Diffusive orthogonal load balancing for Euler–Lagrange simulations

Antoine Stock, Ghislain Lartigue, Vincent Moureau

Received: 14 April 2022 | Revised: 17 February 2023 | Accepted: 25 February 2023

DOI: 10.1002/fld.5191

RESEARCH ARTICLE

WILEY

Diffusive orthogonal load balancing for Euler–Lagrange simulations

Antoine Stock | **Ghislain Lartigue** | **Vincent Moureau**

CORIA, Normandie Université, UNIROUEN, INSA Rouen, CNRS, Saint-Étienne-du-Rouvray, France

Correspondence
Antoine Stock, CORIA, Normandie Université, UNIROUEN, INSA Rouen, CNRS, Avenue de l'Université, Saint-Étienne-du-Rouvray, 76801, France.
Email: antoine.stock@coria.fr

Funding information
Horizon 2020 Framework Programme, Grant/Award Number: 952181

Abstract

In the context of unsteady 3D simulations of particle-laden flows, a new double-constraint load balancing strategy for Euler–Lagrange models is proposed. The method relies on an existing Eulerian partitioning and implements a Lagrangian load balancing step, which is orthogonal to the pre-existing Eulerian balancing. This orthogonality property ensures to keep a near-to-ideal Eulerian load balance while strongly improving the distribution of the Lagrangian particles on the processors. The method has been designed to handle large unstructured 3D meshes on complex geometries. Lagrangian performance measurements performed on massively parallel simulations of realistic spray cases show a CPU cost reduction up to 70% compared to the unbalanced case.

KEYWORDS

double-constraint partitioning, Euler–Lagrange, large-eddy simulation, load balancing, particle-laden flows

1 | INTRODUCTION

Unsteady 3D computational fluid dynamics (CFD) has become a valuable tool for the prediction of particle-laden flows despite its computational power demand. Parallel efficiency in CFD is achieved when the workload is well balanced, that is, by splitting evenly the workload among the cores minimizing idle time. In Eulerian approaches, the load balancing methods often consist in giving each core the same amount of contiguous mesh elements. It is referred to as domain decomposition and achieved by using single-constraint partitioning methods.[1,2] This form of load balancing is highly efficient for pure aerodynamic simulations.[3] This is due to the fact that aerodynamic phenomena are solved using the same set of equations at each mesh element making the cost proportional to the number of elements. However, these methods struggle with multi-physics problems that may introduce space or time irregular workload. For example, in combustion applications, the chemical source terms may be solved iteratively, thus resulting in a large cost in hot regions and negligible cost in cold regions.[4] Dispersed two-phase flows solved with an Euler–Lagrange approach are also a typical example: the carrier phase is continuous but the dispersed phase only exist locally.

Particle-laden flows can be handled in two separate ways. The first one, consists in distributing the particles evenly among the cores with no regards to their spatial localization on the mesh. This is known as particle sharing algorithm[5-8] or as dual grid approach in AMReX[9] and is represented on Figure 1A. It ensures a close to perfect load balance since the Eulerian and Lagrangian constraints are fully decoupled: a particle and its containing mesh cell are not owned by the same core. As a consequence, every Euler–Lagrange interaction will require parallel communications which can result in high overheads and thus bad parallel scalability on large core counts. Mirror domain decomposition has been proposed

by Capecelatro and Desjardins[10] to solve this issue. A complete copy of the data, mesh and particles is given to each core. However, this drastically increases memory usage and forces many communications to keep data synchronized, rendering it inefficient on more than 32 cores.[10]

In large parallel simulations of particle-laden flows, frequent data exchanges between the Euler and Lagrangian phases often impose to have a consistent spatial partitioning of the two phases. The particles and mesh elements are split among the cores based on the same domain decomposition. This method is known as spatial particle partitioning and is represented in Figure 1B communications due to Euler–Lagrange interactions are reduced, but particles can be ill balanced. For rather homogeneous particle distributions, spatial particle partitioning is efficient.[10] However, if the particle distribution is heterogeneous, Lagrangian parallel efficiency is likely to be deteriorated.[11] Heterogeneous particle distributions occur in many common CFD applications, like spray injectors used in combustion and atomization processes. A typical spray injector particle distribution is characterized by a high particle concentration close to the injection point. Further downstream the particle concentration decreases sharply due to the particles spreading out or being evaporated.

Hybrid approaches combine particle sharing and spatial particle partitioning. Computational cores are grouped into a larger structures called nodes. Inside a node, all cores benefit from shared memory access, meaning that each core can access every particle and mesh part on the node. A particle sharing algorithm can be used inside a node without any data exchange or data duplication. Therefore the load is well balanced inside of each node, without any of the previously mentioned downsides. But since the memory is not shared among the nodes, spatial partitioning must still be performed at the node level, which can finally result in load imbalance among the nodes. This method can provide great performance improvement and it has been shown to scale well for application with 100,000 particles.[12] Recent developments have been proposed to improve this method by adding an asynchronous Euler–Lagrange framework.[13,14] The downside of this method is its heavy dependency on the hardware and it still requires a load-balancing algorithm at the node level.

This article focuses on improving the efficiency of spatial particle partitioning. Hybrid approaches are not considered but could also benefit from the improvements of the spatial particle partitioning method as mentioned above. A first idea to perform an ideal partitioning for both Euler and Lagrange weights would be to gather those weights into a single constraint. Then any classical partitioning tool such as METIS[15] or SCOTCH[16] could be used to perform this coloring. This is relevant when the particle distribution is close to homogeneous but this usually fails as soon as the particle load becomes very irregular. The second idea is to use a double constraint approach where both Lagrange and Euler weights are considered separately by the coloring algorithm. Double constraint partitioning is a more complex problem than single constraint partitioning as both constraints may contradict one another. This is the case for Euler–Lagrange simulations, especially when the particle distribution is very heterogeneous. On the one hand the Lagrangian constraint attempts to maximize the number of cores in regions with high particle concentration. On the other hand the Eulerian constraint attempts to provide a very homogeneous partitioning without any local core concentration. Standard partitioning libraries like METIS[15] provide double constraint load balancing routines. However, these methods rarely converge to a solution when contradictory constraints are requested, making them too unreliable to be used.

Load refinement algorithms are able to bypass this issue: they take a basic partitioning as an input and attempt to improve its balance. Usually a single constraint Eulerian partitioning serves as the initial state and the Lagrangian constraint is improved by shifting load away from the most loaded parts. The algorithm stops when no more improvements can be made and the resulting partitioning is at least as good as the initial one if not better. Methods based on hierarchical domain decomposition[17] and diffusion have been developed.[18] In these approaches the load is refined by shifting the boundaries between the cores and notable improvement on the balance is made. However refinement tends to deteriorate

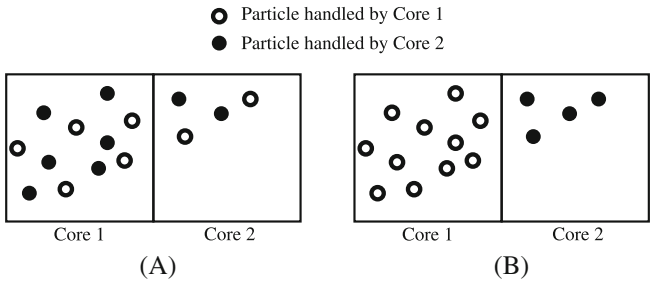


FIGURE 1 Illustration of (A) particle sharing and (B) spatial particle partitioning mechanisms.

one constraint to improve another, resulting into a compromise between both constraints. In these cases, it is likely for the efficiency gain on one constraint to be absorbed by the efficiency loss on the other. A solution to this issue is provided in this article.

This article presents a new double constraint load balancing strategy. It is referred to as DOB-EL (diffusion based orthogonal load balancing for Euler–Lagrange simulations). Its key feature, orthogonality, implies that the Lagrangian balance can be improved without any alteration to the already balanced Eulerian load. The method has been designed to be generic and work on most simulation set-ups. It is able to handle massively parallel simulations with complex geometries and unstructured 3D meshes. It can be performed either statically or dynamically during a simulation.

A few guidelines are provided below to assess if DOB-EL should be used to optimize a given simulation based on the results described in this article. Because DOB-EL starts from a given partitioned mesh and particle sets, it performs best on large heterogeneous particle distributions. It should be avoided to perform DOB-EL on:

- Large particle distributions with low particle density gradient within the particle populated area, this includes among others fluidized beds. This is because the subpart exchange process bases itself on the particle gradient between the subparts, if no gradient is present no exchanges will be done and the final state will be equal to the initial state.
- Small simulations, the method has a some overhead that would deteriorate performance on these small simulation.
- The Lagrangian cost should not completely dominate the Eulerian cost, if it does, not considering the Eulerian constraint and doing single-constraint Lagrangian load balancing is likely to be more efficient.

