Elsevier Editorial System(tm) for Journal Of Computational Physics
Manuscript Draft

Manuscript Number: JCOMP-D-13-01421R2

Title: Parallel Load Balancing Strategy for Volume-of-Fluid Methods on 3-D Unstructured Meshes

Article Type: Regular Article

Keywords: interface-capturing; load balancing; parallelization; unstructured mesh; Volume-of-Fluid

Corresponding Author: Mr. Lluís Jofre Cruanyes,

Corresponding Author's Institution: Technical University of Catalonia

First Author: Lluís Jofre Cruanyes

Order of Authors: Lluís Jofre Cruanyes; Ricard Borrell; Oriol Lehmkuhl; Assensi Oliva

Abstract: Volume-of-Fluid (VOF) is one of the methods of choice to reproduce the interface motion in the simulation of multi-fluid flows. One of its main strengths is its accuracy in capturing sharp interface geometries, although requiring for it a number of geometric calculations. Under these circumstances, achieving parallel performance on current supercomputers is a must. The main obstacle for the parallelization is that the computing costs are concentrated only in the discrete elements that lie on the interface between fluids. Consequently, if the interface is not homogeneously distributed throughout the domain, standard domain decomposition (DD) strategies lead to imbalanced workload distributions. In this paper, we present a new parallelization strategy for general unstructured VOF solvers, based on a dynamic load balancing process complementary to the underlying DD. Its parallel efficiency has been analyzed and compared to the DD one using up to 1024 CPU-cores on an Intel SandyBridge based supercomputer. The results obtained on the solution of several artificially generated test cases show a speedup of up to 12x with respect to the standard DD, depending on the interface size, the initial distribution and the number of parallel processes engaged. Moreover, the new parallelization strategy presented is of general purpose, therefore, it could be used to parallelize any VOF solver without requiring changes on the coupled flow solver. Finally, note that although designed for the VOF method, our approach could be easily adapted to other interface-capturing methods, such as the Level-Set, which may present similar workload imbalances.

SUMMARY OF CHANGES

Accordingly to the comments raised by the reviewer, the changes on the manuscript are:

1. As suggested by Reviewer #1: (1) Eq. 3 is valid only for divergence-free velocity fields, hence, this condition has been introduced in the text; (2) the limits of the sum in the numerator of Eqs. 9 - 11 were incorrect, therefore, they have been corrected to start at 0 and end at P-1; (3) the description of Fig. 10 has been improved.

2. The reviewer observed that Alg. 4 did not scale in terms of memory, since vector WI gathered information from all processes and, thus, grew with the size of the problem and number of CPU-cores. This may had become a bottleneck on larger scale tests. Therefore, we have developed a new version of the algorithm overcoming such limitations. The new version is based on non-blocking point-to-point communications. The new algorithm has substituted the previous one on the manuscript, and all tests for the LB strategy have been repeated and the results updated. Since Alg. 4 represents only 5% of the algorithm time, its upgrading has provided rather small improvements on the results. Even so, the important point is that the memory bottleneck has been eliminated.

3. Furthermore, we have corrected some minor errors and we have tried to improve the wording across all the manuscript.

ANSWER TO REVIEWER 1

First of all, we really want to thank all your comments and suggestions about our work. The manuscript has been revised and improved taking all of them into account. In the next paragraphs we would like to give you our point of view.

Reviewer #1: I would like to thank the authors for addressing my questions and concerns and submitting a revised manuscript. I would like to revise my earlier assessment that in practice the cost of the VOF algorithm is typically small compared to other portions of a coupled flow solver. While this is likely the case for Cartesian, split advection VoF schemes, the here employed unstructured meshes and/or unsplit VoF schemes on structured meshes may incur significant computational cost due to geometric operations making the load balancing strategy proposed in this manuscript a worthwhile contribution to the literature. Unfortunately, the referenced "Importance and Effectiveness" document was missing from the resubmission documents, thus I still think that dual-constraint load balancing would be an option that should outperform the DD approach and may or may-not outperform the here proposed algorithm. Furthermore, the shown results suffer from a shortcoming outlined below (see 2). In summary, I would be inclined towards recommending publication, however, I would like to ask the authors to address the following additional comments:

1. If I'm not mistaken, Eq. (3) is incorrect. It is valid only for divergence free velocity fields but this simplification wasn't introduced.

Your assessment is correct, consequently, we have introduced the divergence free condition: "*Assuming that the fluids are immiscible and that their movement is defined by a unique divergence-free velocity field, i.e.,* $u^k = \boldsymbol{u}$ *for each fluid k with* $\nabla \cdot \boldsymbol{u} = 0$ *, ...*".

2. I agree with the authors' general assessment that any portion of an algorithm that is not properly load balanced may eventually slow the overall computation on massively parallel systems. This does indeed motivate the need for a load balancing algorithm for the VOF portion. Unfortunately, the presented results do not include a massively parallel case although the employed hardware would support it. It would be worthwhile to test the approach on significantly more cores than the currently employed 1024. This might yield different results due to additional communication costs going over larger "distances" of the supercomputer. I would not make this an absolute requirement for publication, though.

Although the MareNostrum supercomputer of the Barcelona Supercomputing Center (BSC) is composed of approximately 50K processors, it is difficult, as an individual research group, to carry out persistent tests with significantly larger numbers of processors than 1024 CPU-cores, due to the high demand of computing resources for scientific projects. Since this was a new algorithmic approach for us and we required several tests to understand all the aspects of the algorithm, we decided to limit ourselves to 1024 CPU-cores, otherwise it would have been difficult to repeat the tests many times.

Nevertheless, the dimension of the problems considered is proportional to the number of CPU-cores used, note that the loads per CPU using 1024 CPU-cores were only 3, 24 and 192 interface cells for the 2x2x2, 4x4x4 and 8x8x8 case, respectively. These workloads are very low! so the effects of the communication costs can be clearly observed, using larger number of CPU-cores on larger problems may show similar results.

Finally, despite that any parallelization deficiency of an algorithm is potentially a performance killer on massively parallel systems, so it must be attended, we have already shown important benefits of our LB strategy on relatively modest number of CPU cores, which is the most common user case.

3. In Eq. 9, should the divisor not read (P+1) or alternative the sum in the numerator start at 1?

This error has been corrected. Now, the sum in the numerator starts at 0 and ends at P-1. The error has been also corrected in Eqs. 10 and 11.

4. Some portions of the load balancing algorithm appear not to scale concerning memory, for example WI. If this is the case, it should be noted in the text.

This observation has been very important in order to improve the algorithm!! It is is really true that on the previous version of Alg. 4, the algorithm for the tasks reassignment (considering weights) uses an Allghaterv communication to obtain the weight of all tasks being reassigned on each process. In this way, since all processes had the same input data, they could all evaluate the same distribution without requiring communications. However, this does not scale in memory! We have substituted this step by a new algorithm where each overloaded process evaluates the distribution of its tasks according with the initial global workload distribution. This algorithm is based on non-blocking point-to-point communications and perfectly scales concerning memory (see new Alg. 4).

In consequence, the tests with the new parallelization strategy (LB) have been repeated substituting the previous version of Alg. 4 by the new one. The new version of Alg. 4 results to be faster than the previous one, up to 33.5% for the cases with higher number of CPU-cores, however since the cost of Alg. 4 within the overall load balancing algorithm is only 5%, the effect of this optimization is not so noticeable. In any case, the most important aspect is that the memory scale bottleneck identified on the previous version has been eliminated.

5. The description of Fig.10 in the text does not agree with the Fig. For example, (1) the percentage of communication costs grows with the number of CPU cores is not true for 2x2x2 and 64, 128, 256 cores.

When describing the figure, we were focusing on the general picture, as stated in the beginning of the sentence. Even so, it is true that for case 2x2x2 with 128 and 256 cores, the description is not correct. Therefore, the sentence has been made more precise: "*(1) in general,* the *percentage of the communications cost grows with the number of CPU-cores (there are some exceptions)*". However the main conclusion of this figure is that the communications costs represent less than 1.7% of the total solution time for any case, so the conclusion is that the DD approach is not limited at all by the communication costs.

Significance and Novelty of this paper

Volume-of-Fluid (VOF) is one of the methods of choice to reproduce the interface motion in the simulation of multi-fluid flows. One of its main strengths is its accuracy on capturing sharp interface geometries, but, unfortunately, at high computing costs. Under these circumstances, achieving parallel performance on current supercomputers is a must.

The main obstacle for the parallelization is that the computing costs are concentrated in the discrete elements that lie on the interface between fluids. Consequently, if the interface is not homogeneously distributed throughout the domain, standard domain decomposition (DD) strategies lead to imbalanced workload distributions.

After an exhaustive search on the specific literature, we found that there are not many studies on the parallel performance of VOF methods, neither alternatives to the DD. Consequently, the motivation of the work presented in this paper was the development of a new parallelization strategy for general unstructured VOF solvers. After discarding other alternatives, such as adapting the mesh partition to the fluids interface distribution, we have adopted a methodology based on a load balancing process complementary to the underlying domain decomposition. It has been developed for the general case of unstructured meshes, and all the details to implement it are given. On its core, it includes a fast process to approximate the solution of the Np-complete load balancing problem with 99% precision in few iterations.

An exhaustive analysis and comparison of the standard domain decomposition and our load balancing strategy has been performed. Several test cases, based on grids of spheres (representing the interface between fluids) distributed within a cubic domain, have been generated in order to measure the influence of the initial imbalance and of the problem size. These tests have been executed on an Intel SandyBridge based supercomputer, engaging up to 1024 CPU-cores. We have observed speedups up to 12x respect to the DD for the most ill-conditioned situations, but even in situations where the interface is almost spread throughout all the domain, so the imbalance is not singularly detrimental, the speedup achieved is 1.5x in average.

With this scenario in mind, the new parallelization strategy presented may be a feasible option to be considered when solving multi-fluid flows by means of VOF methods. Moreover, although designed for the VOF method, our approach could be easily adapted to other interface-capturing methods, like the Level-Set, which may suffer from a similar workload imbalance.

Highlights:

- A new parallelization strategy for Volume-of-Fluid methods is presented.
- Based on a load balancing process complementary to the domain decomposition.
- Suitable for Cartesian and unstructured 3-D meshes.
- Speedup up to 12x with respect to the domain decomposition approach.
- Easily adaptable to other interface-capturing methods.

Parallel Load Balancing Strategy for Volume-of-Fluid Methods on 3-D Unstructured Meshes

Lluís Jofre[a], Ricard Borrell[a,b], Oriol Lehmkuhl[a,b], Assensi Oliva[a,*]

[a] *Heat and Mass Transfer Technological Center (CTTC), Universitat Politècnica de Catalunya - Barcelona Tech, ETSEIAT, Colom 11, 08222 Terrassa (Barcelona), Spain*
[b] *Termo Fluids S.L., Av. Jacquard 97 1-E, 08222 Terrassa (Barcelona), Spain*

Abstract

Volume-of-Fluid (VOF) is one of the methods of choice to reproduce the interface motion in the simulation of multi-fluid flows. One of its main strengths is its accuracy in capturing sharp interface geometries, although requiring for it a number of geometric calculations. Under these circumstances, achieving parallel performance on current supercomputers is a must. The main obstacle for the parallelization is that the computing costs are concentrated only in the discrete elements that lie on the interface between fluids. Consequently, if the interface is not homogeneously distributed throughout the domain, standard domain decomposition (DD) strategies lead to imbalanced workload distributions. In this paper, we present a new parallelization strategy for general unstructured VOF solvers, based on a dynamic load balancing process complementary to the underlying DD. Its parallel efficiency has been analyzed and compared to the DD one using up to 1024 CPU-cores on an Intel SandyBridge based supercomputer. The results obtained on the solution of several artificially generated test cases show a speedup of up to $\sim 12\times$ with respect to the standard DD, depending on the interface size, the initial distribution and the number of parallel processes engaged. Moreover, the new parallelization strategy presented is of general purpose, therefore, it could be used to parallelize any VOF solver without requiring changes on the coupled flow solver. Finally, note that although designed for the VOF method, our approach could be easily adapted to other interface-capturing methods, such as the Level-Set, which may present similar workload imbal-

*Corresponding author. Tel.: +34 93 739 81 92; Fax: +34 93 739 89 20.
Email address: cttc@cttc.upc.edu (Assensi Oliva)

ances.

