VII International Conference on Particle-Based Methods
PARTICLES 2021
P. Wriggers, M. Bischoff, E. Oñate, M. Bischoff, A. Düster & T. Zohdi (Eds)

A SCALABLE LAGRANGIAN PARTICLE TRACKING METHOD

Giacomo Baldan*†, Tommaso Bellosta† and Alberto Guardone†

† Department of Aerospace Science and Technology
Politecnico di Milano
Via La Masa 34, 20156 Milano, Italy
e-mail: giacomo.baldan@mail.polimi.it

Key words: Particle tracking, Mesh partitioning, Unsteady simulation, Parallel computation

Abstract. Particle tracking within an underlying flow field is routinely used to analyse both industrial processes and natural phenomena. In a computer code running on a distributed-memory architecture, the different behaviour of fluid-particle systems must be taken into account to properly balance element-particle subdivision among processes. In unsteady simulations, the parallel efficiency is even more critical because it changes over time. Another challenging aspect of a scalable implementation is the initial particle location due to the arbitrary shapes of each subdomain. In this work, an innovative parallel ray tracing particle location algorithm and a two-constrained domain subdivision are presented. The former takes advantage of a global identifier for each particle, resulting in a significant reduction of the overall communication among processes. The latter is designed to mitigate the load unbalance in the particles evolution while maintaining an equal element distribution. A preliminary particle simulation is performed to tag the cells and compute a weight proportional to the probability to be crossed. The algorithm is implemented using MPI distribute memory environment. A cloud droplet impact test case starting from an unsteady flow around a 3D cylinder has been simulated to evaluate the code performances. The tagging technique results in a computational time reduction of up to 78% and a speed up factor improvement of 44% with respect to the common flow-based domain subdivision. The overall scalability is equal to 1.55 doubling the number of cores.

1 Introduction

Particle-laden flows can be found in industrial processes and natural phenomena. In literature, several applications are reported such as dry-powder inhalers [1], environmental DNA transport in the coastal ocean [2], complex interactions in molecular dynamics [3], or in-flight ice accretion problems [4, 5]. Lagrangian and Eulerian approaches have been proposed to model particle motion in fluids. In the Lagrangian model, the motion of each particle is simulated. In contrast, particles are regarded as a continuum in the Eulerian

model and their average spacing is described by a particle density function [6].
The need of a parallel Lagrangian particle tracking is mainly connected to two factors. First of all, the increasing size of computational domains can be incompatible with the available resources. In addition, smaller elements need a more refined cloud to maintain the number of impinging particles on each cell statistically significant. Therefore, evolving a considerable number of particles leads to an excessive running time. There are some difficulties in a scalable parallelization such as the quite unpredictable behaviour of the flow solution and the particle evolution. Then, the computational cost at each time step is dictated by the process with the highest number of particles to integrate. Finally, the subdomain partitioning must account for memory limit and computational power involved in the particles evolution. If an unsteady flow solution is considered all the just mentioned points are even more critical. In addition, the initial particle location cannot be performed using a standard serial algorithm.
In this work, a two-constrained mesh partitioning based on a graph representation is developed in Section 2. Two- and three-dimensional hybrid meshes are supported. The k-way subdivision aims at equally partition the number of elements and the particle workload among processes. The code can be split in three different steps. Firstly, a simulation, using the standard domain subdivision based only on the flow solution, evolves few particles to suitably tag the crossed cells. Secondly, the domain is repartitioned taking into account also the constraint linked to the cloud distribution. Finally, the real simulation is performed. In Section 3, an innovative parallel ray-tracing algorithm is described to locate particles inside the mesh at the beginning of the simulation. In Section 4, the flow unsteadiness effects on the efficiency of the code are highlighted. Finally, a cloud droplet impact test case starting from an unsteady flow around a 3D cylinder has been simulated to evaluate both the collection efficiency at different positions and the code performances.