The article is organized as follows. Section 2 introduces the method and the underlying theory. Section 3 details the implementation of the method and validates its core aspects. In Section 4, the method is applied to large scale cases and its efficiency and cost are assessed. Concluding remarks are given in Section 5.

2 | DOB-EL METHOD

This section introduces the theory behind the load balancing method and the method itself. First the double domain decomposition technique and the diffusive load-balancing strategy are detailed, then both notions are combined in the DOB-EL method.

2.1 | Double domain decomposition

Dealing with each mesh element individually is not a viable strategy for load balancing as meshes can be composed of billions of elements. The solution is to use a double domain decomposition to create small contiguous groups of elements that will act as elementary bricks for all mesh manipulations. This double domain decomposition technique is illustrated in Figure 2 and can be decomposed into two steps:

A first domain decomposition is performed to split the mesh into contiguous parts with a similar amount of mesh elements. The number of parts created equals the number of computational cores, thus each core receives a single part. This is commonly done in parallel CFD codes. Figure 2B represents the first domain decomposition.

A second domain decomposition is then performed to split each part into contiguous subparts with a similar amount of mesh elements. These will act as the elementary bricks for the load balancing algorithm. The impact of the number of subparts on the efficiency of the DOB-EL method will be discussed extensively in Section 2.3.4. Figure 2C represents the second domain decomposition.

Both decompositions are achieved by using a single constraint partitioning algorithm: the multilevel k-way partitioning[19] available in METIS[15] has been used throughout this work.

Finally, a non-oriented graph based on this double domain decomposition can be built and used in the load balancing process. Each node of the graph represents a single mesh subpart and the edges represent the connectivity between theses subparts. The core to graph vertex correspondence is referred to as the graph coloring, each color representing a given core. Figure 2D represents a graph built based on the double domain decomposition.

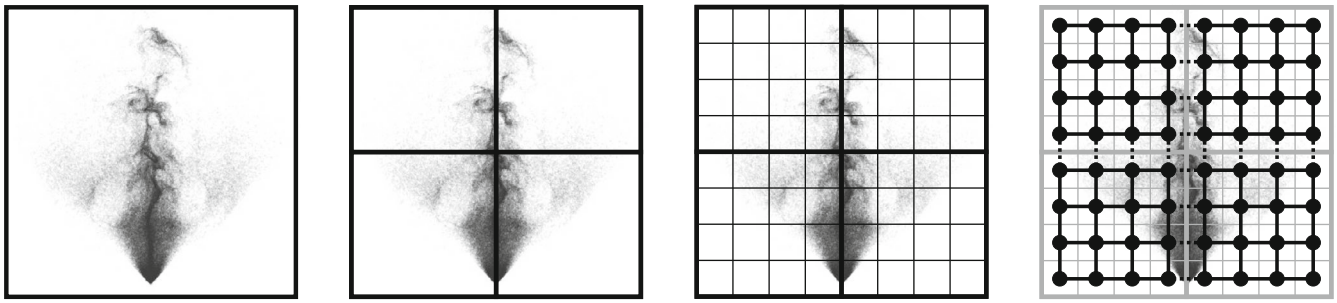


FIGURE 2 Double domain decomposition and associated graph. This is a simplified example with a square shape decomposition but subparts may have irregular shapes forming no pattern. (A) CFD domain. (B) Single domain decomposition with 4 cores. (C) Double domain decomposition with 4 cores and 16 subparts per core. (D) Graph created based on the double domain decomposition.

The following notations are used throughout this work:

- n: total number of vertices of the graph/total number of subparts in the mesh.
- k: number of cores used in the simulation.
- V: set of all n vertices, represents the whole mesh.
- V_c: set of all vertices on core c, represents a mesh part.
- v_i: ith vertex, represents a mesh subpart.

In order to describe the Eulerian and Lagrangian constraints on the graph, the following weight functions are introduced:

- ω_e: Eulerian weight function.
- ω_l: Lagrangian weight function.

The weights are real values associated to each vertex of the graph which are then used by the load balancing algorithm whose goal is to achieve an equipartition of these weights. They can also be evaluated for vertex sets as the sum of the weight of all vertices. Throughout this work the most basic weight model is used, that is, ω_e counts the number of elements per vertex and ω_l counts the number of particles per vertex. This assumes that every element has the same Eulerian cost and that every particle has the same Lagrangian cost. More complex models could be implemented in the same manner.

Balance is attained for a constraint when the associated weight function result is the same for all parts. A more general way to express this is to introduce the imbalance function. The Lagrangian imbalance of a subpart V_i is

$$LI(V_i) = k\frac{\omega_l(V_i)}{\omega_l(V)}, \tag{1}$$

$LI(V_i) < 1$ means that V_i is underloaded, $LI(V_i) > 1$ means that V_i is overloaded and $LI(V_i) = 1$ corresponds to the perfect balance.

Performance is dependent on the maximum load, that is to say the maximum imbalance. Therefore, the aim of the load balancing is to reduce the maximum imbalance. The maximum Lagrangian imbalance will be referred as LI_{MAX}.

2.2 | Diffusion and dimension exchange

The load balance is improved by moving subparts from one core to another, which on the graph consists in changing the color of the associated vertices. While load-balancing the graph, the mesh is left untouched and will be updated later on. The first step is to identify which cores should exchange load and how much load has to be exchanged.[20]

This is achieved by using a diffusion based algorithm.[21] Diffusion is a well-known iterative load-refinement method in which tasks are moved from heavy-loaded parts to lightly-loaded neighbor parts. Assuming that no tasks are created during load balancing, the discrete diffusion of the Lagrangian load can be expressed as a Laplace-Beltrami operator:[22]

$$\omega_l^{t+1}(V_i) = \omega_l^t(V_i) + \sum_j \alpha_{ij} \left(\omega_l^t(V_j) - \omega_l^t(V_i)\right), \tag{2}$$

where t is the current iterative step and α_{ij} is the fraction of load difference to be exchanged. In the present algorithm, we chose to set $\alpha_{ij} = 0$ if V_i and V_j share no edge. It can be noted that the proposed approach can be seen as a first order time scheme. Higher order schemes exist for iterative diffusion but they bring no improvements on systems with coarse grain loads.[23] The convergence of Equation (2) is guaranteed if and only if:[22]

$$\alpha_{ij} \geq 0 \qquad \forall i,j \in [1;k], \tag{3}$$

$$1 - \sum_j \alpha_{ij} \geq 0 \qquad \forall i,j \in [1;k]. \tag{4}$$

Equation (2) assumes that each part interacts with all its neighbors at every step. However, this does not perform well on supercomputers as point to point communications can cause an important network contention. Dimension exchange on hypercubes has been introduced by Cybenko[22] to address this specific issue. The main idea of this algorithm is to traverse all the dimensions of the hypercube sequentially and to perform load exchange between neighboring vertices in this dimension only at a given step. The dimension exchange algorithm is not limited to hypercubes as it has been extended to all graphs.[24,25] A dimension exchange step is expressed as:

$$\omega_l^{t+1}(V_i) = \omega_l^t(V_i) + \alpha_{ij} \left(\omega_l^t(V_j) - \omega_l^t(V_i)\right). \tag{5}$$

The convergence criteria given by Equations (3) and (4) can be simplified as:

$$0 \leq \alpha_{ij} \leq 1 \qquad \forall i,j \in [1;k]. \tag{6}$$

The intermediate value $\alpha_{ij} = \frac{1}{2}$ is commonly used as it usually grants the fastest convergence rate.

On top of its simplified formulation, dimension exchange also converges faster than regular diffusion. An analogy can be made for dimension exchange and diffusion with respectively a Gauss-Seidel and Gauss-Jacobi method.[22]

2.3 | Orthogonal load exchange

While dimension exchange specifies the amount of load to be exchanged, it does not state how to select the load to be exchanged. This subsection will details the selection rules that are applied to exchange load orthogonally while preserving a good aspect ratio of the parts.

2.3.1 | Orthogonality

Orthogonality consists in the ability of changing the balance of one constraint without impacting the other by any mean. In the present case Eulerian balance is considered to be good and should not be altered when improving the Lagrangian balance.

Let us assume that the partitioning algorithm used for the double domain decomposition is ideal. The parts are built based on the Eulerian constraint: this means that after the first domain partitioning each part holds exactly the same Eulerian load:

$$\omega_e(V_i) = \frac{\omega_e(V)}{k} \qquad \forall i \in [1;k]. \tag{7}$$

After the second partitioning, if the same number of subparts are created from each part, every subpart holds exactly the same Eulerian load:

$$\omega_e(v_i) = \frac{\omega_e(V)}{n} \qquad \forall i \in [1; n]. \tag{8}$$

All load exchanges are performed by swapping subparts between processors: each time a subpart is given away, another subpart is received. This swapping mechanism ensures the preservation of the Eulerian balance as the given and received subparts have the same Eulerian load and cancel out. However, both subparts do not have the same Lagrangian load and the swap will change the Lagrangian balance: swaps can then be selected in such a manner that Lagrangian balance gets strictly improved.