Keywords: interface-capturing, load balancing, parallelization, unstructured mesh, Volume-of-Fluid

1. Introduction

The numerical simulation of immiscible multi-fluid flows is fundamental to better understand many phenomena of interest in different disciplines such as engineering, hydrodynamics, geophysics or fundamental physics. Typical examples are the simulation of sprays, injection processes, bubbles, breakup of drops, wave motion, etc. These type of flows are characterized by the existence of an interface, separating the different fluids, which needs to be reproduced by the simulation method. So far, different numerical methods exist to reproduce the interface motion. These can be classified into two main groups: interface-tracking and interface-capturing methods. On the one hand, the interface-tracking approaches chase the interface as it moves: (1) defining the interface as a boundary between subdomains of a moving mesh [1, 2, 3]; (2) following the Lagrangian trajectories of massless particles [4, 5, 6]. On the other hand, the interface-capturing approaches describe the motion of the interface by embedding the different fluids into a static mesh. In particular, from this last group, the two main options of choice are the Volume-of-Fluid (VOF) [7, 8, 9] and Level-Set (LS) [10, 11, 12] methods, as well as algorithms based on combinations of both. From all these options, this paper is focused on the VOF method. This is based on geometrically reconstruct the fluids interface and, according to it, evaluate the portion of advected volumetric flux corresponding to each fluid. Its major strength is the accuracy achieved by some of its implementations on capturing sharp interfaces and their complex deformation, including breakups, while complying with the volume preservation constraint. This accuracy results in high computational costs. However, in the last decade, with the increase of the available computing power, different interfacial problems have been successfully tackled using it. Examples are the simulation of the drop breakup phenomenon by Renardy [13], the bubble motion by Annaland et al. [14], the solution of wave impact problems by Kleefsman et al. [15] or the numerical study of primary and impinging jet atomizations by Fuster et al. [16], Tomar et al. [17] and Chen et al. [18].

In general, on the simulation of interfacial multi-fluid flows with VOF

methods, the computing costs are dominated by the Navier-Stokes (NS) flow solver, and specifically by the solution of the Poisson system derived from the incompressibility constraint. Even so, the cost of the VOF calculations is not negligible at all. Its relative weight depends on different factors, such as the algorithms chosen, the effectiveness of its implementation, the physical case being considered, the type of geometric discretization used, the computing system employed, etc. As an example, on the sequential simulation of the Richtmyer-Meshkov instability [19] with an unstructured tetrahedral mesh of 250K cells, our VOF solver represents 22% of the computing costs. A similar percentage was reported by Le Chenadec and Pitsch [20], on the solution of a diesel jet with a Cartesian mesh of $256\times256\times1152$ cells. Anyway, beyond the percentage obtained for any particular simulation, it is a certainty that, in the high performance computing context, the cost of the VOF calculations will become more and more important while the algorithmic solutions adopted disregard parallel performance issues. Besides, by contrast, many efforts are employed by the scientific community on the parallelization of NS flow solvers and, in particular, on Poisson solvers [21, 22]. Considering, for example, the aforementioned Richtmyer-Meshkov instability case, with the DD approach we measured a raise of the VOF cost up to 84% when engaging 128 CPU-cores while, with the new parallelization strategy presented in this paper, the percentage is kept at 24%.

The parallel performance limitations of the standard DD approach can also be observed in the work by Aráujo et al. [23] focused on the 3-D simulation of injection processes. Their tests show a maximum parallel efficiency of 50% with up to 80 CPU-cores, including both the momentum and the VOF solvers. Another study on parallel algorithms for multiphase flows is the work of Sussman [24], based on solving the pressure Poisson equation by means of a multi-level solver and the interface motion through a coupled LS and VOF method [25]. This last work, however, is mainly focused on the performance of the pressure solver and, after all, no more than 16 CPU-cores were used in the parallel performance tests. Surprisingly, we could not find other relevant works on the literature presenting new alternatives for the parallelization of VOF methods.

Broadening the literature search to LS-based interface-capturing approaches, we found an additional parallelization alternative studied by Herrmann [26], which may be adapted to VOF methods. In particular, LS methods require the solution of an extra partial differential equation (PDE) to maintain the interface sharp. Similarly to VOF methods, this interface re-initialization

process is not well balanced if the interface is not homogeneously distributed throughout the domain. Herrmann proposes to generate two independently adapted grids for the solution of the flow and interface motion, respectively. While no restrictions are imposed on the Navier-Stokes grid, an equidistant Cartesian grid is adopted for the interface motion solution, with enough resolution to ensure accuracy of the LS method at any part of the domain, avoiding the application of complex adaptive mesh refinement (AMR) algorithms. This configuration also simplifies the LS parallelization since, in order to achieve a good workload balance, tasks can be easily reassigned between parallel processes without geometric information exchange. This approach was tested on the solution of the Zalesak's disk case, obtaining a slightly sub-optimal speedup with up to 128 CPU-cores [26]. In a later work, Herrmann applied the same strategy on a multi-scale Eulerian/Lagrangian two-phase flow algorithm [27], where the LS grid method was used for the Eulerian part, the overall algorithm showed an excellent speedup with up to 2048 CPU-cores.

Therefore, considering the good parallel performance achieved by Herrmann with his load balancing strategy, our purpose has been to develop a similar strategy for the parallelization of VOF methods on general unstructured meshes. Moreover, we solve both the motion of the flow and of the interface in the same mesh, without imposing any restriction to it. Consequently, the load balancing algorithm and its computing profile undergo major changes with respect to the Herrmann approach. For instance, when a task is reassigned, in the Cartesian case no geometric information needs to be transmitted since the mesh is homogeneous, contrary, in the general case the geometric characteristics of the discrete elements engaged on the task need to be transmitted as well. Additionally, our load balancing approach is based on a precise optimization algorithm, rather than iteratively reassign tasks until some threshold imbalance is reached or the process stalls. Finally, note that although our algorithm has been developed for VOF methods, it could be easily adapted to the parallelization of LS methods on unstructured grids.

Hence, this paper presents a new strategy for the parallelization of VOF methods on unstructured meshes, which is based on a dynamic load balancing process complementary to the DD. The rest of the document is organized as follows: in the next section, the mathematical formulation of the VOF method on unstructured meshes is presented. The standard domain decomposition and our new load balancing parallelization strategy are detailed in

Sec. 3. An exhaustive analysis and comparison of the parallel performance issues of both methods are presented in Sec. 4. Finally, conclusions are drawn in Sec. 5.

2. Volume-of-Fluid method

Volume-of-Fluid methods capture the fluids interface by embedding it into a fixed grid. In particular, a fraction scalar field, C_k, is defined for each fluid k, determining the fraction of volume that it occupies within each grid cell. Basically, $C_k = 0$ for cells that do not contain fluid k, $C_k = 1$ for cells which only contain the k'th fluid, and finally $0 < C_k < 1$ if part but not all of a cell's volume is occupied by the k'th fluid. These cells in which different fluids coexist are referred to as interface cells. Indeed, C_k can be defined as the normalized integral of a fluid's characteristic function $C_k(\mathbf{x}, t)$, such that

$$C_k(\mathbf{x}, t) = \begin{cases} 1 & \text{if there is fluid } k \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathbf{x}$ is a position in space and t refers to time instant. Therefore, for each cell c, its k'th fluid volume fraction value is evaluated as

$$C_k[c, t] = \frac{\int C_k(\mathbf{x}, t) dV_c}{V_c}, \tag{2}$$

where V_c refers to the cell volume.

Assuming that the fluids are immiscible and that their movement is defined by a unique divergence-free velocity field, i.e., $u^k = \mathbf{u}$ for each fluid k with $\nabla \cdot \mathbf{u} = 0$, the interface motion can then be captured by solving the respective conservation equation

$$\frac{\partial C_k}{\partial t} + \nabla \cdot (C_k \mathbf{u}) = 0. \tag{3}$$

Applying the divergence theorem and using a first-order explicit time scheme, the relative discrete equation reads

$$C_k^{n+1} - C_k^n + \frac{1}{V_c} \sum_{f \in F(c)} V_{k,f}^n = 0, \tag{4}$$

where the superscript n refers to the discrete time level, $F(c)$ to the set of faces of cell c, and $V_{k,f}$ is the volumetric flow of fluid k across face f.

VOF methods are characterized by the geometric evaluation of the volumetric flows, which is split into two consecutive phases: (1) the interface reconstruction according to the volume fraction fields; (2) the evaluation of the advection of each fluid, in accordance with the velocity field and the interface geometry previously reconstructed. Both phases are described in more detail in the following subsections. Additionally, further details can be found in our previous work [28].

2.1. Interface reconstruction

In this work, the fluids interface is reconstructed following the piecewise linear interface calculation (PLIC) approach. This means that within each grid cell, the interface is represented by a plane described with the equation

$$\mathbf{n} \cdot \mathbf{x} - d = 0, \tag{5}$$

where $\mathbf{n}$ is a unit normal vector to the plane and d sets its position.

Specifically, we evaluate $\mathbf{n}$ by means of the standard first-order Youngs method [29]. This is based on the normalized gradient of the volume fraction scalar field,

$$\mathbf{n} = \frac{-\nabla C_k}{|\nabla C_k|}. \tag{6}$$

In particular, with the aim of obtaining smooth solutions avoiding sharp angles between adjacent planes, we evaluate the gradient by means of a vertex-connectivity least-squares method [30].

Once fixed the unitary normal vector $\mathbf{n}$, d is found by placing the plane at the position that fulfills the initial condition

$$C_k = \frac{V_k}{V_c}, \tag{7}$$

where V_k is the volume occupied by fluid k within the cell. Particularly, we perform this search by means of the iterative Brent's minimization method [31].

It is important to note that the interface reconstruction within each cell is an independent process. In other words, for any interface cell, given its geometric description and some values of the field C_k, its interface reconstruction can be evaluated independently. This is a crucial point for our load balancing strategy, since it means that the global reconstruction calculation can be decomposed into unitary tasks, which can be then reassigned

through the parallel processes in order to balance the workload. In particular, in the load balancing process we are only reassigning the evaluation of constant d, which is the most time-consuming part of the reconstruction process. Therefore, when the interface reconstruction within a cell is reassigned, the information to be transmitted is the geometric description of the cell and the corresponding values of the fields C_k and ∇C_k.

2.2. Interface advection

Once the interface has been reconstructed, the advection is performed by geometrically calculating the volumetric fluxes $V_{k,f}$; see Eq. 4. The interface geometry evaluated in the previous step is necessary in order to discriminate, in the zones where two or more fluids coexist, which part of the volumetric flux corresponds to each fluid. Note that when two fluids coexist, it is only necessary to advect one of them, the solution of the other is obtained as the complement. The steps required to evaluate the volumetric fluxes, $V_{k,f}$, at any face f are presented below:

1. *Quantify the total volumetric flux.* The value of the total advection volume is calculated as

$$V_f = |\mathbf{u}_f \cdot \mathbf{n}_f| A_f \Delta t, \tag{8}$$

   where Δt is the time step, $\mathbf{u}_f$ the velocity at face f, and $\mathbf{n}_f$ and A_f correspond, respectively, to the unit-outward normal and the area of face f. Particularly, in order to limit the stencil of neighboring cells engaged, the CFL restriction is fixed to one. Thus, the flux polyhedron will always be contained in the stencil of cells that share at least one vertex with the face being considered. Consequently, this is the stencil of neighboring cells being used on the calculations.
2. *Construct the polyhedron representing the volumetric flux.* A polyhedron with volume V_f is constructed over face f. In particular, we are employing a vertex-matched approach with the aim to minimize flux over/underlapping situations that degrade the volume conservation principle. This approach is based on setting the direction of the extrusion edges equal to the velocity vectors at the face vertices. A 2-D illustration of it is shown in Fig. 1a, while an extended description can be found in [28].

3. *Truncate the part of the volumetric flux polyhedron corresponding to each fluid.* If the polyhedron only contains one fluid, truncation is not necessary, since $V_{k,f}$ is equal to 0 or V_f; this situation is illustrated by cases A and C of Fig. 2, respectively. Therefore, in this case, computing costs are negligible. Otherwise, it is necessary to truncate the part of the polyhedron corresponding to each fluid; case B in Fig. 2. This operation is performed independently on each cell of the face neighboring cells stencil; see Fig. 1b. In particular, three actions are performed for each of these neighboring cells: (1) evaluate its intersection with the flux polyhedron; (2) if it is an interface cell, truncate the resulting polyhedron by the interface plane; (3) add to $V_{k,f}$ the volume of the polyhedron resulting from the two previous actions. Note that the basic geometric operation used in the first two steps is the truncation of a polyhedron by a plane. A general algorithm to perform it is described in the work by López et al. [32].

As in the reconstruction phase, the evaluation of the fluids advection through any face is an independent process. Therefore, the advection calculation can also be decomposed into unitary tasks. Note that in the load balancing process, we only count as unitary task the evaluation of the volumetric fluxes at faces with a neighboring interface cell. As explained above, the other cases are trivial and irrelevant in terms of computing cost. In particular, the information required to evaluate the fluids advection through any face is: the velocity vector at its vertices and, for all the elements of the stencil of neighboring cells, its geometric description, the respective volume fraction value and, in the case of being an interface cell, the interface reconstruction plane.

