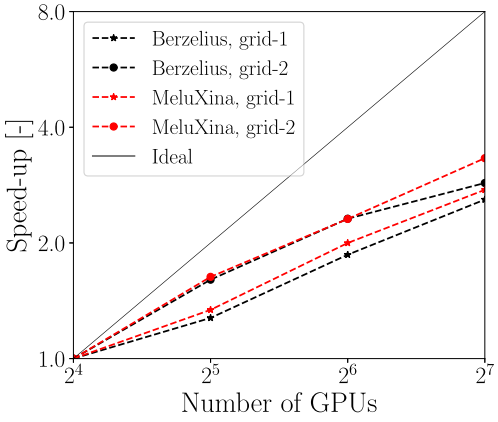


Fig. 7. Strong scaling test performed on Berzelius (black-dashed lines) and MeluXina (red-dashed lines) clusters for two different grids: $1024 \times 512 \times 1024$ (grid-1) and $1024 \times 1024 \times 1024$ (grid-2). The black continuous line indicates the ideal behaviour desired for the strong-scaling test.

Results of the strong scaling tests are reported in Fig. 7. Here we use two different grids, *i.e.* $1024 \times 512 \times 1024$ (grid-1) and $1024 \times 1024 \times 1024$ (grid-2) for the Rayleigh-Bénard problem discussed in §7.2. Tests are performed on Meluxina and Berzelius as for the weak scaling. While keeping the problem sizes fixed, the number of GPUs is progressively increased up to a maximum of 128, starting from $N_{GPU} = 16$ which represents the minimum amount required to fit the two computational domains in the available GPU memory. Despite a speed-up is always achieved, the code shows a progressive loss in performance, i.e. a reduction of the benefits derived by increasing the number of GPUs. Note, however, that a larger number of grid point (e.g. grid-2) leads to lesser performance loss, as a higher GPU occupancy can be obtained.

The decrease in performance observed in Fig. 7 is caused by two factors: the increase in communication among GPUs, and the reduction in local problem size, which does not leverage the full compute capacity of each GPU. While these effects are present in a strong scaling test, weak scaling allows us to isolate the effects of multi-GPU communication while keeping a higher GPU saturation. Thus, we argue that weak scaling represents a better tool to identify communication bottlenecks on multiple GPUs. Conversely, the strong scaling is more useful to estimate how much a fixed domain can be partitioned while keeping an efficient use of the computational resources.

[2] Berzelius uses a HGX 8-way platform, 8 A100 GPU connected all together, designed primarily for heavy AI workloads. MeluXina uses a HGX 4-way platform, 4 A100 GPU connected all together, which is better suited for scale-out HPC workloads. The majority of GPU-accelerated HPC clusters used primarily for simulation workloads in various scientific fields adopt the HGX 4-way configuration (including many European HPC systems funded by EuroHPC).

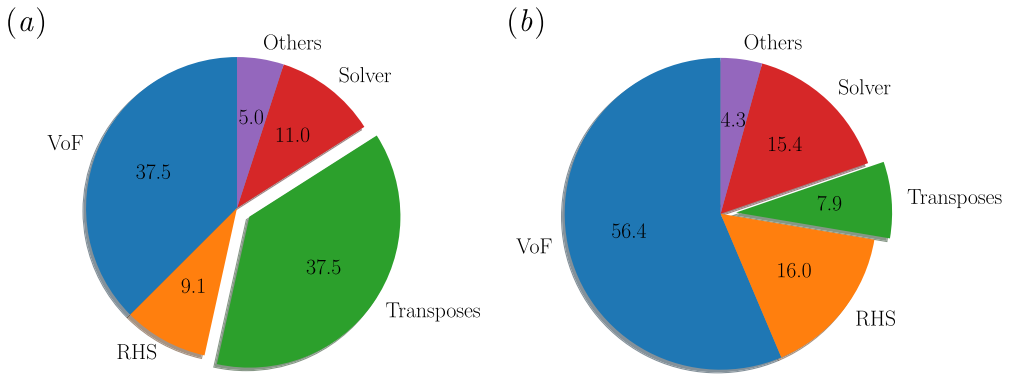


Fig. 8. Comparison of code-section load percentage on the total simulation time for GPUs (panel *a*) and CPUs (panel *b*). The different "slices" represent different code sections: 1) VoF (i.e. interface reconstruction and advection, update of the thermophysical properties), 2) RHS (i.e. discretization of the governing equations), 3) Transposes (i.e. transpose operation in the solver), 4) Solver (i.e. only Gaussian elimination) and others (i.e. correction step, divergence/time-step checks, output and post-processing routines).

Overall, the previous analysis suggests an important guideline for the user: in presence of unbalanced compute-vs-network architectures (e.g., node-to-node networking connection less efficient than the connection among GPUs within the same node), the optimal number of GPUs to be employed should be chosen as close as possible to the minimum amount required to fit the computational domain in the available GPU memory. Indeed, this is not always the case with older HPC architectures using previous generations of GPU hardware, where NVLink connections across GPUs inside a node were typically missing. For a fixed problem size, modern cards with high compute throughput will exhaust the required computation faster, leaving the remaining part of the computation as communication bound. In older GPU hardware, the acceleration is lower and communication becomes the dominant component affecting scalability for a larger number of GPUs. Hence, best practice dictates to use the least possible number of GPUs; on modern units with 80 GByte of HBM memory, if possible, it is therefore convenient to use one single 8-way GPU node (like a DGX A100) where all GPU are also connected via NVLink reducing communication overhead dramatically.

6.2. CPU-GPU comparison

In conclusion, we perform a comparison between the code performances on a CPU and a GPU architecture. It is worth mentioning that such comparison is notoriously not trivial. First, no exact and standard procedures to compare the two systems are established. Next, code performances may exhibit large variations among different architectures and using the same hybrid CPU-GPU node to perform tests on both may be misleading. CPU-only nodes and CPU-GPU nodes are intrinsically different in terms of network configuration and GPU/CPU interconnection, hence an unbiased test may not be performed directly on hybrid architectures (as CPU-GPU cluster would hardly be used to perform CPU-only jobs). Therefore, the following analysis has to be taken as a first-approximation estimate.

Here, we repeat the weak-scaling simulation with $n_{GPU} = 8$ GPUs on Berzelius on $n_{CPU} = 512$ CPUs on Tetralith, in both cases employing a slab parallelization along z. The test shows that for GPUs the average wall-clock time per-timestep is $t_{8,GPU} = 0.191\ s$, while for CPUs $t_{512,CPU} = 1.075\ s$. This results in an equivalent number of GPUs $n_{eq} = (t_{512,CPU} n_{CPU})/(t_{8,GPU} n_{GPU}) \approx 359$.

Finally, a comparison in terms of computing-load percentage for each code section is displayed in Fig. 8. As previously anticipated, transposes during GPU simulations (panel *a*) represents more than half of the computing load. The remaining parts, mainly composed of stencil operations enclosed in `for` loops, largely benefit from GPU-offload, while in CPUs (panel *b*) these account for more than 70% of the total wall-clock time per time-step.

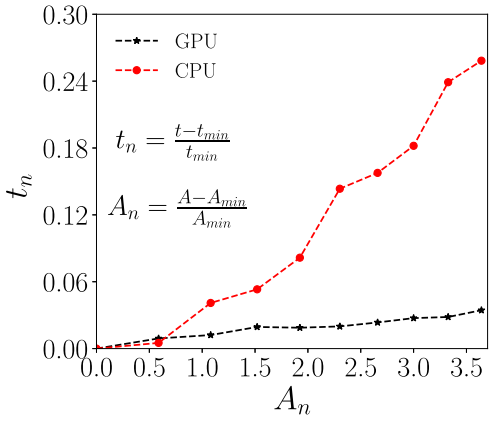


Fig. 9. Performance of the functions composing the VoF code section, expressed through normalized time t_n as a function of the normalized area A_n, where each variable increment is normalised by its minimum value. Each point represents a simulation performed on a problem at 512^3 grid points for 100 time steps. In order to increase the area, different spheres have been initialized with increasing volume, going from 5% to 50% of the total domain volume. The GPU simulations were performed on MeluXina on 1 node (4 GPUs), while the CPU simulations were performed on Berzelius (SNIC) on 1 node (128 CPUs).