2 Two-constrained mesh partitioning

Mesh partitioning is mandatory if a parallel distribute memory architecture is adopted. The standard subdivision technique, based only on the connectivity of the mesh, create a set of subdomains that are composed of an equal number of elements or minimize data exchange at the interfaces [7]. When a multi-physics problem is analysed, such as a particle-laden flow, a multi-constrained subdivision should be used.
In this work, an efficient mesh partitioning that balances flow solution and particle evolution workload is implemented. The goal is reached using Parmetis library [8, 9] that offers a parallel multi-level and multi-constraint k-way subdivision [10, 11] through a C/C++ API. It has a sequential complexity of the serial multi-constraint algorithm of $O(mn)$ where m is the number of constraints and n of nodes. The isoefficiency functions, using p processes, are $O(p^2 \log p)$. If compared to other methods, like geometric, spectral, and combination, it takes less time. The domain is treated as a nodal graph, where each mesh node corresponds to a graph node. The alternative approach is to convert the mesh into the dual graph, where each graph node represents an element in the original domain. In this way, the conversion, which is time demanding as much as the graph subdivision, is

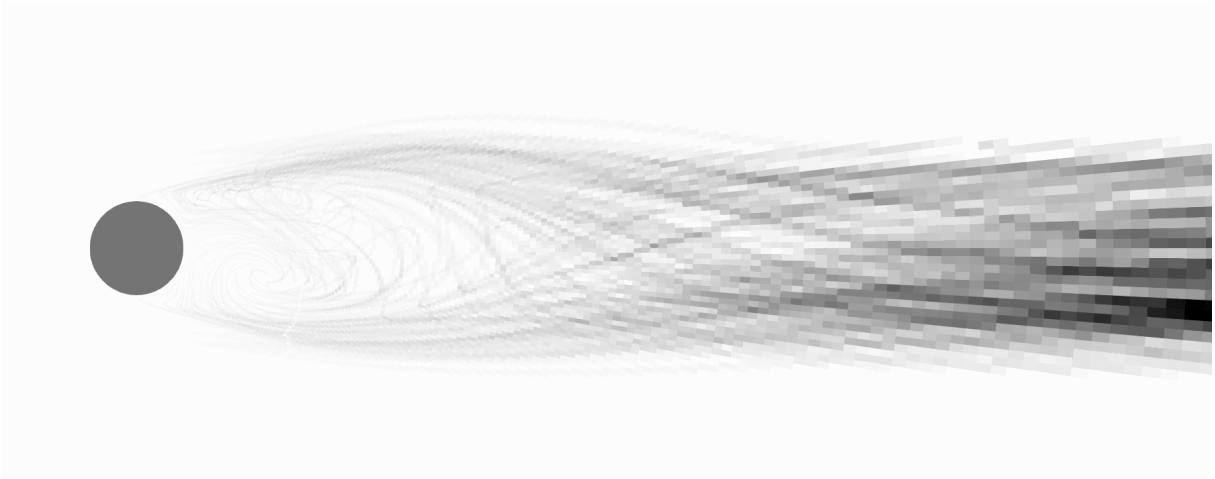

Figure 1: Tagged elements during the preliminary simulation

avoided. Where the domain is cut new non-physical boundaries are created. Since the partitioning is based on nodes, a layer of overlapping elements is present at the interfaces among processes. The cells owned by more than a process are called halo elements. In the particle tracking, their presence is crucial because the particle is located in the new process just using a local to global, and vice-versa, element mapping.
The two-constrained mesh partitioning is based on a cell tagging technique. Firstly, a preliminary simulation is performed using the standard geometric mesh subdivision. Only few significant particles are evolved. The algorithm counts the particles that cross each element of the mesh. Figure 1 is an example of an unsteady 2D cylinder. Then, the cell weights are converted into node ones thanks to the mesh connectivity. In such a way, the other constraint linked to the cloud distribution is retrieved. Finally, ParMetis routine is called a second time to perform the two-constrained subdivision and the real simulation is computed efficiently. A detailed description of the domain partitioning algorithm is presented in [12].

3 Parallel ray tracing

At the beginning of the simulation, the position of each particle within the mesh is to be computed. The three main particle location methods present in literature are brute force, tree based [13, 14] and ray tracing [15]. Brute force works in every situation but it is inefficient especially for large domains. Tree based algorithms are not sufficiently robust if the domain is not simply connected or is made up of more than one part. Indeed, particles can be near a boundary element and, at the same time, be outside the subdomain. Ray tracing requires a convex geometry that is not granted due to the arbitrary subdomain shapes. The present code solves the issues considering a parallel ray tracing algorithm in order to recover the original mesh convexity and locate efficiently the cloud. A ray always ends at the position of the particle to be located. The starting point is copied from the previous particle if it has been located, otherwise the code searches the closest element centroid among 50 random cells. Therefore, the ray covers a shorter distance.

