# Efficient parallel algorithms for coupled fluid-particle simulation

Giacomo Baldan, Tommaso Bellosta, Alberto Guardone


**INFORMATION**

**Keywords:**
fluid-particle systems
particle tracking
mesh partitioning

**DOI:** 10.23967/coupled.2021.021

**Published:** 12/07/2021

IX International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2021
E. Oñate, M. Papadrakakis and B. Schrefler (Eds)

# EFFICIENT PARALLEL ALGORITHMS FOR COUPLED FLUID-PARTICLE SIMULATION

**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:** Fluid-particle systems, Particle tracking, Mesh partitioning

**Abstract.** Coupled fluid-particle simulations are routinely used in a variety of applications, ranging from respiratory droplet spreading to internal combustion engines, from ink-jet printing to in-flight ice accretion. The efficiency of parallel algorithms to simulate fluid-particle systems is strongly influenced by the different evolution of the flow and the particles dynamics. Indeed, a domain partitioning based on particle workload is possibly sub-optimal in terms of the number of fluid volume elements associated to each process. In this work, an efficient mesh partitioning based on graph representation is implemented. It can handle unstructured hybrid meshes composed by triangles and quadrilaterals in two spatial dimensions, and by tetrahedra, hexahedra, prisms, and pyramids in three dimensions. In order to obtain a domain decomposition to efficiently follow the particle trajectories, a preliminary solution is computed to suitably tag the fluid domain cells. The obtained weights represent the element probabilities to be crossed by particles. The algorithm is implemented using MPI distribute memory environment. The proposed approach is tested against reference cases for the coupled flow-particle simulation of ice accretion over 2D and 3D geometries. Two different cloud droplet impact test cases have been simulated: a NACA 0012 wing section and a NACA 64A008 swept horizontal tail. The computed collection efficiency compares fairly well with reference numerical and experimental data. The parallel efficiency of the algorithm is verified on a distributed memory cluster.

## 1 Introduction

Fluid-particle systems are present in many applications ranging from natural phenomena to industrial processes. Recent studies adopted Lagrangian particle tracking to simulate respiratory droplet in turbulent flows [1] or internal combustion engine [2, 3], ink-jet printing [4] or in-flight ice accretion problems [5, 6].