Figure 3 gives a representation of this swapping process. Both cores have 4 subparts at any time, that is to say the same Eulerian load. Initially the core 2 holds the majority of the particles, the Lagrangian balance is evened out after two swaps, as both parts have the same amount of particles in the final state. This is an ideal example, on real cases the Lagrangian balance will be improved but won't reach a perfectly balanced state as shown later on in this article in Sections 3.3 and 4.2

2.3.2 | Selection of swapped vertices

Let V_H and V_L be two adjacent parts such that

$$\omega_l(V_H) - \omega_l(V_L) = D > 0. \tag{9}$$

Let v_H and v_L be the swapped vertices so that

$$\omega_l(v_H) - \omega_l(v_L) = d. \tag{10}$$

In order to ensure optimal swaps the following rules are used:

- (R1): Convergence towards load homogeneity
  Convergence requirement expressed by Equation (6) can be reformulated as:

$$0 < d < D. \tag{11}$$

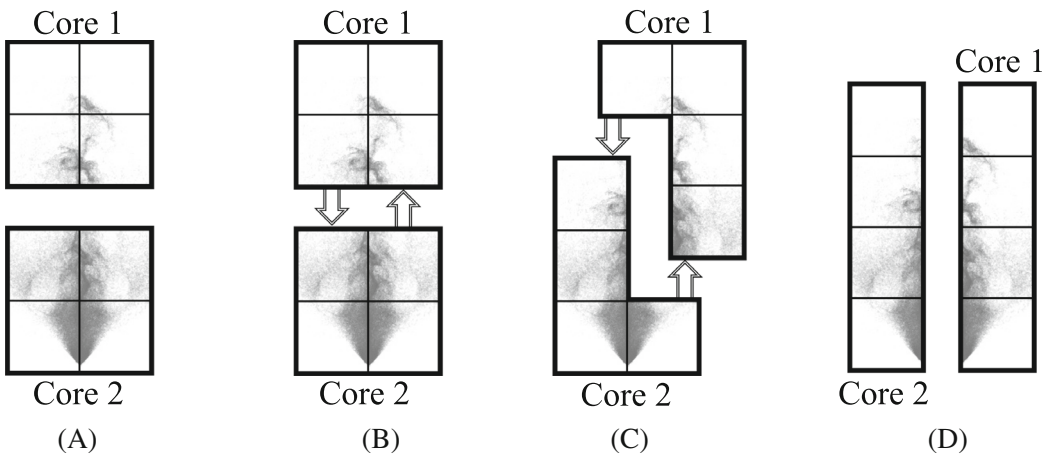


FIGURE 3 Load balancing using the orthogonal swapping process. This is a simplified example with a square shape decomposition but subparts may have irregular shapes forming no pattern. (A) Initial state. (B) First swap. (C) Second swap. (D) Final state.

In order to speed up convergence, Equation (11) can be narrowed:

$$c < d < D - c, \tag{12}$$

with c a positive constant

- (R2): Contiguity
  V_H and V_L have to remain contiguous at all times. This is a vital condition for some CFD applications and increases part quality.
- (R3): Edgecut control
  Many pair of vertices satisfy (R1) and (R2). Using the pair ensuring the faster convergence would result in very entangled parts and large edgecut. The edgecut of a part is the number of its external edges, that is, the number of edges connecting it with another part. It is thus directly linked to the size of the interfaces between parts. It can be expressed globally on the graph or locally for a color. Edgecut should always be kept as low as possible to ensure a low communication overhead in parallel codes (Section 2.3.3 will provide more in-depth explanations on this matter). The variation of the maximum local edgecut is expressed as:

$$\Delta Ec = \max\left[Ec(V_L), Ec(V_H)\right]_{\text{after swap}} - \max\left[Ec(V_L), Ec(V_H)\right]_{\text{before swap}}. \tag{13}$$

If there is a node pair inducing $\Delta Ec \leq 0$, it should be swapped. If all the possible swaps imply an increase of the edgecut, the pair with the highest balance gain over edgecut loss is swapped:

$$(v_H, v_L) = \text{argmax}\left(\frac{\frac{D}{2} - \left|d - \frac{D}{2}\right|}{\Delta Ec}\right). \tag{14}$$

Section 3.3.3 will demonstrate the benefits of using edgecut control on a 2D example.

2.3.3 | Importance of edgecut control

The edgecut of a part can be related to its geometrical shape. Minimizing the edgecut of a part can actually be interpreted physically as minimizing the capillary force arising from surface tension in liquids and thus driving the part to a spherical shape. Figure 4 represents two parts as an example of good and bad edgecut.

When edgecut control is not included in the DOB-EL algorithm, most parts quickly tend to have a tree or filament shape instead of a rather round shape which can create a variety of issues. The most important one in CFD simulation is because of point to point communications that take place at core to core interfaces. Point to point communication cost consist of a constant overhead occurring at the beginning of the communication, referred to as latency, and a variable part that scales with the amount of data to be transferred. An increased edgecut implies an increased interface size between

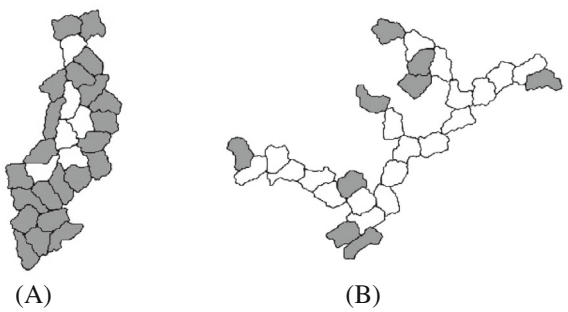
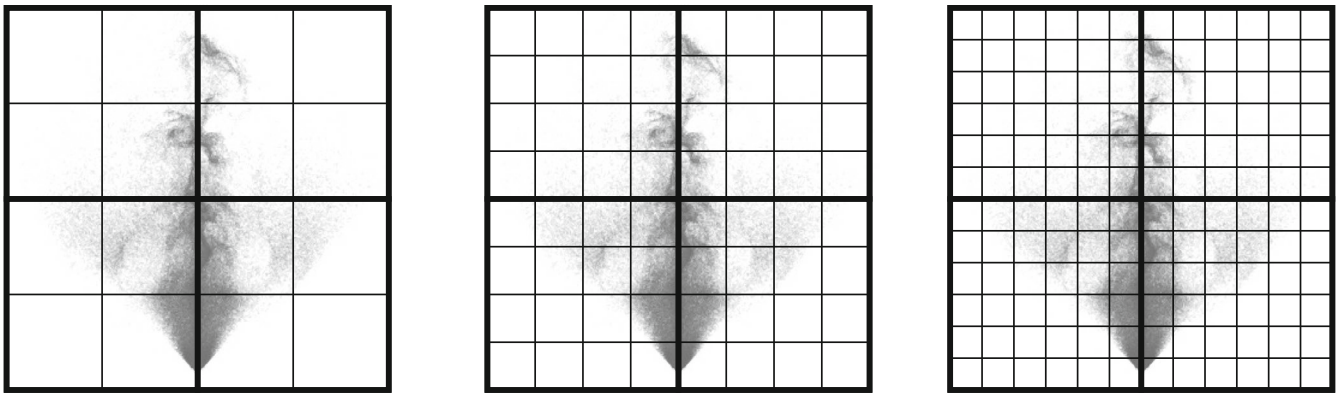


FIGURE 4 Comparison between parts with good and bad edgecut, subparts that could be swapped without breaking contiguity are grayed. For this example edgecut of part (B) is 2.2 times greater than part (A). (A) Part with low edgecut. (B) Part with high edgecut.



(A) (B) (C)

FIGURE 5 Illustration of double domain decomposition for different grain sizes. This is a simplified example with a square shape decomposition but subparts may have irregular shapes forming no pattern. (A) Large grain size. (B) Medium grain size. (C) Small grain size.

the cores and therefore more data to be transferred between the cores, and finally an increase in the variable part of the communication cost. Additionally if a mesh part is very elongated it is susceptible to interact with an increased number of neighboring cores, increasing also the number of communications to be performed and the latency cost. Impact of edgecut on performance will not be measured in this article as it is very dependent on the CFD code and the physics that are solved.

During the swapping process, having many swappable subparts without breaking the contiguity is a big advantage as it increases greatly the likelihood of finding a good swap. When the parts are elongated the number of swappable subparts decreases greatly due to the presence of filaments. This phenomenon is represented in Figure 4. In these cases the parts can get stuck in that shape and become inactive in the load balancing process. This happens especially when the load balancing is used dynamically. It should be noted that controlling the edgecut does not forbid elongated shapes, but helps elongated shapes to recover a rounder shape later on.