6.3. VoF performances

So far, we have presented results assuming that the total amount of operations is exactly the same for all points and only depends on the total number of grid points. This is valid for most code sections (such as solver and advection/diffusion terms), but not for the VoF calculations, when the amount of operations is indeed constant for each point at the interface, but the number of interfacial points changes with the physical problem under study. Increasing the surface area generally implies higher computational costs, as each interface point is computed sequentially through *for-loops*. Finally, the total time associated with the VoF computation is equal to the total time used for the parallel process with a higher concentration of interface points.

On GPUs, the computation of the VoF is significantly different and the total amount of points at the interface has only a weak effect on the overall code performance. To illustrate this, we show in Fig. 9 the normalized execution time $t_n = (t - t_{min})/t_{min}$ (with t being the wall-time for the VoF code section) as a function of the normalized total area $A_n = (A - A_{min})/A_{min}$, where the normalization factor is the minimum variable value. Each point corresponds

to a 100 time steps simulation performed for a spherical droplet of volume ranging from 5% to 50% of the computational domain. The data in the figure clearly show that an increase of the total number of points at the interface does not imply higher computational costs on GPUs (roughly an overall 2.7% time increment), while on CPU the same simulations would require a significant increment of wall time.

The limited increment in computational cost on GPUs (compared to CPUs) can be explained by considering some basic principles in the way GPUs hardware and software work together. Let us present as example a simulation with a total of N_d grid points and N_s points at the interface, performed on a GPU which can simultaneously accommodate $N_{t,G}$ threads on a $N_{c,G}$ number of physical GPU cores. By definition $N_d >> N_s \gtrsim N_{c,G}$. To efficiently exploit parallelism on GPU, the machine schedules (or *launches*) many more GPU threads than physical GPU cores (meaning, $N_{t,G} >> N_{c,G}$). Each of the N_s surface points requires a long series of operations, while the treatment of the remaining $N_d - N_s$ points needs an almost negligible number of operations. When the VoF functions are launched on the GPU, all the domain points N_d will be mapped to GPU threads and rapidly fill all the available GPU cores $N_{c,G}$. During execution, the GPU will switch thread from active to inactive (and viceversa) in the attempt to keep GPU resources busy while memory operations are completed. Because $N_d >> N_{t,G}$, all the points that cannot be immediately dealt with are dynamically queued and executed as soon as one of the computing GPU core becomes available. The threads related to non-surface points will be rapidly executed and the available empty GPU cores filled with one of the queued threads, mapped either to surface or non-surface points. In other words, all the non-interface points will have an almost negligible computational overhead, while all the interface points will rapidly fill the available threads. As threads are efficiently queued, and assuming the amount of interface points is of a similar magnitude compared to the GPU cores, the overall computation time on the GPU varies very little with N_s. Finally, it is worth noticing that in the limit of extreme interface deformability (i.e. infinite Weber number), larger variations of computational costs may be observed when increasing the total area. However, these cases require higher resolution to correctly resolve the interface and all the relevant length and temporal scales, so that more GPUs should be used to avoid memory saturation. This will reduce the amount of interface points per GPU and thus reduce the risks associated to large interface deformability.

7. Applications

7.1. Emulsions in HIT

In many multiphase flows, the dispersed phase interacts with the surrounding turbulence induced through large-scale stirring mechanisms. While turbulence introduces a vast range of scales, usually spanning over several order of magnitudes, the presence of an interface introduces further complexity, offering alternative paths for energy transmission through scales and generating polydispersed droplet/bubble distributions. Resolving the interplay between all these mechanisms leads to an extremely complex scenario to simulate numerically. Even in simplified conditions, represented by Homogeneous and Isotropic Turbulence (HIT), the number of grid points could rapidly exceed $N \geq 1024^3$, quite challenging for multiphase flows. Furthermore, variations of density and viscosity, and variations of the surface tension coefficient may introduce smaller scales as lower viscosity in the dispersed phase may accelerate vortices. Fully developed turbulence is usually reached for Taylor-Reynolds number $Re_\lambda = u_{rms}\lambda/\nu \gtrsim 200$, where λ is the Taylor scale and u_{rms} is the root-mean-square fluctuation velocity. In multiphase flow simulations, these intensities are rarely reached as typical values are $Re_\lambda \lesssim 100$. In this example, we present a simulation of a turbulent emulsion at $Re_\lambda \approx 137$ performed on a grid of 512^3 on a cube of side length 2π and with the main physical parameters reported in Table 3. Turbulence is sustained at large scale using the Arnold-Beltrami-Childress forcing (see [66,67]) throughout the whole simulation.

Table 3
Physical dimensionless parameters of the present configuration: $Re_\lambda = u_{rms}\lambda/\nu_2$ and $We_\mathcal{L} = \rho_1 u_{rms}^2 \mathcal{L}/\sigma$ are the corresponding Taylor microscale Reynolds number and the large scale Weber number, $\mathcal{L}$ is the large-scale at which turbulence is forced, $\mathcal{V}$ is the volume fraction (ratio between the volume occupied by the disperse phase and the total volume). The simulation is performed at matching density and viscosity (i.e., $\rho_1/\rho_2 = \mu_1/\mu_2 = 1$).

Re_λ	$We_\mathcal{L}$	$\mathcal{V}$	$\mathcal{L}$	ρ_1/ρ_2	μ_1/μ_2
137	42.6	0.06	π	1	1

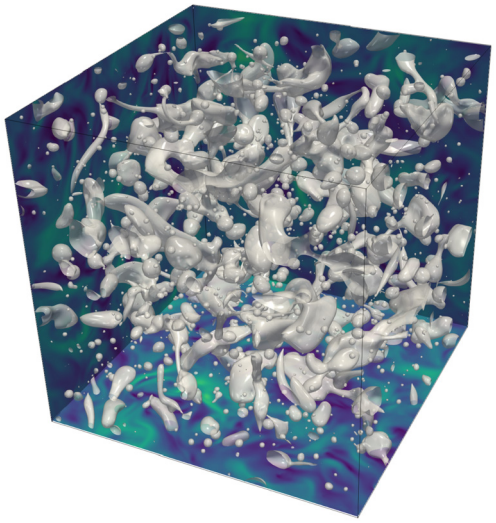


Fig. 10. Render for the iso-contour of the volume fraction value $\phi = 0.5$. Lateral planes show the modulus of the vorticity field.

Fig. 10 shows a render at statistically-stationary state. Turbulence is first simulated in single phase; when statistically-stationary conditions are reached, the dispersed phase is introduced through a random distribution of droplets and let to develop until convergence is reached. This is monitored in terms of droplet-size-distribution and one-dimensional spectra, see Fig. 11. Despite the simplicity of this configuration, HIT offers a relevant framework to study multiphase turbulence in complex configurations. To reach properly convergent statistics, several large-eddies turnovers are required, corresponding to $\sim 10^7$ time-steps. Hence, GPU acceleration will be invaluable to reach fully-developed turbulent conditions in future studies.