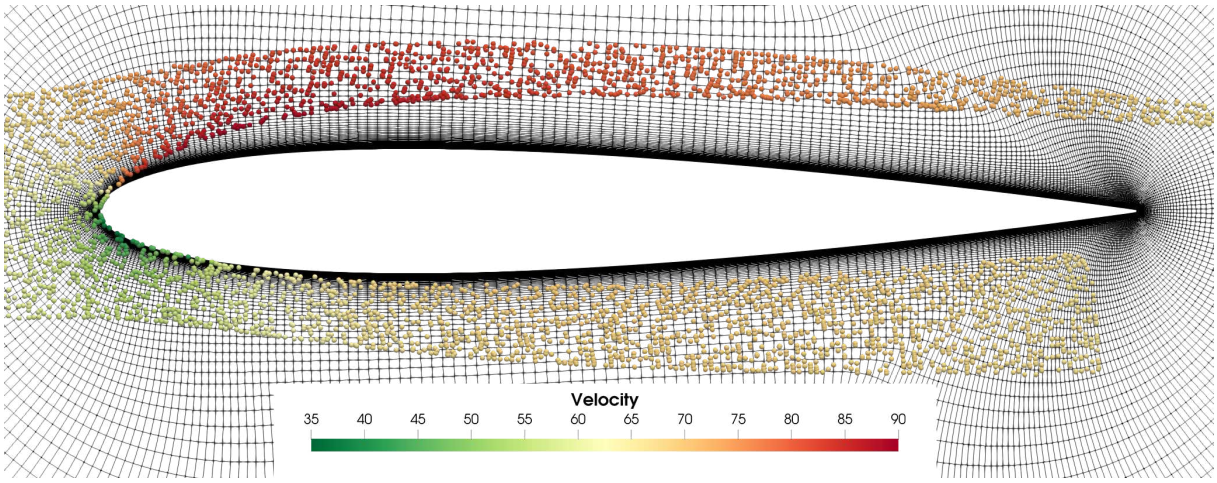


Figure 1: Particles around a NACA 0012 profile

In a Lagrangian particle tracking algorithm, the solution accuracy is proportional to the number of sample particles. Furthermore, the number of cells within the domains determines the accuracy of the flow description. Parallel algorithms can be adopted to speed up simulations. In some cases, they become a requisite because of resource limits. However, there are some critical aspects to be considered in an efficient parallelization. An effective technique to subdivide the domain based only on the flow solution [7] usually leads to an unbalanced particle workload. In addition, during particle integration, a direction is usually predominant. The problem physics is, in some sense, reflected also in the movement of particles among processes. A standard all-to-all communication performs slower than the ad-hoc implemented strategy. It consists in non-blocking send and receive calls only when at least one particle has to be communicated. Indeed, a lot of empty requests are saved especially when the number of cores increases. An example is given in Figure 1, where water droplets are advected by the flow around a NACA 0012 airfoil. This simulations is typical of ice accretion problems, where cloud droplets impinge on the aircraft.
In this work, the initial domain partitioning based on a nodal graph is presented in Section 2. Unstructured hybrid meshes are considered. In two dimensions, triangular and quadrilateral elements are supported, while in three-dimension cells can be tetrahedra, hexahedra, prisms, and pyramids. After computing the first partitioning, a preliminary simulation with few particles is performed to suitably tag the cells. In this manner, a constraint, linked with the particle distribution, is added to the graph partitioning. The cell tagging technique is described in Section 3. Then, the changes in the particle tracking algorithm are reported in Section 4. Finally, in Section 5, the numerical tool is tested against reference cases for the coupled flow-particle simulation of ice accretion over a NACA 0012 wing section and a three-dimensional NACA 64A008 swept horizontal tail. Also, the speed up and the parallel efficiency improvements of the code are presented.

## 2 Initial domain partitioning

Parallel distribute memory algorithms require mesh partitioning in subdomains. To distribute memory usage, one usually requires to have the same number of nodes and elements in each subdomain while, at the same time, reduce the information exchange among processes.
In literature, diverse graph partitioning methods are presented such as geometric, spectral, combination, multilevel, dynamic repartitioning, parallel graph, multi-constraint and multi-objective [8]. Multilevel algorithms [9] are the most recent techniques and they are implemented in different libraries such as Chaco, Jostle, Party, Scotch, Metis and ParMetis [10, 11]. They offer better performances if compared to other methods, having a sequential complexity of the serial multi-constraint algorithm of $O(mn)$ where $m$ and $n$ are the number of constraints and nodes, respectively. The isoefficiency functions are $O(p^2 \log p)$, with $p$ processes. In this work, we take advantage of the multi-constraint parallel partitioning offered by ParMetis to implement an efficient mesh subdivision that considers flow solution and particle evolution [12]. Indeed, ParMetis provides a straightforward API integration in C/C++ codes.
The multilevel partition can be summarised in three different steps: coarsening, initial partitioning and uncoarsening. During the coarsening phase, the initial graph is reduced to a coarser one by eliminating selected vertices, until a sufficiently small graph is obtained. Then, the initial partitioning is computed over the coarsest graph. Eventually, through a refining and optimizing procedure the final partition is obtained.
We present a domain partitioning based on a nodal graph. The partitioning procedure is briefly recalled as follows. The mesh connectivity and the node coordinates are loaded according to a linear partition. To build the nodal graph, the element connectivity is redistributed among the processes. A cell is owned by a process if at least one node of the cell is in its initial linear partition. Therefore, an element can be possessed by more than one process and duplicates must be handled by the code. Then, the graph edges are built. Only local information are required during this phase avoiding communication among processes. After, k-way Parmetis routine is called to subdivide the graph and the new node owners are obtained. Nodes coordinates, element and boundary connectivity are redistributed according to the just obtained partitioning. At this point, the full mesh connectivity is computed. The interfaces generated during the subdivision are identified. New non-physical boundaries are created where the domain is cut. Therefore, a shared layer of elements, called halo cells, is created. They avoid numerical problems in the particle integration. Once the face connectivity is available, it is quite straightforward to discriminate boundary faces because they are linked with only one element. The following step consists in mapping the boundary faces with the physical ones comparing connectivity. The non-mapped faces compose the internal interfaces, assuming that the loaded mesh is complete. The implemented procedure is summarized in Algorithm 1.
Two more data structures are created to link the local element id with the global one, and