3. Parallelization strategy

3.1. Standard domain decomposition

The domain decomposition is a standard strategy for the parallel solution of PDEs. The initial discretized domain is divided into P subdomains with similar number of cells, distributed then between P parallel processes to perform the computations. The subset of discrete elements assigned to a parallel process is referred to as its *owned* elements, while the rest of elements are named *external*. Thus, for any parallel process, we may talk about owned

cells, owned nodes, owned components of a scalar field, external nodes, external faces, etc. Since the system of equations generally links unknowns owned by different subdomains, to perform calculations in parallel is necessary the transmission of data between parallel processes. Here we refer to the external elements required by any parallel process as its *halo* elements. Each parallel process obtains its halo elements from neighboring subdomains by means of communications throughout the network, referred to as halo updates. In particular, note that a halo element that varies on its owner parallel process needs to be updated before being used, otherwise, the parallel and sequential executions would differ.

The DD approach has been extensively used in many VOF-based codes for the simulation of immiscible multi-fluid flows; see for example [33, 34, 35]. Using the DD for the VOF calculations is relatively simple, since it is just necessary to define the halo requirements of the parallel processes and introduce some halo updates. Moreover, since the rest of calculations, like the solution of the Navier-Stokes equations, may be parallelized using the same DD strategy, it becomes easy to assemble the solution of the whole system. In particular, two halo updates are needed to solve Eq. 4: (1) to the volume fraction scalar fields, before the reconstruction phase; and (2) to the field composed of the interface reconstruction planes, before the advection phase.

Since in the VOF calculations the work is concentrated on the discrete elements around the fluids interface, the workload of the parallel processes will only be well balanced if the interface is homogeneously distributed through the different subdomains. Unfortunately, the contrary occurs in many situations. For example, the simulation of gas bubbles within a liquid media may produce really imbalanced distributions or, in hydrodynamics simulations, the sea surface is generally located in a specific zone of the domain, involving only the subdomains covering it. In particular, Fig. 3 illustrates an imbalanced situation for a simplified case where two fluids coexist in a discrete domain divided in four parts.

In order to overcome the degradation of the parallel performance produced by the load imbalance, a possible strategy is to adapt the mesh partition to the interface distribution. In cases with predictable and constant interface location this adaptive strategy can be very convenient. However, in a general case some drawbacks appear: (1) the location of the interface may be not known a priori; (2) it may vary during the simulation, having to readapt the domain partitions; (3) the new partition optimized for

VOF calculations may be inappropriate or perform poorly for other parts of the code. For example, the numerical simulation of gas bubbly flows [14] requires, usually, a random initialization of the bubbles pattern inside the domain. Thus, in these cases the adapted mesh partition cannot be evaluated a priori. Moreover, any possible adapted partition would no longer suit the pattern of the bubbles as they evolve in time, having to readapt the partition several times. In addition, it has been found that adapting the mesh partition to the interface distribution, instead of prioritizing the minimization of halo requirements, optimizes the parallelization of the VOF algorithm but can decrease significantly the parallel performance of the Navier-Stokes solver [36].

3.2. New parallelization strategy

We propose a new parallelization strategy based on a dynamic load balancing process that reduces the common imbalance obtained from the standard domain decomposition. With this objective in mind, some reconstruction and advection unitary tasks are reassigned to new parallel processes overpassing the initial mesh partition. Consequently, when an unitary task is reassigned to a new parallel process, all the discrete data required to perform it (geometric and algebraic information), need to be transmitted to the new committed parallel process.

Note that, in the advection process, the geometry of the interface around any face is required in order to discriminate the portion of the volumetric flux corresponding to each fluid (terms $V_{k,f}^n$ of Eq. 4). This coupling between the resconstruction and advection phases makes it difficult to perform only one communication episode for all the algorithm. Indeed, a second level of data transfer, after the interface reconstruction and before the advection, seems that would be inevitable to ensure the availability of the interface geometry around any face through which two or more fluids are advected. Under these circumstances, in order to avoid a complex data interdependence management and better adjust the result, we prefer to perform separately the load balancing of the reconstruction and advection phases.

The load balancing algorithm presented in this work consists in the five main steps outlined in the next items. Further details about them are given in the subsections below.

1. *Determine the workload.* Each parallel process, p, evaluates its workload, W_p. When the cost of the tasks is variable, weights are used in

order to optimize the accuracy of the assigned loads. Further details are given in Sec. 3.2.3.

2. *Define a new balanced distribution.* This is performed in two steps. First, an optimal workload per parallel process, W_{opt}, is determined taking into account possible overheads on the solution of the tasks being reassigned; see Sec. 3.2.5. Second, a new tasks distribution is determined according to the previous load per process target. The corresponding algorithm, namely Alg. 3, defines also the communication scheme to transfer the data.
3. *Move data.* The data needed to perform the reassigned tasks is transported accordingly to the scheme determined in the previous step. This redistribution is performed by means of non-blocking point-to-point communications. However, to avoid inconsistencies, any parallel process does not start the next step until the communications in which it is involved are completed. Buffering is used to group all the data transactions between two parallel processes in only one message, using an independent buffer for each communication.
4. *Solve VOF tasks.* These tasks may be a set of interface reconstructions within interface cells, or fluids advection evaluations at faces around the interface. The parallel processes committed to solve both external (received from other parallel processes) and owned tasks, start with the solution of the external ones. In this way, the communications required to send back the results to the owner processes can be overlapped with the solution of the owned tasks.
5. *Collect solutions.* The processes which reassigned part of their tasks to others, receive the solutions back in buffers and store them in the corresponding memory space.

To summarize, the main steps of our load balancing strategy are outlined in Alg. 1. Remaining details are attained in the following subsections. Note that Alg. 1 is applied twice: first on the reconstruction phase and, afterward, on the advection phase.

3.2.1. Analysis of the algorithm

The diagram shown in Fig. 4 illustrates the computing time distribution for the VOF algorithm using the new parallelization strategy. In particular, the test case represented is a translation applied to 64 spheres contained in a cubic domain discretized by means of an unstructured mesh of 1000K cells;

Algorithm 1 Parallel load balancing strategy

1: Determine the workload
2: Define a new balanced distribution
3: Move data
4: Solve VOF tasks
5: Collect solutions

see Fig. 8b. This test was executed using 128 CPU-cores. Note that the advection costs dominate the VOF execution, while the overhead produced by the load balancing is around $\sim$5%. In the rest of the tests presented in the next section, we have observed that the relative weight of the load balancing step varies with the number of parallel processes engaged. On the contrary, the ratio between the reconstruction and advection phases has shown to be almost constant, meaning that similar parallel performance is obtained for both.

More in detail, Fig. 5 shows the distribution of the computing time through the different steps of Alg. 1 for the advection phase. The left part of the figure illustrates the flowchart for a parallel process *overloaded*, i.e., which reassigns some of its tasks to others. While the right part represents an *underloaded* parallel process, receiving tasks from the overloaded ones. The height of each rectangular box is proportional to the cost of the corresponding step of Alg. 1. The communications between both groups are illustrated with lines or boxes across the two columns. These occur in steps 3 and 5 ("Move data" and "Collect solutions"). Note that the communications of step 5 are asynchronous and, consequently, are represented by means of a line that couples different levels of the flowcharts. Step 2 ("Define a new balanced distribution") is also represented with a horizontal box across both columns because collective communications are required to perform it. These three steps constitute the part of the algorithm which increases its cost with the number of parallel processes. Therefore, it becomes a degradation factor for the speedup. The rest of the algorithm can be executed independently by each parallel process and reduces its cost when the number of parallel processes increases. More details on these aspects are shown in the numerical tests of Sec. 4.

The main difference in the flowchart of the overloaded and underloaded parallel processes occurs around the data movement of step 3 ("Move data"). Before it, the first ones pack in buffers the information to be sent while the

last ones become idle. After it, the underloaded parallel processes need to unpack the required information from the received buffers before performing any external VOF task. Note that the tasks distribution can be balanced in order to compensate the overcosts produced by the unpacking operations and, hence, reduce idle times; see Sec. 3.2.4. The same situation is repeated on the communication required to collect the solution of the reassigned tasks in step 5 ("Collect solutions"). However, in this case, the size of the communication is much smaller and its cost, together with the required pack and unpack operations, is almost negligible. For this reason, they are all represented by means of a simple line.

Finally, note that all the steps of Alg. 1, except the solution of the VOF tasks (step 4), can be considered pure overcosts, since they are not part of the solution itself but part of the balancing process. However, in the next section it is demonstrated that these overcosts are widely outweighed by the gain achieved with the load balancing.

3.2.2. Buffering

The geometric and algebraic data required to perform the VOF tasks are heterogeneous and not continuously stored in memory. Consequently, in order to move them through the network, we have explicitly defined pack functions, to store them into communication buffers, and unpack functions, to read the received information and reconstruct the stored objects before performing the calculations. Moreover, buffers are also used to group all data moves between two parallel processes in only one message, and thus reduce latency costs.

In particular, we have optimized our implementation of the pack and unpack functions for unstructured tetrahedral meshes, which is the type of meshes that we have used in the numerical experiments. In this case, in order to reassign a reconstruction task, 16 floating point elements need to be sent: 12 accounting for the vertices components, 1 for the volume fraction value and 3 for the gradient of the volume fraction field; an example is illustrated in Fig. 6. Note that the faces of a tetrahedron are just defined by the different combinations of its vertices, therefore, it is not necessary to explicitly determine its composition. Similar strategies are possible for prismatic cells, however, in a general case with more complex polyhedra, information of the faces composition may be required for the cell description into the buffer.

On the other hand, the advection tasks require many more elements to be

transmitted. For each mesh face, any element of the stencil of neighboring cells sharing at least one vertex with it, could be engaged on the calculation of the fluids advection through it. However, in order to minimize the communication costs, we try to discard some of the neighboring cells that are not required for the calculations. In particular, we can restrict to the neighboring cells that: (1) contain the fluid being considered and (2) have at least one vertex at the upstream side of the face plane (with respect to the flux), since the volumetric flux polyhedron is built into that side; see Fig. 7. Therefore, for each reassigned advection task are packed: 9 floating point elements, describing the components of the velocity field in the face vertices, and up to 17 floating point elements for each engaged neighboring cell — 12 to describe its geometry, 1 for its volume fraction and, in case of interface cell, 4 more defining the interface plane.

3.2.3. Weight of a task

In the first step of Alg. 1 each parallel process evaluates its workload, W_p. This is performed by assigning a weight to each owned unitary task and then adding up all these weights.

In the reconstruction phase, different weights are not necessary because reconstructing the interface has almost the same cost for any interface cell. In this case, for each parallel process, p, the workload can be set equal to the number of its owned interface cells, N_p.

A different situation occurs in the evaluation of the fluids' advection. As explained in Sec. 2.2, the advection calculation in a face only has a significant cost when its flux polyhedron intersects an interface cell. The evaluation of the advection for the faces matching this condition composes the set of unitary tasks to be distributed. At any of these faces, the advection evaluation requires geometric calculations with its neighboring cells that meet three conditions: (1) share at least one vertex with the face, (2) contain the fluid being advected and (3) have no null intersection with the advection polyhedron. The number of cells of this subset may be a good approach for the weight of the corresponding unitary task. Nevertheless, since the evaluation of the workloads is not part of the solution but just part of the process to find a good distribution, it must be a relative fast process. On this regard, constructing the advection polyhedron and checking its intersection with the neighboring cells (condition 3) would be too costly. Consequently, when defining the weight of a task, we substitute condition 3 by the less restrictive but easier to evaluate condition: "being in the upstream side of

the face" — the same condition used in the pack function described in the previous subsection. For example, in Fig. 7, considering the advection of fluid 1, with our approximation the relative weight of the task represented would be 5, while the precise (but too costly to evaluate) weight is 3. Even so, as shown in the numerical tests, using weights has a clear positive impact on the load balancing of the advection phase.

3.2.4. Overcost of external tasks

On the "Solve VOF tasks" step of Alg. 1, external tasks have an additional cost due to the buffers unpacking. This overcost should be taken into account when defining the new balanced distribution. We introduce it by means of a coefficient α, such that when a task is reassigned the cost of its solution is multiplied by $(1 + \alpha)$. Note that only the overcost produced by the unpack process is taken into account. The communication costs are not included in the definition of α because affect both overloaded ("senders") and underloaded ("receivers") parallel processes, so they do not produce an additional imbalance. On the other hand, the overcost produced by the pack process, which is executed only in the overloaded processes, hardly can be compensated by the tasks distribution, because the subsequent communication synchronizes the parallel processes.