Algorithm 1: Parallel ray-tracing

```
locate first particle;
forall particles do
    if previous particle has been located then
        dummy = previous particle;
    else
        dummy = nearest element centroid;
    end
end
forall non located particles do
    if located in adjacent subdomain then
        remove in the other;
    end
end
while dummies are not all located do
    forall non located particles do
        send the dummy to the right process;
    end
    forall received dummies do
        if located then
            remove in the owner;
        else
            send the dummy to the right process;
        end
    end
end
```

The position of the particles is obtained from an uniformly distributed cloud enclosed by a rectangular parallelepiped. The edges are oriented along the Cartesian directions. Three input parameters are needed: the spatial coordinates of two opposite vertexes and the number of particles for each axis. Since the subdomain shapes are not regular, and usually the cloud size is bigger than the subdomain extension, the limits of the initial cloud are adapted in each process. Maximum and minimum assume the strictest mesh node coordinates. Therefore, less particles have to be located. In Figure 2 an example is reported. On the right-hand side, it is visible how some particles are outside the subdomain. At this stage, some particles are present in more than a subdomain and the algorithm have to determine which is the right owner. Another key aspect is the definition of a global rule to treat particles inside halo cells. The code assigns a particle included in a ghost element to the process with the lowest id.
When a mesh is partitioned, the subdomain shapes can be quite challenging for a ray tracing algorithm. A ray can exit and re-enter from a subdomain multiple times. Another possibility is that it crosses more than one subdomain and then it returns to the original

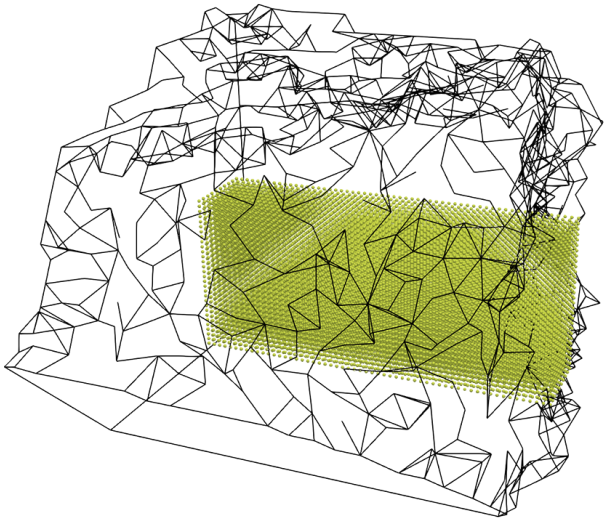

Figure 2: Particles before being located

one. Another rare event is that a ray goes through a process that has no particles to initialize. These situations are more frequent in three-dimensional meshes but can arise also in bidimensional ones. For instance, Figure 3 shows a sliced 3D domain with the localized particles. The present parallel code robustly handles all the possibilities in an efficient way. It is summarised in Algorithm 1. Firstly, all particles are located in each process using the serial algorithm. If a ray reaches an internal boundary the particle is marked. Before continuing to send the rays in all the domains, the algorithm checks if the marked particles have already been located in an adjacent subdomain. If this is the case the particle is eliminated, otherwise the ray is sent to the new process. Checking adjacent processes allows a reduction of communication in the following stages. After, all the rays, initialized in the first phase, are tracked until they reach the final position. At the end, when the cloud is located, extra particles are removed.
Several improvements are included in the code to speed up execution. Firstly, a global id is assigned to each particle allowing a direct comparison between two particles in different subdomains. Also, less information are exchanged among processes to identify a particle. Furthermore, when a ray is communicated, the new starting point coincides with the centroid of the element at which it belongs. Only the cell id has to be sent instead of a three-dimensional array containing the position. The last implementation stratagem regards the ray representation. Since the code integrates the ray tracing algorithm for particle integration, a simplified dummy particle is adopted to evolve a ray.
The location algorithm has been tested on a 16-node cluster. Each node has two 6-core Intel Xeon X5650 @2.66GHz equipped with 32 GB of DDR3 memory. They are connected through a dual Gigabit Ethernet network. An unstructured tetrahedral mesh has been generated starting from a unitary cube geometry. During the tests, two nodes have been used at which corresponds 24 cores. In Figure 4, one million particles are located in