**Algorithm 1:** Domain subdivision

```
load nodes coordinates and element connectivity into a linear partition;
distribute element connectivity according to nodes distribution;
build graph edges;
subdivide the graph;
forall nodes do
    send nodes coordinates to new owner;
    send element connectivity to new owner;
    send boundaries connectivity to new owner;
end
create full mesh connectivity;
forall faces do
    if linked with only one element then
        if physical then
            map boundary;
        else
            internal interface;
        end
    end
end
```

vice versa. If a particle is in a ghost cell and its trajectory crosses an internal interface, it must be sent to another process. Two steps have to be performed. The first is to identify the receiver that is found among the node interface owners. The other one is to localize the particle in the new subdomain. Firstly, the local element id is transformed to the global identifier. Then, the receiver can easily convert it according to its local numbering. If the particle is inside an element in the sending process, then it will be also in the receiving one. An alternative is to not use halo cells and evolve each particle until it reaches the interface face. Small position errors can be significant in the inclusion test leading to wrong results since each element is possessed uniquely by one process.

## 3 Cell tagging

The domain partitioning, based only on the equal node subdivision, leads to an inefficient parallel particle tracking as reported in Section 5. To improve the overall performances further constraint in the graph partitioning is therefore added. This grants also a comparable particle workload on each process. Parmetis interface requires a node weight for each constraint. One, in order to balance elements, is set equal to the number of edges departing from a specific node. The other is proportional to the probability of adjacent elements to be crossed by particles.
The cell tagging technique is divided in two phases: a preliminary simulation and the

