Computers and Fluids 256 (2023) 105856

Contents lists available at ScienceDirect

Computers and Fluids

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

Efficient Lagrangian particle tracking algorithms for distributed-memory architectures

Giacomo Baldan *, Tommaso Bellosta, Alberto Guardone

Department of Aerospace Science and Technology, Politecnico di Milano, Via La Masa 34, Milano 20156, Italy

ARTICLE INFO

Keywords:
Particle tracking
Mesh partitioning
Parallel computing
High performance computing
Distributed memory
Computational fluid dynamics

ABSTRACT

This paper focuses on the solution of the dispersed phase of Eulerian–Lagrangian one-way coupled particle laden flows. An efficient two-constraint domain partitioning for 2D and 3D unstructured hybrid meshes is proposed and implemented in distributed memory architectures. A preliminary simulation, using a set of representative particles, is performed first to suitably tag the cells with a weight proportional to the probability of being crossed by a particle. In addition, an innovative parallel ray-tracing location algorithm is presented. A global identifier is assigned to each particle resulting in a significant reduction of the overall communication among processes.

The proposed approaches are verified against two steady reference cases for ice accretion simulation: a NACA 0012 profile and a NACA 64A008 swept horizontal tail. Furthermore, a cloud droplet impact test case starting from an unsteady flow around a 3D cylinder is performed to evaluate the code performances on unsteady problems.

1. Introduction

Particle-laden turbulent flows have significant importance in various natural phenomena and industrial processes [1,2]. Lagrangian particle tracking has been adopted in recent studies to simulate respiratory droplets in turbulent flows [3] or internal combustion engines [4,5], ink-jet printing [6] or in-flight ice accretion problems [7–9].

The accuracy of Lagrangian methods is proportional to $1/\sqrt{N}$ with N the number of sample particles. In order to obtain high-fidelity results, a large set of particles is required [10]. Parallel algorithms can be adopted to reduce computational time. In addition, a distributed-memory parallelization becomes a requisite if large datasets are to be used. Nevertheless, due to the a-priori unpredictable behavior of the flow solution and the particle evolution, some critical aspects have to be considered in an efficient parallelization. First, an effective technique to subdivide the domain based only on the flow solution usually leads to an unbalanced particle workload [11]. Secondly, the physics of the problem is also reflected in the particle movement among processes. Indeed, during particle integration, a direction is usually predominant. A standard all-to-all communication tends to perform slower since most of the processes do not exchange information. A set of non-blocking send and receive calls is more efficient. Indeed, communication is performed between two processes only when at least one particle has to be sent. Finally, the computational cost at each time step is dictated by the process with the highest number of particles to integrate, considering an equal integration time step for all particles. If an unsteady flow solution is analyzed, all the mentioned points are even more critical since the flow solution is updated multiple times during the simulation. Another aspect to consider is the initial particle location. The computation of the initial position within the computational mesh cannot be performed using a standard serial algorithm. The arbitrary shape of each subdomain due to the mesh partitioning can lead to non-simply connected and multi-part subdomains.

In literature, several parallel implementations have been proposed. In [12], a Lagrangian particle tracking model in a mixed Eulerian–Lagrangian commercial CFD software is presented. Specifically, the parallelization treats the fluid phase solution and the particle integration. In addition, a domain decomposition scattering technique is discussed to improve the parallel efficiency. Another example of parallel code is presented in [13]. The implementation focuses on finite element applications using 2D and 3D unstructured meshes. An extensive description of the particle tracking algorithm is reported. Finally, in [14], a 3D coupling algorithm between Volume of Fluid and Lagrangian particle tracking is presented for the open-source CFD library OpenFOAM.

This work focuses on the efficient parallelization of the computation of a Lagrangian dispersed phase in a given Eulerian continuous flow.

* Corresponding author.
E-mail address: giacomo.baldan@polimi.it (G. Baldan).

https://doi.org/10.1016/j.compfluid.2023.105856
Received 31 October 2021; Received in revised form 1 December 2022; Accepted 10 March 2023
Available online 15 March 2023

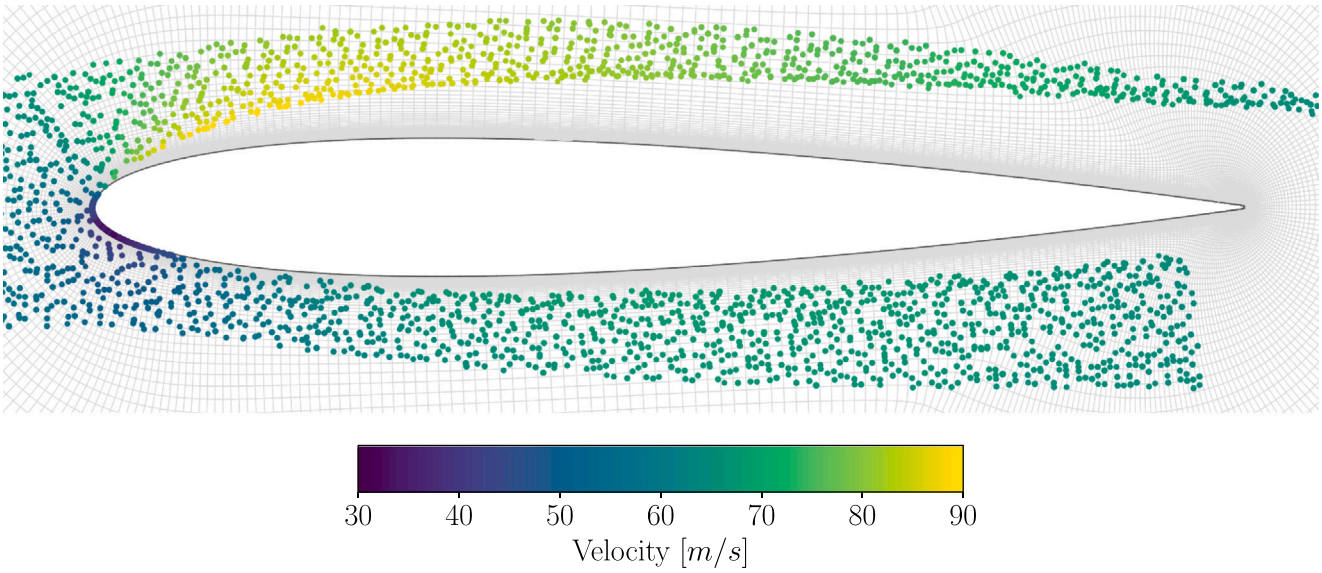


Fig. 1. Particles around a NACA 0012 profile.

The narrow scope of this work is justified by the target application, that is the computation of the water mass captured by a flying object in icing conditions. Such flows are usually one-way coupled, and the particle phase is obtained as a post-processing of the continuous aerodynamic field [15,16]. In this work, a two-constrained mesh partitioning based on a graph representation is developed in Section 2. The code can handle unstructured hybrid meshes. In two dimensions, triangular and quadrilateral elements are considered, while in three-dimension, cells can be tetrahedra, hexahedra, prisms, and pyramids. A standard partitioning based only on the geometric constraint is performed using Parmetis. After, a preliminary simulation with few significant particles is computed to suitably tag the cells. In this manner, a constraint proportional to the probability of being crossed by particles is added to the graph partitioning. The changes from the serial particle tracking algorithm are reported in Section 3. In particular, the main loop is described to obtain a non-blocking communication of particles among processes. In Section 4, a parallel ray-tracing procedure is described to locate particles inside the mesh at the beginning of the simulation. The performances of the location algorithm are reported. In Section 5, two steady reference cases for ice accretion simulation are studied: a two-dimensional NACA 0012 profile (Fig. 1) and a three-dimensional NACA 64A008 swept horizontal tail. In addition, a cloud droplet impact test case starting from an unsteady flow around a 3D cylinder is performed. In both cases, steady and unsteady, the speed-up factor is evaluated.

2. Cell tagging

A mesh partitioning algorithm is usually adopted when using a domain decomposition approach. The standard subdivision technique, based only on the mesh connectivity, creates a set of subdomains composed of a similar portion of nodes to avoid imbalances among the processes, while the number of cut edges is minimized to reduce the communication. In a particle-laden flow, simulated with a Lagrangian approach, a multi-constrained subdivision should be used to balance all the contributions to the computation. In the present work, mesh partitioning is carried out using Parmetis library [17,18] that offers a parallel multi-level and multi-constraint k-way subdivision [19,20] through a C/C++ API. Parmetis guarantees a sequential complexity of the serial multi-constraint algorithm of $O(mn)$ where m is the number of constraints and n of graph nodes. Also, other libraries, such as PT-Scotch [21] and Zoltan [22], were successfully used for large-scale parallel CFD computations [23]. Parmetis has been chosen since the API allows straightforward integration with the existing serial code, granting the required parallel performances.