The coefficient α depends on the ratio between the cost of executing a VOF task, and the cost of the process of unpacking the data required to perform it. In the reconstruction phase, both magnitudes are almost constant for all reassigned tasks. On the contrary, both are variable in the advection phase. In this case, we evaluate α as an average overcost for all the reassigned tasks. Note that α mainly depends on the type of grid (e.g., orthogonal or tetrahedral), the VOF algorithm implemented and the computing equipments being used. While, on the other hand, since we are considering unitary tasks, α is independent of the mesh size, of the number of parallel processes and of the interface size and distribution within the domain. Under these conditions, we measure α by running a test with a rather coarse mesh and few parallel processes. This measurement is then valid for any execution with the same equipment and mesh type. In particular, for all the numerical tests shown in Sec. 4, the value of α for the reconstruction and advection phases was fixed to 0 and 0.1, respectively. Therefore, with the algorithms being used, in the reconstruction phase, the cost of the unpacking process is negligible, compared to the cost of finding the linear reconstruction of the interface (which requires an iterative Brent's root search). While, in

the advection phase, the average overcost produced by the unpacking process represents 10%.

3.2.5. Define a new balanced distribution

The second step of Alg. 1 is divided in the two substeps described in detail below.

Optimal workload per CPU. We first determine the optimal workload per parallel process, W_{opt}, independently of the particular reassignment of tasks required to achieve it.

When the cost of the external and owned tasks is equivalent ($\alpha = 0$), the theoretical optimum, referred as W^*_{opt}, is the average workload

$$W^*_{opt} = W_{avg} = \frac{\sum_{p=0}^{P-1} W_p}{P}. \tag{9}$$

Nevertheless, as stated in the previous subsection, in general, external tasks may suffer an overcost produced by the unpack process, which multiplies its cost by $1 + \alpha$, with $\alpha \geq 0$. This means that in general $W^*_{opt} \geq W_{avg}$. Under this circumstances, given an initial distribution, finding an optimal redistribution or, what is the same, an optimal workload per parallel process, becomes a NP-complete problem. Equivalent formulations of it can be found in [37]. Therefore, we have to focus on heuristic approaches.

In the present application, there is an important advantage: given an estimated optimal workload per process, W_{opt}, it can be easily determined if it is higher or lower than the theoretical optimum, W^*_{opt}. On the one hand, the leftover workload from parallel processes with $W_p > W_{opt}$ is

$$L\left(W_{opt}\right) = \sum_{p=0}^{P-1} max(0, W_p - W_{opt}). \tag{10}$$

On the other hand, the workload required to reach W_{opt} by the underloaded processes, is

$$R\left(W_{opt}\right) = \sum_{p=0}^{P-1} \frac{max(0, W_{opt} - W_p)}{1 + \alpha}. \tag{11}$$

Finally, the balance is

$$B\left(W_{opt}\right) = L\left(W_{opt}\right) - R\left(W_{opt}\right). \tag{12}$$

Hence, if $B\left(W_{opt}\right) = 0$, the optimal workload has been found. Otherwise, if $B\left(W_{opt}\right) > 0$, it means that $W_{opt} < W^*_{opt}$ and, finally, $B\left(W_{opt}\right) < 0$ indicates that $W_{opt} > W^*_{opt}$. In fact, $B\left(W_{opt}\right)$ is a continuous function and the more closer to zero is its value, the better is the corresponding approximation. Under these circumstances, a root finding algorithm can be used in order to approach the theoretical optimum. In particular, we adopt the simple and well known bisection method, detailed in Alg. 2.

Algorithm 2 Iterative calculation of the optimal weight per process

```
1: W^a_opt = W_avg
2: W^b_opt = (1 + α)W_avg
3: for 0 ≤ i < numIte do
4:     if B((W^a_opt + W^b_opt)/2) < 0 then
5:         W^b_opt = (W^a_opt + W^b_opt)/2
6:     else
7:         W^a_opt = (W^a_opt + W^b_opt)/2
8: W_opt = (W^a_opt + W^b_opt)/2
```

The bisection method requires two initial guesses, W^a_{opt} and W^b_{opt}, such that $B\left(W^a_{opt}\right)$ and $B\left(W^b_{opt}\right)$ have different sign. On the one hand, we can take $W^a_{opt} = W_{avg}$ with $B(W^a_{opt}) \geq 0$, since, as previously stated, $W_{avg} \leq W^*_{opt}$. On the other hand, given an initial tasks distribution and considering $\alpha > 0$, note that the optimal workload depends on the percentage of tasks that need to be reassigned. The larger the movements required, the larger the number of tasks that multiply its cost by $1+\alpha$ and, consequently, the larger becomes W^*_{opt}. In particular, if all the tasks were reassigned, the theoretical optimum would be $(1 + \alpha)W_{avg}$. However, this extreme is not possible because some tasks will always remain in its owner parallel process. Hence, we can affirm that $W^*_{opt} \leq (1 + \alpha)W_{avg}$ and, therefore, $B((1 + \alpha)W_{avg}) \leq 0$. In conclusion, we can take as initial guesses $W^a_{opt} = W_{avg}$ and $W^b_{opt} = (1+\alpha)W_{avg}$. Note that the length of the initial interval is αW_{avg}. In our case, this is 0 and $0.1W_{avg}$ for the reconstruction and advection phases, respectively. Therefore, in the advection case, since the initial maximal error is 10% and each iteration of the bisection method halves it, we can affirm that in 4 iterations the error of our approach is less than 1%. This precision is more than enough for our application context and, what is more, the cost of these 4 iterations is almost negligible compared to the overall solution time. For the reconstruction it is not necessary any iterative process, being $\alpha = 0$ the optimal solution is just the average workload.

Tasks reassignment algorithm. Once an optimal workload per parallel process is calculated with the algorithm defined above, it is necessary to determine a new distribution of tasks fulfilling it. With the aim of better understandability, in Alg. 3 we first describe this process for the case of tasks with equal cost.

For each parallel process p the only input of the Alg. 3 is its initial workload $W_p = N_p$, while the output are two arrays, $SendTo$ and $RecvFrom$, of dimension P, storing in the k'th position the number of tasks to be sent and to be received to/from process k, respectively. For instance, $SendTo[0] = 5$ would mean that the process being considered, i.e., process p, has to reassign 5 of its owned tasks to process 0. Note that, since we are assuming that all tasks have the same cost, it is not relevant which particular tasks are redistributed.

At the first line of the algorithm, a collective all-gather communication is performed in order to get the whole interface distribution on each parallel process. This is stored in an array I such that $I[k] = N_k$. The rest of the algorithm is executed independently at each parallel process. This implies that some calculations are repeated but, since their cost is very low, it is more efficient to replicate calculations rather than using additional communications.

In the second line of Alg. 3 it is executed the Alg. 2, described in the previous subsection, in order to find the optimal workload W_{opt}.

From lines 3 to 6, the vectors S and R of dimension P are evaluated, these contain respectively the number of owned tasks that will sent (reassign) and the number of external tasks that will receive each process on the redistribution. Their evaluation is straightforward from the comparison of $I[k]$ with N_{opt}, where N_{opt} refers to the closest integer to W_{opt}. Some adjustment may be necessary to minimize the errors produced by the integer round-offs. Note that for any $k \in [0, ..., P-1]$, it is not possible that both $S[k]$ and $R[k]$ are different than zero. For instance, $S[k] > 0$ indicates that process k is overloaded, i.e., $I[k] > N_{opt}$. Thus, some of its tasks need to be reassigned to other processes. Obviously, this means that it does not require additional external tasks, i.e., $R[k] = 0$. In the same way, if $R[k] > 0$ then $S[k] = 0$. Finally, at line 6, the total number of tasks to be reassigned on the load balancing process, referred as N_{re}, is evaluated as $N_{re} = \sum_p S[p]$, which equals $\sum_p R[p]$.

In the next loop of the algorithm, lines 7-16, the reassignment of tasks is organized. In detail, for each of the N_{re} tasks that need to be reassigned, an

overloaded and an underloaded parallel process are committed to send and receive it, respectively. This information is stored in the arrays $SendTask$ and $RecvTask$, of dimension N_{re}, storing in the i'th position the rank of the process sending and receiving the i'th reassigned task, respectively. There is not a unique form to organize this redistribution, in this case we arrange it by the rank of the parallel processes.

Finally, once the overall tasks redistribution is defined, the evaluation of the vectors $SendTo$ and $RecvFrom$, which define the particular communications involving process p, is straightforward. This is performed in the last loop of the algorithm, corresponding to lines 17-21.

Algorithm 3 Tasks reassignment for process p (weights not considered)

1: **AllGather** communication of initial tasks distribution: $I[k] = N_k$
2: Apply **Alg. 2** to find W_{opt}
3: **for** $0 \leq k < P$ **do**
4: $\quad S[k] = max\,(0, I[k] - N_{opt})$
5: $\quad R[k] = max\left(0, \frac{N_{opt} - I[k]}{1+\alpha}\right)$
6: $N_{re} = \sum_k S[k]$
7: $count\_send = count\_recv = 0$
8: **for** $0 \leq k < P$ **do**
9: $\quad$ **if** $S[k] > 0$ **then**
10: $\quad\quad$ **for** $0 \leq i < S[k]$ **do**
11: $\quad\quad\quad SendTask[count\_send] = k$
12: $\quad\quad\quad ++\,count\_send$
13: $\quad$ **else**
14: $\quad\quad$ **for** $0 \leq i < R[k]$ **do**
15: $\quad\quad\quad RecvTask[count\_recv] = k$
16: $\quad\quad\quad ++\,count\_recv$
17: **for** $0 \leq i < N_{re}$ **do**
18: $\quad$ **if** $SendTask[i] == p$ **then**
19: $\quad\quad ++\,SendTo[RecvTask[i]]$
20: $\quad$ **if** $RecvTask[i] == p$ **then**
21: $\quad\quad ++\,RecvFrom[SendTask[i]]$

In the case of tasks with different computing costs additional complexities need to be considered. In particular, the new distribution is defined according to the weight of the tasks being reassigned. This was not necessary in the previous case since all tasks had the same cost. The new implementation is shown in Alg. 4. For each parallel process, the inputs of the algorithm are its initial workload, W_p; its number of owned elements, N_p; and an array of dimension N_p containing the weight of each owned task, WI. The outputs

are the same $SendTo$ and $RecvFrom$ vectors obtained with Alg. 3. The steps of the new algorithm are described next.

At the first line of Alg. 4, a collective all-gather communication is performed in order to get the whole workload distribution on each parallel process. This is stored in the vector W such that $W[k] = W_k$. Subsequently, in the second line, it is executed the Alg. 2 described in the previous subsection in order to find the optimal workload, W_{opt}.

In the next loop, lines 3-5, the vectors S and R of dimension P are evaluated, these contain the workload that needs to be sent or received by each parallel process, respectively. Their evaluation is straightforward from the comparison of $W[k]$ with W_{opt}. According to this initial situation, in the next loop, lines 6-21, process p evaluates the vector $\overline{SendTo}$ or $\overline{RecvFrom}$, depending if it is an overloaded or an underloaded process. This could be considered as a continuous approach of our solution, containing the weight that will be sent or received by process p to/from the others. The final solution determines the elements that need to be reassigned to approximate this target. Note that this part of the algorithm is designed such that the processes find coherent solutions, so these vectors determine the processes that will need to communicate.

In the next part of the algorithm, comprising lines 22-33, the processes sending part of its workload find a discrete approximation to the target $\overline{SendTo}$ and store it in the vector $SendTo$. For instance, $SendTo[k] = 5$ means that process p will reassign 5 of its tasks to process k, and those tasks are such that the sum of its weights approaches $\overline{SendTo}[k]$. Some adjustment may be necessary in order to minimize the errors produced by the integer round-offs.

In the last loop, lines 34-38, non-blocking point-to-point communications are performed between each overloaded process and the receptors of its tasks in order to communicate the number of tasks that will be reassigned on the solution process, these values are stored in vector $RecvFrom$ by the receivers.

4. Numerical tests

In this section, the new load balancing (LB) strategy is tested and compared with the standard DD approach. Both methods have been implemented within the TermoFluids (TF) parallel CFD software platform [38]. Therefore, its comparison accounts only for differences on the parallelization

Algorithm 4 Tasks reassignment for process p (weights considered)

AllGather communication of initial workload distribution: $W[k] = W_k$
Apply **Alg. 2** to find W_{opt}
for $0 \leq k < P$ **do**
    $S[k] = max(0, W[k] - W_{opt})$
    $R[k] = max(0, W_{opt} - W[k])$
if $W_p > W_{opt}$ **then**
    for $0 \leq k < P$ **do**
        for $0 \leq j < P$ **do**
            $trans\_weight = min(S[k], R[j])$
            if $k == p$ **then**
                $\overline{SendTo}[j] = trans\_weight$
            $S[k] - = trans\_weight$
            $R[j] - = trans\_weight$
else
    for $0 \leq k < P$ **do**
        for $0 \leq j < P$ **do**
            $trans\_weight = min(R[k], S[j])$
            if $k == p$ **then**
                $\overline{RecvFrom}[j] = trans\_weight$
            $R[k] - = trans\_weight$
            $S[j] - = trans\_weight$
if $W_p > W_{opt}$ **then**
    $count\_solve = 0$
    for $0 \leq k < P$ **do**
        if $\overline{SendTo}[k] > 0$ **then**
            $count\_weight = 0$
            for $count\_solve \leq i < N_p$ **do**
                if $(count\_weight + WI[i]) \leq \overline{SendTo}[k]$ **then**
                $count\_weight + = WI[i]$
                $++SendTo[k]$
                else
                $count\_solve = i$
                break;
for $0 \leq k < P$ **do**
    if $\overline{SendTo}[k] > 0$ **then**
        ISend to process k the number of elements reassigned to it: $SendTo[k]$
    if $\overline{RecvFrom}[k] > 0$ **then**
        IRecv from process k the number of elements fo be solved from it: $RecvFrom[k]$

strategy. Tests have been performed on the IBM MareNostrum-III supercomputer at the Barcelona Supercomputing Center [39]. MareNostrum-III is based on Intel SandyBridge 8-core processors at 2.6 GHz (2 per node), iDataPlex Compute Racks, a Linux Operating System and an Infiniband

FDR10 interconnection network. The number of CPU-cores engaged in our numerical experiments ranges between 16 and 1024 units.