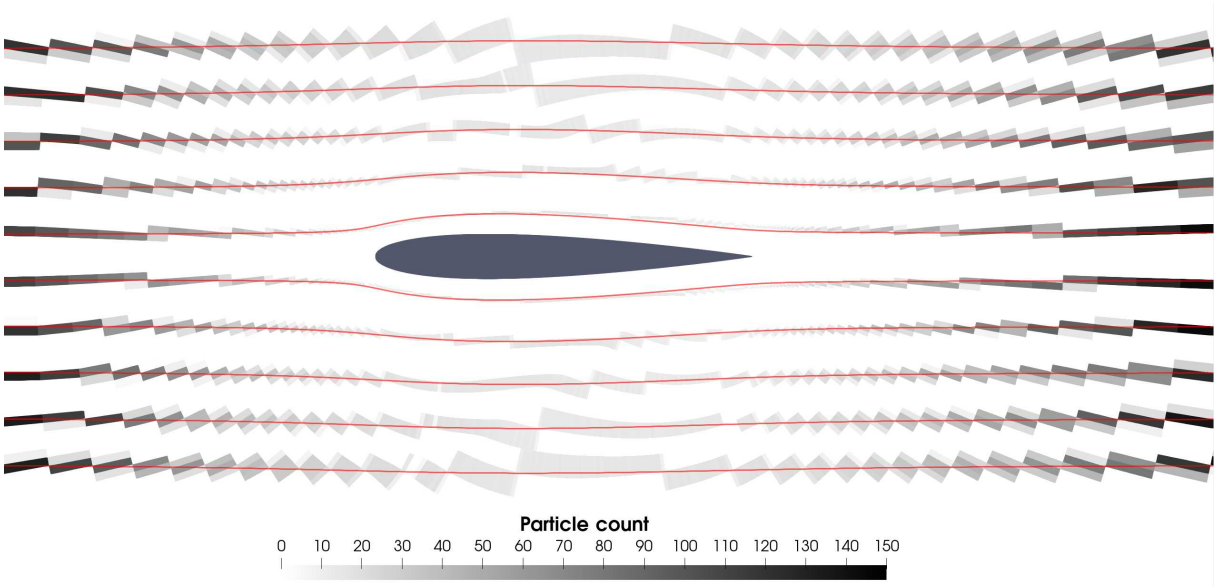


Figure 2: Trajectories and number of particles that crossed each cell

domain repartitioning. In the first step, during the tracking, when a particle crosses an element, a count is updated. In this manner, a cell weight is obtained. In Figure 2, it is possible to see the outcome. Since a node weight is requested, thanks to the mesh connectivity previously calculated, the cell count is converted into a node one. Parmetis routine is called another time to compute the final partitioning accounting for the nodal weights. The procedure is the same presented before in Algorithm 1, except for the fact that the mesh has not to be loaded. The process is significantly faster because it starts from an already computed subdivision and not from a random one as in the previous partitioning. A two-dimensional example of mesh partition with and without the tagging strategy is presented in Figure 3. It is noticeable how the subdomain shapes change in according to the particle trajectories. Without the cell tagging technique lots of the available resources are concentrated around the profile where element size is lower. In the second domain subdivision, instead, a portion of elements near the geometry is owned by each process and this is achieved also for the farther ones. A drawback of the implementation is the possibility of having non simply connected or multiple part subdomains. However, the code can handle all these possible shapes.

## 4 Particle tracking

The particle tracking algorithm presents some differences from the serial one presented in References [13, 14, 15]. Two new features have been added in the parallel implementation: particle communication and the possibility to handle a cloud where each particle is evolved until an arbitrary time. The last aspect allows to avoid blocking MPI communication in favour of non-blocking one reducing the overall idle time. The main loop

Figure 3: Mesh subdivision in 8 parts without (under) and with (below) cell tagging technique, each colour corresponds to a subdomain

is composed of three distinct steps. Firstly, all the owned particles are evolved until the current simulation time. Secondly, each process receives from the previous time step the new particles. Finally, the evolved particles are sent if their trajectories intersected an internal interface.
At each time step, the particles to move are sorted by the receiver id. Then, all the information are serialized and communicated to the new owners. The flow solution at the particle position is not transferred because it can be easily computed once received.
The asynchronous evolution is effective but presents also some disadvantages. The first one is, as mentioned before, the arbitrary time of each particle. Another one is related to the necessity of having the cloud synchronized, for instance when it is saved. A while loop is adopted to evolve the cloud maintaining constant the simulation time. The control condition consists in checking if there are no more particles to be send and each individual time is equal to the saving one.

## 5 Results

The code assessment is performed computing the collection efficiency of two test cases. The first one is a classical NACA 0012 wing section analysed at different angles of attack and flow conditions, the second one is a three-dimensional NACA 64A008 swept horizontal tail at negative incidence. The computed collection efficiency, namely, the water impinging per unit surface, is compared with data obtained in previous experimental campaigns or by other codes. In this work, all the flow solutions are calculated using SU2 suite [16], the adopted turbulent models to close the RANS equations are SST Mentor [17] in the two-dimensional case and SA [18] in the three-dimensional one. A 16-node cluster has been used in all the simulations. 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.