2.3.4 | Grain size

Grain size refers to the smallest unit of load that can be exchanged. In the current context the grain size corresponds to the size of the subparts as shown on Figure 5. The grain size has a major impact on the behavior of the algorithm:

- A high number of subparts leads to a better balance but edgecut and load balancing cost increase.
- Conversely, a low number of subparts leads to a lower load balancing cost but has worse and less consistent results.

A detailed analysis of the impact of the number of subparts on the performance of DOB-EL is provided in Section 3.3 on a 2D test case and in Section 4.2 on large scale cases.

3 | IMPLEMENTATION AND VALIDATION

This section gives an overview of the CFD code YALES2 and the implementation of the load balancing method. A 2D validation case is used as a pedagogical example to show the behavior of the DOB-EL algorithm.

3.1 | About YALES2

YALES2[26] is a massively parallel low-Mach number code for the DNS and LES of reacting two-phase flows in complex unstructured geometries. The code includes several solvers allowing for simulations in a wide range of scientific fields.

This includes among other, combustion, wind turbines and biomedical modeling. YALES2 is also capable of dynamic adaptive mesh refinement during runtime.

The structure of YALES2 relies on a double domain decomposition. This property is used throughout the code for optimization purposes allowing better CPU cache usage. It is also used to design highly efficient algorithms like the DPCG[27] to solve the Poisson equation that arises in all low-Mach number applications. Lagrangian methods, implemented in particle-in-cell formalism, also benefit from the double domain decomposition and particles are always bound to a given subpart. If many particles are located on a same subpart, several particle groups are used to optimize once again the CPU cache usage, adding a third level of decomposition.

3.2 | Implementation

The implementation of the load balancing process is illustrated in Figure 6 and is decomposed into three algorithms going from the largest scope to the smallest.

Algorithm 1 describes the high level implementation of the method. A double domain decomposition is performed on each core and the associated local graph is created. This graph is then assembled on a single core, using MPI gathering. This graph is a light weight representation of the subpart weights and connectivity, as represented in Figure 2D. A sequential coloring is performed using DOB-EL, no communication is required during the coloring process. Afterwards, the new coloring is scattered among all cores and mesh parts are exchanged accordingly using non-blocking MPI point-to-point communications. This implementation has the benefit of simplicity and performing the least communications. Its scalability could be questioned because a single core performs the coloring of the full mesh but it's cost is low as shown in Section 4.3.

The main stage of the method is depicted in Algorithm 2. The cores are paired two by two sequentially, starting with the most loaded cores and highest load differences, and subparts are exchanged on the graph. After all the cores have performed an iteration of dimension exchange, another iteration is to be performed only if a favorable load exchange

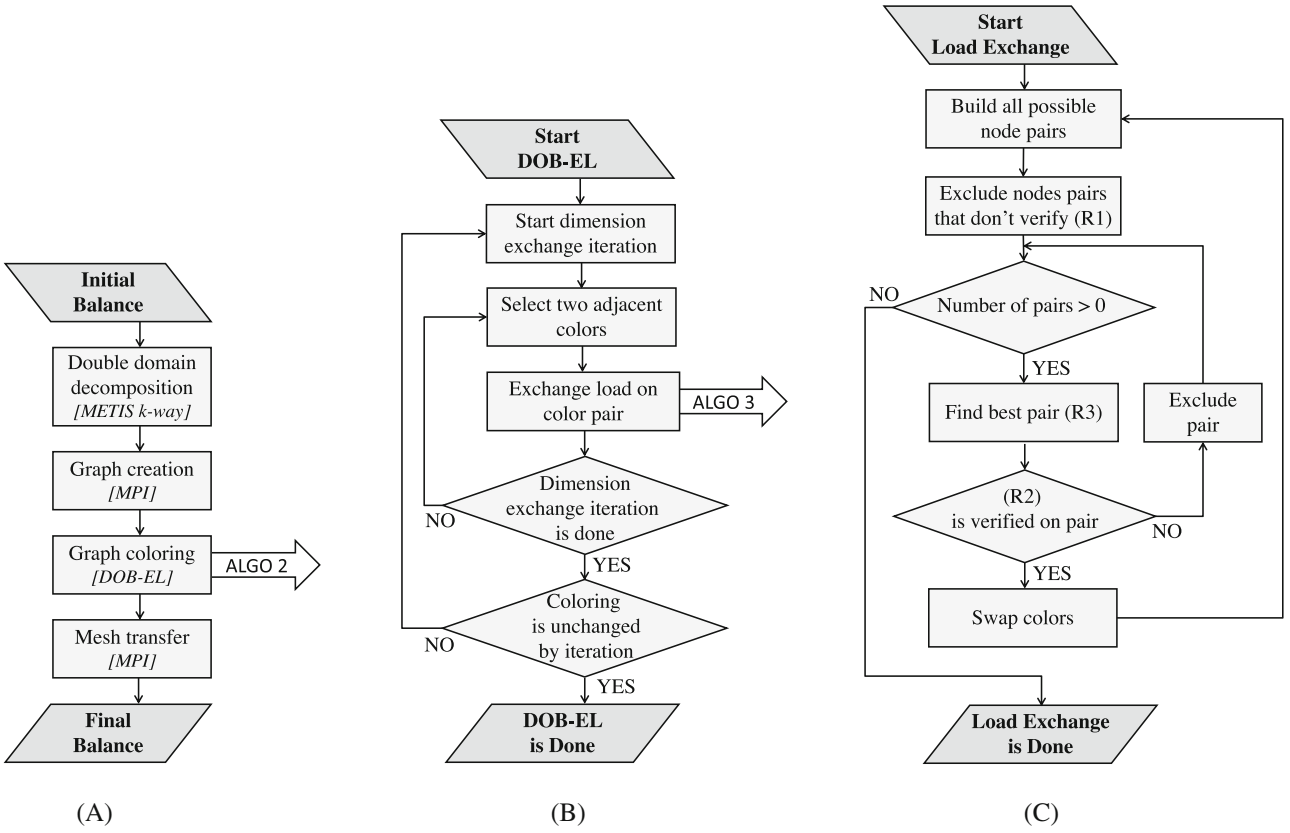


FIGURE 6 Algorithmic implementation. (A) Algorithm 1: Main. (B) Algorithm 2: DOB-EL. (C) Algorithm 3: Detailed load exchange.

has happened. As the main objective of the algorithm is to reduce LI_{MAX}, only pairs including a highly loaded core are considered. In the proposed implementation, a core is considered highly loaded if its Lagrangian imbalance exceeds 90% of the current LI_{MAX}.

Algorithm 3 details the swapping process between two parts. All the subparts connected to their opposing color by an edge are selected. All the pairs built using these subparts are considered. If a pair does not verify (R1) it is rejected. Afterwards, the optimal pair is considered (R3) and contiguity is checked (R2). If contiguity is verified, the swap is performed and Algorithm 3 is repeated. If contiguity is not verified, the pair is rejected and a new optimal pair is searched. When no selectable pairs remains, the load exchange between the two parts ends.

3.3 | 2D validation case

A simple test case is used to give a visual representation of how the algorithm works and to validate the behavior of the Eulerian and Lagrangian balance. The considered domain is a 2D square box in which particles are injected at the center of the domain and spread out in every direction. Figure 7A shows the resulting particle distribution. Particles density decreases as the inverse of the distance to the injection point. The particle distribution is similar to a 3D injector distribution projected on a plane perpendicular to the injection direction.

The case is composed of 1,835,664 triangular elements, 13,344 particles and is run on 25 cores.

It must be noticed that the DOB-EL algorithm will produce different results depending on the initial partitioning that is provided. Therefore the following analysis is conducted statistically on a large batch of runs, each run being based on a different initial state. A multitude of distinct initial states is achieved by modifying the seeding of the random number generator in the k-way partitioning algorithm of the METIS library. For the present study, a total of 10,000 runs have been performed.

A typical initial partitioning is represented on Figure 7B and the corresponding final partitioning is represented on Figure 7C.

3.3.1 | Eulerian balance

The Eulerian balance is assumed to be ideal in the model by Equation (8). However, in real cases, balance is slightly off due to the error margin of single constraint partitioning. A maximum imbalance equal to 1.01 is imposed on the k-way partitioning algorithm of the Eulerian mesh. This means that a part can exceed the ideal load by 1% at maximum.

On average, the initial maximum Eulerian imbalance is 1.0092 and it never exceeds the 1.01 threshold as expected. The same threshold is used to create the subparts during the second decomposition. As a consequence, the load exchange process is no longer perfectly orthogonal to the Eulerian load as Equation (8) is no longer verified. However this effect is minute and will be neglected in the remaining of this study.