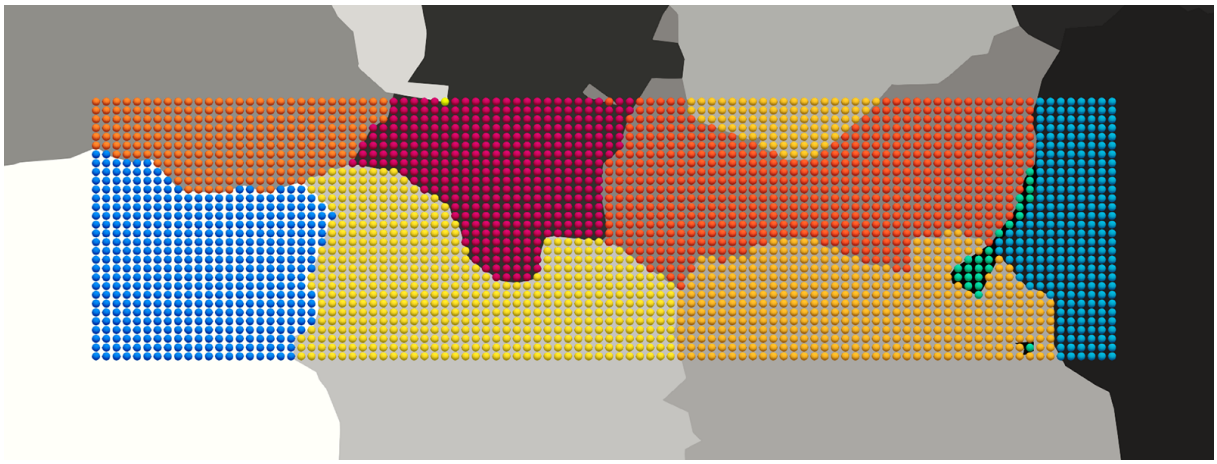

Figure 3: Mesh and particle slice, each colour corresponds to a subdomain

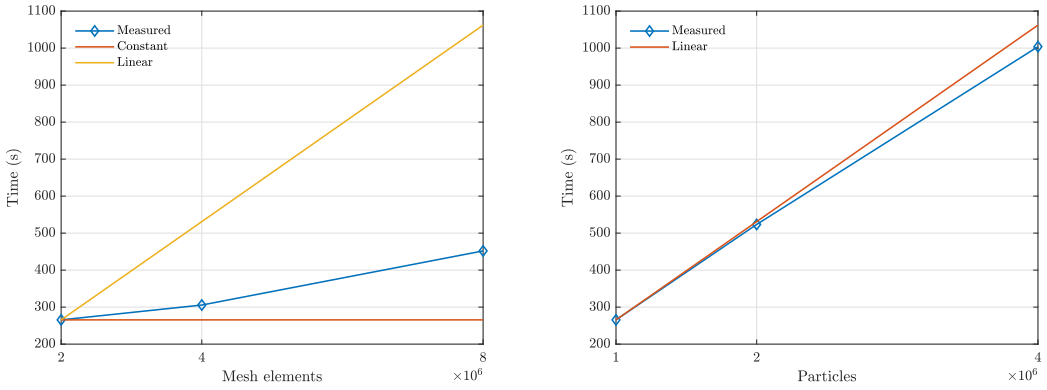

Figure 4: Location time for 1M particles in different size meshes (left) and for different cloud size on a 2M tetrahedral mesh (right) using 24 cores

different mesh sizes. It is noticeable as the algorithm is slightly influenced by the mesh size. The other test is performed varying the number of particles located in a mesh composed by two million of elements. The location takes a bit less than the linear scaling. Both results confirm the robustness of the implementation. In addition, in Figure 5 is reported the overall speed up factor and parallel efficiency of the algorithm. 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, included the ones that are owned by more than one process, and we normalize the result it scales a bit more than ideal.