### 5.1 NACA 0012 wing section

The NACA 0012 symmetric wing section is used here because it was tested many times in the icing tunnel [19, 20, 21]. Furthermore, collection efficiency simulated with NUAA-ICE2D [22] and LEWICE [23] codes is available. Firstly, we compare the implemented code with the one presented by Guo et al. [22]. Then, we consider another flow condition, maintaining the geometry unchanged, to verify if the computed collection efficiency agrees with results presented in literature [20, 21]. In both cases, we firstly asses convergence of the computed collection efficiency by changing the number of particles.
The flight and icing conditions imposed in the first analysis are equal to the ones presented in Guo et al. [22]. The free-stream velocity is set equal to $V_\infty = 67$ m/s, while ambient pressure $P_\infty = 98900$ Pa, static temperature $T_\infty = -7°$C, angle of attack $\alpha = 0°$ and $\alpha = 3°$, median volume diameter MVD $= 25$ $\mu$m, and the profile chord measures 0.914 m. In Figure 4 and 5 are reported the results at $0°$ and $3°$, respectively. The first angle of attack has been chosen since a symmetric solution is expected, the implemented code catches the behaviour quite well and it slightly differs from the data reported in [22]. At positive incidence, the data computed by Guo et al. is really close to the one presented in this work.
The second flight and ice conditions are set in according to the ones presented by Silviera group in [20]. The free-stream velocity is set equal to $V_\infty = 44.39$ m/s, while ambient pressure $P_\infty = 1$ atm, static temperature $T_\infty = 265$ K, angle of attack $\alpha = 0°$, median volume diameter MVD $= 20$ $\mu$m, and the profile chord measures 0.914 m. In Figure 6, particle convergence and the computed collection efficiency is presented. It is in good agreement with experimental and LEWICE data near the profile nose. Moving away from the stagnation point, the code underestimates the collection efficiency with respect to the experimental one, while LEWICE overestimates it.

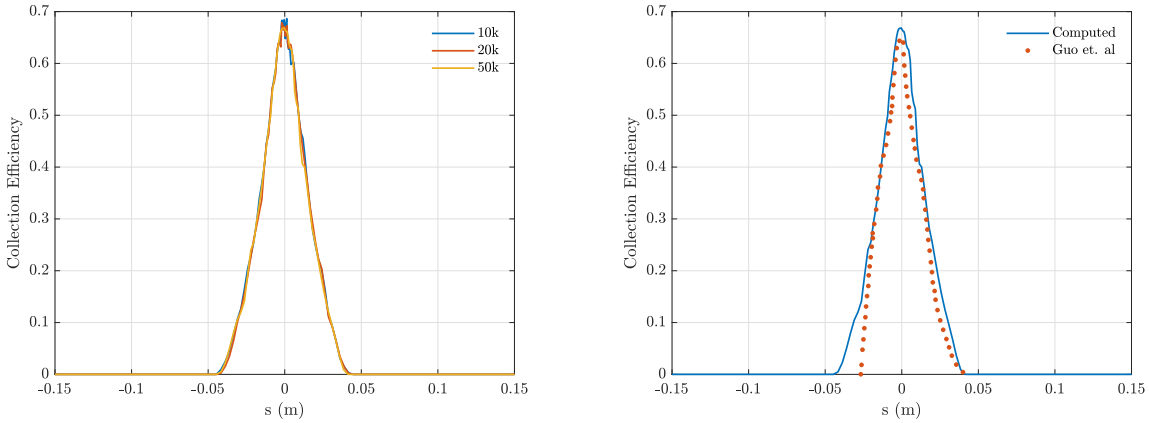


Figure 4: Particle convergence (left) and collection efficiency comparison (right) at $\alpha = 0°$