The mesh can be treated as a nodal graph or a dual graph. In the first case, each mesh node corresponds to a graph node and all mesh edges coincide with graph edges. After the mesh redistribution, based on the computed graph subdivision, each mesh node is owned by only one process, while elements can be present in multiple ones. In the second case, each graph node refers to an element, and the graph edges represent the mesh faces. Opposite to the nodal graph representation, nodes can be shared among multiple processes, and elements are present in only one. In this work, the mesh partitioning is based on the nodal graph. The main reason is linked to particle communication. If the dual graph representation is adopted, in the position where the domain is cut, new non-physical boundaries are created, and adjacent processes share only nodes and faces. This usually leads to finite precision arithmetic issues when a particle has to be sent to the new owner because the two processes own only 1D and 2D geometric entities. The nodal graph representation, instead, naturally generates a layer of overlapping elements, called halo cells, that allows straightforward particle communication. The particle that has to be moved automatically satisfies the inclusion test in all the processes that own that halo cell. More details about the particle communication are provided in Section 3. Another key aspect that has been considered in the type of graph representation is the computational time. Indeed, converting the original domain into the dual graph can be time demanding as much as the graph subdivision [24]. Adopting the nodal graph representation avoids the expensive graph extraction from the domain since the mesh connectivity is the same as in the graph. The last reason is linked to the CFD solver SU2, the node-based Finite Volume code used to compute the fluid solution.

The adopted two-constrained mesh partitioning aims at equally subdividing the mesh nodes and the particle workload among the processes. Parmetis API requires an integer weight per constraint for all the nodes. The geometric constraint, related to the mesh connectivity, is accounted for by assigning each node a weight equal to the number of edges that own that node. The second constraint, linked to the particle workload, is computed using the cell tagging technique. The main idea is to perform a preliminary simulation to tag the cells crossed by particles to obtain a weight proportional to the number of particles that pass through each cell. The preliminary simulation uses mesh partitioning based only on mesh connectivity. This strategy works, and it is adopted for steady and unsteady flow simulation since it is

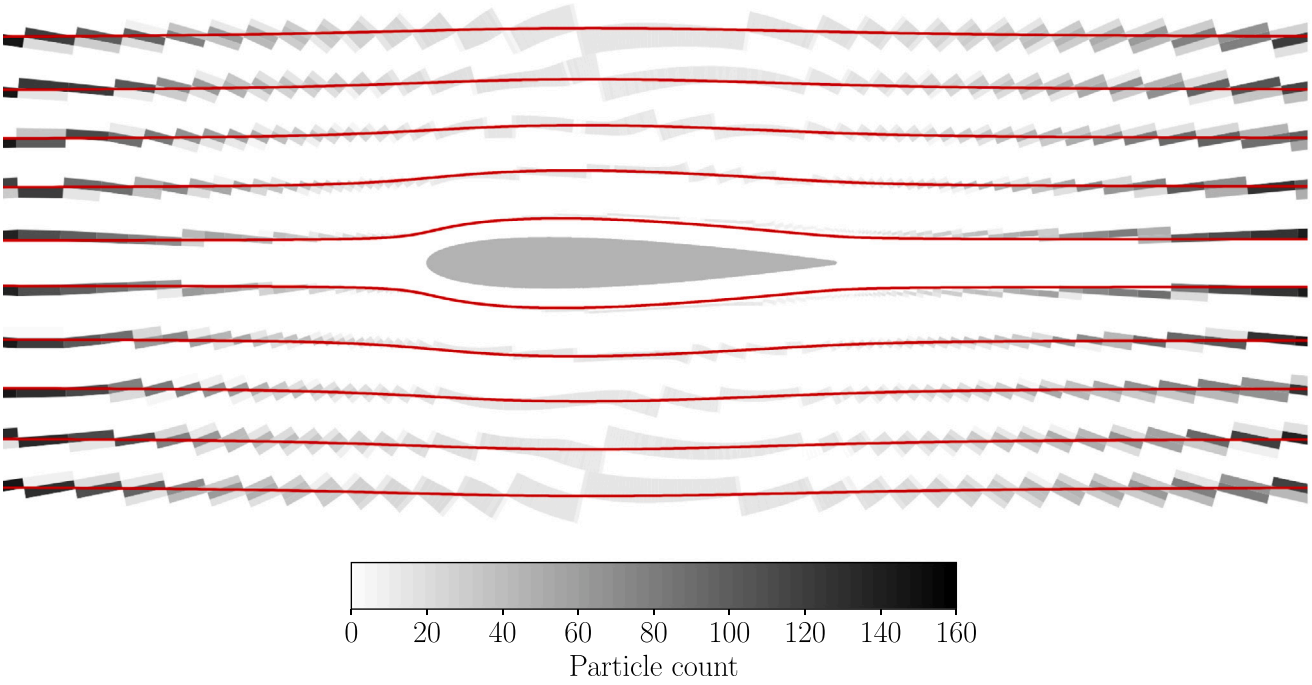


Fig. 2. Trajectories and number of particles that crossed each cell.

equivalent to a standard particle simulation with a reduced particle number. The particle workload can be unbalanced among the processes, but the overall execution time of this phase is negligible compared with the actual simulation. Indeed, only a few significant particles are evolved to tag the cells. The present algorithm initializes a uniform particle distribution, also referred to as cloud, enclosed in a Cartesian box. Usually, ten particles per Cartesian direction are sufficient to describe the behavior of the cloud containing all the particles of the actual simulation. In 3D, the number of particles for the actual simulation is at least in the order of millions making the preliminary cloud integration three orders of magnitude faster or more. During the tracking, when a particle crosses an element, a cell counter is updated. If the distance traveled by a particle in a time step is smaller than the cell size or it is not sufficient to exit from the cell, the counter is updated multiple times. This is why in Fig. 2, where it is possible to see a color representation of the cell tagging technique outcome, the elements on the left and the right of the domain have a higher particle count. In this manner, a cell weight is obtained. Thanks to the full mesh connectivity previously calculated, the cell count is converted into a node one. Parmetis is called a second time to compute the final partitioning that accounts for both constraints. The process is significantly faster than the previous partitioning because it starts from an already computed subdivision, not a random one. A two-dimensional example with and without the tagging strategy is presented in Fig. 3. It is clear how the subdomain shapes change according to the particle trajectories. Without the cell tagging technique, many available resources are concentrated around the airfoil where the element size is small. Instead, in the second domain subdivision, a portion of elements near the geometry is owned by each process, which is also achieved for the farther ones. A drawback of this implementation is the higher possibility of having non-simply connected or multiple-part subdomains. However, the code can handle all these possible shapes.

3. Particle tracking

The parallel particle tracking algorithm presents some differences from the serial one reported in Refs. [25,26]. The code is a one-way coupled Lagrangian particle tracking developed at Politecnico di Milano and supports forward Euler and explicit Runge–Kutta integration schemes.

Two new features have been added in the current implementation: particle communication and the possibility to handle a cloud where each particle is evolved until an arbitrary time. The last feature allows for avoiding blocking MPI communication in favor of non-blocking one, thus reducing the overall idle time.

During the cloud evolution, when a particle reaches a non-physical boundary created by the mesh partitioning, it has to be sent to a new process. The current code checks if the element in which the particle is included is a halo cell and if the final trajectory point for the current time step is outside the subdomain. If this is the case, the algorithm verifies the number of processes that own the halo cell. If it is two, the new process is easily retrieved. Otherwise, if multiple processes share the same halo cell, the algorithm finds the face that intersects the particle trajectory, and the new owner is the process that includes that face. All the information about the neighbor processes is built from the node distribution computed with Parmetis. For instance, a face is in a subdomain if all the nodes that compose it are also part of the subdomain. In order to speed up the particle integration, all the data structures related to neighbor processes and halo cells are precomputed just after the mesh partitioning. The particle tracking code relies on a mapping from local ID to global ID, and vice-versa, for the elements and nodes of the mesh, where local refers to a numbering for the geometric entities in each process, and global is the original mesh numbering for the entire domain. The element mapping is fundamental when a particle is communicated because the global ID is used to identify the halo cell in the entire domain uniquely, but the mesh connectivity in each subdomain is based on the local IDs.

Each particle in the code is described with a C++ structure containing all the relevant quantities. The data structure contains physical quantities like the position, the velocity, the diameter, and the current time and support variables such as the global particle ID. When the cloud is integrated, the code loops over the particles. Integration starts at the particle time and is evolved until the current simulation time is reached or when a non-physical boundary is encountered, and the particle has to be communicated. In order to respect the maximum

(a) Without cell tagging technique, only geometric constraint is used.

(b) With cell tagging technique, geometric and particle workload constraints are used.

Fig. 3. Example of mesh subdivision in 8 parts, each color corresponds to a subdomain.