4 Unsteady simulations

In unsteady simulations, flow structures evolve over time and their evolution is a-priori quite unpredictable. Indeed, the available resources must be concentrated in the areas in which the number of particles is greater. The implemented code aims to uniformly subdivide the computational power avoiding local bottlenecks that arise in the naive

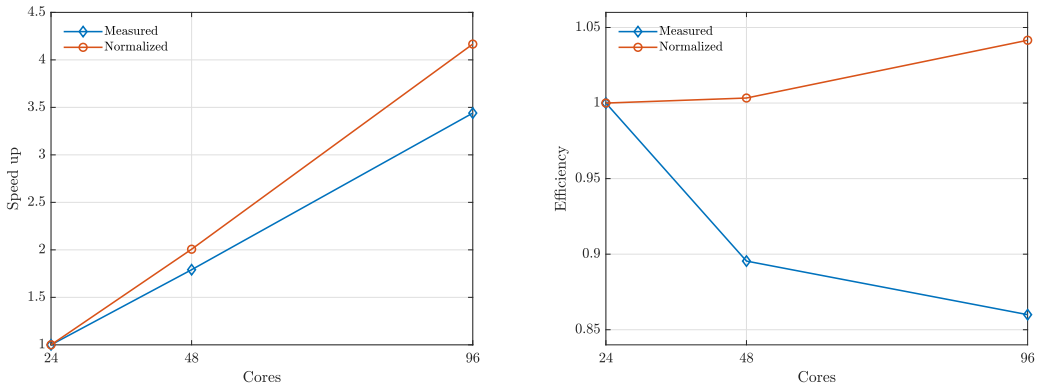

Figure 5: Speed up (left) and efficiency (right) of the location algorithm

implementation. Taking advantage from a preliminary simulation, the algorithm is able to compute a domain subdivision considering the cloud distribution during the entire evolution even if the flow solution is unsteady.
The procedure, reported in Algorithm 2, has an additional step with respect to the steady implementation presented in [12]. The flow solution has to be updated in according to the particle simulation time. Before reloading the fluid solution, the cloud has to be synchronized because during the particle evolution non-blocking communication are used. For instance, when a particle is received, it is not evolved until the current simulation time. A while loop is adopted to integrate particles until the prescribed time. Also, when the cloud is saved, it has to be synchronized. This leads to a reduction of performances as it is highlighted in Section 5. Another drawback of unsteady simulation is related to the preliminary particle evolution. Even if only few particles are integrated, the flow solution must be updated leading to an increased execution time.
The flow solution is loaded from binary VTK files. The code use the parallel I/O interface offered by MPI library [16]. The fluid quantities are known at the nodes of the mesh and they are subdivided among processes according to the previously computed partitioning. An MPI hindexed variable is created to describe the non-contiguous data and the collective MPI-I/O function is used to efficiently read the file [17].

5 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 [18]. The cylinder length is ten times the diameter. The flow solution is computed using SU2 [19] on a hexahedral mesh composed by 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 turbulence model is the SST Mentor [20]. The free-stream conditions are: Mach number $M_\infty = 0.1$, temperature $T_\infty = 288.15$ K and $\mathrm{Re}_\infty = 5000$. Two different particle simulations are performed, the first using a Median Volume Diameter (MVD) = 20 μm and the second MVD = 50 μm.

Algorithm 2: Unsteady particles evolution

```
while t ≤ t_final do
    forall particles do
        integrate;
        if internal boundary intersected then
            send to new owner;
        end
    end
    if t = t_save then
        synchronize cloud;
        save cloud;
    end
    if t = t_unsteady flow then
        synchronize cloud;
        load new flow solution;
    end
end
```

Table 1: 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 first test case aims to evaluate the code performances and evolves 1M droplets. The particle diameter, and consequently the inertia, is too small to impact the cylinder, so the impinging droplet density per unit surface, namely, the collection efficiency, is negligible. In Figure 6, a graphical representation of the cloud is given at different time. The speed up factor and the parallel efficiency, for the specific case, are reported in Figure 7. It is apparent how the performances improve 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 when using 96 cores as reported in Table 1. Not surprisingly, the performances are a bit lower than in the steady case reported in Reference [12].
The second simulation shows the effect of the unsteady flow on the collection efficiency computation. It is calculated at the longitudinal symmetry plane and at ± 2 diameters. In Figure 8, the particle convergence is reported. Starting from 4M droplets the collection efficiency does not vary significantly. In a steady flow around a cylinder a symmetric result is expected as reported in [21]. In Figure 9, it is evident how the behaviour differs