Since we are only interested in parallel performance issues, we consider a canonical test case consisting of a translation applied to a set of spheres, which represent an interface between two fluids, and are placed in a cubic domain; see Fig. 8. In this way, we can easily control the size and distribution of the interface within the domain, and measure their influence on the parallel performance. In a general case, the interface may be deformed by a shifting velocity field. However, the computing pattern of the VOF part of the code would be exactly the same than the one of our canonical test case. Therefore, the conclusions about the parallel performance of VOF algorithms derived from this paper are generic.

Our measurements have been obtained after averaging over several iterations of the same time step in order to avoid dispersion by canceling outlier results. In particular, the translation applied is defined by the vector $\mathbf{u}_t = 1/\sqrt{3}(1,1,1)$, the radius of the spheres is 0.0425 and they are uniformly distributed in a $1\times1\times1$ cubic domain. Unless otherwise stated, the underlying geometric discretization is a mesh of 1000K tetrahedral cells, and the domain decomposition is performed by means of the graph partitioning tool METIS [40]. In Fig. 8, three interface configurations used in the following numerical experiments are shown. Moreover, their detailed characteristics are given in Tab. 1.

The first test considered is the strong speedup of the complete VOF algorithm using the standard DD approach. Note that with the DD strategy, acceleration can only be achieved when the overall domain partition further splits the interface and, consequently, divides the VOF computing costs. Results are shown in Fig. 9 for the interface configurations mentioned above, ranging the number of CPU-cores between 16 and 1024. Two a priory expected trends are clearly observed: (1) the strong speedup improves with the size and the extension covered by the interface within the domain; (2) as in most of parallel algorithms, increasing the number of parallel processes engaged in the execution implies that the parallel efficiency (PE) falls. In particular, regarding the second trend, for the $2\times2\times2$ configuration the PE decreases from 59% (with 32 CPU-cores) down to 3% (with 1024 CPU-cores). In fact, the total acceleration achieved in this case from 16 to 1024 CPU-cores is around $2\times$, while the number of parallel processes increases $64\times$. Note that in this case the interface is relatively very small and concentrated around eight points; see Fig. 8a. The situation improves when the interface

covers a larger part of the domain and, consequently, a larger percentage of CPU-cores become involved in the VOF calculations: for the 4×4×4 configuration the PE varies from 87% to 11%, and for the 8×8×8 one from 92% to 48%. Note also that with 1024 CPU-cores the workload per parallel process is rather low: in ascending order of number of spheres, the ideal workload per process would be around 3, 24 and 192 interface cells, respectively. This fact relativizes the poor results achieved with the highest number of CPU-cores for the coarser interfaces. However, these small cases allow us to better analyze aspects of the speedup degradation that may become hidden when the computing costs dominate.

Considering the causes that degrade the acceleration of the DD approach, we have, on the one hand, the effects of the poor workload distribution and, on the other hand, the cost of the communications required on the halo updates. The influence of the second aspect is shown in Fig. 10, where the percentage of the communications cost over the total cost of the VOF algorithm is presented. Again, the general picture looks as expected: (1) in general, the percentage of the communications cost grows with the number of CPU-cores (there are some exceptions); (2) when the size of the interface grows, and thus do the computing costs, the relative weight of the communications falls. Nevertheless, the most relevant aspect shown in Fig. 10 is that in all cases the percentage of the communications cost is below 1.7%. Even when the interface is relatively very small and up to 1024 CPU-cores are used (case 2×2×2). The subsequent conclusion is that the communications cost is negligible compared to computations. Therefore, the acceleration depends only on the workload distribution condition. In other words, the cause of the PE drop is the decreasing percentage of CPU-cores involved on the VOF calculations as their number grows, rather than the associated communication costs. This statement is reinforced by the result shown in Fig. 11, where the imbalance obtained for each of the mesh partitions used in the previous tests is represented. The imbalance is evaluated as the difference between the number of interface cells of the most overloaded parallel process and the average of interface cells per process, divided by the latter. The values obtained agree with our statement. For example, looking at the 8×8×8 case, the imbalance obtained using 1024 CPU-cores is 1.2× the average number of interface cells, thus, the parallel process with maximum workload has to solve $(1\times)+(1.2\times) = 2.2\times$ the average. According to this, and assuming for simplicity an ideal load balance with 16 CPU-cores and an equal solution cost for all interface cells, the strong speedup obtained should be $64/2.2 = 29\times$,

which is close to the observed strong speedup of 30×.

The strong speedup is now analyzed for the new parallelization strategy, with identical test conditions to those set for the DD approach. The results, depicted in Fig 12, show a speedup qualitetively similar to that obtained with the DD algorithm, but quantitatively better. The improvement achieved is more noticeable the more imbalanced the case is. In the most extreme situation, using 1024 CPU-cores, the leap obtained in the PE by using the LB instead of the DD is from 48% to 67% for case 8×8×8, from 11% to 50% for case 4×4×4 and from 3% to 28% for case 2×2×2; see Figs. 9 and 12. This improvement is also evident with lower numbers of CPU-cores. For example, with 128 CPU-cores, the leap is from 70% to 92%, from 42% to 83% and from 16% to 62% for the 8×8×8, 4×4×4, and 2×2×2 interfaces, respectively. In conclusion, we observe that the LB strategy consistently outperforms the DD one.

In Fig. 13, as previously done for the DD strategy, we show the relative cost of the communications over the total cost of the VOF algorithm. These communications occur in the steps 2, 3 and 5 of Alg. 1. Again, as expected, the relative weight of the communications is proportional to the number of CPU-cores engaged and inversely proportional to the size of the interface, i.e., the workload. However, there is a major difference with respect to the results obtained for the DD parallelization: while the communications cost always represents less than 1.7% of the total time for the DD strategy, it reaches up to 50% with the LB one. In particular, in ascending order of number of spheres, with 1024 CPU-cores, communications represent 48%, 21% and 5% of the total time, respectively. Therefore, in contrast to what happens with the DD strategy, the speedup of the LB approach is limited by the cost of the communications required to move the data between parallel processes. Nevertheless, the degree of initial imbalance determines also the parallel performance, since the more imbalance there is, the larger is the amount of data that needs to be moved.

All tests presented up to this point refer to the strong speedup of the DD and LB strategies. Nevertheless, in order to contrast their real performance, we must compare their solution times instead of their acceleration with respect to themselves. Accordingly, the ratio between both solution times is shown in Fig. 14 for the test cases studied in the previous figures. At the initial point, with 16 CPU-cores, all cases present a certain imbalance that favors the LB approach. This produces a speedup that ranges from 1.14 for the 8×8×8 case up to 1.34 for the 2×2×2 one. The rest of values derive from

the differences already shown in the acceleration trends of both methods; see Figs. 9 and 12. Consequently, the initial speedup widens much more for the coarser interfaces. At the end, with 1024 CPU-cores, the speedup achieved by using our new approach ranges from 1.5× for the 8×8×8 configuration, up to 11.6× for the coarser interface case.

The next test is devoted to further analyze the effects of the initial interface distribution on the parallel performance. For this purpose, three new configurations are considered; see Fig. 15. On the one hand, the new configuration 4×8×8 is obtained by removing the spheres of the 8×8×8 interface that are located in one half of the domain. On the other hand, the configurations R-8×8×8 and R-4×8×8 are obtained by assigning random positions to the spheres of the respective grids, restricting them to one half of the domain for the case R-4×8×8. In Fig. 16, it is compared the VOF solution time for these three new configurations together with the 8×8×8 one, using 512 CPU-cores and both parallelization strategies. Analyzing first the effect of randomly placing the grids of spheres, i.e., comparing the first and second, and the third and forth columns of the DD and LB blocks in Fig. 16; it is clear that it produces a negative effect only for the DD strategy. The DD degradation was predictable, since setting the spheres positions randomly widens the imbalance. On the other hand, for the LB one, this additional imbalance should increase a little the data transfer requirements on the load balancing process. However, this effect is not perceptible in the solution time because, as shown in Fig 13, the weight of communications is relatively low in this case (∼5%). Note that we have carried out all the previous tests on grids of spheres uniformly distributed throughout the domain, the imbalance was therefore principally produced by the sparsity of the interface that leaves many processes with little or zero workload. As shown in the present test, random distributions may increase the imbalance and, thus, the performance of the LB with respect to the DD methodology.

In the test shown in Fig. 16 we have also proposed the artificially generated situation in which half of the domain is empty. By doing this, the VOF workload is almost halved. Therefore, we would ideally expect that the solution time was halved as well. The real effect is observed by comparing columns first and third, and second and fourth of the DD and LB blocks, respectively. Using the DD approach, there is no time reduction, since the processors of the non-emptied half retain its workload. On the other hand, with the LB strategy the workload is redistributed and the solution time is halved in both situations. This test shows the robustness of the new strat-

egy in situations of imbalance in which the parallel performance of the DD is seriously degraded.

In the above tests different interface configurations have been considered, however, the underlying 3-D discretization has been kept constant. The effects of varying it are shown in Fig. 17. In particular, the strong speedup of the LB algorithm on the solution of the 4×4×4 interface for two additional 3-D meshes of sizes 250K and 4000K, together with the results presented previously for the 1000K mesh, are depicted. When the 3-D discretization is varied, the number of interface cells also varies but in a lower degree, since the fluids interface is bidimensional. On the other hand, the distribution of the interface within the domain remains constant. However, this does not ensure that the partitions imbalance is the same, since the mesh partitioning is not based on geometrical criteria, but on topological criteria. The results obtained look as expected: the speedup improves by increasing the computing load, i.e., the mesh size. There are two main reasons for this: (1) the relative weight of the communications decreases; (2) the relative weight of any residual imbalance remaining after the load balancing process decreases as well. Particularly, the strong speedup results shown in Fig. 17 are very similar for all the 3-D meshes up to 128 CPU-cores. Indeed, in this range the communication overcosts remain rather low for all cases. As these costs grow and become more significant, differences appear on the speedup. For instance, with 1024 CPU-cores, where the differences are the largest, the communications cost represents 47%, 21% and 15% of the solution time for the 250K, 1000K and 4000K mesh, respectively. Additionally, note that with 1024 CPU-cores, for the 250K mesh the ideal workload per CPU-core is minimal, only around 9 interface cells. As a consequence, a residual imbalance of only one interface cell on the new distribution degrades the imbalance by 10%. On the contrary, for the 4000K mesh the ideal load per CPU grows up to 63 interface cells, so this potential degradation is below 1.5%. In any case, note that the situation described in this test is practically the same one which occurs when the interface is varied by increasing or decreasing its size on a fixed 3-D grid. In fact, since only the interface cells are engaged on the VOF calculations, the size of the 3-D mesh is only important as it determines the size of the interface.

Once our LB parallelization strategy has been extensively compared to the standard DD approach, the influence of the improvements introduced during the development of the algorithm are analyzed in the last test. In particular, these are two: (1) considering the overcost caused by the unpack

process on the solution of the reassigned tasks (coefficient α); (2) assigning a weight to each task according to its relative cost. Neither of these two optimizations are necessary on the reconstruction phase because, on the one hand, the unpack operation cost is negligible with respect to the reconstruction calculations ($\alpha = 0$) and, on the other, the reconstruction unitary tasks have all the same cost. Therefore, results are shown only for the advection phase. Indeed, with our implementation the advection phase represents always around 85% of the solution time, so it essentially determines the overall performance. In Fig. 18 the reduction achieved on the advection solution time by the different optimizations is shown: (1) considering coefficient α (LB Alpha); (2) introducing weights (LB Weight); (3) considering both optimizations together (LB Optimal). The tests are executed on the 1000K mesh for the 4×4×4 interface configuration, on the range of CPU-cores previously considered. At first sight, it is clear that the larger the CPU-cores engaged, the larger is the influence of the optimizations. Indeed, the optimizations are more relevant because the load balancing itself becomes more significant too. In particular, the benefit obtained by considering only the unpack overcosts (LB Alpha) is rather limited, not reaching 4%. In fact, there is an intrinsic limitation of 10%, since $\alpha = 0.1$. For VOF algorithms, geometric discretizations or computing systems that result in a larger α, this optimization could be much more important. On the other hand, using weights in order to optimize the balancing process (LB Weight) results in greater benefits that reach above 20%. Finally, by setting both optimizations together (LB Optimal), the benefits reach over 25%. Note also that both optimizations are mutually beneficial since, in general, the time reduction achieved by their interaction is superior to the sum of the reductions achieved separately.