Algorithm 1: Cloud evolution.

```
while simulation time < max time do
    forall particles do
        while particle time < simulation time do
            update particle position;
            if particle has to be communicated then
                store particle ID;
                break while;
            end
        end
    end
    wait and complete communication from previous time step;
    add received particles to the cloud;
    forall stored particle IDs do
        buffer particle data sorted by receiver rank;
    end
    send particles to new owners;
end
```

time step imposed by the user, a nested while loop is adopted to split the integration from the particle time to the current simulation time or until another non-physical boundary is reached.

The main loop to evolve the cloud is composed of three main steps and is summarized in Algorithm 1. First, all the particles owned by a specific process are evolved until the current simulation time. Since the communication is non-blocking, the particle integration is overlapped with the communication of the previous time step. During the particle update, if a particle reaches a non-physical boundary and has to be communicated, its ID is stored, and the particle is no more updated because the trajectory is outside the subdomain. Then, the communication from the previous step is completed, and the code waits until all the particles have been received to add them into the new process cloud. Finally, the particles are sent to the new owners if their trajectories intersect an internal interface looping over the stored IDs. All the particle data is buffered and sorted by the receiver MPI rank. The flow solution at the particle position is not transferred because it can be easily computed once received. This choice does not affect the results because it takes advantage of halo cells and the flow solution is interpolated starting from the flow values at the same nodes. The asynchronous evolution is effective but also presents some disadvantages. The first one is, as mentioned before, the possibility that each particle is evolved until a physical time different from the simulation one. Another drawback relates to the necessity of synchronizing the cloud, for instance, when it is saved. In unsteady simulations, the cloud is also synchronized when the flow solution has to be updated according to the particle simulation time. This leads to a reduction in performances, as highlighted in Section 5. Another disadvantage of unsteady simulations is related to the preliminary particle evolution needed for the cell tagging technique introduced earlier. Even if only a few particles are integrated, the flow solution must be updated multiple times, leading to an increased execution time. The code uses the parallel MPI I/O

interface to speed up the loading time of each flow solution. The binary VTK files [27], where the solution is stored, are loaded using an MPI-indexed variable. This is created to describe the non-contiguous data among the processes, and the collective MPI-I/O function is used to read the file efficiently [28].

4. Parallel ray-tracing

Before the actual particle simulation can start, the location of each particle within the mesh is to be computed. It consists in finding per each particle the cell in which it is included. In literature, three main methods are present: brute force, tree-based [29,30], and ray-tracing [31]. A brute force approach works in every situation but is inefficient, especially when the number of particles is high, or the domain is large. Vicinity algorithms are not sufficiently robust if the domain is not simply connected or comprises more than one part, as is the case here. Indeed, particles can be near a boundary element and, at the same time, be outside the subdomain. A possible solution to the problem is to verify each time if the particle is actually inside the element through an inclusion test, but it is computationally expensive. Ray-tracing requires a convex mesh portion that encloses the cloud, which is not granted due to the arbitrary subdomain shapes. The present code solves the issues by considering a parallel ray-tracing algorithm to recover the convexity and efficiently locate the cloud. In this perspective, a ray always starts from a known point and ends at the position of the particle to be located. The computational time of the ray-tracing algorithm is proportional to the number of crossed cells. For this reason, the starting point coincides with the previous particle if it has been located. Otherwise, the code searches for the closest element centroid among 20 random elements. Therefore, the ray covers a shorter distance. Since the code integrates the ray-tracing algorithm for particle integration, a simplified dummy particle, in which the position and the cell ID are stored is adopted to represent and evolve a ray.

The particle position is computed from a uniformly distributed cloud enclosed by a rectangular parallelepiped oriented along the three Cartesian directions. Each subdomain's bounding box is compared with the initial cloud's box to avoid searching for particles in empty subdomains. At this stage, the code has to determine the correct owner because the particles near the boundaries can be present in more than a subdomain. The code assigns a particle in a halo element to the process with the lowest MPI ID. This rule applies only when the particle are located the first time inside the computational domain, and, even if it slightly increases the number of particles owned by lower MPI ranks, it does not affect the particle evolution where particles in the halo cells are naturally evolved based on their trajectories. The subdomain shapes can be quite challenging for the ray-tracing algorithm. For instance, a ray can cross a subdomain multiple times. Another possibility is that it passes across more than one subdomain, ending in the original one. Another rare event is that a ray goes through a process with no particles to initialize, so all the processes can compute trajectories even if they are empty. These situations arise more frequently when three-dimensional meshes are considered, but they can be present also in two-dimensional ones. For instance, Fig. 4 shows a sliced 3D domain with located particles. The developed parallel code is summarized in Algorithm 2. Each part of the present algorithm is executed in parallel to avoid bottlenecks when using a large set of processes. First, one particle is located starting from the closest element centroid among twenty randomly selected. Then, all the other particles are located in each process using the ray-tracing algorithm in each subdomain. In this phase, there is no communication among the processes. If a particle is located, the last element ID of the dummy particle is assigned to the actual particle. On the contrary, if a ray reaches an internal boundary, the particle is marked. Before sending the rays to all the subdomains, the algorithm checks if the marked particles have already been located in an adjacent subdomain. If this is the case, the particle is eliminated. Checking adjacent processes allows a significant reduction of communication in the following stages. Then, all the rays, initialized in the first phase, are tracked until they reach the final position. The starting process is stored in the dummy particle to keep track of all the rays. When a particle is located, its ID is sent to the starting process that deletes it. While the process in which the particle is located stores, as done previously, the last element ID of the ray. If multiple rays reach a particle, the code does not need any modification because the element ID is always the same, and the particles are eliminated in the starting process. If this occurs, some computational power is wasted.

Algorithm 2: Parallel ray-tracing.

```
locate first particle;
forall particles do
    if previous particle has been located then
        dummy = previous particle;
    else
        dummy = nearest element centroid among 20 random
         elements;
    end
    trace ray from dummy to particle position, namely try to
     locate particle;
    if located then
        particle element ID = last dummy element ID;
    end
end
communicate to adjacent subdomains the particle IDs that have
 been located;
forall non located particles do
    if located in adjacent subdomain then
        remove in the current process;
    end
end
while dummies are not all located do
    forall non located particles do
        send the dummy to the new process;
    end
    forall received dummies do
        try to locate dummy;
        if located then
            particle element ID = last dummy element ID;
            remove particle in the starting process;
        end
    end
end
```

In order to speed up the execution and increase scalability, some improvements are included in the code. First, the global ID assigned to each particle allows a direct comparison between two particles in different subdomains. Also, less information is exchanged among processes to identify a particle because only an integer number has to be specified instead of the position vector. Furthermore, when a ray is communicated, the new starting point coincides with the centroid of the element it belongs to. Even in this case, only the cell ID has to be communicated instead of a three-dimensional array.

4.1. Performances

The location algorithm has been tested on a 16-node cluster. Each node has two 6-core Intel Xeon X5650 @2.66 GHz equipped with 32 GB of DDR3 memory. A dual Gigabit Ethernet network connects all the nodes. The performances of the location algorithm are evaluated inside an unstructured tetrahedral mesh generated from a unitary cube geometry. During the tests of the location algorithm, 24 cores were