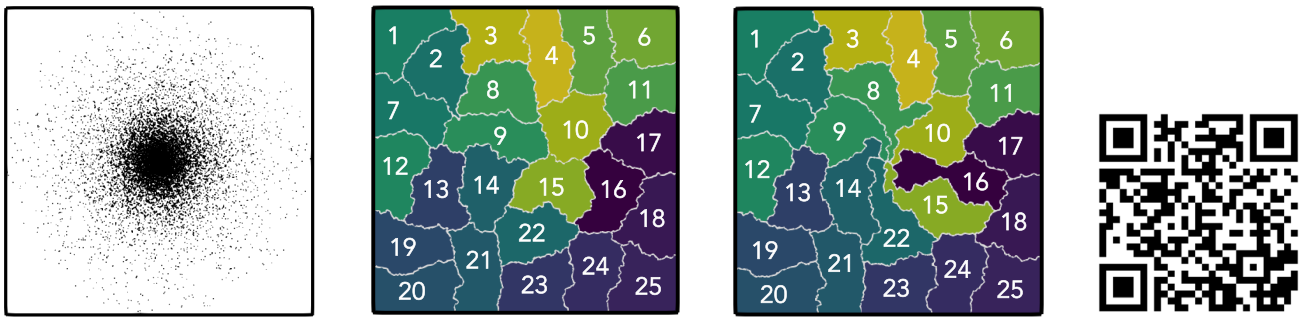


FIGURE 7 Representation of the 2D case and load balancing, on 25 cores with 40 subparts/core. (A) Particle distribution in the 2D square domain (outline). (B) An initial single constraint Eulerian partitioning. (C) Partitioning after DOB-EL. (D) Link to video of the load balancing process. [Colour figure can be viewed at wileyonlinelibrary.com]

In 61% of the 10,000 runs, the maximum Eulerian imbalance remained the same, as the part with the maximum load did not exchange any load. This happens when the part is skipped by the diffusion process because its Lagrangian load is low and not worth diffusing.

In 28% of the cases, the maximum Eulerian imbalance decreased and in 11% of the cases, the maximum Eulerian imbalance increased slightly. The worst recorded increase is 0.23% relative to the initial balance, which remains a negligible increase.

3.3.2 | Lagrangian balance

The Lagrangian load is shifted away from the most loaded cores. After DOB-EL is done there is no longer one core with a peak load but instead several cores sharing most of the load, as shown by Figure 8. The balance has been improved substantially but is still not ideal because the cores located too far away from the areas with high particle load couldn't receive a part of the load without losing contiguity or deteriorating the edge cut a lot.

Load balancing is performed using four different grain sizes and results are compared to the reference case without load balancing. Figure 9 represents the probability density function (PDF) of the final LI_{MAX} for all cases. The initial LI_{MAX} (reference) is very inconsistent and ranges from 6 to 15. This stresses the need for a load balancing algorithm.

The final LI_{MAX} decreases significantly when load balancing is performed even with large grain size. Reducing grain size leads to smaller imbalance and consistency increases. In very rare cases the imbalance remains rather high as the algorithm gets stuck early because no more satisfactory swaps can be performed. Using a smaller grain size or reiterating the algorithm with another double domain decomposition can alleviate this issue.

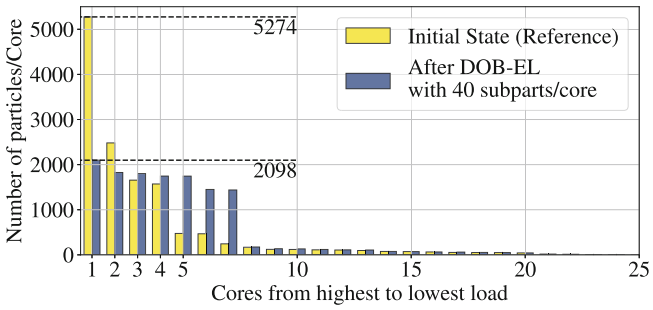


FIGURE 8 Distribution of the Lagrangian load among the cores before and after DOB-EL for a given run. [Colour figure can be viewed at wileyonlinelibrary.com]

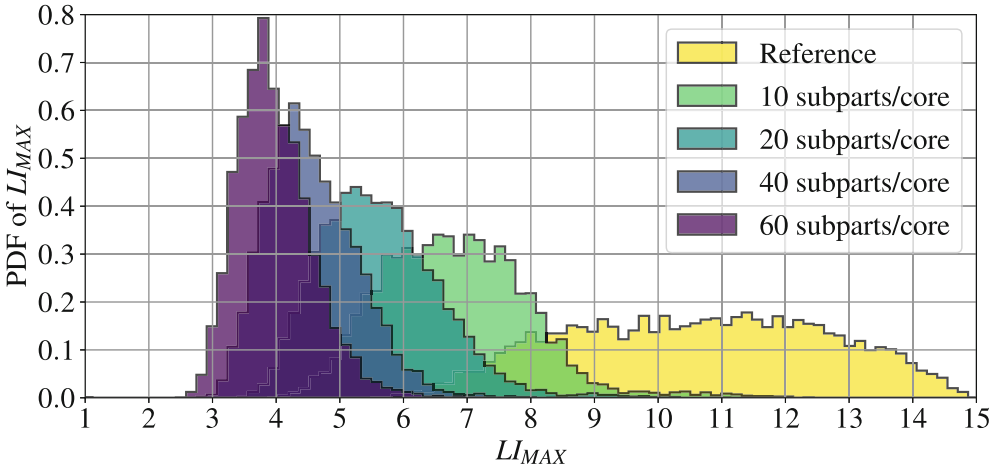


FIGURE 9 PDF of Lagrangian imbalance for different grain sizes. [Colour figure can be viewed at wileyonlinelibrary.com]

3.3.3 | Edgecut control

The efficiency of edgecut control (R3), detailed in Section 2.3.2, is demonstrated by comparing the results on the 2D case with edgecut control and without it. When edgecut control is enabled the ideal swap is maximizing the ratio of load balance improvement over local edgecut increase. Without edgecut control the ideal swap is the one maximizing load balance improvement regardless of local edgecut increase.

Table 1 shows by how much the local edgecut increase with and without (R3) for different numbers of subparts/core. It appears that (R3) limits the edgecut increase by approximately a factor 2. It should also be noted that local edgecut increases when the number of subparts per cores increases.

It is also important to consider the impact of (R3) on the final LI_{MAX}. Results in Table 2 indicate that (R3) benefits DOB-EL as a slightly better final load balance is reached with it.

3.3.4 | Comparison to single constraint Euler–Lagrange approach

When performing Euler–Lagrange load balancing using a single constraint approach, the simplest approach that can be envisioned is to define a composite weight function as:

TABLE 1 Influence of edgecut control on maximum edgecut increase.

	Number of subparts/core			
	20	**40**	**60**	**80**
With edgecut control	+13.7%	+18.9%	+21.0%	+21.1%
Without edgecut control	+23.6%	+35.4%	+42.1%	+46.8%

TABLE 2 Influence of edgecut control on LI_{MAX}.

	Number of subparts/core			
	20	**40**	**60**	**80**
Final LI_{MAX} with edgecut control	5.27	4.30	3.95	3.84
Final LI_{MAX} without edgecut control	5.58	4.93	4.65	4.57

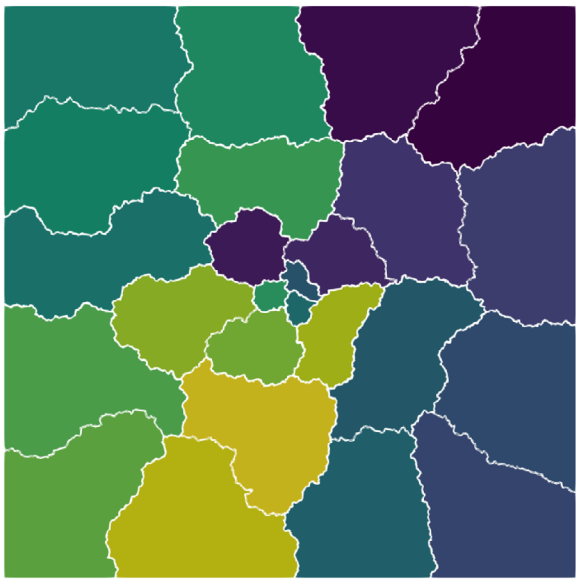


FIGURE 10 Example of single constraint Euler–Lagrange partitioning. [Colour figure can be viewed at wileyonlinelibrary.com]