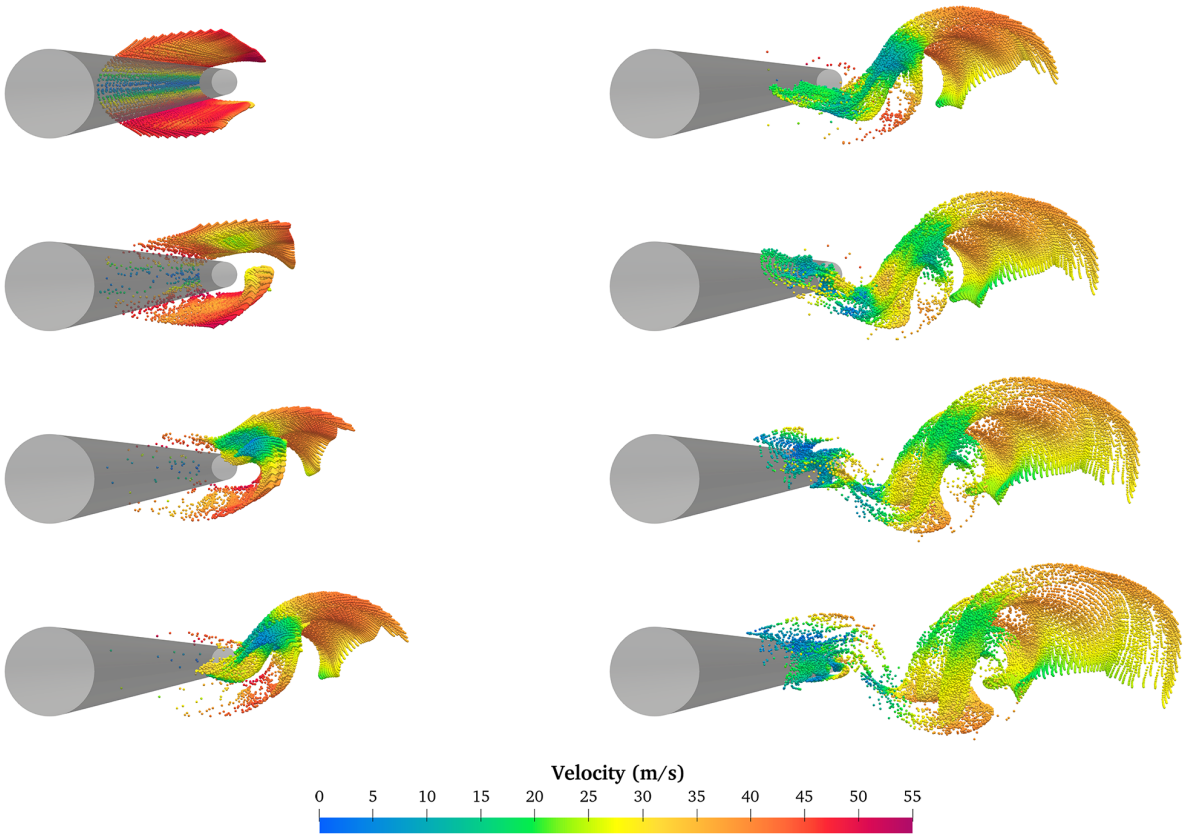

Figure 6: Cloud evolution (MVD = 20 μm), each figure is taken after a $\Delta t = 0.0025$ s

due to the wake influence. The collection efficiency for each cell is computed according to the following equation $\beta_i = \rho_i/\rho_\infty$ where ρ_i is the particle density at the i-th element and ρ_∞ the initial one.

6 Conclusions

In this work, an efficient parallel ray tracing algorithm to locate particles in unstructured meshes is presented. It can handle arbitrary shaped subdomains and shows an almost ideal scaling in the test case. In addition, a preliminary cell tagging technique applied to an unsteady flow simulation allows a significant saving of resources if compared to the naive implementation. Finally, a cloud droplet impact test case starting from an unsteady flow around a 3D cylinder has been simulated to evaluate the effects of the unsteadiness on the collection efficiency computation.

REFERENCES

[1] M. Sommerfeld, Y. Cui, and S. Schmalfuß. Potential and constraints for the application of CFD combined with Lagrangian particle tracking to dry powder inhalers. *European Journal of Pharmaceutical Sciences*, 128:299–324, 2019.