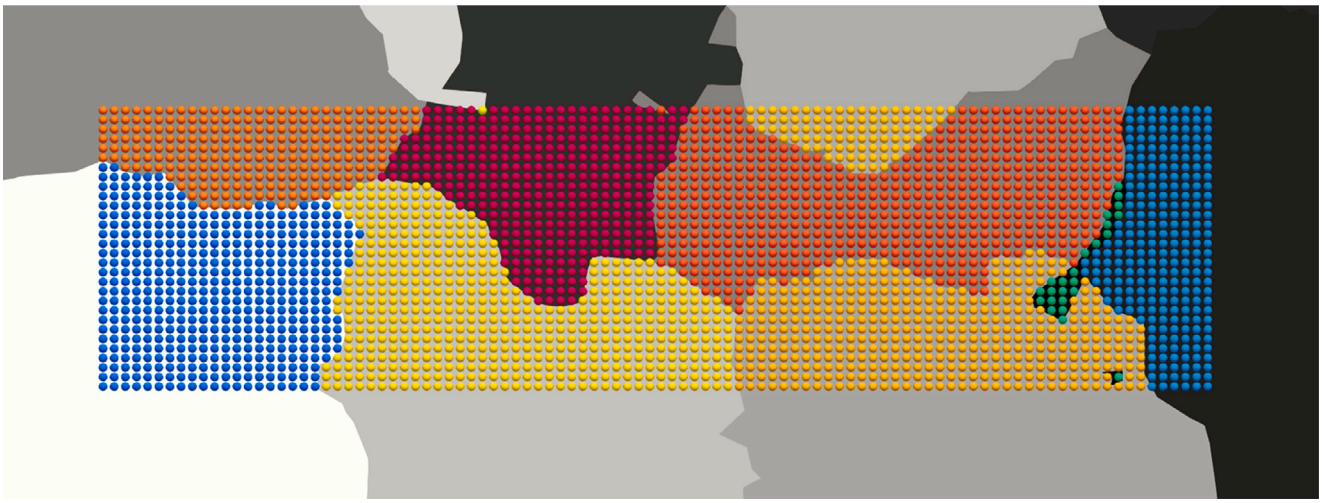
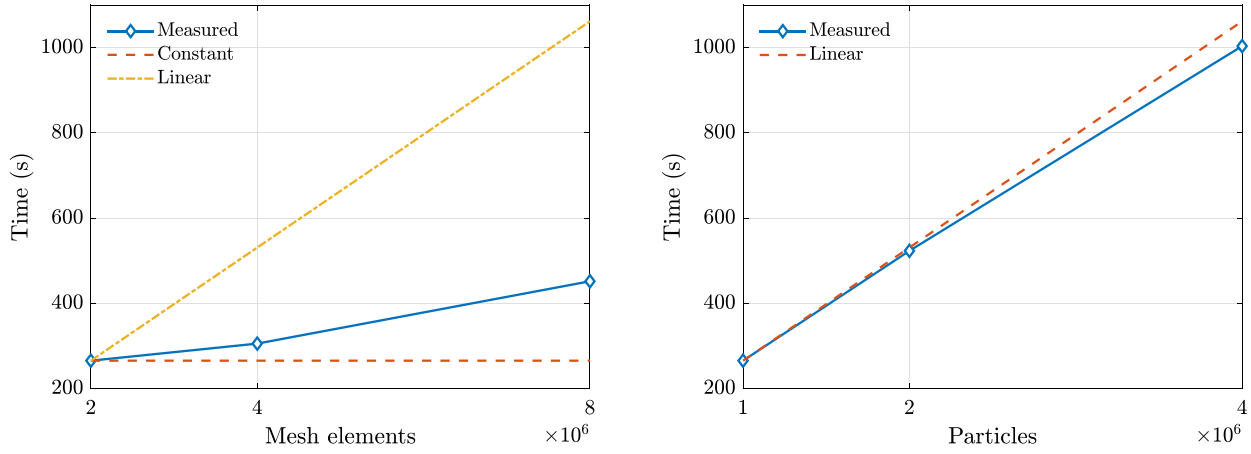


Fig. 4. Mesh and particle slice, each color corresponds to a subdomain.



(a) Mesh size influence for 1M particles.

(b) Cloud size influence on a 2M tetrahedral mesh.

Fig. 5. Location time using 24 cores.

used. In Fig. 5(a), one million particles are located in different mesh sizes. It is noticeable that the mesh size slightly influences the algorithm because the computational time is mainly related to the number of crossed cells. Starting a ray from the previously located particle allows keeping this value almost constant, taking advantage of the ordered distribution of the particles. The other test is performed by varying the number of particles in a mesh composed of two million elements. The implementation scales almost linearly with the number of particles, as reported in Fig. 5(b). Also in this case, the result is expected since the algorithm traces a ray for each located particle. In addition, Fig. 6 reports the overall speed-up factor of the algorithm for a strong scaling test. The specific test case shows a speed-up factor of 3.5 moving from 24 to 96 cores. If we consider the real number of initialized particles, including the ones that are localized multiple times since they are owned by more than one process, and we normalize the result, scaling is almost superlinear.

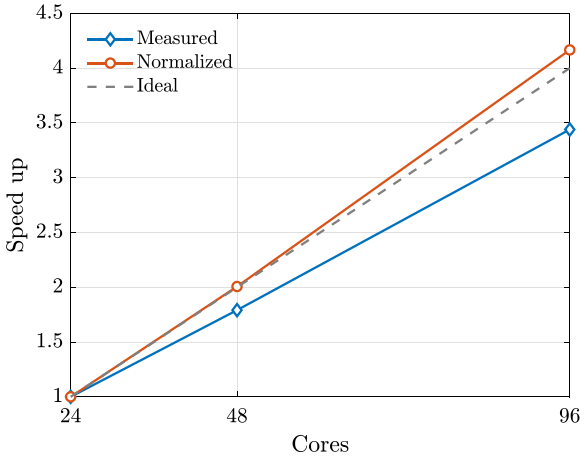
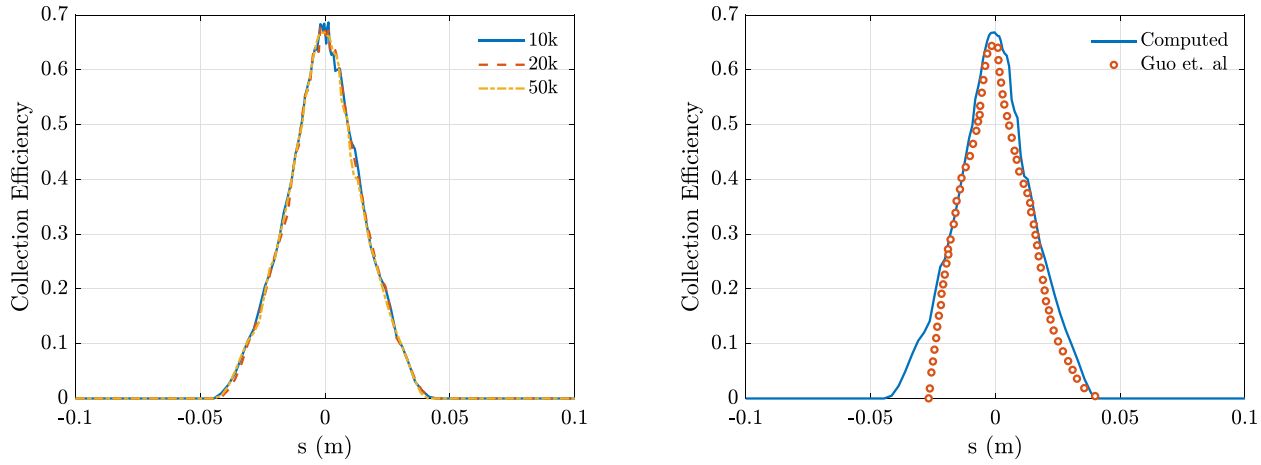


Fig. 6. Speed-up factor of the location algorithm.

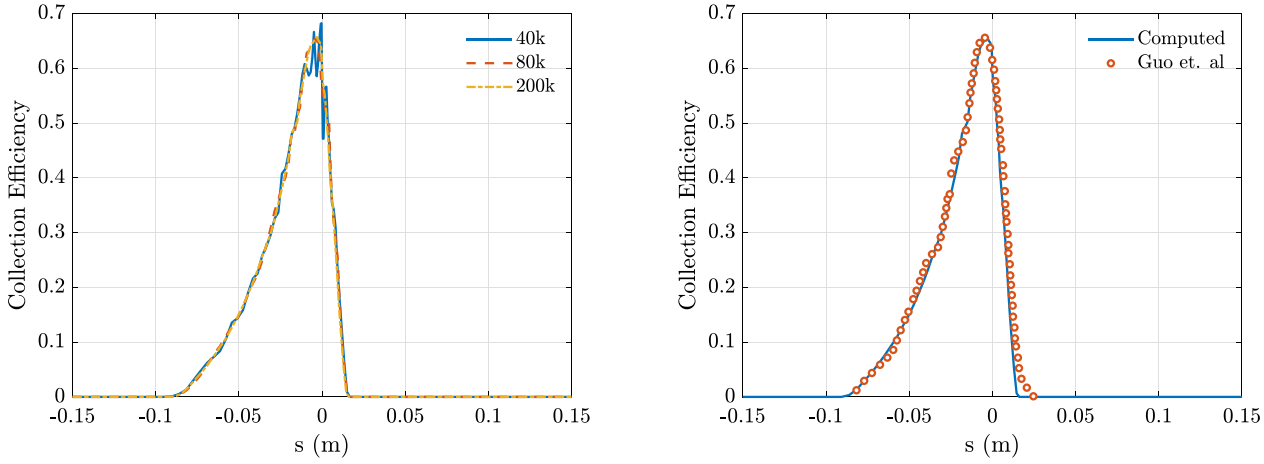
5. Results

The collection efficiency of two steady flow test cases, and an unsteady one, is computed to assess the current code. All the flow solutions are calculated using the SU2 suite. The compressible Reynolds averaged Navier–Stokes equations are discretized using a vertex-based finite volume method. Convective fluxes are discretized using a limited second-order MUSCL scheme with an Approximate Riemann Solver of Roe type. The Venkatakrishnan slope limiter is employed. Viscous



(a) Particle convergence. (b) Collection efficiency.

Fig. 7. Comparison at $\alpha = 0°$ with Guo et al. [34].



(a) Particle convergence. (b) Collection efficiency.