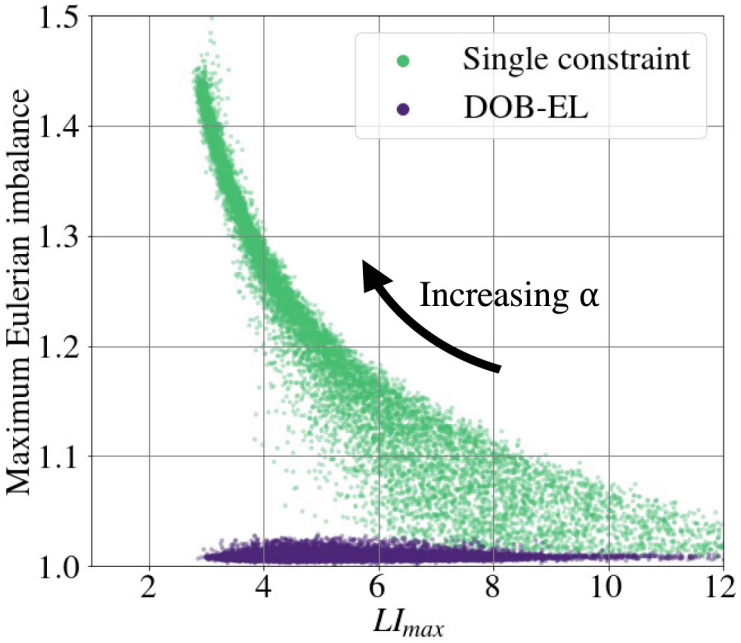


FIGURE 11 Dependency between Eulerian and Lagrangian imbalance for the Euler–Lagrange single constraint and DOB-EL approach. Each point represents a run. [Colour figure can be viewed at wileyonlinelibrary.com]

$$\omega = (1-\alpha)\,\omega_e + \alpha\omega_l \qquad \forall\alpha \in [0;1]\ , \tag{15}$$

α is a user-defined variable that describes how much importance is given to the Eulerian weight relative to the Lagrangian weight. An example of partitioning based on this approach is given in Figure 10. In regions with high particle density, partition will have an under-average element count, thus covering less space and including fewer particles. Overall, it can be interpreted as a compromise between Eulerian and Lagrangian balance.

A set of 10.000 runs using the single constraint approach have been performed for various $\alpha \in [0;1]$. The dependence between the Eulerian and Lagrangian imbalance is shown in Figure 11. When one imbalance decreases, the other increases. In contrast, for the DOB-EL approach, no dependence exists between the two constraints. Reducing the Lagrangian imbalance does not increase the Eulerian balance, thus orthogonality is achieved.

4 | LARGE SCALE APPLICATIONS

4.1 | Presentation of the simulation set-ups

Three large scale simulations are considered. These applications share the common aspect of being spray injectors. As spray injector feature very heterogeneous particle distributions, they are well fit to demonstrate the efficiency of the diffusive load balancing method.

- The CRSB (Coria Rouen Spray Burner) injector is a spray burner. The particles are injected and are quickly fully evaporated due to the presence of a flame close by. In this set up, particles are driven by the air co-flow and clump up along the injection center-line. The experimental set-up is detailed in Reference 28 and the associated simulation is available in Reference 29.

  Finite rate chemistry with transported species is used in the Eulerian phase.
- The HERON (High prEssuRe facility for aerO-eNgine combustion) injector is also a spray burner but distinguishes from the CRSB injector due to the particles inertia being greater and therefore not driven by the co-flow. A cone shaped particle distribution is observed. The experimental set-up is detailed in References 30 and 31 and the associated simulation is available in Reference 32.

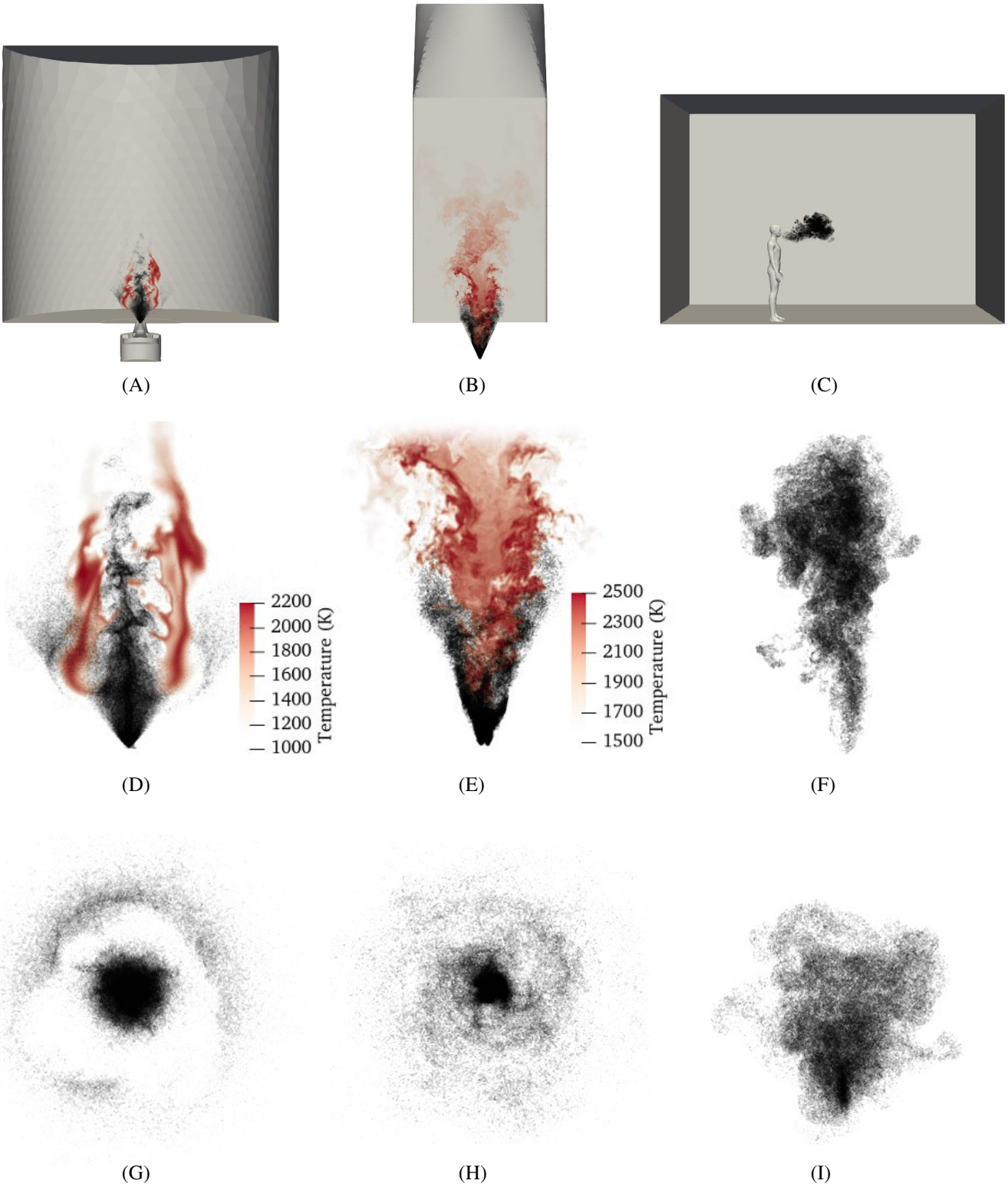


FIGURE 12 Spatial particle distributions and flame topology (only for CRSB and HERON cases). (A) CRSB domain. (B) HERON domain. (C) SALIVA domain. (D) CRSB side view. (E) HERON side view. (F) SALIVA side view. (G) CRSB face view. (H) HERON face view. (I) SALIVA face view. [Colour figure can be viewed at wileyonlinelibrary.com]

Tabulated chemistry is used in the Eulerian phase.

- The SALIVA injector replicates saliva droplets in human breath. These particles are only partially evaporated and stay suspended in the stagnating surrounding air. Their residence time in the domain is greater than for the spray burner particles. The flow and particles are gradually slowed down and form a cloud like distribution. The simulation of this set-up is detailed in Reference 33.

An overview of the spatial particle distributions of these injectors is given in Figure 12 and additional information about the set-ups is given in Table 3.

4.2 | Load balancing of an instantaneous distribution

The DOB-EL algorithm is applied on the three configurations described above. Each case is studied for three different load balancing grain sizes: 10, 20, and 30 subparts per core. A reference without any particle load balancing is also considered. A total of 100 runs is performed for each of those subcases, using different initial partitionings. All the runs have been performed on the Occigen supercomputer from CINES, France.[34]

The LI_{MAX} is measured directly after load balancing and is presented in Table 4. The LI_{MAX} decreases as the number of subparts is increased like for the 2D case. However, most of the decrease is already achieved with 10 subparts per core, using 30 subparts only improves slightly the LI_{MAX}.

The evolution of LI_{MAX} is a theoretical estimation of the evolution of the Lagrangian performance. The actual Lagrangian performance is assessed based on the wall clock time (WCT) per solver iteration using internal timers of the YALES2 code. The results are displayed in Table 5. A notable performance gain is achieved, up to −70% for HERON with 30 subparts.