7.2. Two-layer Rayleigh-Bénard convection

Rayleigh–Bénard convection is the flow developed inside a fluid layer that is heated from below and cooled from above. It is driven by the density differences that arise due to the temperature variation inside the fluid. Even though it is a seemingly simple configuration, it encapsulates rich physics that are encountered in a range of engineering applications and physical phenomena. Beyond the classical setting, the study of the two-layer Rayleigh–Bénard variant is crucial from both a fundamental and an applied standpoint. First, regardless of the application, there is always some dissolved gas in every liquid. Therefore, it is inevitable that a gaseous phase will be formed in any realistic natural convection flow. Second, physical phenomena such as the convection in the earth's mantle [70] or engineering applications such as the heat transfer inside

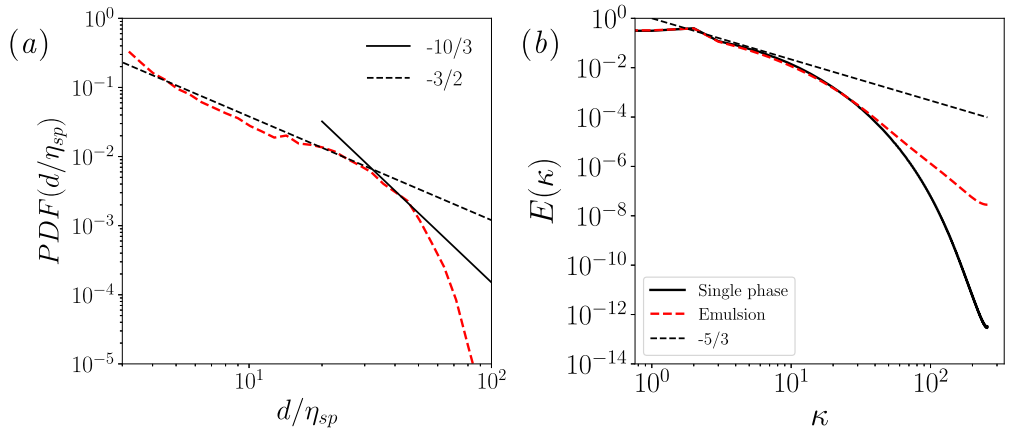


Fig. 11. Results at statistical-stationary state for turbulent emulsions simulation. Panel (a) shows the droplet size distribution, where the droplet diameter d is normalized by the single-phase Kolmogorov scale η_{sp}. Also in black, the $-10/3$ and $-3/2$ power-laws are shown (see [68,69]). Panel (b) shows the one-dimensional turbulent kinetic energy spectra, comparing the single-phase and multiphase (i.e. emulsion) cases. Here, the $-5/3$ law for the inertial range is shown, which applies for almost a decade.

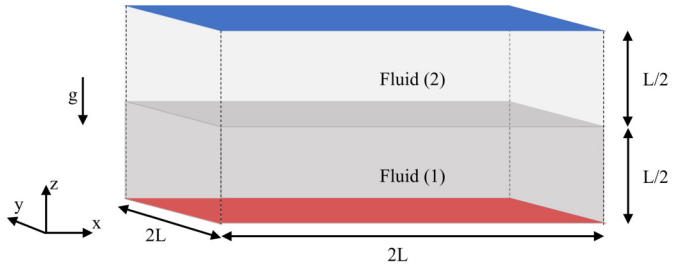


Fig. 12. Schematic representation of the geometry used for the two-layer Rayleigh–Bénard convection. The bottom heated surface is depicted in red and the top cooled surface in blue.

Table 4
Dimensionless parameters adopted for the study of two-layer Rayleigh–Bénard convection. All other property ratios are equal to 1.

$\lambda_\rho = \frac{\rho_2}{\rho_1}$	Pr= $\frac{\nu_1}{\alpha_1}$	Ra= $\frac{\lvert g\rvert \beta_1 \Delta T L^3}{\nu_1 \alpha_1}$	We= $\frac{\rho_1 \lvert g\rvert \beta_1 \Delta T L^2}{\sigma}$	Fr= $\sqrt{\beta_1 \Delta T}$
0.1	1	$10^6, 10^7, 10^8$	100	1

magnetic confinement systems in fusion reactors [71] are more accurately modelled as two-layer convection, where the two fluid layers are dynamically coupled.

Fig. 12 shows a schematic representation of the domain used for the numerical simulations, as used previously in [72], in two dimensions. The bottom and top walls are modelled as solid isothermal surfaces at constant temperatures of 328 K and 318 K respectively. The x- and y-directions are considered periodic, and the aspect ratio between the horizontal and vertical dimensions of the cavity is $\Gamma = L_x/L_z = L_y/L_z = 2$. The dimensionless parameters adopted are shown in Table 4. The property ratios between the two fluids are considered equal to 1 (kinematic viscosity ν, thermal diffusivity α, specific heat c_p and thermal expansion coefficient β), except the density ratio $\lambda_\rho = \rho_2/\rho_1 = 0.1$. This mismatch in densities is the reason behind the arrangement of the fluids in a two-layer configuration. Preliminary simulations revealed that a grid of $1024 \times 1024 \times 512$ cells and a CFL number of 0.50 were adequate for obtaining grid- and time step-independent solutions.

Fig. 13 shows instantaneous temperature fields in the x–y plane for the three Rayleigh numbers considered. With increasing Rayleigh number, the thermal structures in the cavity become finer, indicating increased turbulent activity. This increased thermal agitation is not enough to induce any significant interface movement, even at the highest Rayleigh number considered. Furthermore, in all three cases, the temperature drop at the bottom

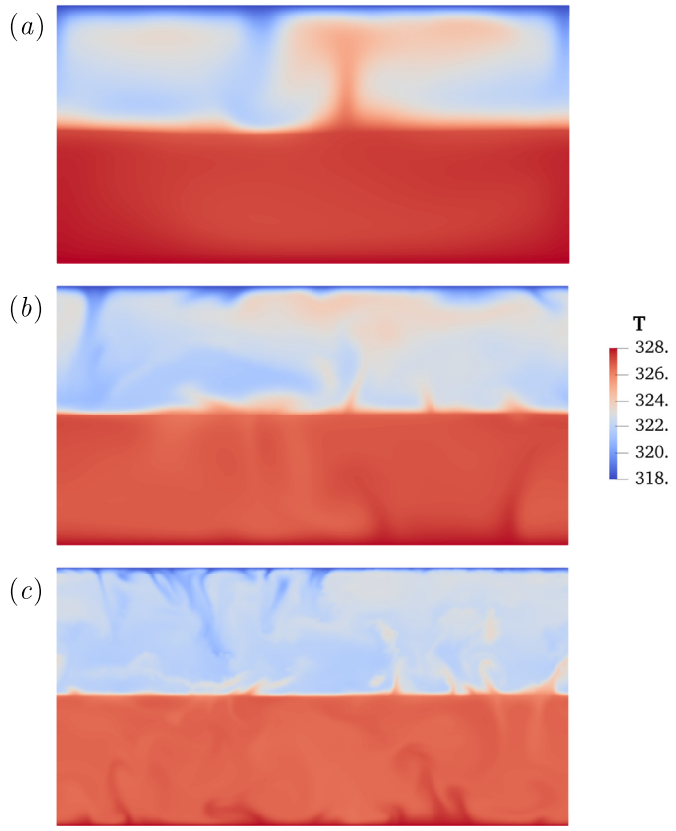


Fig. 13. Instantaneous temperature fields in the x–y plane for the two-layer Rayleigh–Bénard convection. (a) $Ra = 10^6$, (b) $Ra = 10^7$, (c) $Ra = 10^8$.

fluid layer is much smaller than the top layer. This is explained by the fact that the two layers have the same thermal diffusivity and specific heat but different densities, leading to a top layer with a smaller thermal conductivity ($k = \alpha \rho c_p$) compared to the bottom layer. Therefore, the top wall conducts heat less effectively than the bottom wall, which explains the larger temperature gradients at the top layer. Focusing on the highest Rayleigh number case, Fig. 14 shows instantaneous temperature contours for $Ra = 10^8$. As in classical Rayleigh–Bénard convection, hot and cold plumes are ejected from the bottom and top walls. In the presence of a single fluid layer, these plumes would typically get organized in large scale circulation structures, extending from the bottom to the top wall. Here, the existence of two fluid layers changes the clas-