Fig. 8. Comparison at $\alpha = 4°$ with Guo et al. [34].

fluxes are discretized using a standard corrected average of gradients approach. Gradients are obtained via a weighted least-squares approach. For steady computations, a time-marching approach is used to drive the RANS system to a steady solution using an implicit Euler scheme, whereas unsteady simulations are performed via a dual time stepping technique yielding formal second-order accuracy in time. The turbulence model of Menter [32] is used in the three-dimensional cases, whereas the one-equation Spalart–Allmaras model [33] is used in the two-dimensional ones to close the RANS system.

For each test case, proper mesh resolution was obtained via a grid independence study. The pressure coefficient distribution on the clean airfoil was used to assess grid independence. Where available, the computed pressure coefficient was compared to the experimental one to identify possible wind tunnel corrections to the angle of attack. The particles are integrated using an explicit Euler scheme and a maximum time step of 10^{-5} in all the following computations.

The first test case is a classical NACA 0012 wing section analyzed at different angles of attack and flow conditions. The second is a three-dimensional NACA 64A008 swept horizontal tail at negative incidence. The unsteady simulation starts from a flow over a 3D cylinder at Reynolds equal to 5000. The computed collection efficiency, namely, the water impinging per unit surface, is compared with data obtained in previous experimental campaigns or by other codes. The collection efficiency β_i for each cell is defined according to the following equation

$$\beta_i = \frac{\rho_i}{\rho_\infty}$$

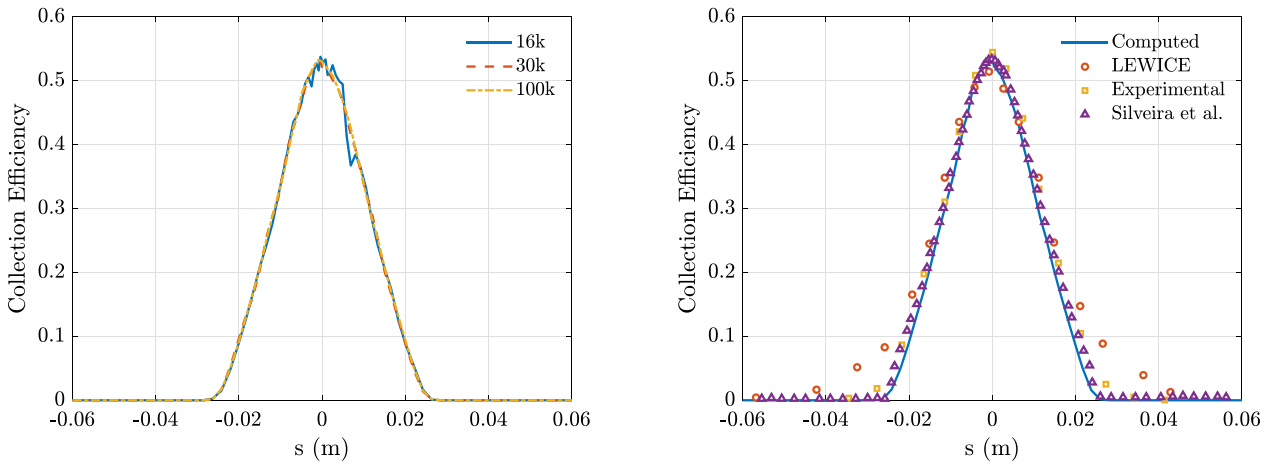
where ρ_i is the particle density at the ith element and ρ_∞ the undisturbed one. If the cloud is uniformly distributed, the equation can be simplified as follows

$$\beta_i = \frac{m_i/A_i}{\mathrm{LWC}}$$

where m_i and A_i are the collected water mass and the area of the ith element, respectively, and the LWC is the liquid water content of the free-stream cloud.

5.1. NACA 0012 wing section

The NACA 0012 symmetric wing section is used here because it was thoroughly tested in icing wind tunnel experiments [35–37]. Furthermore, collection efficiency simulated with NUAA-ICE2D [34] and LEWICE [38] codes is available. First, we compare the implemented code with the one presented by Guo et al. [34]. Then, we consider another flow condition, maintaining the geometry unchanged, to verify if the computed collection efficiency agrees with results presented in literature [36,37]. In both cases, we first assess the convergence of the computed collection efficiency by changing the number of particles. An



(a) Particle convergence.

(b) Collection efficiency.

Fig. 9. Comparison with Silveira et al. [36].

O-Grid mesh has been adopted for the computation with 55.7k elements and 56.1k nodes.

The flight and icing conditions in the first analysis are taken from Guo et al. [34]. The free-stream velocity is set equal to V_∞ = 67 m/s, while ambient pressure is P_∞ = 98900 Pa, the static temperature is T_∞ = −7 °C, the angle of attack is α = 0° and α = 4°, the median volume diameter is MVD = 25 μm, and the profile chord measures 0.914 m.

In Figs. 7 and 8 are reported the results at 0° and 4°, respectively. The first angle of attack has been chosen since a symmetric solution is expected. The symmetry of the solution is well captured, and results slightly differ from the data reported in [34]. At positive incidence, the data computed by Guo et al. [34] is close to the one presented in this work. The second conditions are set according to the ones presented by Silveira et al. [36]. The free-stream velocity is set equal to V_∞ = 44.39 m/s, while the ambient pressure is P_∞ = 1 atm, the static temperature is T_∞ = 265 K, the angle of attack is α = 0°, the median volume diameter is MVD = 20 μm, and the profile chord measures 0.914 m. In Fig. 9, particle convergence and computed collection efficiency are presented. All solutions are in good agreement near the profile nose. Moving away from the stagnation point, the present code and the one presented in [36] slightly underestimate the experimental collection efficiency. On the opposite, the LEWICE code significantly overestimates it. This is due to the experimental particle diameter distribution adopted in the LEWICE computation. The presence of droplets with a diameter greater than the mean value used in the other cases tends to increase the collection efficiency [39,40].

5.2. NACA 64A008 swept horizontal tail

The three-dimensional NACA 64A008 swept horizontal tail has been chosen since numerical results from the LEWICE code and NASA experimental data [41] are available. In addition, it is a test case of the 1st AIAA Ice Prediction Workshop. The hybrid mesh adopted provided as part of the workshop is composed of 16.8M cells (tetrahedra and prisms) and 3.82M nodes [42]. The air and ice conditions are equal to Papadakis et al. [41] and correspond to V_∞ = 78.68 m/s, P_∞ = 95147 Pa, T_∞ = 280 K, Mach M_∞ = 0.23, Reynolds $Re_\infty = 5 \cdot 10^6$, and median volume diameter MVD = 21 μm. First, we compare the pressure distributions with the experimental ones. Wind tunnel effects are compensated in the numerical flow simulation changing the angle of attack α from −6° to −6.25°. In Fig. 10, it is visible how the two solutions are almost overlapping. The comparison of collection efficiency, Fig. 11, presents little discrepancies on the upper part of the profile. On the lower, LEWICE data and the computed result overlap, while the experimental campaign presents a higher value.

Table 1
Steady simulation execution time to evolve 1M particles.

Cores	Without tagging (s)	With tagging (s)	Time reduction
24	47244	15144	67.95%
48	41552	8858	78.68%
96	36819	5183	85.92%

Table 2
Unsteady simulation execution time to evolve 1M particles.

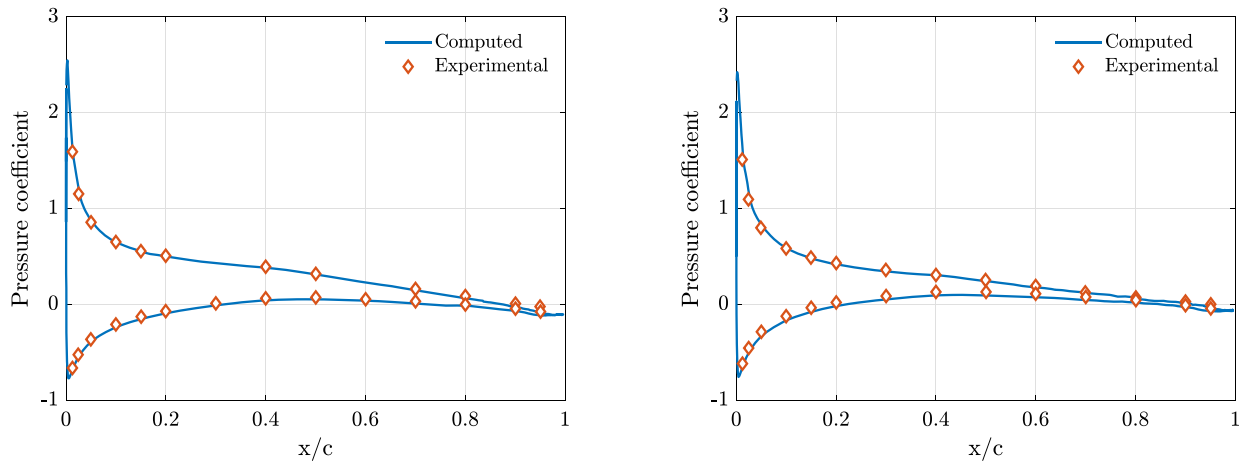
Cores	Without tagging (s)	With tagging (s)	Time reduction
24	33343	16100	51.71%
48	30789	10401	66.22%
96	29213	6421	78.02%

The parallel performances are tested by evolving one million particles in the three-dimensional geometry. The simulations are run using 2, 4, and 8 nodes corresponding to 24, 48, and 96 cores, respectively. The particle integration is computed twice, with and without the cell tagging technique. The results are summarized in Fig. 12 and Table 1. The speed-up factor using a standard domain subdivision is almost flat. The behavior is entirely changed by adopting the proposed improvements, reducing the execution time by a factor of 3, passing from 24 to 96 cores. Also, the overall time is significantly lower, becoming about one-sixth in the last case.