5. Conclusions

A new parallelization strategy for VOF methods has been presented and studied in detail. It has been developed with the aim of overcoming the workload imbalance obtained with the standard domain decomposition when the fluids interface is not homogeneously distributed throughout the domain. Basically, it consists in a dynamic load balancing process, complementary to the underlying domain decomposition, that reassigns tasks from processes with higher workload to processes with lower workload. This process is applied separately to the reconstruction and advection phases of the VOF algorithm. Since the initial domain decomposition is surpassed and the algorithm is ap-

plied to general unstructured discretizations, all the geometric and algebraic data required to perform any reassigned task need to be transmitted with it. In particular, communications are managed by means of buffers, and specific pack and unpack functions to, respectively, read and write data from them. To better achieve the desired load balance, two important issues need to be considered: the variable cost of the tasks being distributed and the overcost produced when a task is reassigned. An optimal workload balance leads to an NP-complete problem, for which a fast heuristic has been found giving a solution with 99% precision in few steps. Moreover, all the algorithms necessary to implement the new strategy have been described in detail.

An exhaustive analysis and comparison of the standard domain decomposition and our load balancing strategy has been performed. Several test cases, based on grids of spheres (representing the interface between fluids) distributed within a cubic domain, have been generated in order to measure the influence of the initial imbalance and of the problem size. These tests have been executed in the MareNostrum-III supercomputer of the Barcelona Supercomputing Center, engaging up to 1024 CPU-cores. It has been asserted that the efficiency of the DD strategy depends only on the load balancing or, equivalently, the interface distribution within the domain. Our LB strategy overcomes the imbalance, but the redistribution cost cancels part of the gains achieved from it. Anyway, when directly comparing both strategies, the result is that the larger the initial imbalance, the larger the speedup achieved by the LB algorithm respect to the DD one. We have observed speedups up to $\sim 12\times$ for the most ill-conditioned situations, but even in situations where the interface is almost spread throughout all the domain, the speedup achieved is $\sim 1.5\times$ in average.

With this scenario in mind, the new parallelization strategy presented may be a feasible option to be considered when solving multi-fluid flows by means of VOF methods. Moreover, our approach could be easily adapted to other interface-capturing methods, like the Level-Set, which suffer from a similar workload imbalance.

Acknowledgements

This work has been financially supported by the Ministerio de Economía y Competitividad, Secretaría de Estado de Investigación, Desarrollo e Innovación, Spain (ENE-2010-17801), a FPU Grant by the Ministerio de Educación, Cultura y Deporte, Spain (AP-2008-03843) and by Termo Fluids

S.L.

The computations presented in this work have been carried out on the IBM MareNostrum-III supercomputer at the Barcelona Supercomputing Center (BSC), Spain (FI-2012-3-0021 and FI-2013-1-0024). The authors thankfully acknowledge this Institution.

We also thank the anonymous reviewers for their comments and remarks which helped to improve the quality of this work.

References

[1] C. W. Hirt, J. L. Cook, T. D. Butler, A Lagrangian Method for Calculating the Dynamics of an Incompressible Fluid with Free Surface, Journal of Computational Physics 5 (1970) 103–124.

[2] C. W. Hirt, A. A. Amsden, J. L. Cook, An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds, Journal of Computational Physics 135 (1997) 203–216.

[3] H. H. Hu, N. A. Patankar, M. Y. Zhu, Direct Numerical Simulations of Fluid-Solid Systems Using the Arbitrary Lagrangian-Eulerian Technique, Journal of Computational Physics 169 (2001) 427–462.

[4] F. H. Harlow, J. E. Welch, Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface, Physics of Fluids 8 (1965) 2182–2189.

[5] C. S. Peskin, Numerical Analysis of Blood Flow in the Heart, Journal of Computational Physics 25 (1977) 220–252.

[6] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawashi, W. Tauber, J. Han, S. Nas, Y. J. Jan, A Front-Tracking Method for the Computations of Multiphase Flow, Journal of Computational Physics 169 (2001) 708–759.

[7] C. W. Hirt, B. D. Nichols, Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries, Journal of Computational Physics 39 (1981) 201–225.

[8] P. Liovic, M. Rudman, J. L. Liow, D. Lakehal, D. Kothe, A 3D Unsplit-Advection Volume Tracking Algorithm with Planarity-Preserving Interface Reconstruction, Computers & Fluids 35 (2006) 1011–1032.

[9] T. Marić, H. Marschall, D. Bothe, voFoam - A Geometrical Volume of Fluid Algorithm on Arbitrary Unstructured Meshes with Local Dynamic Adaptive Mesh Refinement using OpenFOAM, arXiv:1305.3417 (2013) 1–30.

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

[11] E. Olsson, G. Kreiss, A Conservative Level Set Method for Two Phase Flow, Journal of Computational Physics 210 (2005) 225–246.

[12] N. Balcázar, L. Jofre, O. Lehmkuhl, J. Castro, J. Rigola, A Finite-Volume/Level-Set Method for Simulating Two-Phase Flows on Unstructured Grids, International Journal of Multiphase Flow 64 (2014) 55–72.

[13] Y. Renardy, Effect of Startup Conditions on Drop Breakup under Shear with Inertia, International Journal of Multiphase Flow 34 (2008) 1185–1189.

[14] M. S. Annaland, N. G. Deen, J. A. M. Kuipers, Numerical Simulation of Gas Bubbles Behaviour using a Three-Dimensional Volume of Fluid Method, Chemical Engineering Science 60 (2005) 2999–3011.

[15] K. M. T. Kleefsman, G. Fekkena, A. E. P. Veldman, B. Iwanowski, B. Buchner, A Volume-of-Fluid Based Simulation Method for Wave Impact Problems, Journal of Computational Physics 206 (2005) 363–393.

[16] D. Fuster, A. Bagué, T. Boeck, L. Le Moyne, A. Leboissetier, S. Popinet, P. Ray, R. Scardovelli, S. Zaleski, Simulation of Primary Atomization with an Octree Adaptive Mesh Refinement and VOF Method, International Journal of Multiphase Flow 35 (2009) 550–565.

[17] G. Tomar, D. Fuster, S. Zaleski, S. Popinet, Multiscale Simulations of Primary Atomization, Computers & Fluids 39 (2010) 1864–1874.

[18] X. Chen, D. Ma, V. Yang, S. Popinet, High-Fidelity Simulations of Impinging Jet Atomization, Atomization and Sprays 23 (2013) 1079–1101.

[19] P. R. Chapman, J. W. Jacobs, Experiments on the Three-Dimensional Incompressible Richtmyer-Meshkov Instability, Physics of Fluids 18 (2006) 074101.

[20] V. Le Chenadec, H. Pitsch, A 3D Unsplit Forward/Backward Volume-of-Fluid Approach and Coupling to the Level Set Method, Journal of Computational Physics 233 (2013) 10–33.

[21] S. P. MacLachlan, J. M. Tang, C. Vuik, Fast and Robust Solvers for Pressure-Correction in Bubbly Flow Problems, Journal of Computational Physics 227 (2008) 9742–9761.

[22] R. Borrell, O. Lehmkuhl, F. X. Trias, A. Oliva, Parallel Direct Poisson Solver for Discretizations with one Fourier Diagonalizable Direction, Journal of Computational Physics 230 (2011) 4723–4741.

[23] B. J. Araújo, J. C. F. Teixeira, A. M. Cunha, C. P. T. Groth, Parallel Three-Dimensional Simulation of the Injection Molding Process, International Journal for Numerical Methods in Fluids 59 (2009) 801–815.

[24] M. Sussman, A Parallelized, Adaptive Algorithm for Multiphase Flows in General Geometries, Computers & Structures 83 (2005) 435–444.

[25] M. Sussman, E. G. Puckett, A Coupled Level Set and Volume-of-Fluid Method for Computing 3D and Axisymmetric Incompressible Two-Phase flows, Journal of Computational Physics 162 (2000) 301–337.

[26] M. Herrmann, A Balanced Force Refined Level Set Grid Method for Two-Phase Flows on Unstructured Flow Solver Grids, Journal of Computational Physics 227 (2008) 2674–2706.

[27] M. Herrmann, A Parallel Eulerian Interface Tracking/Lagrangian Point Particle Multi-Scale Coupling Procedure, Journal of Computational Physics 229 (2010) 745–759.

[28] L. Jofre, O. Lehmkuhl, J. Castro, A. Oliva, A 3-D Volume-of-Fluid Advection Method Based on Cell-Vertex Velocities for Unstructured Meshes, Computers & Fluids 94 (2014) 14–29.

[29] D. L. Youngs, Time-Dependent Multi-Material Flow with Large Fluid Distortion, in: Numerical Methods for Fluid Dynamics, Academic Press, New York, 1982, pp. 273.

[30] A. Haselbacher, V. Vasilyev, Commutative Discrete Filtering on Unstructured Grids based on Least-Squares Techniques, Journal of Computational Physics 187 (2003) 197–211.

[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes in C++, Cambridge University Press, 2002.

[32] J. López, J. Hernández, Analytical and Geometrical Tools for 3D Volume of Fluid Methods in General Grids, Journal of Computational Physics 227 (2008) 5939–5948.

[33] W. Wall, S. Genkinger, E. Ramm, A Strong Coupling Partitioned Approach for Fluid-Structure Interaction with Free Surfaces, Computers & Fluids 36 (2007) 169–183.

[34] P. Liovic, D. Lakehal, Interface-Turbulence Interactions in Large-Scale Bubbling Processes, International Journal of Heat and Fluid Flow 28 (2007) 127–144.

[35] Z. Wang, J. Yang, B. Koo, F. Stern, A Coupled Level Set and Volume-of-Fluid Method for Sharp Interface Simulation of Plunging Breaking Waves, International Journal of Multiphase Flow 35 (2009) 227–246.

[36] L. Jofre, O. Lehmkuhl, R. Borrell, J. Castro, A. Oliva, Parallelization Study of a VOF/Navier-Stokes Model for 3D Unstructured Staggered Meshes, in: Proceedings of the Parallel CFD 2011 Conference, pp. 1–5.

[37] J. Y-T. Leung, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, 2004.

[38] O. Lehmkuhl, C. D. Pérez-Segarra, R. Borrell, M. Soria, A. Oliva, TERMOFLUIDS: A New Parallel Unstructured CFD Code for the Simulation of Turbulent Industrial Problems on Low Cost PC Cluster, in: Proceedings of the Parallel CFD 2007 Conference, pp. 1–8.

[39] Barcelona Supercomputing Center (BSC), Webpage: www.bsc.es.

[40] G. Karypis, K. Vipin, A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM Journal on Scientific Computing 20 (1998) 359–392.

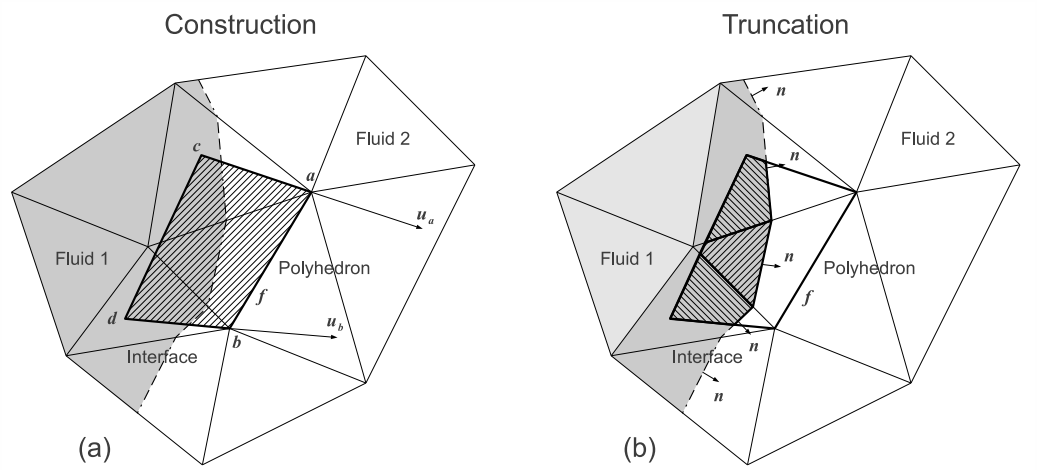

Figure 1: (a) Construction of the total volumetric flux polyhedron (**abdc**), point **c** is evaluated by tracing back the Lagrangian trajectory of point **a** for the time step Δt, i.e., $\mathbf{c} = \mathbf{a} - \Delta t \mathbf{u}_a$; idem for point **d**. (b) Truncation of the part of the volumetric flux polyhedron corresponding to Fluid 1.