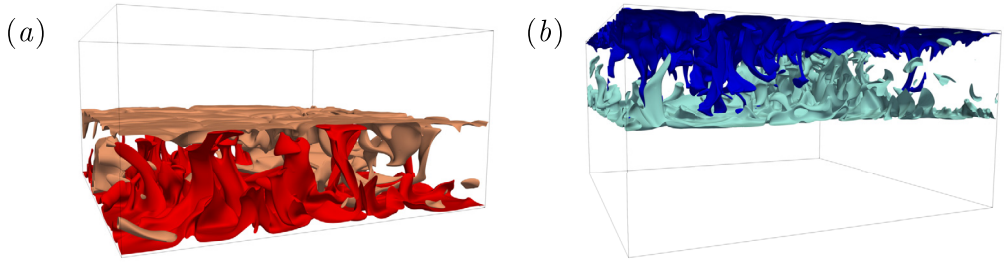


Fig. 14. Instantaneous temperature contours for the two-layer Rayleigh–Bénard convection at $Ra = 10^8$. (*a*) Bottom half of the cavity with red plumes at 327 K and orange plumes at 326.5 K. (*b*) Top half of the cavity with cyan plumes at 323 K and blue plumes at 321 K.

sical picture; the interface acts as a barrier, confining the thermal plumes in each fluid layer. The interface also acts as a thermal conductor, promoting the exchange of heat between the two layers. More specifically, the hot plumes ascending from the bottom wall are cooled off when reaching the interface, forming colder plumes that travel downwards. The opposite is true at the top layer, where the descending cold plumes are heated by the interface and hotter plumes emerge from the interface travelling upwards. Fig. 14 clearly illustrates this behaviour, revealing the existence of regions dominated by ascending plumes and regions dominated by descending plumes, hinting towards the organization of the flow in three-dimensional large scale circulation structures in each layer.

8. Conclusions and further developments

We present the code FluTAS, a numerical framework tailored for direct numerical simulations of multiphase flows, with the option of heat transfer, able to run efficiently on CPU-based standard architectures and on GPU-based accelerated machines. The open source version, released under MIT license, includes a pressure-correction algorithm for two-phase flows extended with an algebraic Volume-of-Fluid method (MTHINC) for capturing the interface dynamics.

We provide here a description of the employed numerical algorithm, with details on the solution of the governing equations and of the advection of the interface. After presenting different validation benchmarks both in single and multiphase configurations, we discuss the code performance focusing on three aspects: i) its current limitation when the communications among GPUs in different nodes are considerable less efficient than the communication among GPUs within the same node, ii) its advantages compared to CPUs in terms of "time-to-solution", iii) the performance stability of the VoF code section on GPUs for increasing amount of interface. Finally, we report results from two configurations of fundamental interests in multiphase turbulence: emulsions in homogeneous isotropic turbulence and the two-layer Rayleigh-Bénard convection. In the future, we aim to improve the code maintainability and portability (both on CPU and GPU) and to release additional modules under development, e.g. weak compressibility and phase change [43,44]. Further efforts will be devoted to enhance the code performance on multiple GPUs nodes, reducing the current communication bottlenecks. To this end, a promising strategy is the one proposed in [73], i.e., implement the solution of the tridiagonal system for the third direction on a distributed memory. The main advantage of this approach is the elimination of *all-to-all* operations in the Poisson solver. This improvement combined with future enhancements in the software frameworks for collective data communications will allow tackling several multiphase problems while keeping an efficient use of the computational resources.

Declaration of competing interest

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

Data availability

Data will be made available on request.

Acknowledgements

We would also like to thank the staff at CINECA, notably Massimiliano Guarrasi and Fabio Pitarri for their help in the development of the GPU functionalities at an early stage of the work. We thank Francesco De Vita for the help with the first implementation of the VoF MTHINC method.

Funding

M.C-E., N.S., A.D. and L.B. acknowledge the support from the Swedish Research Council via the multidisciplinary research environment INTERFACE, Hybrid multiscale modelling of transport phenomena for energy efficient processes and the Grant No. 2016-06119. M.E.R. was supported by the JSPS KAKENHI Grant Number JP20K22402 and by the Okinawa Institute of Science and Technology Graduate University (OIST) with subsidy funding from the Cabinet Office, Government of Japan. P.C. was supported by the University of Iceland Recruitment Fund grant No. 1515-151341, *TURBBLY*. The computer time was provided by SNIC (Swedish National Infrastructure for Computing), by the National Infrastructure for High Performance Computing and Data Storage in Norway (project No. NN9561K) and by CINECA (Marconi100) in Italy. Scaling tests on multiple GPUs has been performed on Berzelius (under the project No. Berzelius-2022-29 and operated by SNIC) and MeluXina (under the project number No. EHPC-BEN-2022B01-027 with the EuroHPC Benchmark Access and managed by LXP). M.E.R. acknowledges the computer time provided by the Scientific Computing section of Research Support Division at OIST.

Appendix A. Implementation details of MTHINC method

A.1. Calculation of the reconstructing polynomial

In the current work, we consider a polynomial up to second order and, therefore, $\mathcal{T}$ can be expressed using the following general quadratic form:

$$\mathcal{T}(\tilde{\mathbf{x}}) = \mathbf{c}\cdot\left[\mathcal{Q}_{\mathcal{T},1}\cdot\left(\tilde{\mathbf{x}}\cdot\tilde{\mathbf{x}}^T\right)^T\right] + \mathbf{c}\cdot\mathbf{c}^R\cdot\mathcal{Q}_{\mathcal{T},2}\cdot\left(\tilde{\mathbf{x}}\cdot\tilde{\mathbf{x}}^R\right)^T + \mathcal{L}_{\mathcal{T}}\cdot\tilde{\mathbf{x}}^T. \tag{A.1}$$

where $\tilde{\mathbf{x}}^R = (\tilde{y}, \tilde{z}, \tilde{x})$ while the components of the flag vectors (equal to 0 or 1) $\mathbf{c}$ and $\mathbf{c}^R$ will be discussed later in Appendix A.1.2. Accordingly, to determine $\mathcal{T}$, one needs just to compute the components of the vectors $\mathcal{Q}_{\mathcal{T},i=1,2}$ and $\mathcal{L}_{\mathcal{T}}$. This can be done by first imposing that the first-order and second-order gradient of $\mathcal{T}$, evaluated for $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_c$, are equal to the normal vector $\mathbf{n}$ and the curvature tensor $\mathbf{K}$:

$$\begin{cases} \nabla\mathcal{T}|_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_c} & = \mathbf{n}, \\ \nabla\left(\nabla\mathcal{T}\right)|_{\tilde{\mathbf{x}}=\tilde{\mathbf{x}}_c} & = \mathbf{K}. \end{cases} \tag{A.2}$$

Expanding equation (A.1) and the equations (A.2) lead after some manipulation to a unique expression for the components of the vectors $\mathcal{Q}_{\mathcal{T},i=1,2}$ and $\mathcal{L}_{\mathcal{T}}$:

$$\mathcal{Q}_{\mathcal{T},1} = \left[a^x_{Q,1}, a^y_{Q,1}, a^z_{Q,1}\right] = \frac{1}{2}\left[K^{xx}, K^{yy}, K^{zz}\right], \tag{A.3}$$

$$\mathcal{Q}_{\mathcal{T},2} = \left[a^x_{Q,2}, a^y_{Q,2}, a^z_{Q,2}\right] = \left[K^{xy}, K^{yz}, K^{xz}\right], \tag{A.4}$$

$$\begin{aligned} \mathcal{L}_{\mathcal{T}} = \left[a^x_L, a^y_L, a^z_L\right] = \Bigg[& n^x - \frac{c^x}{2}\left(K^{xx} + c^y K^{xy} + c^z K^{xz}\right), \\ & n^y - \frac{c^y}{2}\left(K^{yy} + c^x K^{xy} + c^z K^{xz}\right), \\ & n^z - \frac{c^z}{2}\left(K^{zz} + c^x K^{xz} + c^y K^{yz}\right)\Bigg]. \end{aligned} \tag{A.5}$$

Note that to recover a linear reconstruction, the curvature tensor is set identically zero and thus also the components of the vector $\mathcal{Q}_{\mathcal{T},i=1,2}$ in equations (A.3) and (A.4). Accordingly, from equation (A.5), $\mathcal{L}_{\mathcal{T}} = [n^x, n^y, n^z]$ and equation (A.1) reduces to $\mathcal{T}(\tilde{\mathbf{x}}) = \mathcal{L}_{\mathcal{T}} \cdot \tilde{\mathbf{x}}^T$ which represents the equation of a plane in the three-dimensional Cartesian space.