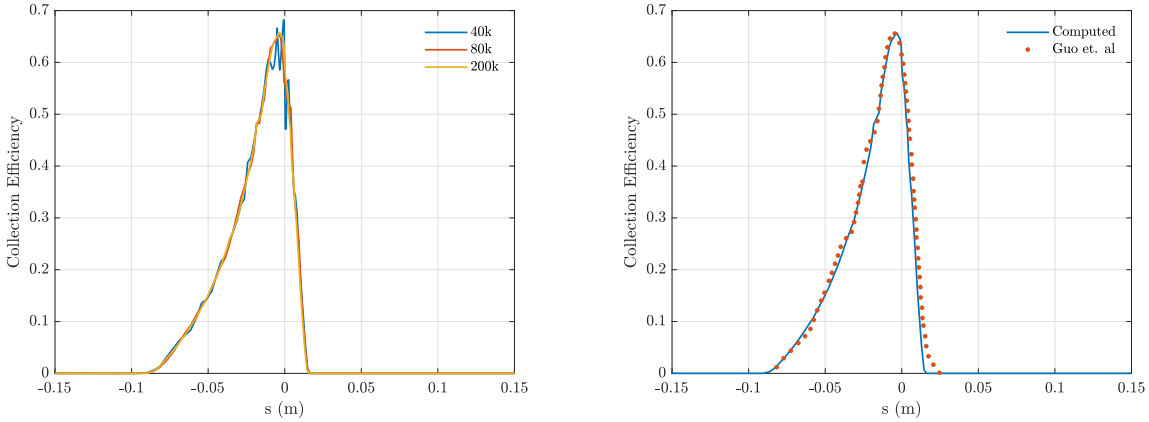


Figure 5: Particle convergence (left) and collection efficiency comparison (right) at $\alpha = 3°$

### 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 [24] are available. In addition, it is a test case of the 1st AIAA Ice Prediction Workshop. The air and ice conditions are equal to [24] and correspond to $V_\infty = 78.68$ m/s, while static pressure $P_\infty = 95147$ Pa, static temperature $T_\infty = 280$ K, Mach $M_\infty = 0.23$, Reynolds $Re_\infty = 5 \cdot 10^6$, and median volume diameter MVD $= 21$ $\mu$m. Firstly, we compare the pressure distributions with the experimental ones. Wind tunnel effects are compensated in the numerical flow simulation changing the angle of attack $\alpha$ from $-6°$ to $-6.25°$. In Figure 7, it is clearly visible how the two solutions are almost overlapping. The comparison of collection efficiency, Figure 8, presents little discrepancies on the upper part of the profile. On the lower, LEWICE data and the computed result are overlapping, while the

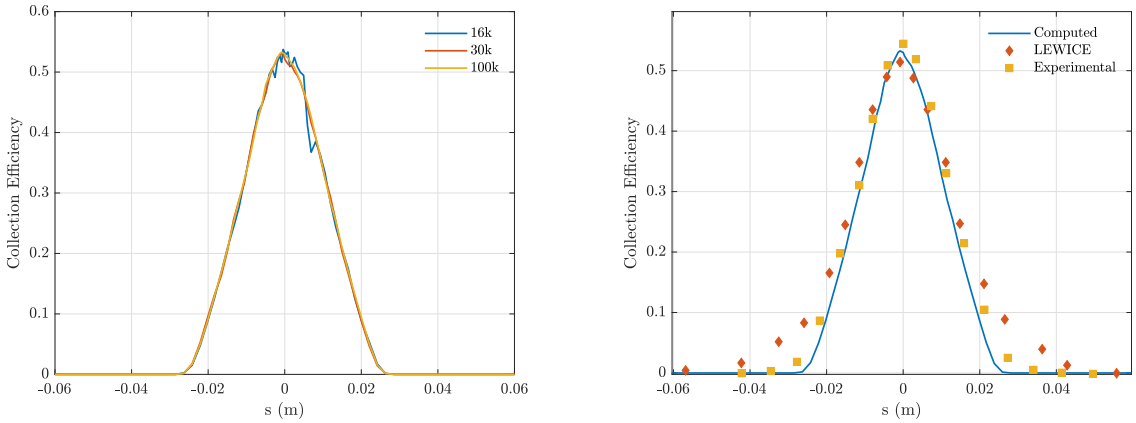


Figure 6: Particle convergence (left) and collection efficiency comparison (right)

