A Massively Parallel Implementation of the Actuator Line Method for High-Fidelity Large Eddy Simulation

Etienne Muller, Simone Gremmo, Félix Houtin-Mongrolle, Pierre Benard

ParCFD2022
33^{rd} International Conference on Parallel Computational Fluid Dynamics
May 25-27 2022, Alba Italy

A MASSIVELY-PARALLEL IMPLEMENTATION OF THE ACTUATOR LINE METHOD FOR HIGH-FIDELITY LARGE EDDY SIMULATION

Etienne MULLER*, Simone GREMMO*, Félix HOUTIN-MONGROLLE* AND Pierre BENARD*

* CORIA, CNRS UMR6614, Normandie Universite, UNIROUEN, INSA of Rouen, France
Department of Turbulence, Atomization, Sprays and Chaos
Avenue de l'Université, 76801 Saint-Étienne-du-Rouvray, France
e-mail: etienne.muller@coria.fr, web page: https://www.coria.fr

Key words: CFD methodology, Parallel methodology, Wind energy, ALM, LES

Abstract. This paper presents a parallel implementation of the Actuator Line Method (ALM), designed for massively parallel large eddy simulations (LES) of wind farms. The performance obtained with this implementation shows to be close to ideal, based on both Wall Clock Times (WCTs) and Reduced Computation Times (RCTs) measurements.

1 INTRODUCTION

In the recent years, the wind energy sector has known a rapid growth, sustained by the democratization of numerous numerical tools. Mostly, those rely on low order engineering models, as their computational cost is low. However, the development, tuning and validation of such models often require to use high fidelity tools to first get a deep knowledge of the multiphysics phenomena involved. For instance, the Large Eddy Simulation (LES) can be profitable to better understand the flow crossing a wind farm. This approach may prove to be costly, so efficient massively parallel CFD codes are usually necessary to make it computationally achievable. State-of-the-art turbine modeling approach consists in the Actuator Line Method (ALM) [1]. Still, for multi-turbine configurations, a proper parallel implementation of the method is required to avoid workload imbalances. Such an implementation is reported in [2]. In the Section 2 of this paper, we present an alternative implementation of the ALM in YALES2 [3], also designed to be computationally efficient. The performance obtained with our approach is then evaluated in Section 3.

2 IMPLEMENTATION OF THE ACTUATOR LINE METHOD

2.1 The aerodynamic solver

YALES2 [3] is a low-Mach, massively-parallel finite volume solver, able to solve the incompressible Navier-Stokes equations on unstructured grids with up to several billions cells. The solver high performance is mainly obtained thanks to an in-house two-level grid partitioning. The first level consists in splitting the computational domain into

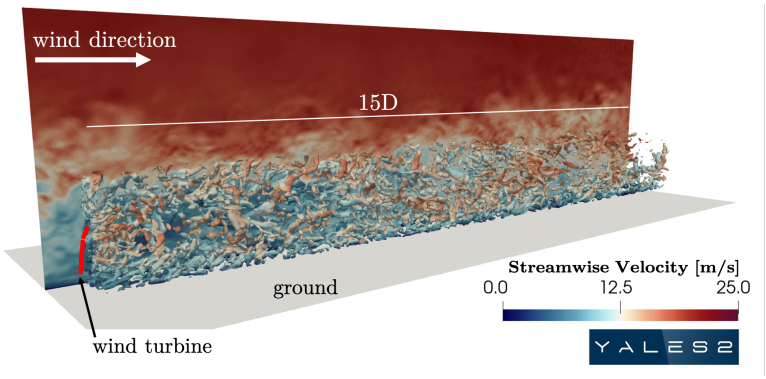

Figure 1: LES simulation of a wind turbine operating in the wake of another, featuring the ALM. An iso-contour of the Q-criterion, colored by the streamwise velocity, is used to depict the vortical structures in the wake up to a distance of 15 diameters.

partitions, one per MPI rank. Each partition is then further decomposed into a collection of cell groups, whose size is limited to avoid any cache-memory miss.

2.2 The actuator line method

The ALM allows to model the forces applied by a wind turbine blades on an incident flow and thus to simulate the development of the wake (see Figure 1). Each blade geometry is replaced by a virtual line discretized into 1D elements. All elements have the same width w and their center point, also called *particle*, is associated with airfoil-specific tabulated polars. The aerodynamic force F, assumed constant on each 1D element, is computed at the particle locations as $F = 0.5\,\rho v_{rel}^2 c(r) w C_F(\alpha, r)$, where c is the airfoil chord, v_{rel} is the magnitude of the fluid velocity seen by the airfoil and ρ is the density. The force coefficient (lift or drag), denoted C_F here, is tabulated for each airfoil profile as function of the angle of attack α.

The resulting body force distribution is singular and must then be smoothed before being added as a source term in the Navier-Stokes equations. In the ALM framework, this projection operation is known as mollification process and it acts as a spatial filtering operation, typically based on a Gaussian-like kernel η [4]:

$$\eta(d) = \frac{1}{\epsilon^3 \pi^{3/2}} \exp\left[-\left(\frac{d}{\epsilon}\right)^2 \right], \tag{1}$$

where d is the distance to the kernel center, and ϵ is the kernel radius. The latter is usually chosen as twice the maximum cell-size encountered in the rotor region.

2.3 Implementation details

To help the ALM implementation in YALES2, dedicated structures are created. Each rotor blade is registered as an *actuator* object. A whole rotor is represented by an *actuator-set* object, which is a collection of actuator objects.