5.3. Unsteady 3D cylinder

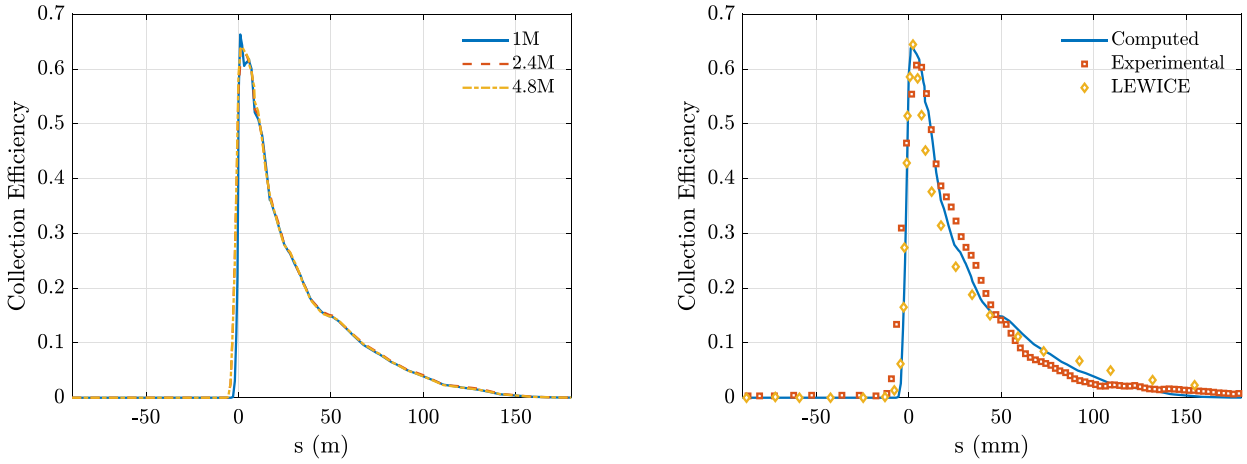
An unsteady three-dimensional cylinder at Reynolds equal to 5000 is tested. This case has been chosen because the flow solution is strongly time-dependent and presents recirculating bubbles and 3D structures in the wake [43]. The cylinder length is ten times the diameter. The flow solution is computed using SU2 [44] on a hexahedral mesh composed of 3.5M elements. A second-order dual time stepping technique is used to solve the unsteady RANS equations with a 0.01 s time step. The free-stream conditions are: Mach number M_∞ = 0.1, temperature T_∞ = 288.15 K and Re_∞ = 5000. Two different particle simulations are performed, the first using a Median Volume Diameter (MVD) = 20 μm and the second MVD = 50 μm. In both cases, the collection efficiency is evaluated over one period (Fig. 13).

The first test case aims to evaluate the code performances, evolving 1M droplets. The particle diameter, and consequently the inertia, is too small to impact the cylinder surface, so the impinging droplet density



(a) Inboard. (b) Outboard.

Fig. 10. Pressure coefficient comparison with Papadakis et al. [41].



(a) Particle convergence. (b) Collection efficiency.

Fig. 11. Comparison with Papadakis et al. [41].

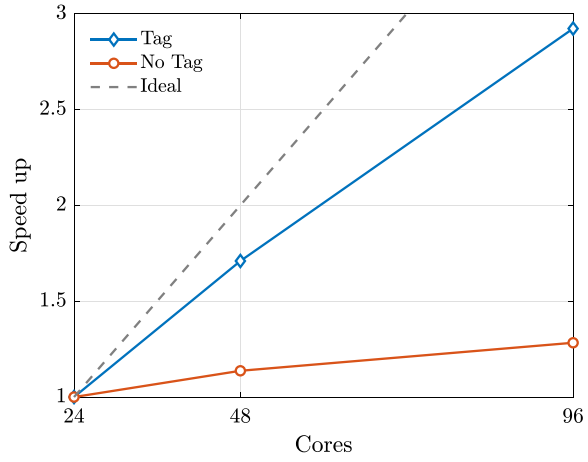


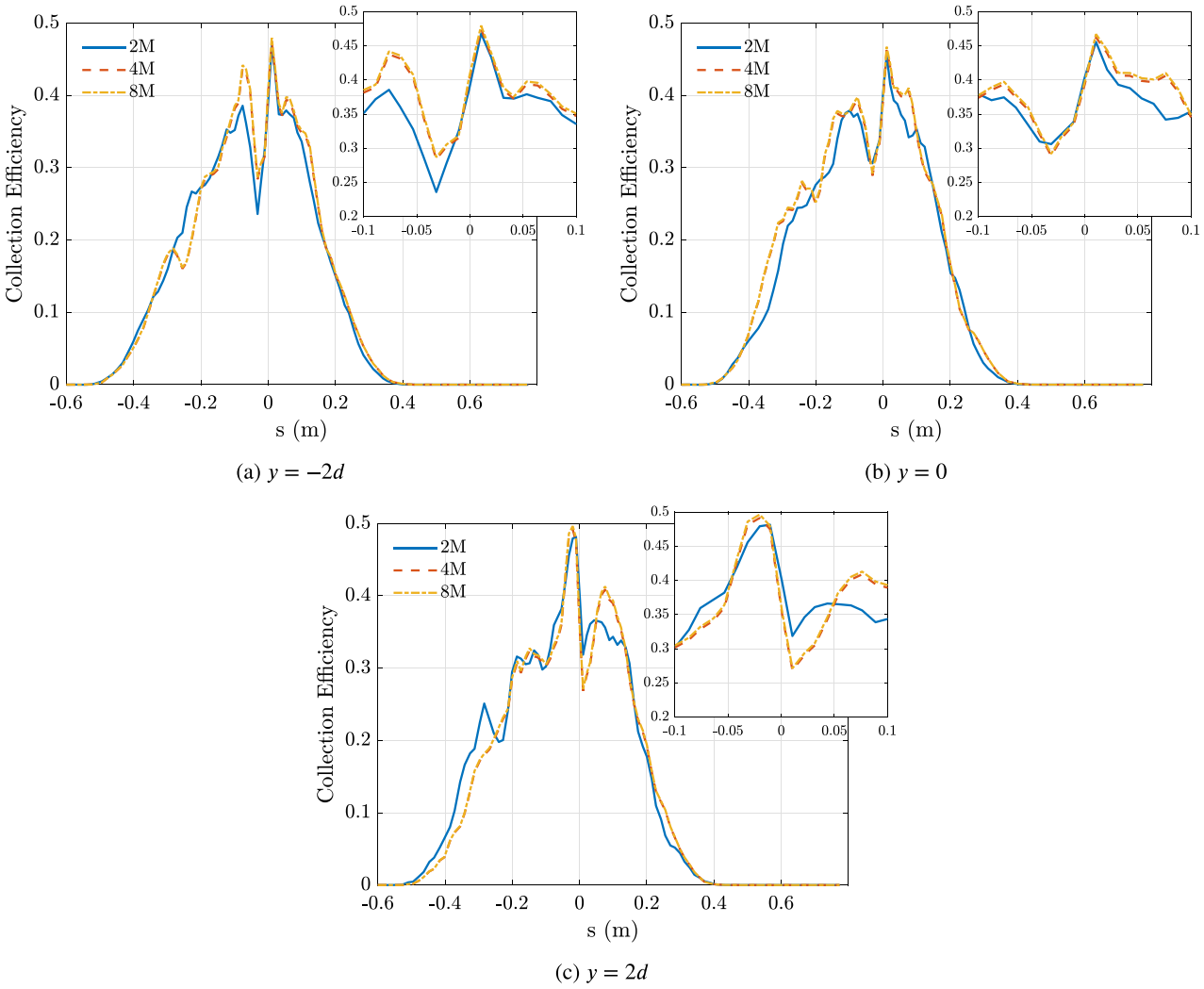
Fig. 12. Steady simulation speed-up factor evolving 1M particles.

per unit surface, namely, the collection efficiency, is negligible. In Fig. 15, a graphical representation of the cloud is given at a different time. For the specific case, the speed-up factor is reported in Fig. 14. It is clear how the performances improve by adopting the cell tagging technique. Doubling the number of cores in the naive implementation leads to a 1.07 speed-up factor, while in the presented implementation, it reaches 1.55. A reduction of the execution time up to 78% is achieved using 96 cores as reported in Table 2. Not surprisingly, the performances are lower than in the steady case mainly due to the need to synchronize the particle time at each CFD time level [45].