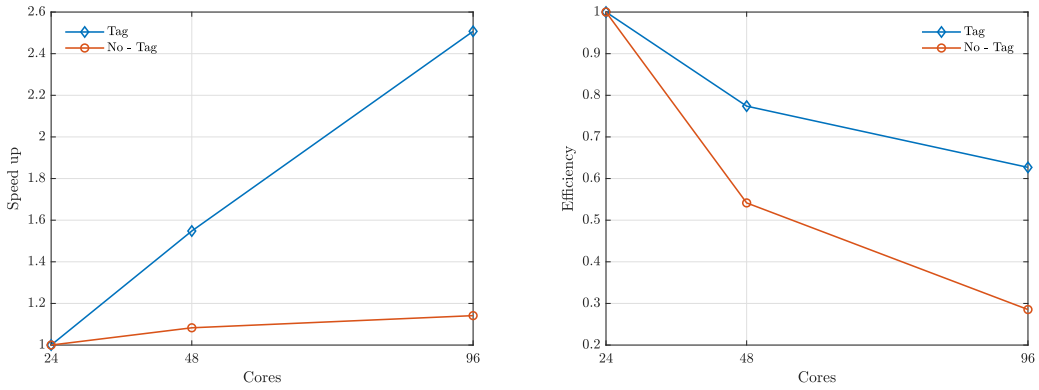

Figure 7: Speed up (left) and efficiency (right) of the particle algorithm

[2] Elizabeth A. Andruszkiewicz, Jeffrey R. Koseff, O. Fringer, N. Ouellette, A. B. Lowe, C. Edwards, and A. Boehm. Modeling environmental DNA transport in the coastal ocean using Lagrangian particle tracking. *Frontiers in Marine Science*, 6, 2019.

[3] J. Dunn and S. G. Lambrakos. Calculating complex interactions in molecular dynamics simulations employing Lagrangian particle tracking schemes. *Journal of Computational Physics*, 111:15–23, 1994.

[4] A. Hamed, K. Das, and D. Basu. Numerical simulations of ice droplet trajectories and collection efficiency on aero - engine rotating machinery. 2005.

[5] G. Gori, M. Zocca, M. Garabelli, A. Guardone, and G. Quaranta. PoliMIce: A simulation framework for three-dimensional ice accretion. *Appl. Math. Comput.*, 267:96–107, 2015.

[6] M. Widhalm, A. Ronzheimer, and J. Meyer. Lagrangian particle tracking on large unstructured three-dimensional meshes. 2008.

[7] Giacomo Capodaglio and Eugenio Aulisa. A particle tracking algorithm for parallel finite element applications. *Computers & Fluids*, 159:338–355, 2017.

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

[9] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel algorithms for multi-constraint graph partitioning. *Euro-Par*, 2000.

[10] L. Zhang, G. Zhang, Y. Liu, and H. Pan. Mesh partitioning algorithm based on parallel finite element analysis and its actualization. *Mathematical Problems in Engineering*, 2013.

[11] K. George and K. Vipin. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 1998.