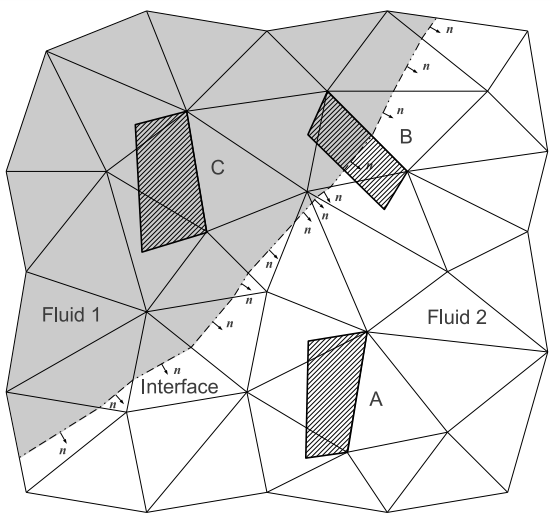

Figure 2: Possible initial situations on the evaluation of the fluids advection: (1) The volumetric flux polyhedron contains only one fluid (cases A and C); (2) Different fluids coexist within the volumetric flux polyhedron (case B).

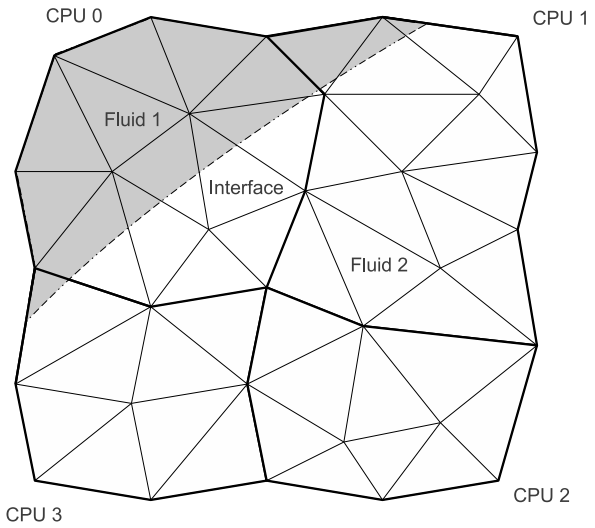

Figure 3: Decomposition of an unstructured grid where two fluids coexist. The interface between fluids is not homogeneously distributed throughout the domain.

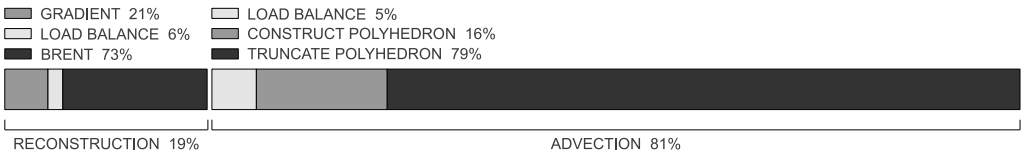

Figure 4: Computing time distribution for the new parallelization strategy.

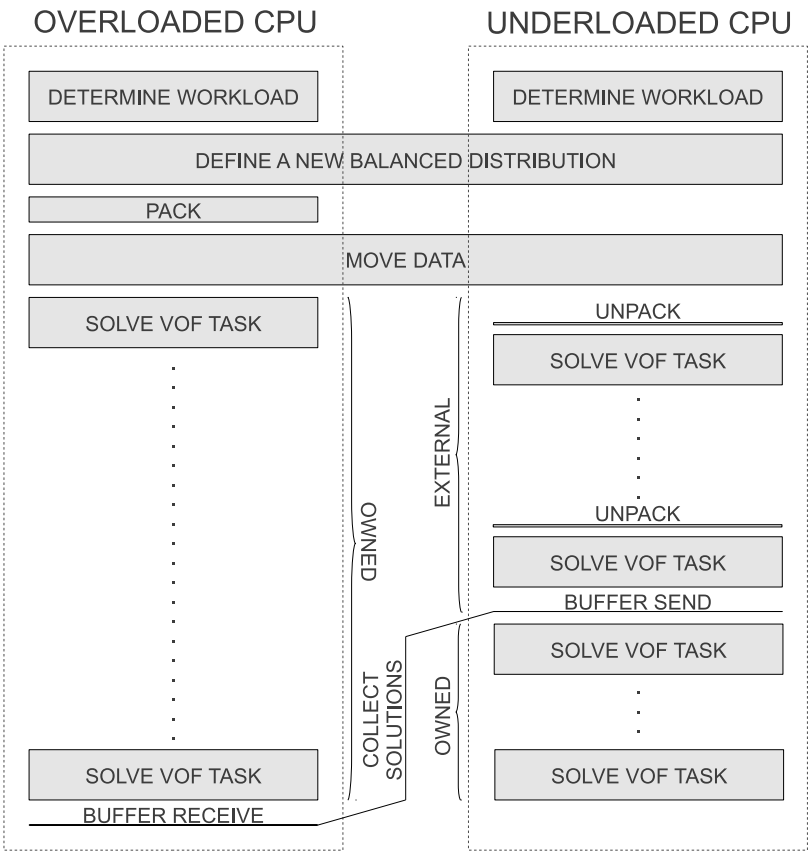

Figure 5: Flowchart of the advection process, from the perspective of an "overloaded CPU" (left) and an "underloaded CPU" (right). The height of each rectangular box is proportional to the cost of the corresponding step of the algorithm.

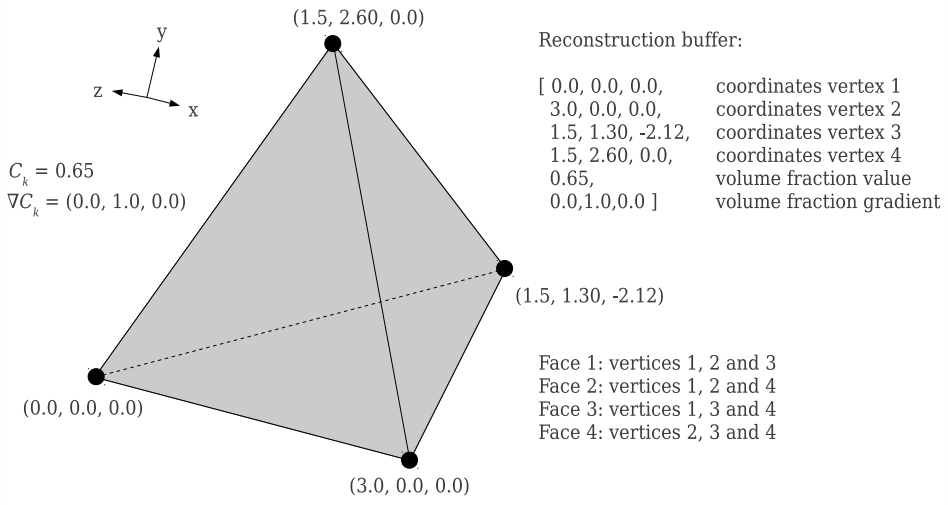

Figure 6: Illustration of the data packed into the communication buffer for a reassigned reconstruction task.

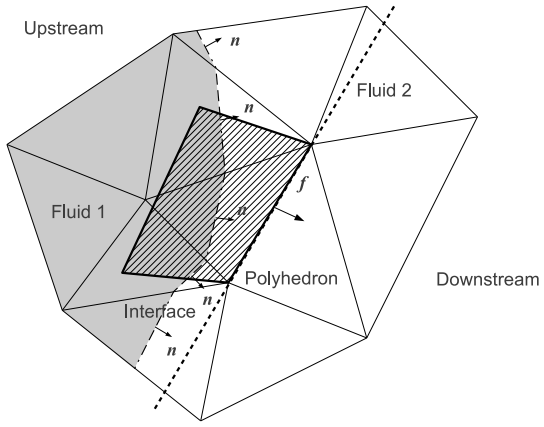

Figure 7: Representation of the flux polyhedron used on the fluids advection evaluation at face f.

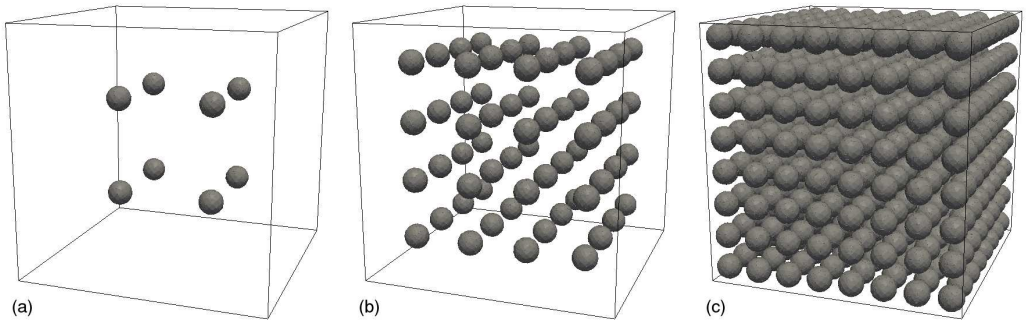

Figure 8: Representation of different grids of spheres, which define the interface between two fluids, used in the numerical experiments: (a) 2×2×2, (b) 4×4×4 and (c) 8×8×8.

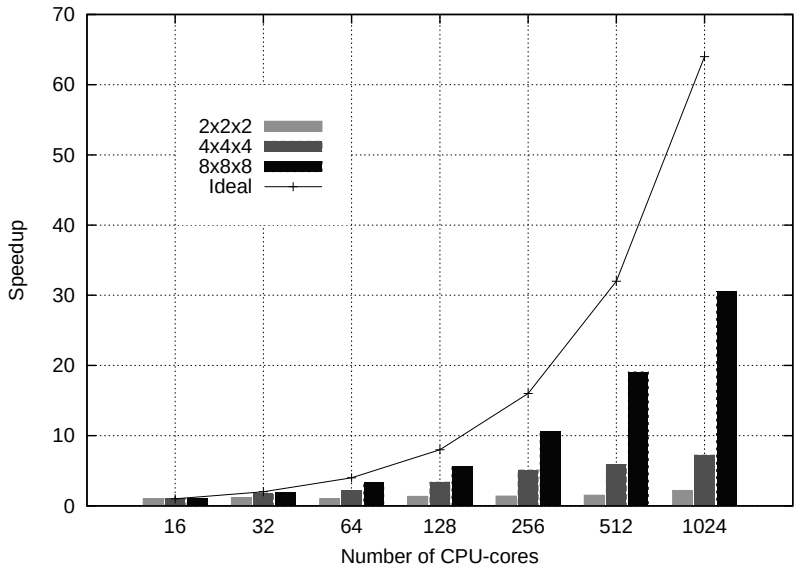

Figure 9: Speedup of the VOF algorithm using the DD strategy for the 2×2×2, 4×4×4 and 8×8×8 interface configurations.

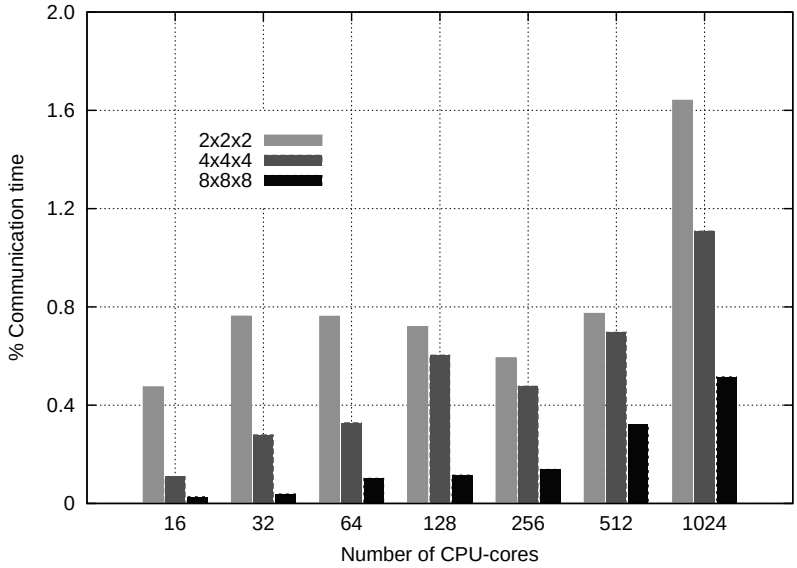

Figure 10: Percentage of the communication costs over the total cost of the VOF algorithm with the DD parallelization strategy.