A.1.1. Normal vector and curvature calculation

Given the smooth nature of the colour function, the normal vector $\mathbf{n}$ and the curvature tensor $\mathbf{K}$ can be computed directly from the corresponding geometrical definitions:

$$\mathbf{n} = \frac{\nabla\phi}{|\nabla\phi|}, \tag{A.6}$$

$$\mathbf{K} = -\nabla\mathbf{n}, \tag{A.7}$$

where $\mathbf{m} = \nabla\phi = (m^x, m^y, m^z)$. Following the Youngs' method [74,75], the three components m^x, m^y and m^z (the partial derivative of ϕ in each direction) are computed by first evaluating the derivatives at the cell corners and then each corner-value is averaged to find $\partial\phi/\partial x$, $\partial\phi/\partial y$ and $\partial\phi/\partial z$. Once $\mathbf{n}$ is known, the curvature tensor components are computed using (A.7) whereas the geometrical curvature is derived from the sum of the three diagonal components of $\mathbf{K}$, i.e., $\kappa = -\left(K^{xx} + K^{yy} + K^{zz}\right)$.

A.1.2. Calculation of normalization parameter d_{th}

Once $\mathcal{T}$ is known, the last step to obtain $H(\tilde{\mathbf{x}})$ is to compute the normalization parameter d_{th}. To this purpose, we impose the volume conservation at discrete level by setting that the volume integral of $\hat{H}$ over local grid cell is equal to the VoF field in that cell. Using the normalized Cartesian coordinate, this results in:

$$\mathcal{I}: \int_0^1\int_0^1\int_0^1 \hat{H}\, d\tilde{x}\, d\tilde{y}\, d\tilde{z} = \phi. \tag{A.8}$$

As remarked in [1], an exact integration of (A.8) is not possible while it exists for the one dimensional integration. Taking for example the $\tilde{x}$ direction for an exact integration, we get:

$$\int_0^1 \hat{H}(\tilde{x})\, d\tilde{x} = \frac{1}{2}\log\left[\tilde{x} + \frac{\log\left(\cosh\left(\beta_{th}\left(\mathcal{T}\left(\tilde{x},\tilde{y},\tilde{z}\right)\right)\right)\right)}{\beta_{th}\left(\partial\mathcal{T}_{\tilde{x}}/\partial\tilde{x}\right)}\right]_0^1. \tag{A.9}$$

Note that in the integration of (A.9), we assume that the derivative of $\mathcal{T}$ with respect to $\tilde{x}$ just depends on $\tilde{y}$ and $\tilde{z}$. This is achieved by setting $\mathbf{c} = [0,1,1]$ and $\mathbf{c}^R = [1,1,0]$. In the other two directions, numerical integration should be performed and in this work we employ a two-point Gaussian quadrature method. Therefore, the integral $\mathcal{I}$ in equation (A.8) results in:

$$8\phi = \sum_{p=1}^{4}\left[\left(\tilde{x} + \frac{\log\left(\cosh\left(\beta_{th}\left(\mathcal{T}\left(\tilde{x}, r_p(p), r_m(p)\right) + d_{th}\right)\right)\right)}{a^x_L}\right)\right]_0^1, \tag{A.10}$$

where $r_p = 1 + \sqrt{3}/2[-1,+1,-1,+1]$ and $r_m = 1 + \sqrt{3}/2[-1,-1,+1,+1]$. In general, the direction along which the exact integration is performed cannot be decided a-priori. On the other hand, a criterion for this choice is based on the magnitude of the normal vector components and, therefore, three cases are possible.

Case 1

If, $|n^x| \geq (|n^y|, |n^z|)$, the exact integration is performed only along x and we set $\mathbf{c} = [0,1,1]$ and $\mathbf{c}^R = [1,1,0]$ in equation (A.1). Therefore $\mathcal{I}$ becomes:

$$8\phi = \sum_{p=1}^{4}\left[\left(\tilde{x} + \frac{\log\left(\cosh\left(\beta_{th}\left(\mathcal{T}\left(\tilde{x}, r_p(p), r_m(p)\right) + d_{th}\right)\right)\right)}{a^x_L}\right)\right]_0^1. \tag{A.11}$$

Case 2

On the other hand, if $|n^y| \geq (|n^x|, |n^z|)$, the exact integration is performed only along y and we set $\mathbf{c} = [1,0,1]$ and $\mathbf{c}^R = [0,1,1]$ in equation (A.1). Therefore, $\mathcal{I}$ becomes:

$$8\phi = \sum_{p=1}^{4}\left[\left(\tilde{y} + \frac{\log\left(\cosh\left(\beta_{th}\left(\mathcal{T}\left(r_p(p), \tilde{y}, r_m(p)\right) + d_{th}\right)\right)\right)}{a^y_L}\right)\right]_0^1. \tag{A.12}$$

Case 3

Finally, if $|n^z| \geq (|n^x|, |n^y|)$, the exact integration is performed only along z and we set $\mathbf{c} = [0,0,1]$ and $\mathbf{c}^R = [1,0,1]$ in equation (A.1). Therefore, $\mathcal{I}$ results:

$$8\phi = \sum_{p=1}^{4}\left[\left(\tilde{z} + \frac{\log\left(\cosh\left(\beta_{th}\left(\mathcal{T}\left(r_p(p), r_m(p), \tilde{z}\right) + d_{th}\right)\right)\right)}{a^z_L}\right)\right]_0^1. \tag{A.13}$$

Depending on the three different cases, equations (A.11) or (A.12) and or (A.13) can be solved for the only unknown d_{th}. To this purposed, they can be re-written first as:

$$\begin{aligned} & 4 + \frac{1}{\beta_{th} a^p_L} \\ & \quad \times \log\left(\frac{(AB^- D + 1)(AB^+ D + 1)(AC^- D + 1)(AC^+ D + 1)}{A^2(B^- D + 1)(B^+ D + 1)(C^- D + 1)(C^+ D + 1)}\right) \\ & = 8\phi, \end{aligned} \tag{A.14}$$

where a^p_L is a^x_L, a^y_L or a^z_L according to the three cases above. Finally, equation (A.14) can be further recast in a more convenient quartic equation:

Table A.5
Coefficients a_L^p and $\mathbf{a}_v$ (as given by equations (A.3), (A.4) and (A.5)) needed to compute A, $B^{\pm}$, $C^{\pm}$ and D depending on the three cases.

Case	a_L^p	$\mathbf{a}_v$
1	a_L^x	$[a_{Q,2}^z, a_{Q,2}^z, a_{Q,1}^{yz}, a_L^y, a_L^z]^T$
2	a_L^y	$[a_{Q,2}^x, a_{Q,2}^z, a_{Q,1}^{xz}, a_L^x, a_L^z]^T$
3	a_L^z	$[a_{Q,2}^x, a_{Q,2}^y, a_{Q,1}^{xy}, a_L^x, a_L^y]^T$

Table A.6
Coefficients $\mathbf{r}_{p,B}$ and $\mathbf{r}_{p,C}$ needed to compute $B^{\pm}$ and $C^{\pm}$ depending on the three cases. Note that $r_p^+ = 1+\sqrt{3}/2$ and $r_p^- = 1-\sqrt{3}/2$.