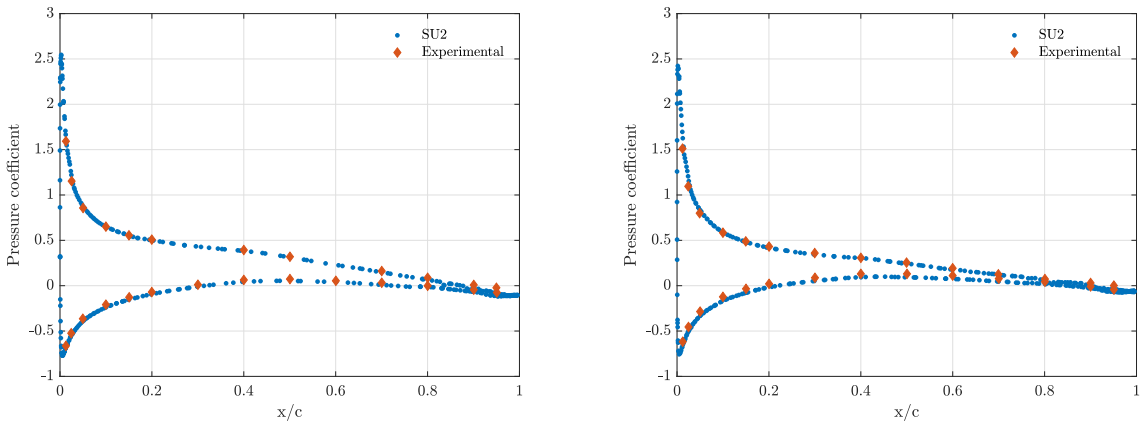


Figure 7: Pressure coefficient comparison at inboard (left) and outboard (right)

experimental campaign presents higher value.
The parallel performances are tested evolving one million particles in the three-dimensional geometry. The simulations are run using 2, 4, and 8 nodes at which corresponds 24, 48, and 96 cores respectively. The particle integration is computed twice, with and without the cell tagging technique. In Figure 9 and Table 1 the results are summarized. The speed up factor using a standard domain subdivision is almost flat and the efficiency is very low. The behaviour is completely changed 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.

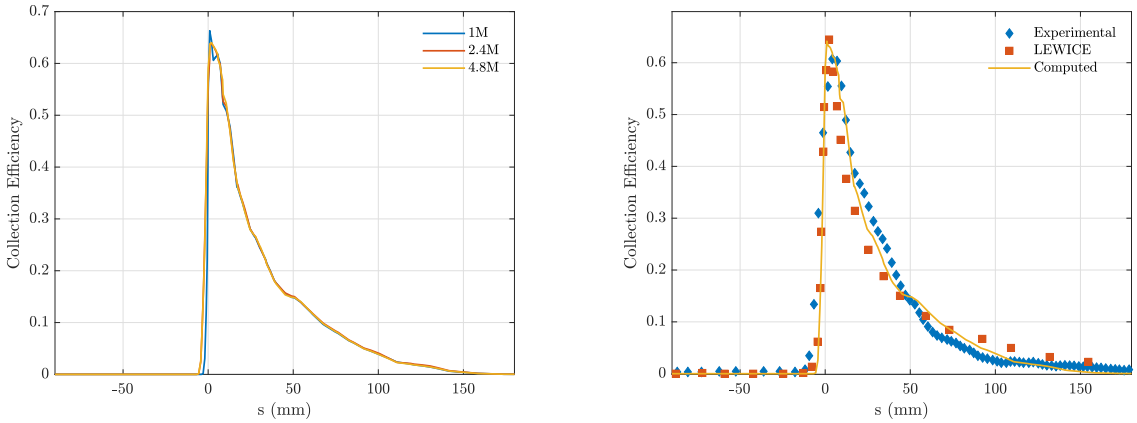


Figure 8: Particle convergence (left) and collection efficiency comparison (right)