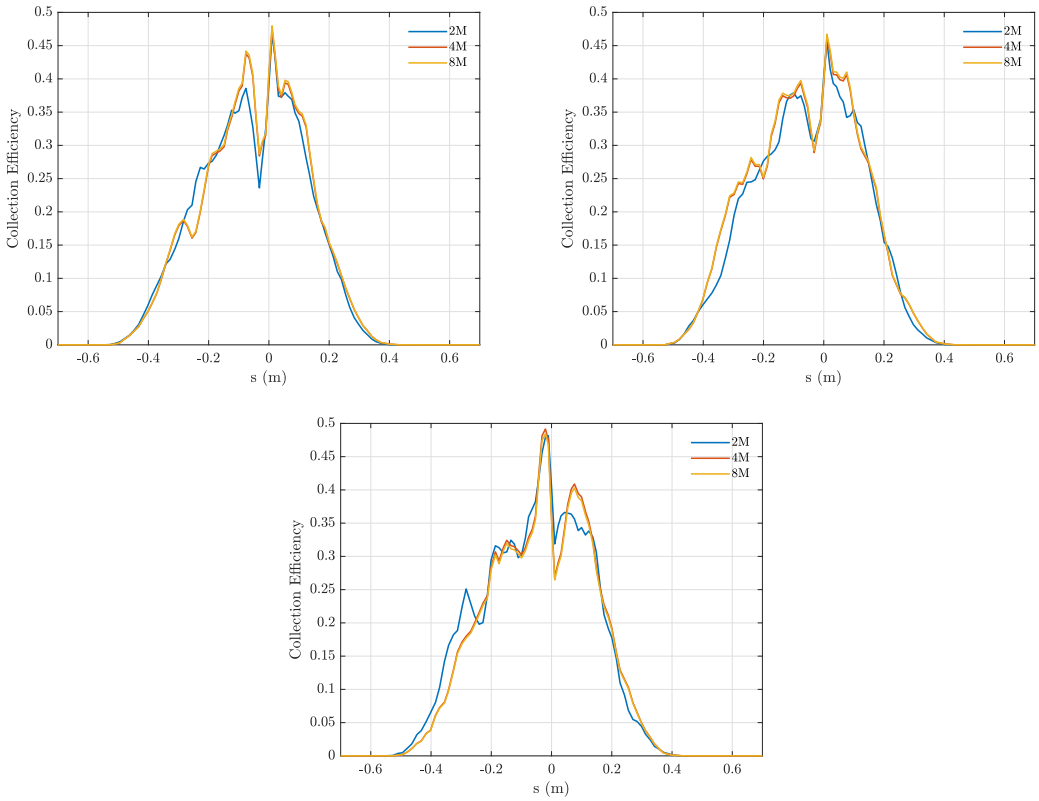

Figure 8: Convergence of particles (MVD = 50 μm)

[12] G. Baldan, T. Bellosta, and A. Guardone. 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.

[13] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45:891–923, 1998.

[14] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[15] Severin Strobl, Marcus Bannerman, and Thorsten Pöschel. Robust event-driven particle tracking in complex geometries. *Computer Physics Communications*, 254:107229, 02 2020.

[16] M. Snir, S. Otto, D. Walker, J. Dongarra, and S. Huss-Lederman. MPI: The complete reference. 1996.

[17] Kohei Sugihara and O. Tatebe. Design of locality-aware MPI-IO for scalable shared file write performance. *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1080–1089, 2020.

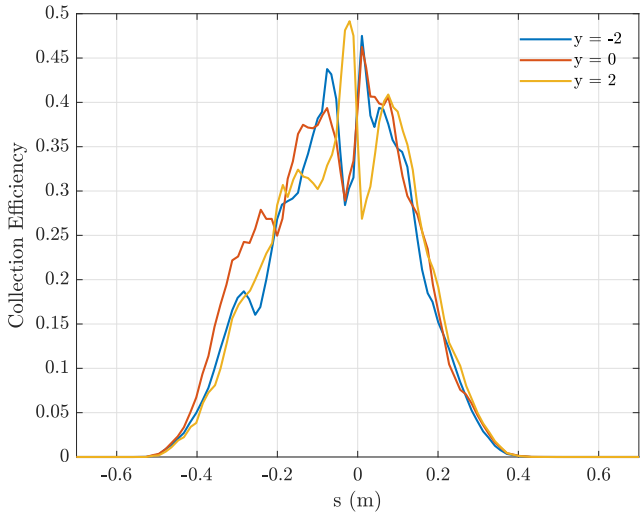

Figure 9: Collection efficiency comparison at different positions (MVD = 50 μm)

[18] D. E. Aljure, O. Lehmkhul, I. Rodríguez, and A. Oliva. Three dimensionality in the wake of the flow around a circular cylinder at Reynolds number 5000. *Computers & Fluids*, 147:102–118, 2017.

[19] F. Palacios, Thomas D. Economon, Aniket C. Aranake, S. R. Copeland, Amrita K. Lonkar, T. Lukaczyk, David E. Manosalvas, K. Naik, A. Padr, Brendan D. Tracey, Anil Variyar, and J. Alonso. Stanford University Unstructured (SU2): Open-source analysis and design technology for turbulent flows. 2014.

[20] F. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA Journal*, 32:1598–1605, 1994.

[21] T. Guo, C. Zhu, and C. Zhu. An efficient and robust ice accretion code: NUAA-ICE2D. *IEEE International Conference on Aircraft Utility Systems*, 2016.