Case	$\mathbf{r}_{p,B}$	$\mathbf{r}_{p,C}$
1	$[(r_p^{\pm})^2, (r_p^-)^2, (r_p^{\pm} r_p^-), r_p^{\pm}, r_p^-]^T$	$[(r_p^-)^2, (r_p^{\pm})^2, (r_p^- r_p^{\pm}), r_p^-, r_p^{\pm}]^T$
2	$[(r_p^{\pm})^2, (r_p^-)^2, (r_p^{\pm} r_p^-), r_p^{\pm}, r_p^-]^T$	$[(r_p^-)^2, (r_p^{\pm})^2, (r_p^- r_p^{\pm}), r_p^-, r_p^{\pm}]^T$
3	$[(r_p^{\pm})^2, (r_p^-)^2, (r_p^{\pm} r_p^-), r_p^{\pm}, r_p^-]^T$	$[(r_p^-)^2, (r_p^{\pm})^2, (r_p^- r_p^{\pm}), r_p^-, r_p^{\pm}]^T$

$$
\begin{aligned}
0 = & \underbrace{A^2 B^- B^+ C^- C^+ (A^2 - Q)}_{\alpha_4} D^4, \\
& + \underbrace{A^2 (B^- B^+ (C^- + C^+) + (B^- + B^+) C^- C^+)(A - Q)}_{\alpha_3} D^3, \\
& + \underbrace{A^2 ((B^- B^+)(C^- + C^+) + B^- B^+ + C^- C^+)(1 - Q)}_{\alpha_2} D^2, \\
& + \underbrace{A (B^- + B^+ + C^- + C^+)(1 - AQ)}_{\alpha_1} D, \\
& + \underbrace{(1 - A^2 Q)}_{\alpha_0}.
\end{aligned} \tag{A.15}
$$

where $D = \exp(2\beta_{th} d_{th})$ while the constants A, $B^{\pm}$, $C^{\pm}$ and Q are given by:

$$
\begin{cases}
A & = \exp\left(2\beta_{th} a_L^p\right), \\
B^{\pm} & = \exp(2\beta_{th} \mathbf{a}_v \cdot \mathbf{r}_{p,B}), \\
C^{\pm} & = \exp(2\beta_{th} \mathbf{a}_v \cdot \mathbf{r}_{p,C}), \\
Q & = \exp(4\beta_{th} a_L^p (2\phi - 1)).
\end{cases} \tag{A.16}
$$

The coefficients a_L^p and the expressions for $\mathbf{a}_v$, $\mathbf{r}_{p,B}$ and $\mathbf{r}_{p,C}$ are reported in Tables A.5 and A.6, differentiated among the three cases previously described. Once the coefficients $\alpha_{i=1,4}$ of the quartic equation (A.15) are computed, a solution for D can be found. In the current work, we adopt the approach proposed in [1]. Instead of computing all the four complex roots, we look for the real and positive solution, fulfilling the constraint given by equation (A.8). To this purpose, we first write equation (A.15) as:

$$
D^4 + \gamma_3 D^3 + \gamma_2 D + \gamma_1 D + \gamma_0 = 0, \tag{A.17}
$$

with $\gamma_3 = a_3/a_4$, $\gamma_2 = a_2/a_4$, $\gamma_1 = a_1/a_4$ and $\gamma_0 = a_0/a_4$. Next, equation (A.17) is recast in a quadratic form as:

$$
(D^2 + \varepsilon_1 x + \varepsilon_2)^2 - (\varepsilon_3 D + \varepsilon_4)^2 = 0, \tag{A.18}
$$

where $2\gamma_3 = 2\varepsilon_1$, $\gamma_2 = \varepsilon_1^2 + 2\varepsilon_2 - \varepsilon_3^2$, $\gamma_1 = 2\varepsilon_1\varepsilon_2 - 2\varepsilon_3\varepsilon_4$ and $\gamma_0 = \varepsilon^2 - \varepsilon_4^2 = \gamma_0$. Finally, by introducing the variable $z = 2\varepsilon_2$ and by comparing equation (A.17) with equation (A.18), a cubic equation can be derived:

$$
z^3 + \eta_2 z^2 + \eta_1 z + \eta_0 = 0, \tag{A.19}
$$

with $\eta_2 = -\gamma_2$, $\eta_1 = \gamma_1\gamma_3 - 4\gamma_0$ and $\eta_0 = \gamma_0(4\eta_2 - \eta_3^2) - \eta_1^2$. Equation (A.19) can be easily solved with the Cardano's formula. Excluding the complex solutions, setting $\lambda = -\eta_2^2/9 + \eta_1/3$, $\mu = 2\eta_2^3/27 - \eta_1\eta_2/3 + \eta_0$ and $\Delta = \lambda^2 + 4\mu^3$, the real one z_r is given by:

$$
z_r = \begin{cases}
\left(\dfrac{-\lambda + \sqrt{\Delta}}{2}\right)^{1/3} - \left(\dfrac{+\lambda + \sqrt{\Delta}}{2}\right)^{1/3} - \dfrac{\eta_2}{3} & \text{if } \Delta \geq 0, \\
2\sqrt{-\mu}\cos\left[\dfrac{1}{3}\tan^{-1}\left(\dfrac{-\Delta}{-\lambda}\right)\right] - \dfrac{\eta_2}{3} & \text{if } \Delta < 0.
\end{cases} \tag{A.20}
$$

Once z_r is known, the coefficients $\varepsilon_{i=1,4}$ of equation (A.18) can be found: $\varepsilon_1 = \gamma_1/2$, $\varepsilon_2 = z_r/2$, $\varepsilon_4 = \sqrt{\varepsilon_2^2 - \gamma_0}$ and $\varepsilon_3 = (-\gamma_3/2 + \varepsilon_1\varepsilon_2)/\varepsilon_4$.
The last step requires the solution of equation (A.18). Once more, four solutions are possible, but since $D > 0$ by numerical constrains, only one positive and real solution is acceptable. This can be easily computed as:

$$
D = \frac{-(\varepsilon_1 - \varepsilon_2) + \sqrt{(\varepsilon_1 - \varepsilon_2)^2 - 4(\varepsilon_2 - \varepsilon_4)}}{2}, \tag{A.21}
$$

from which, the normalization parameter d_{th} can be evaluated simply as:

$$
d_{th} = \frac{1}{2\beta_{th}} \log(D). \tag{A.22}
$$

A.2. Computation of the numerical flux

The approximate expression of H is also used for the calculation of the numerical fluxes in the interface advection step. These are evaluated as:

$$
f^x_{i\pm1/2,j,k} = \frac{1}{\Delta y \Delta z} \int_{t^n}^{t^{n+1}} \left[\int_{\Delta y} \int_{\Delta z} u H(\mathbf{x}, t)|^n_{i\pm1/2,j,k} \; dy \; dz \right] dt, \tag{A.23}
$$

$$
f^y_{i,j\pm1/2,k} = \frac{1}{\Delta x \Delta z} \int_{t^n}^{t^{n+1}} \left[\int_{\Delta x} \int_{\Delta z} v H(\mathbf{x}, t)|^n_{i,j\pm1/2,k} \; dx \; dz \right] dt, \tag{A.24}
$$

$$
f^z_{i,j,k\pm1/2} = \frac{1}{\Delta x \Delta y} \int_{t^n}^{t^{n+1}} \left[\int_{\Delta x} \int_{\Delta y} w H(\mathbf{x}, t)|^n_{i,j,k\pm1/2} \; dx \; dy \right] dt. \tag{A.25}
$$