6. Conclusions

An efficient parallel Lagrangian particle tracking is presented to solve dispersed flows. The main novelty concerns the subdivision of the domain among processes. Indeed, the problem's multi-physics nature is also reflected in the partitioning algorithm. The naive domain subdivision based only on the fluid mesh elements leads to unacceptable performances during particle integration. Adopting the two-constrained partitioning allows an equal subdivision of cells for each process, accounting for the memory limit and, simultaneously, the computational power necessary to evolve the cloud. The innovative tagging technique can reduce the resources significantly, providing the same results in steady and unsteady simulations.

An efficient parallel ray-tracing algorithm to locate particles in unstructured meshes is presented. It can handle arbitrarily shaped subdomains and shows an almost ideal scaling in the test cases. The



(a) $y = -2d$ (b) $y = 0$

(c) $y = 2d$

Fig. 13. Convergence of particles (MVD = 50 μm).

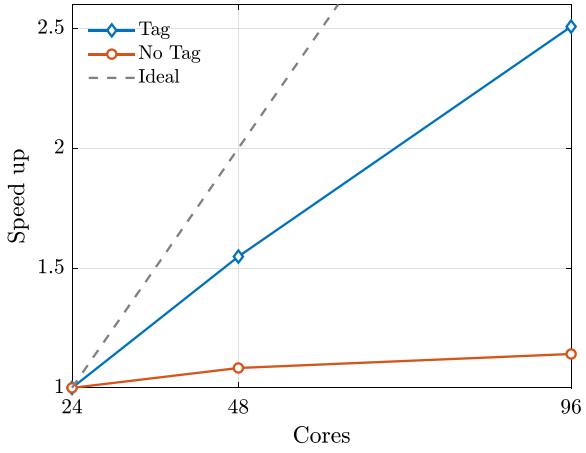


Fig. 14. Unsteady simulation speed-up factor evolving 1M particles.

algorithm can perform cloud localization using less than one order of magnitude of computational time than a brute force procedure.

Since the problem assumes a collision-less one-way coupling between fluid and particle systems, each particle is evolved independently. A shared-memory parallelization is perfectly suitable for cloud integration and can be implemented on top of the present work, obtaining a hybrid parallelization. However, the cell tagging technique aims at providing an efficient and straightforward strategy for using multi-node and distributed architectures.

CRediT authorship contribution statement

Giacomo Baldan: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Tommaso Bellosta:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing. **Alberto Guardone:** Conceptualization, Data curation, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

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

Data availability

The data that has been used is confidential.

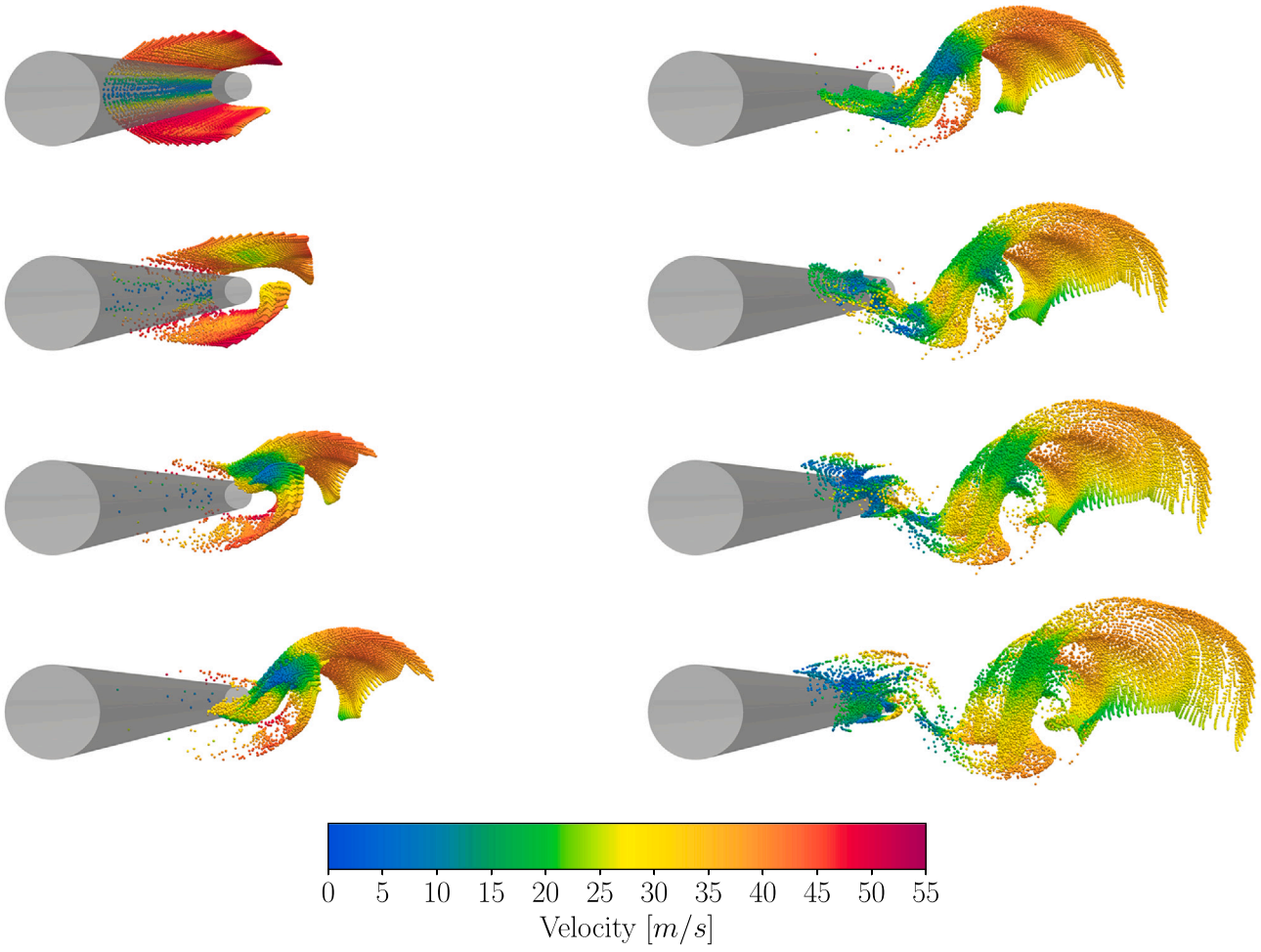


Fig. 15. Cloud evolution (MVD = 20 μm), each figure is taken after a $\Delta t = 0.0025$ s.

References

[1] Widhalm M, Ronzheimer A, Meyer J. Lagrangian particle tracking on large unstructured three-dimensional meshes. In: 46th AIAA aerospace sciences meeting and exhibit. 2008, http://dx.doi.org/10.2514/6.2008-472.

[2] Kopper P, Schwarz A, Copplestone SM, Ortwein P, Staudacher S, Beck A. A framework for high-fidelity particle tracking on massively parallel systems. 2022, http://dx.doi.org/10.48550/ARXIV.2211.05458.

[3] Wedel J, Steinmann P, Štrakl M, Hriberšek M, Ravnik J. Can CFD establish a connection to a milder COVID-19 disease in younger people? Aerosol deposition in lungs of different age groups based on Lagrangian particle tracking in turbulent flow. Comput Mech 2021;1–17. http://dx.doi.org/10.1007/s00466-021-01988-5.

[4] Vångö M. CFD modelling of direct gas injection using a Lagrangian particle tracking approach. 2015.

[5] Kaario OT, Vuorinen V, Kahila H, Im HG, Larmi M. The effect of fuel on high velocity evaporating fuel sprays: Large-Eddy simulation of Spray A with various fuels. Int J Engine Res 2020;21(1):26–42. http://dx.doi.org/10.1177/1468087419854235.

[6] Ikegawa M, Ishii E, Harada N, Takagishi T. Development of ink-particle flight simulation for continuous inkjet printer. In: ASME international mechanical engineering congress and exposition. Volume 7A: Fluids engineering systems and technologies, 2013, http://dx.doi.org/10.1115/IMECE2013-63094.