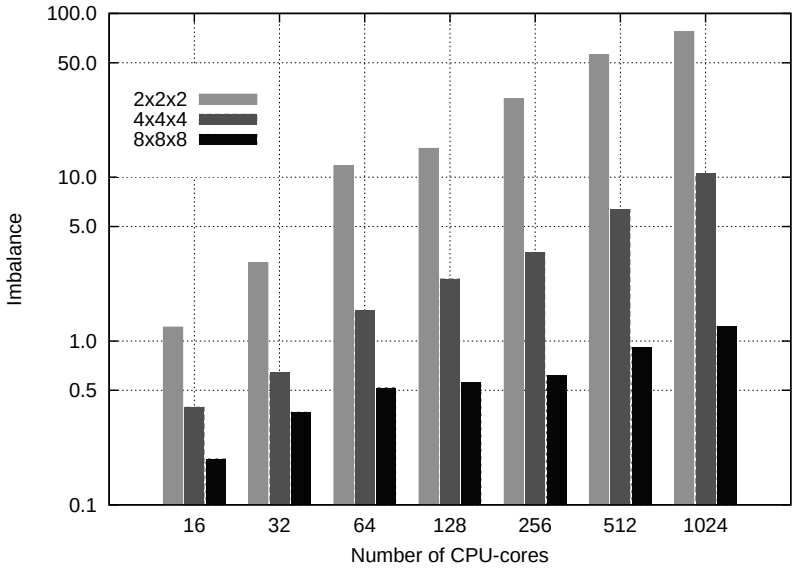

Figure 11: Imbalance obtained on each of the test cases studied. The imbalance is evaluated as the difference between the number of interface cells for the most overloaded parallel process and the average of interface cells per process, divided by the average.

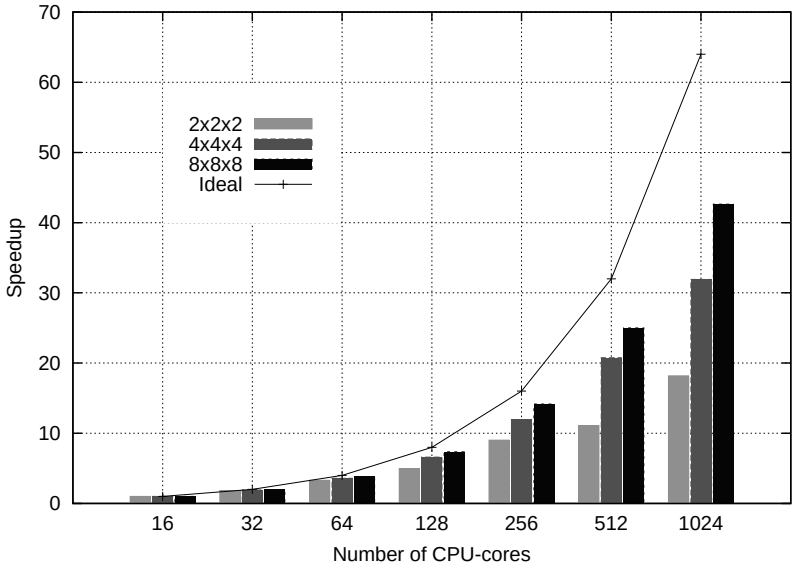

Figure 12: Speedup of the VOF algorithm using the LB strategy for the 2×2×2, 4×4×4 and 8×8×8 interface configurations.

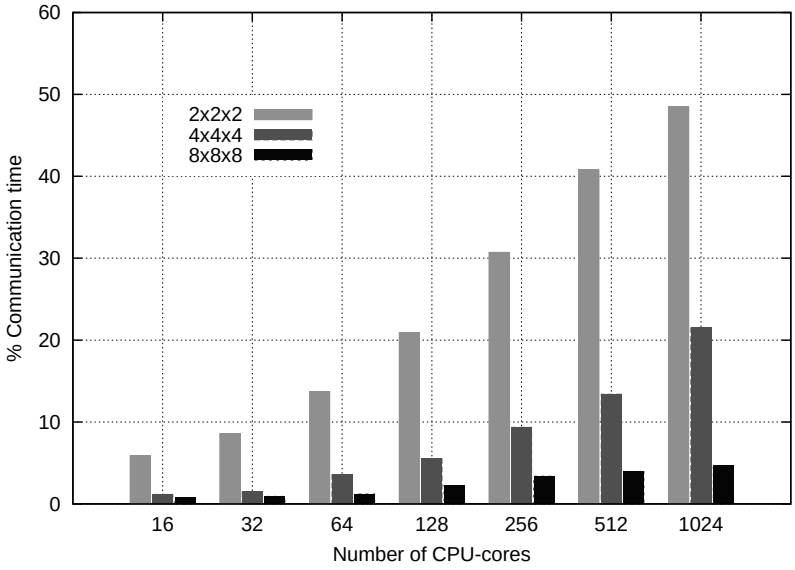

Figure 13: Percentage of the communication costs over the total cost of the VOF algorithm with the LB parallelization strategy.

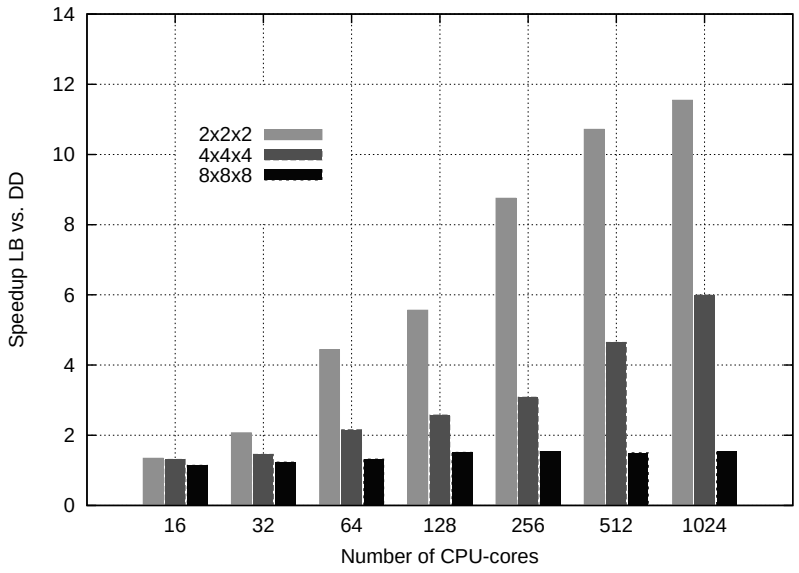

Figure 14: Speedup of the LB strategy versus the DD one for the VOF solution of the 2×2×2, 4×4×4 and 8×8×8 interface configurations with different number of CPU-cores.

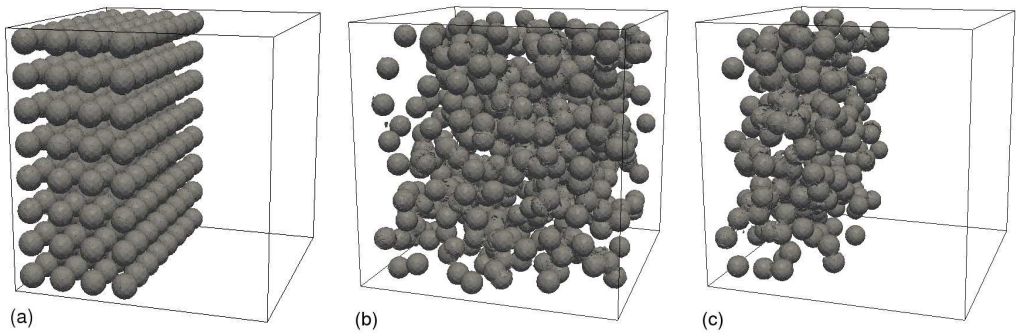

Figure 15: Three additional configurations of spheres: (a) 4×8×8, (b) R-8×8×8 and (c) R-4×8×8.

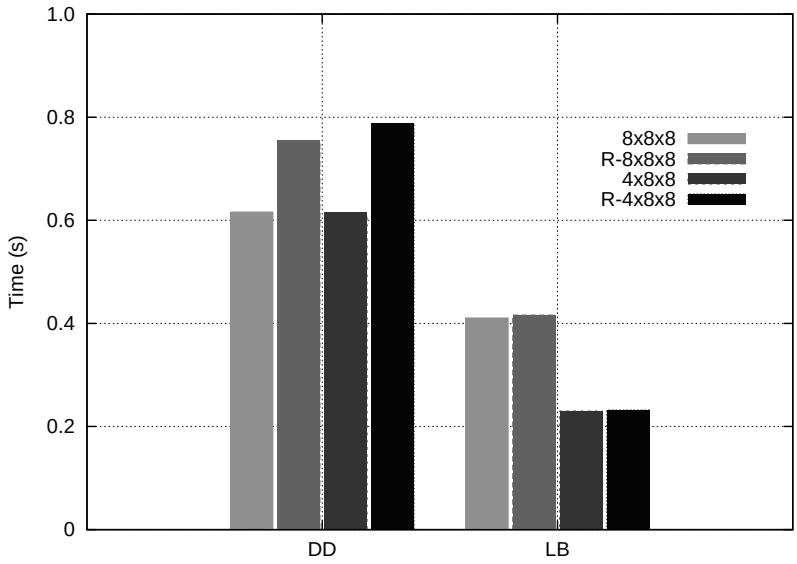

Figure 16: Comparative of the time required by the DD and LB strategies to solve the interface configurations 4×8×8 and 8×8×8 using 512 CPU-cores.

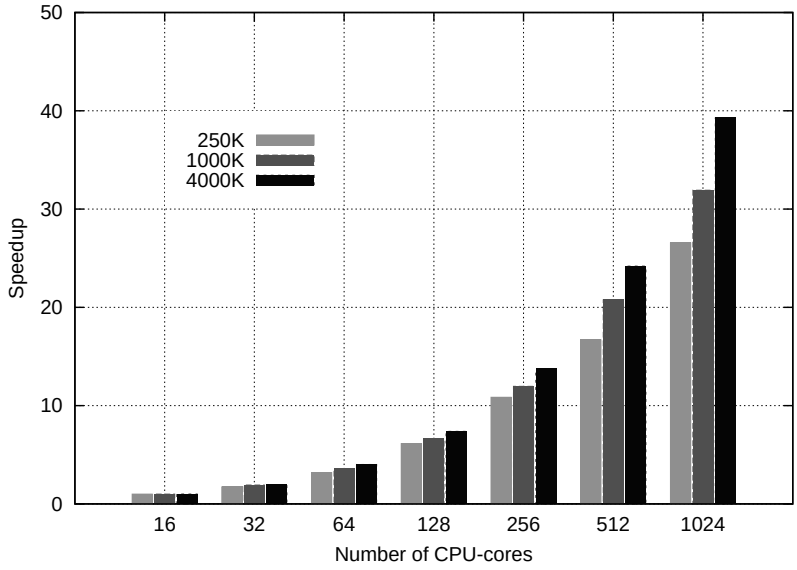

Figure 17: Speedup of the LB strategy on the solution of the 4×4×4 interface configuration for three different 3-D meshes of sizes 250K, 1000K and 4000K.

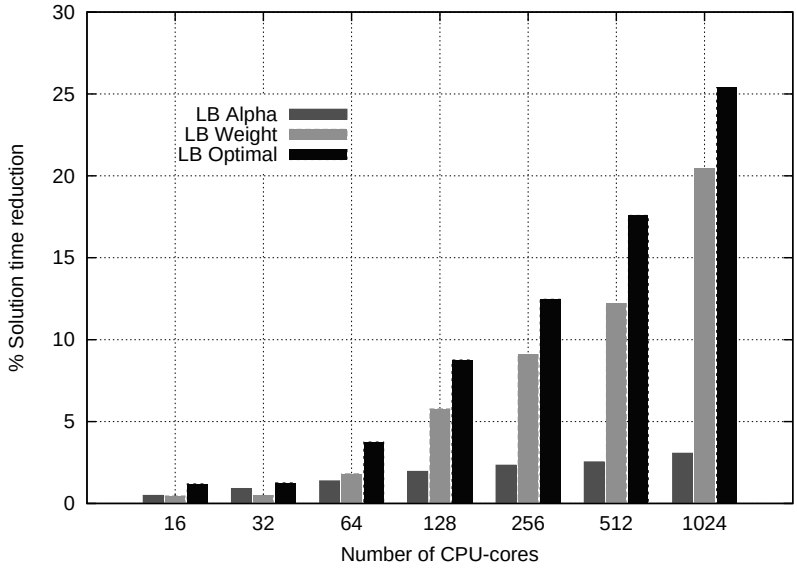

Figure 18: Reduction achieved on the advection solution time by different optimizations of the LB algorithm: (1) considering coefficient α (LB Alpha); (2) introducing weights (LB Weight); (3) considering both optimizations together (LB Optimal). The test case is the 4×4×4 interface configuration on the 1000K mesh.

Name	No. interface cells	% interface cells
2×2×2	3120	0.3
4×4×4	25254	2.5
8×8×8	204778	20.0

Table 1: Detailed characteristics of different interface configurations used in the numerical experiments.