At solver initialization, all the actuator-set objects and their respective bounding boxes are registered. For each actuator-set, a specific MPI sub-communicator is then created, which contains all the cores having at least one cell group whose bounding box intersects the one of the rotor. The MPI rank registering the largest number of intersections is finally selected as the master rank of the communicator. In multiple-rotor configurations, this choice will tend to prevent two rotors from sharing the same master rank.

During the temporal loop, the ALM process follows several steps: interpolation of the fluid velocity at the particle position, angle of attack evaluation, computation of the aerodynamic forces, and forces mollification. To help balance the computation, the steps for a particular actuator-set are handled only by the MPI ranks belonging to the corresponding MPI sub-communicator. In addition, to minimize the number of idle cores, a loop over the actuator-set objects is implemented for each step of the process. We also take advantage of the two-level grid partitioning for as many steps as possible.

For instance, to interpolate the fluid velocity at the actuator particle position, a multi-step search of the cell containing the particle is performed in each relevant MPI partition. First, we loop over the cell groups, and we check if their bounding box is intersecting the one of the actuator-set. Only if it does, we loop over the cell-group cells. This strategy, which dramatically reduces the number of tests, continues until the cell containing the particle is found. Similarly, additional bounding boxes, computed around each actuator line, are exploited during the mollification process to accelerate it. Indeed, a grid node will be in range of a Gaussian kernel only if: 1) the cell group containing the node is in the rotor bounding box; 2) the node lies within an actuator bounding box.

As an additional optimization, the ALM can also count on a sub-stepping feature. This mechanism is triggered when the ALM time-step, limited by the particle CFL condition, is lower than the fluid time-step based on the flow CFL. The actuator lines are then progressively advanced on several sub-iterations, allowing a larger fluid time-step and so a decreased return time.

3 ASSESSMENT OF THE RESULTING PERFORMANCE

We here consider several configurations with a growing number of rotors: 1, 5, 10, 20, and 30. The domain size and the 5-million tetrahedron mesh remain the same for all configurations, and the *wall clock time* (WCT) and *reduced computation time* (RCT) are measured to assess the computational performance. A RCT is defined as:

$$\mathrm{RCT} = \frac{WCT \cdot N_c}{N_{\Delta t} \cdot N_n}\,, \tag{2}$$

where N_c is the number of cores, $N_{\Delta t}$ is the number of completed time steps, and N_n is the number of vertices in the mesh. The results are compared to those obtained with a *serial* implementation of the ALM, where *all* the MPI ranks have to handle serially *all* the actuator-set objects at each time-step.

The results are depicted in Figure 2. The grey lines refer to what an *ideal* parallel implementation of the ALM would lead to. In such an implementation, the ALM-related

cost would be the same regardless of the number of rotors. Our parallel implementation of the ALM, when compared to the serial one, results in a significantly better performance, close to the ideal one. The saving obtained on the total WCTs and global RCTs appear however lower. This indicates that a serial implementation of the ALM can quickly act as a cost bottleneck.

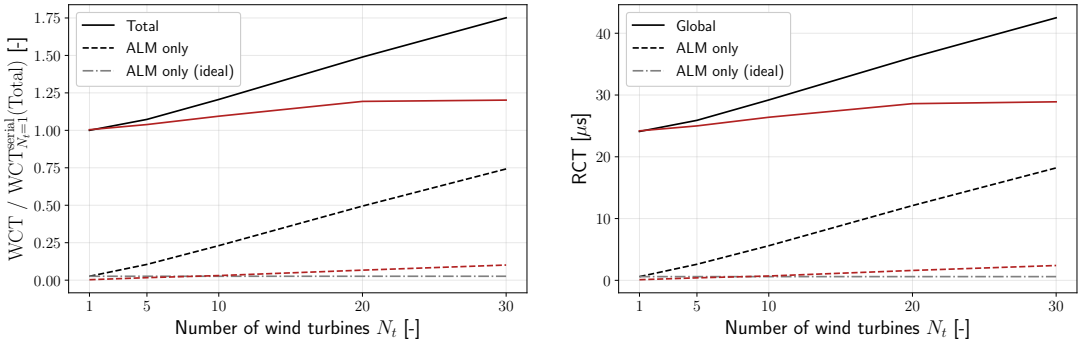

Figure 2: WCT (left) and RCT (right) obtained for a growing number of wind turbines in the computational domain. The results given by the *serial* and *parallel* implementations of the ALM appear as black and red lines, respectively.

4 CONCLUSION

In this paper, we outlined an implementation of the actuator line method in a massively parallel LES code prioritizing optimal workload balancing and minimal return time. The simulation of an arbitrary number of wind turbines is made parallel and efficient by means of an object-oriented programming, relying on rotor-specific MPI communicators, a two-level grid partitioning and an extensive use of bounding boxes. Preliminary tests, performed on a fairly coarse mesh, showed promising results and the reported trends will be confirmed on more demanding cases in a near future.

References

[1] P. Bénard, A. Viré, V. Moureau, G. Lartigue, L. Beaudet, P. Deglaire, and L. Bricteux, "Large-eddy simulation of wind turbines wakes including geometrical effects," *Computers & Fluids*, vol. 173, pp. 133–139, 2018.

[2] R. Stanly, L. A. Martínez-Tossas, S. H. Frankel, and Y. Delorme, "Large-eddy simulation of a wind turbine using a filtered actuator line model," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 222, p. 104868, 2022.

[3] V. Moureau, P. Domingo, and L. Vervisch, "Design of a massively parallel CFD code for complex geometries," *Comptes Rendus Mécanique*, vol. 339, no. 2-3, 2011.

[4] J. N. Sørensen and W. Z. Shen, "Numerical Modeling of Wind Turbine Wakes," *Journal of Fluids Engineering*, vol. 124, no. 2, p. 393, 2002.