Note that the use of equations (A.23), (A.24) and (A.25) is impractical since they contain both temporal and spatial integrations. For a more simple evaluation, the time integral is replaced by a space integral performing a change of variable. Taking as an example the integral along x and using the cell-centered coordinate system, we define

$$
\Delta\tilde{x}_{i\pm1/2,j,k} = \begin{cases}
\left[1 - \dfrac{\Delta t^{n+1}}{\Delta x} u_{i+1/2,j,k}; 1\right] & u_{i+1/2,j,k} \geq 0, \\
\left[0; -\dfrac{\Delta t^{n+1}}{\Delta x} u_{i+1/2,j,k}\right] & u_{i+1/2,j,k} < 0.
\end{cases} \tag{A.26}
$$

and we set $\Delta\tilde{y} = \Delta\tilde{z} = [0, 1]$. Moreover, the indicator function in equation (A.23) is approximated with the VOF function at the current time step, ϕ^n. Accordingly, $f^x_{i,j\pm1/2,k}$ can be computed as:

$$f^x_{i\pm1/2,j,k} = \begin{cases} +\Delta\tilde{x} \displaystyle\int_{\Delta\tilde{x}_{i+1/2}} \int_{\Delta\tilde{y}} \int_{\Delta\tilde{z}} \hat{H}^{x,n}_{i,j,k}(\phi^n) d\tilde{V} & u_{i+1/2,j,k} \geq 0, \\ -\Delta\tilde{x} \displaystyle\int_{\Delta\tilde{x}_{i-1/2}} \int_{\Delta\tilde{y}} \int_{\Delta\tilde{z}} \hat{H}^{x,n}_{i,j,k}(\phi^n) d\tilde{V} & u_{i-1/2,j,k} < 0. \end{cases} \tag{A.27}$$

where $d\tilde{V} = d\tilde{x}d\tilde{y}d\tilde{z}$. Likewise, $f^{y,n}_{i,j\pm1/2,k}$ and $f^{z,n}_{i,j,k\pm1/2}$ can be computed as:

$$f^y_{i,j\pm1/2,k} = \begin{cases} +\Delta\tilde{y} \displaystyle\int_{\Delta\tilde{y}_{i+1/2}} \int_{\Delta\tilde{y}} \int_{\Delta\tilde{z}} \hat{H}^{y,n}_{i,j,k}(\phi^x) d\tilde{V} & v_{i,j+1/2,k} \geq 0, \\ -\Delta\tilde{y} \displaystyle\int_{\Delta\tilde{y}_{i-1/2}} \int_{\Delta\tilde{y}} \int_{\Delta\tilde{z}} \hat{H}^{y,n}_{i,j,k}(\phi^x) d\tilde{V} & v_{i,j-1/2,k} < 0. \end{cases} \tag{A.28}$$

$$f^z_{i,j,k\pm1/2} = \begin{cases} +\Delta\tilde{z} \displaystyle\int_{\Delta\tilde{z}_{i+1/2}} \int_{\Delta\tilde{x}} \int_{\Delta\tilde{z}} \hat{H}^{z,n}_{i,j,k}(\phi^y) d\tilde{V} & w_{i,j,k+1/2} \geq 0, \\ -\Delta\tilde{z} \displaystyle\int_{\Delta\tilde{z}_{i-1/2}} \int_{\Delta\tilde{y}} \int_{\Delta\tilde{z}} \hat{H}^{z,n}_{i,j,k}(\phi^y) d\tilde{V} & w_{i,j,k-1/2} < 0. \end{cases} \tag{A.29}$$

with:

$$\Delta\tilde{y}_{i,j\pm1/2,k} = \begin{cases} \left[1 - \dfrac{\Delta t^{n+1}}{\Delta y} v_{i,j+1/2,k}; 1\right] & v_{i,j+1/2,k} \geq 0, \\ \left[0; -\dfrac{\Delta t^{n+1}}{\Delta y} v_{i,j+1/2,k}\right] & v_{i,j+1/2,k} < 0. \end{cases} \tag{A.30}$$

$$\Delta\tilde{z}_{i,j,k\pm1/2} = \begin{cases} \left[1 - \dfrac{\Delta t^{n+1}}{\Delta z} w_{i,j,k+1/2}; 1\right] & w_{i,j,k+1/2} \geq 0, \\ \left[0; -\dfrac{\Delta t^{n+1}}{\Delta z} w_{i,j,k+1/2}\right] & w_{i,j,k+1/2} < 0. \end{cases} \tag{A.31}$$

A.2.1. Overall VoF algorithm

Below, we report the overall VoF algorithm in the pseudocode 2. As a final remark, note that the entire algorithm has been described assuming that the first directional split is always oriented along x (i.e., $x \to y \to z$. Nevertheless, this solution proves to be only first-order accurate in time. To improve the time accuracy of the solution, one possibility is to alternate the splitting direction as suggested in [76].

Algorithm 2 Overall VoF algorithm.

1: Set $(u^x, u^y, u^z) = (u, v, w)$;
2: **for** $p = x, y, z$ **do**
3: Compute the numerical fluxes f^p;
4: Compute ϕ^p;
5: Set the boundary conditions on ϕ^p;
6: **if** $p! = z$ **then**
7: Using ϕ^p, update $\mathbf{n}$ and κ with the procedure described in A.1.1 and d_{th} with the procedure in A.1.2;
8: **end if**
9: **end for**
10: Compute ϕ^{n+1} using ϕ^x, ϕ^y and ϕ^z.
11: Using ϕ^{n+1}, update $\mathbf{n}$ and κ with the procedure described in A.1.1 and d_{th} with the procedure in A.1.2;