[7] Hamed A, Das K, Basu D. Numerical simulations of ice droplet trajectories and collection efficiency on aero-engine rotating machinery. In: 43rd AIAA aerospace sciences meeting and exhibit. 2005, http://dx.doi.org/10.2514/6.2005-1248, URL: https://arc.aiaa.org/doi/abs/10.2514/6.2005-1248.

[8] Zhang D, Chen W. Numerical investigation of supercooled droplets impingement on complicated icing surface. Modern Phys Lett B 2009;23(03):469–72. http://dx.doi.org/10.1142/S0217984909018679.

[9] Yassin K, Kassem H, Stoevesandt B, Klemme T, Peinke J. Numerical simulation of roughness effects of ice accretion on wind turbine airfoils. Energies 2022;15(21). http://dx.doi.org/10.3390/en15218145, URL: https://www.mdpi.com/1996-1073/15/21/8145.

[10] Ge W, Sankaran R, Chen JH. Development of a CPU/GPU portable software library for Lagrangian–Eulerian simulations of liquid sprays. Int J Multiph Flow 2020;128:103293. http://dx.doi.org/10.1016/j.ijmultiphaseflow.2020.103293.

[11] Ansari SU, Hussain M, Mazhar S, Manzoor T, Siddiqui KJ, Abid M, et al. Mesh partitioning and efficient equation solving techniques by distributed finite element methods: A survey. Arch Comput Methods Eng 2019;26:1–16. http://dx.doi.org/10.1007/s11831-017-9227-2.

[12] Kaludercic B. Parallelisation of the Lagrangian model in a mixed Eulerian–Lagrangian CFD algorithm. J Parallel Distrib Comput 2004;64(2):277–84. http://dx.doi.org/10.1016/j.jpdc.2003.11.010.

[13] Capodaglio G, Aulisa E. A particle tracking algorithm for parallel finite element applications. Comput & Fluids 2017;159:338–55. http://dx.doi.org/10.1016/j.compfluid.2017.10.015.

[14] Heinrich M, Schwarze R. 3D-coupling of Volume-of-Fluid and Lagrangian particle tracking for spray atomization simulation in OpenFOAM. SoftwareX 2020;11:100483. http://dx.doi.org/10.1016/j.softx.2020.100483.

[15] Gent R, Dart N, Cansdale J. Aircraft icing. Phil Trans R Soc A 2000;358(1776):2873–911. http://dx.doi.org/10.1098/rsta.2000.0689.

[16] Cebeci T, Kafyeke F. Aircraft icing. Annu Rev Fluid Mech 2003;35(1):11–21. http://dx.doi.org/10.1146/annurev.fluid.35.101101.161217.

[17] Karypis G, Schloegel K, Kumar V. Parmetis parallel graph partitioning and sparse matrix ordering library. 1997.

[18] Schloegel K, Karypis G, Kumar V. Parallel multilevel algorithms for multi-constraint graph partitioning. In: Euro-Par. 2000.

[19] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. J Parallel Distrib Comput 1998;48(1):96–129. http://dx.doi.org/10.1006/jpdc.1997.1404.

[20] Zhang L, Zhang G, Liu Y, Pan H. Mesh partitioning algorithm based on parallel finite element analysis and its actualization. Math Probl Eng 2013. http://dx.doi.org/10.1155/2013/751030.

[21] Chevalier C, Pellegrini F. PT-scotch: A tool for efficient parallel graph ordering. Parallel Comput 2008;34(6):318–31. http://dx.doi.org/10.1016/j.parco.2007.12.001.

[22] Devine K, Boman E, Heapby R, Hendrickson B, Vaughan C. Zoltan data management service for parallel dynamic applications. Comput Sci Eng 2002;4:90–7. http://dx.doi.org/10.1109/5992.988653.
[23] Shang Z. Large-scale CFD parallel computing dealing with massive mesh. J Eng 2013;2013:1–6. http://dx.doi.org/10.1155/2013/850148.
[24] Baldan G, Borell R, Jägersküpper J. A runtime-based dynamic mesh-partitioning approach. In: The 8th European congress on computational methods in applied sciences and engineering. 2022, http://dx.doi.org/10.23967/eccomas.2022.028.
[25] Gori G, Zocca M, Garabelli M, Guardone A, Quaranta G. PoliMIce: A simulation framework for three-dimensional ice accretion. Appl Math Comput 2015;267:96–107. http://dx.doi.org/10.1016/j.amc.2015.05.081, The fourth European seminar on computing (ESCO 2014).
[26] Bellosta T, Parma G, Guardone A. A robust 3D particle tracking solver for in-flight ice accretion using arbitrary precision arithmetic. 2019, Coupled 2019.
[27] Schroeder W, Martin K, Lorensen B. The visualization toolkit, an object-oriented approach to 3D graphics. 4th ed. Kitware, Inc.; 2006.
[28] Sugihara K, Tatebe O. Design of locality-aware MPI-IO for scalable shared file write performance. In: 2020 IEEE international parallel and distributed processing symposium workshops. IPDPSW, 2020, p. 1080–9. http://dx.doi.org/10.1109/IPDPSW50202.2020.00179.
[29] Arya S, Mount DM, Netanyahu NS, Silverman R, Wu AY. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J ACM 1998;45(6):891–923. http://dx.doi.org/10.1145/293347.293348.
[30] Friedman JH, Bentley JL, Finkel RA. An algorithm for finding best matches in logarithmic expected time. ACM Trans Math Software 1977;3(3):209–26. http://dx.doi.org/10.1145/355744.355745.
[31] Strobl S, Bannerman MN, Pöschel T. Robust event-driven particle tracking in complex geometries. Comput Phys Comm 2020;254:107229. http://dx.doi.org/10.1016/j.cpc.2020.107229.
[32] Menter FR. Two-equation eddy-viscosity turbulence models for engineering applications. AIAA J 1994;32(8):1598–605. http://dx.doi.org/10.2514/3.12149.
[33] Spalart P, Allmaras S. A one-equation turbulence model for aerodynamic flows. In: 30th aerospace sciences meeting and exhibit. 1992, http://dx.doi.org/10.2514/6.1992-439.
[34] Guo T, Zhu C, Zhu C. An efficient and robust ice accretion code: NUAA-ICE2D. In: 2016 IEEE international conference on aircraft utility systems. 2016, p. 739–46. http://dx.doi.org/10.1109/AUS.2016.7748151.
[35] Shin J, Bond T. Results of an icing test on a NACA 0012 airfoil in the NASA Lewis Icing Research Tunnel. In: 30th aerospace sciences meeting and exhibit. 1992, http://dx.doi.org/10.2514/6.1992-647.
[36] Silveira R, Maliska C, Estivam DA, Mendes R. Evalutation of collection efficiency mehods for icing analisys. 2003.
[37] Morency F, Tezok F, Paraschivoiu I. Anti-icing system simulation using CANICE. J Aircr 1999;36(6):999–1006. http://dx.doi.org/10.2514/2.2541.
[38] Wright W. User's manual for LEWICE version 3.2. 2008.
[39] Baldan G, Bellosta T, Guardone A. Efficient parallel algorithms for coupled fluid-particle simulation. In: 9th edition of the international conference on computational methods for coupled problems in science and engineering. CIMNE; 2021, http://dx.doi.org/10.23967/coupled.2021.021.
[40] Bellosta T, Baldan G, Sirianni G, Guardone A. Lagrangian and eulerian algorithms for water droplets in in-flight ice accretion. Journal of Computational and Applied Mathematics 2023;115230. http://dx.doi.org/10.1016/j.cam.2023.115230.
[41] Papadakis M, Hung K, Vu GT, Yeong H, Bidwell C, Breer MD, et al. Experimental investigation of water droplet impingement on airfoils, finite wings, and an S-duct engine inlet. 2002.
[42] Laurendeau E, Bourgault-Cote S, Ozcer IA, Hann R, Radenac E, Pueyo A. Summary from the 1st AIAA ice prediction workshop. In: AIAA AVIATION 2022 forum. http://dx.doi.org/10.2514/6.2022-3398, URL: https://arc.aiaa.org/doi/abs/10.2514/6.2022-3398.
[43] Aljure DE, Lehmkhul O, Rodríguez I, Oliva A. Three dimensionality in the wake of the flow around a circular cylinder at Reynolds number 5000. Comput & Fluids 2017;147:102–18. http://dx.doi.org/10.1016/j.compfluid.2017.02.004.
[44] Palacios F, Economon TD, Aranake AC, Copeland SR, Lonkar AK, Lukaczyk T, et al. Stanford University unstructured (SU2): Open-source analysis and design technology for turbulent flows. In: 52nd aerospace sciences meeting. 2014, http://dx.doi.org/10.2514/6.2014-0243.
[45] Baldan G, Bellosta T, Guardone A. A scalable Lagrangian particle tracking method. In: VII international conference on particle-based methods. CIMNE; 2021, http://dx.doi.org/10.23967/particles.2021.007.