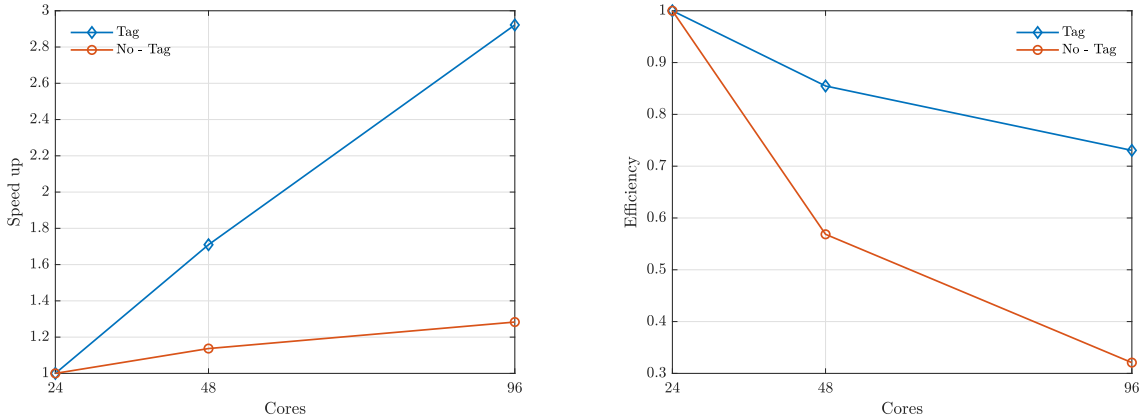


Figure 9: Speed up (left) and efficiency (right) of the code evolving 1M particles

## 6 Conclusions

In this work, an efficient parallel Lagrangian particle tracking is presented to solve coupled fluid-particle systems. The naive domain subdivision leads to unacceptable performances. The innovative tagging technique is capable of providing the same results using much less resources. It has been evaluated in a typical aeronautical test case such as the presented three-dimensional swept horizontal tail. In the specific, comparing the tagging technique with the naive implementation, a time reduction up to 85% and a significant scalability improvement moving from 1.1 to 1.7 doubling the number of cores are achieved.

Table 1: 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% |

## REFERENCES

[1] J. Wedel, P. Steinmann, Mitja Štrakl, M. Hriberšek, and Jure Ravnik. 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. *Computational Mechanics*, pages 1 – 17, 2021.

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

[3] O. Kaario, V. Vuorinen, H. Kahila, H. Im, and M. Larmi. The effect of fuel on high velocity evaporating fuel sprays: Large-eddy simulation of spray a with various fuels. *International Journal of Engine Research*, 21:26 – 42, 2020.

[4] Masato Ikegawa, E. Ishii, N. Harada, and T. Takagishi. Development of ink-particle flight simulation for continuous inkjet printer. 2013.

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

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

[7] Shahab U. Ansari, M. Hussain, S. Mazhar, T. Manzoor, Khalid J. Siddiqui, Muhammad Abid, and H. Jamal. Mesh partitioning and efficient equation solving techniques by distributed finite element methods: A survey. *Archives of Computational Methods in Engineering*, 26:1–16, 2019.

[8] 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.

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

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

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

[12] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14, 2002.

[13] 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.

[14] T. Bellosta, G. Parma, and A. Guardone. A robust 3D particle tracking solver for in-flight ice accretion using arbitrary precision arithmetic. *Coupled 2019*, 2019.

[15] G. Capodaglio and E. Aulisa. A particle tracking algorithm for parallel finite element applications. *Computers & Fluids*, 2017.

[16] 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.

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

[18] P. Spalart. A one-equation turbulence model for aerodynamic flows. 1992.

[19] J. Shin and T. Bond. Results of an icing test on a NACA 0012 airfoil in the NASA Lewis icing research tunnel. 1992.

[20] R. Silveira, C. Maliska, Diego A. Estivam, and R. Mendes. Evalutation of collection efficiency mehods for icing analisys. 2003.

[21] F. Morency, F. Tezok, and I. Paraschivoiu. Anti-icing system simulation using CANICE. *Journal of Aircraft*, 1999.

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

[23] W. Wright. User's manual for LEWICE version 3.2. 2008.

[24] M. Papadakis, K. Hung, Giao T. Vu, H. Yeong, C. Bidwell, M. D. Breer, and T. Bencic. Experimental investigation of water droplet impingement on airfoils, finite wings, and an s-duct engine inlet. 2002.