References

[1] S. Ii, K. Sugiyama, S. Takeuchi, S. Takagi, Y. Matsumoto, F. Xiao, J. Comput. Phys. 231 (2012) 2328–2358.
[2] W.W. Grabowski, L.-P. Wang, Annu. Rev. Fluid Mech. 45 (2013) 293–324.
[3] G. Seminara, Annu. Rev. Fluid Mech. 42 (2010) 43–66.
[4] L. Brandt, F. Coletti, Annu. Rev. Fluid Mech. 54 (2021).
[5] F. Veron, Annu. Rev. Fluid Mech. 47 (2015) 507–538.
[6] T. Dauxois, T. Peacock, P. Bauer, C.-c.P. Caulfield, C. Cenedese, C. Gorlé, G. Haller, G.N. Ivey, P.F. Linden, E. Meiburg, et al., Phys. Rev. Fluids 6 (2021) 020501.
[7] C.T. Crowe, CRC Press, 2005.
[8] G.A. Voth, A. Soldati, Annu. Rev. Fluid Mech. 49 (2017) 249–276.
[9] F. Risso, Annu. Rev. Fluid Mech. 50 (2018) 25–48.
[10] S. Elghobashi, Annu. Rev. Fluid Mech. 51 (2019) 217–244.
[11] V. Mathai, D. Lohse, C. Sun, Annu. Rev. Condens. Matter Phys. 11 (2020) 529–559.
[12] A.U.M. Masuk, A. Salibindla, R. Ni, Int. J. Multiph. Flow 120 (2019) 103088.
[13] A.K. Salibindla, A.U.M. Masuk, S. Tan, R. Ni, J. Fluid Mech. 894 (2020).
[14] A.U.M. Masuk, A.K. Salibindla, R. Ni, J. Fluid Mech. 910 (2021).
[15] S. Mirjalili, S.S. Jain, M. Dodd, Center Turbul. Res. Ann. Res. Briefs 2017 (2017) 13.
[16] S.O. Unverdi, G. Tryggvason, J. Comput. Phys. 100 (1992) 25–37.
[17] R. Scardovelli, S. Zaleski, Annu. Rev. Fluid Mech. 31 (1999) 567–603.
[18] D.M. Anderson, G.B. McFadden, A.A. Wheeler, Annu. Rev. Fluid Mech. 30 (1998) 139–165.
[19] J.A. Sethian, P. Smereka, Annu. Rev. Fluid Mech. 35 (2003) 341–372.
[20] A. Prosperetti, G. Tryggvason, Cambridge University Press, 2009.
[21] G. Soligo, A. Roccon, A. Soldati, J. Fluids Eng. 143 (2021).
[22] A. Khan, H. Sim, S.S. Vazhkudai, A.R. Butt, Y. Kim, in: The International Conference on High Performance Computing in Asia-Pacific Region, 2021, pp. 11–22.
[23] X. Zhu, E. Phillips, V. Spandan, J. Donners, G. Ruetsch, J. Romero, R. Ostilla-Mónico, Y. Yang, D. Lohse, R. Verzicco, et al., Comput. Phys. Commun. 229 (2018) 199–210.
[24] M. Bernardini, D. Modesti, F. Salvadore, S. Pirozzoli, Comput. Phys. Commun. 263 (2021) 107906.
[25] P. Costa, E. Phillips, L. Brandt, M. Fatica, Comput. Math. Appl. 81 (2021) 502–511.
[26] W. Aniszewski, T. Arrufat, M. Crialesi-Esposito, S. Dabiri, D. Fuster, Y. Ling, J. Lu, L. Malan, S. Pal, R. Scardovelli, et al., Comput. Phys. Commun. 263 (2021) 107849.
[27] P. Cifani, J. Kuerten, B. Geurts, Comput. Fluids 172 (2018) 67–83.
[28] K. Eisenschmidt, M. Ertl, H. Gomaa, C. Kieffer-Roth, C. Meister, P. Rauschenberger, M. Reitzle, K. Schlottke, B. Weigand, Appl. Math. Comput. 272 (2016) 508–517.
[29] O. Desjardins, G. Blanquart, G. Balarac, H. Pitsch, J. Comput. Phys. 227 (2008) 7125–7159.
[30] S. Popinet, J. Comput. Phys. 228 (2009) 5838–5866.
[31] S.H. Bryngelson, K. Schmidmayer, V. Coralic, J.C. Meng, K. Maeda, T. Colonius, Comput. Phys. Commun. (2020) 107396, https://doi.org/10.1016/j.cpc.2020.107396.
[32] P. Costa, Comput. Math. Appl. 76 (2018) 1853–1862.
[33] U. Schumann, R.A. Sweet, J. Comput. Phys. 75 (1988) 123–137.
[34] M.E. Rosti, F. De Vita, L. Brandt, Acta Mech. 230 (2019) 667–682.
[35] F. De Vita, M.E. Rosti, S. Caserta, L. Brandt, J. Fluid Mech. 880 (2019) 969–991.
[36] F. De Vita, M.E. Rosti, S. Caserta, L. Brandt, Soft Matter 16 (2020) 2854–2863.
[37] M.E. Rosti, S. Takagi, Phys. Fluids 33 (2021) 083319.
[38] M.E. Rosti, Z. Ge, S.S. Jain, M.S. Dodd, L. Brandt, J. Fluid Mech. 876 (2019) 962–984.
[39] M. Kozul, P.S. Costa, J.R. Dawson, L. Brandt, Phys. Rev. Fluids 5 (2020) 124302.
[40] M. Crialesi-Esposito, M.E. Rosti, S. Chibbaro, L. Brandt, J. Fluid Mech. 940 (2022), https://doi.org/10.1017/jfm.2022.179.
[41] I. Cannon, D. Izbassarov, O. Tammisola, L. Brandt, M.E. Rosti, Phys. Fluids 33 (2021) 085112.
[42] N. Scapin, P. Costa, L. Brandt, J. Comput. Phys. 407 (2020) 109251.
[43] F. Dalla Barba, N. Scapin, A.D. Demou, M.E. Rosti, F. Picano, L. Brandt, Comput. Fluids 216 (2021) 104789.
[44] N. Scapin, F. Dalla Barba, G. Lupo, M.E. Rosti, C. Duwig, L. Brandt, J. Fluid Mech. 934 (2022).
[45] M. Ishii, T. Hibiki, Springer Science & Business Media, 2010.
[46] F.H. Harlow, J.E. Welch, Phys. Fluids 8 (1965) 2182–2189.
[47] E.G. Puckett, A.S. Almgren, J.B. Bell, D.L. Marcus, W.J. Rider, J. Comput. Phys. 130 (1997) 269–282.
[48] E. Aulisa, S. Manservisi, R. Scardovelli, S. Zaleski, J. Comput. Phys. 192 (2003) 355–364.
[49] G.D. Weymouth, D.K.-P. Yue, J. Comput. Phys. 229 (2010) 2853–2865.
[50] M. Castro, B. Costa, W.S. Don, J. Comput. Phys. 230 (2011) 1766–1792.
[51] A.J. Chorin, Math. Comput. 22 (1968) 745–762.
[52] C. Frantzis, D.G. Grigoriadis, J. Comput. Phys. 376 (2019) 28–53.
[53] S. Dong, J. Shen, J. Comput. Phys. 231 (2012) 5788–5804.
[54] M.S. Dodd, A. Ferrante, J. Comput. Phys. 273 (2014) 416–434.

[55] P.N. Swarztrauber, SIAM Rev. 19 (1977) 490–501.
[56] J. Makhoul, IEEE Trans. Acoust. Speech Signal Process. 28 (1980) 27–34.
[57] M. Frigo, S.G. Johnson, in: Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181), vol. 3, IEEE, 1998, pp. 1381–1384.
[58] N. Li, S. Laizet, in: Cray User Group 2010 Conference, 2010, pp. 1–13.
[59] S. Turek, O. Mierka, K. Bäumler, in: European Conference on Numerical Mathematics and Advanced Applications, Springer, 2017, pp. 593–601.
[60] J. Armengol, F. Bannwart, J. Xamán, R. Santos, Int. J. Therm. Sci. 120 (2017) 63–79.
[61] G. de Vahl Davis, I. Jones, Int. J. Numer. Methods Fluids 3 (1983) 227–248.
[62] M. Leal, H. Machado, R. Cotta, Int. J. Heat Mass Transf. 43 (2000) 3977–3990.
[63] https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.7.pdf.
[64] https://luxprovide.lu/technical-structure/.
[65] https://www.nsc.liu.se/systems/berzelius/.
[66] Phys. D: Nonlinear Phenom. 75 (1994) 471–508, https://doi.org/10.1016/0167-2789(94)00031-X.
[67] P.D. Mininni, A. Alexakis, A. Pouquet, Phys. Rev. E, Stat. Nonlinear Soft Matter Phys. 74 (2006) 1, https://doi.org/10.1103/PhysRevE.74.016303.
[68] C. Garrett, M. Li, D. Farmer, J. Phys. Oceanogr. 30 (2000) 2163–2171, https://doi.org/10.1175/1520-0485(2000)030<2163:TCBBSS>2.0.CO;2.
[69] G.B. Deane, M.D. Stokes, Nature 418 (2002) 839–844, https://doi.org/10.1038/nature00967.
[70] F.H. Busse, Phys. Earth Planet. Inter. 24 (1981) 320–324.
[71] F. Wilczynski, D.W. Hughes, Phys. Rev. Fluids 4 (2019) 103502.
[72] H.-R. Liu, K.L. Chong, Q. Wang, C.S. Ng, R. Verzicco, D. Lohse, J. Fluid Mech. 913 (2021).
[73] S. Ha, J. Park, D. You, Comput. Phys. Commun. 265 (2021) 107999.
[74] D.L. Youngs, Numerical Methods for Fluid Dynamics, 1982.
[75] D.L. Youngs, Atomic Weapons Research Establishment (AWRE) Technical Report 44, 1984, 35.
[76] G. Strang, SIAM J. Numer. Anal. 5 (1968) 506–517.