The relation between the Lagrangian performance and LI_{MAX} is represented by Figure 13 for all runs. A very clear correlation between performance and LI_{MAX} is observed, meaning that LI_{MAX} is a good performance predictor. This correlation consists of a static overhead and a linearly scaling part. For small values of LI_{MAX} the overhead is dominant, as it is not improved by the load balancing.

TABLE 3 Additional information about the configurations.

	CRSB	HERON	SALIVA
Number of particles	183k	180k	117k
Number of elements	74M	237M	10M
Number of cores	560	896	384
Number of elements/core	132k	265k	26k
Initial Lagrangian cost/iteration	4.12 s	13.59 s	1.12 s
Eulerian cost/iteration	201.9 s	71.3 s	3.03 s
Lagrangian cost in %	2%	16%	27%
% of cores with particles (of total cores)	84.2%	14.4%	41.3%

TABLE 4 Mean $LI_{\max}$ and associated statistical variance.

	CRSB		HERON		SALIVA	
	$\langle LI_{\max} \rangle$	$\sigma(LI_{\max})$	$\langle LI_{\max} \rangle$	$\sigma(LI_{\max})$	$\langle LI_{\max} \rangle$	$\sigma(LI_{\max})$
Reference	45.1	8.8	275.2	49.0	16.19	1.36
10 subparts/core	30.5	8.0	107.7	19.3	5.85	0.87
20 subparts/core	27.7	6.9	82.6	15.1	4.17	0.62
30 subparts/core	24.7	5.4	73.0	14.3	3.46	0.55

TABLE 5 Lagrangian wall clock time (s) per solver iteration and associated statistical variance.

	CRSB		HERON		SALIVA	
	$\langle WCT \rangle$	$\sigma(WCT)$	$\langle WCT \rangle$	$\sigma(WCT)$	$\langle WCT \rangle$	$\sigma(WCT)$
Reference	4.12	0.67	13.59	2.28	1.12	0.06
10 subparts/core	3.03	0.60	5.70	0.87	0.80	0.05
20 subparts/core	2.82	0.50	4.56	0.68	0.72	0.03
30 subparts/core	2.60	0.39	4.12	0.64	0.70	0.03

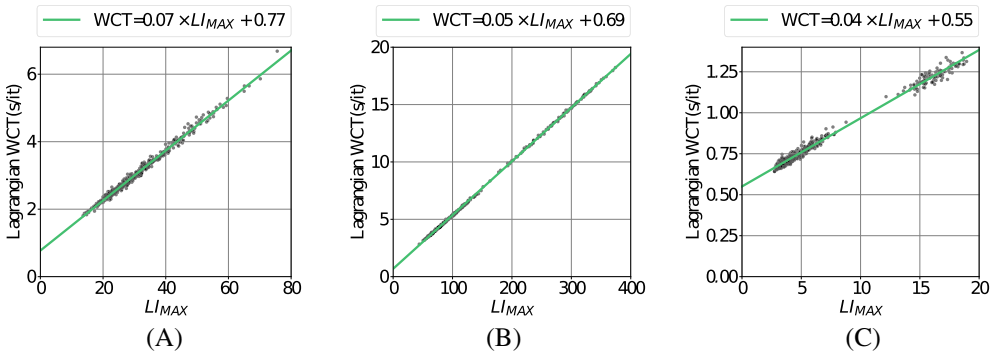


FIGURE 13 Lagrangian performance scaling. (A) CRSB. (B) HERON. (C) SALIVA. [Colour figure can be viewed at wileyonlinelibrary.com]

Eulerian balance remains mostly unchanged in all simulations, which is the expected orthogonal behavior. Some slight Eulerian performance variations have been observed. This is due to the fact that the Eulerian constraint assumes that the cost is perfectly proportional to the number of elements, which it is not. The edgecut increase can also increase slightly the Eulerian cost. The maximum edgecut of a part remained constant for all cases. The global edgecut increase did not exceed 1% for the CRSB and HERON and 7% for the SALIVA injector. Eulerian performance variations never exceed 5% of the total Eulerian cost.

4.3 | Cost of the load balancing method

The cost of the load balance method is crucial as it has to be smaller than the induced performance gain. In order to understand the cost properly it is split into several parts. The detailed results are given by Table 6 and a visual aid is given by Figure 14, the cost decomposition is the following:

- The double domain decomposition cost, using METIS k-way partitioning, increases with the number of elements per core. This is why this cost is low for the SALIVA case and more pronounced for the CRSB and HERON cases. It also increases with the number of created subparts. This cost can be fully skipped on codes based on a double domain decomposition like YALES2, which already provides these subparts.
- The graph creation cost scales with the number of cores and number of subparts but is negligible for less than a thousand cores.
- The diffusive coloring cost scales with the number of subparts because more potential vertex pairs need to be tested to find the optimal swap. Additionally, as smaller parts are exchanged, more swaps are required in the balancing process. However, it is very dependent on the particle distribution and its cost is likely to increase on rather homogeneous cases like SALIVA. The cost does not appear to be scaling with the number cores, this is because the algorithm is performed only on highly loaded cores. On very large core counts, the cost could probably be further reduced using a parallel implementation.

TABLE 6 Detailed load balancing cost for large scale cases and break-even iteration.

	CRSB			HERON			SALIVA		
Subparts/core	**10**	**20**	**30**	**10**	**20**	**30**	**10**	**20**	**30**
WCT METIS (s)	0.392	0.461	0.534	0.795	0.933	1.074	0.067	0.085	0.104
WCT graph (s)	0.005	0.006	0.007	0.007	0.008	0.009	0.003	0.003	0.004
WCT coloring (s)	0.003	0.013	0.042	0.006	0.019	0.048	0.027	0.144	0.530
WCT transfer (s)	0.677	0.817	0.937	1.068	1.097	1.157	0.546	0.876	1.295
WCT post-processing (s)	1.442	1.437	1.449	3.031	3.008	3.510	0.458	0.467	0.471
WCT total (s)	2.519	2.734	2.969	4.907	5.065	5.798	1.101	1.578	2.404
Cost of DOB-EL compared to a Lagrangian iteration prior to load-balancing	61.1%	66.3%	72.1%	36.1%	37.2%	42.7%	98.2%	140.9%	214.6%
Number of iterations to compensate DOB-EL cost	2.31	2.10	1.95	0.62	0.56	0.61	3.44	3.95	5.72

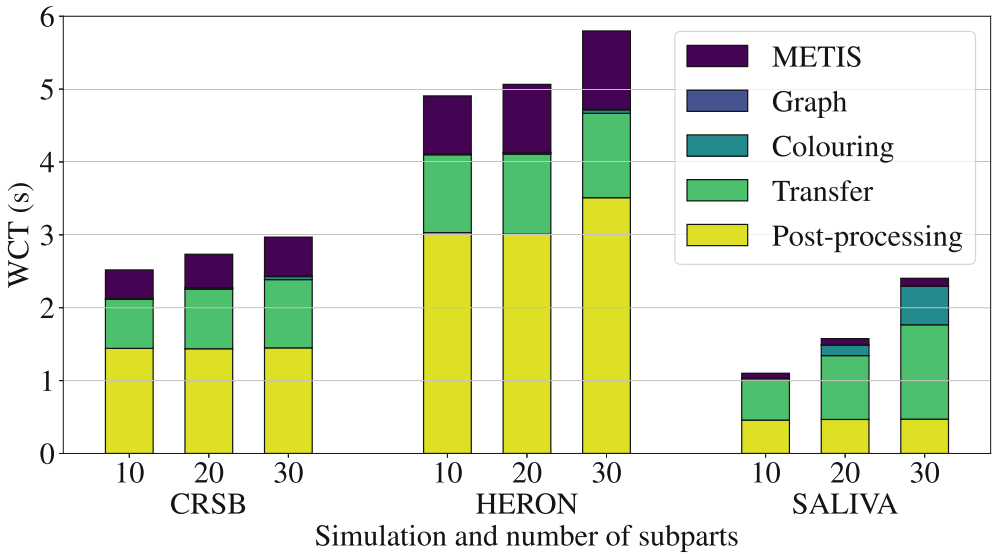


FIGURE 14 Load balancing cost, visual representation of Table 6. [Colour figure can be viewed at wileyonlinelibrary.com]

- The transfer cost corresponds to the communications performed to exchange the mesh parts. It scales with the number of subparts as a better balanced is reached and therefore more mesh elements are likely to be transferred.
- The post-processing cost corresponds to the time required to rebuild data structures and connectivity once mesh subparts have been transferred. This cost is very code dependent and scales with the number of elements per core. This cost is large but can be mitigated for example if the load balancing is done when data-structures are rebuilt anyway, like dynamic adaptive mesh refinement.

Globally, it can be considered that the diffusive load balancing method scales mainly with the number of elements per core and the number of subparts. The number of elements per core is supposed to be similar whatever the case is. It also has been shown that the algorithm works well with a small number of subparts. Therefore, this algorithm should scale well towards larger simulations.

The cost of the method can be compared to the induced cost reduction of Table 5. The load balancing becomes worth it after very few iterations and can therefore be considered as very efficient: it is thus possible to perform the load balancing dynamically during runs if needed.

4.4 | Dynamic load balancing

Load balancing is done on an instantaneous particle distribution. This distribution can change over time as particles are moving and balance may deteriorate. When this happens load balancing needs to be performed again. The three considered cases fall into two categories: steady and unsteady particle distributions.

The particle distributions of CRSB and HERON are steady once the flame is established and does not require repeated load balancing. Figure 15 shows the evolution of LI_{MAX} during a 10 h run, starting from the distribution shown in Figure 12. Small imbalance changes are caused by turbulent structures clumping particle together in some areas. This make the load balancing cost negligible as it is only performed once at the beginning of the run.

The particle distribution of SALIVA changes as the cloud move away from the subject, thus load-balance and performance deteriorate over time. Frequent DOB-EL calls are required to restore a good load-balance. Load-balancing is triggered whenever LI_{MAX} exceed a maximum threshold that is user-set. Additionally, adaptive mesh refinement (AMR) is performed dynamically with Mmg[35,36] based on an error estimator of the mean flow field and vorticity.[37] The AMR process produces a new unbalanced partitioning that requires a DOB-EL call as follow-up. Figure 16 shows the evolution of LI_{MAX} during a 1 h run, starting from the distribution shown in Figure 12. Average behavior has been assessed on 10 runs, LI_{MAX} is reduced by 69% and Lagrangian WCT by 33%. The cost of the load balancing represents less than 5% of the observed gain. LI_{MAX} is not as good on average as in the instantaneous case. This is because imbalance increase over time after load balancing, resulting in a higher imbalance on average. The relation between LI_{MAX} and the WCT remains like the linear regression obtained in Figure 13.

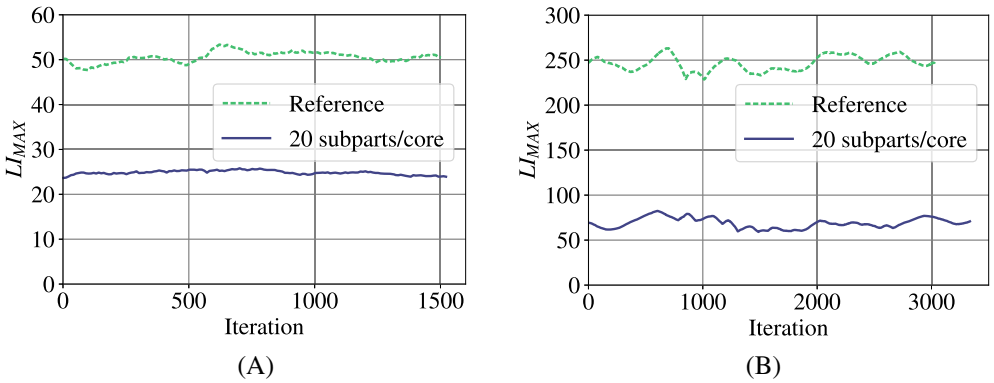


FIGURE 15 Evolution of imbalance over time with load balancing at iteration 0. (A) CRSB. (B) HERON. [Colour figure can be viewed at wileyonlinelibrary.com]

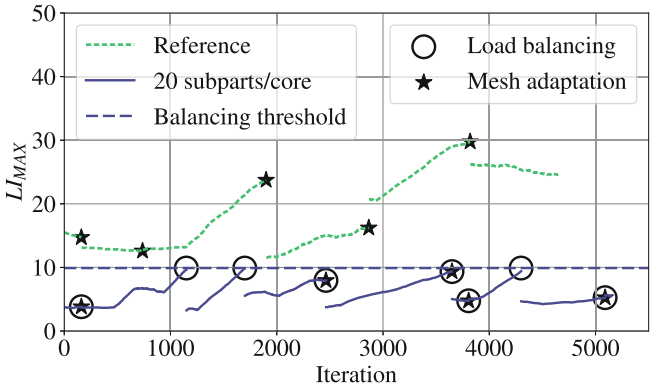


FIGURE 16 Evolution of imbalance over time, on the SALIVA case, with dynamic mesh adaptation and dynamic load balancing. [Colour figure can be viewed at wileyonlinelibrary.com]

5 | CONCLUDING REMARKS

A new load balancing method for Euler–Lagrange applications has been introduced, implemented and tested in the YALES2 code. The behavior of the algorithm has been validated on multiple simulations and major performance improvements have been obtained, the factor of improvement has turned out to be very case dependent. This load balancing algorithm is especially worth it on massive simulations and is expected to scale very well for even larger simulations. Therefore, it can be considered as a steppingstone towards the upcoming exascale simulations.

The load balancing algorithm has been applied exclusively to droplet injectors, as it is one of the most common Lagrangian cases. But other Lagrangian applications exist and could benefit from this load balancing. Soot particle formation involves a very large number of particles and should be investigated in the future.

The article has considered an Euler–Lagrange double constraint system but the load balancing method could easily be extended to another pair of constraints on a CFD mesh.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the CoEC project, Grant agreement no. 952181. This work was granted access to the HPC resources of CINES under the allocation 2020-A0092B06880 made by GENCI and to CRIANN resources under the allocation 2012006. Simon Mendez from IMAG, Montpellier, France is kindly acknowledged for providing the SALIVA case.

CONFLICT OF INTEREST STATEMENT

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 article.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Antoine Stock https://orcid.org/0000-0002-4615-0691
Ghislain Lartigue https://orcid.org/0000-0002-6504-1503
Vincent Moureau https://orcid.org/0000-0003-4574-7709

REFERENCES

1. Chamberlain BL. Graph partitioning algorithms for distributing workloads of parallel computations; 1998. http://citeseerx.ist.psu.edu/viewdoc/citations?doi=10.1.1.28.299
2. Predari M. *Load Balancing for Parallel Coupled Simulations*. Thesis. Université de Bordeaux; 2016.
3. Gourdain N, Gicquel L, Montagnac M, et al. High performance parallel computing of flows in complex geometries: I. Methods. *Comput Sci Discov*. 2009;2:015003. doi:10.1088/1749-4699/2/1/015003
4. Poinsot T, Veynante D. *Theoretical and Numerical Combustion*. R.T. Edwards Inc; 2005.
5. Kafui DK, Johnson S, Thornton C, Seville JPK. Parallelization of a Lagrangian-Eulerian DEM/CFD code for application to fluidized beds. *Powder Technol*. 2011;207:270-278. doi:10.1016/j.powtec.2010.11.008
6. Sbalzarini IF, Walther JH, Bergdorf M, Hieber SE, Kotsalis EM, Koumoutsakos P. PPM—a highly efficient parallel particle-mesh library for the simulation of continuum systems. *J Comput Phys*. 2006;215:566-588. doi:10.1016/j.jcp.2005.11.017
7. Shigeto Y, Sakai M. Parallel computing of discrete element method on multi-core processors. *Particuology*. 2011;9:398-405. doi:10.1016/j.partic.2011.04.002
8. Sitaraman H, Grout R. Balancing conflicting requirements for grid and particle decomposition in continuum-Lagrangian solvers. *Parallel Comput*. 2016;52:1-21. doi:10.1016/j.parco.2015.10.010
9. Zhang W, Myers1 A, Gott K, Almgren A, Bell J. AMReX: block-structured adaptive mesh refinement for multiphysics applications. *Int J High Perform Comput Appl*. 2021;35:508-526. doi:10.1177/10943420211022811
10. Capecelatro J, Desjardins O. An Euler-Lagrange strategy for simulating particle-laden flow. *J Comput Phys*. 2013;238:1-31. doi:10.1016/j.jcp.2012.12.015
11. Pankajakshan R, Mitchell BJ, Taylor LK. Simulation of unsteady two-phase flows using a parallel Eulerian-Lagrangian approach. *Comput Fluid*. 2011;41(1):20-26. doi:10.1016/j.compfluid.2010.09.020
12. Yakubov S, Cankurt B, Abdel-Maksoud M, Rung T. Hybrid MPI/OpenMP parallelization of an Euler-Lagrange approach to cavitation modelling. *Comput Fluid*. 2013;80:365-371. doi:10.1016/j.compfluid.2012.01.020

13. Thari A, Treleaven NCW, Staufer M, Page GJ. Parallel load balancing for combustion with spray for large-scale simulation. *J Comput Phys*. 2021;434:110-187. doi:10.1016/j.jcp.2021.110187
14. Thari A, Staufer M, Page GJ. Asynchronous task based Eulerian-Lagrangian parallel solver for combustion applications. *J Comput Phys*. 2022;458:111103. doi:10.1016/j.jcp.2022.111103
15. Karypis G, Kumar V. METIS: a software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing ordering of sparse matrices. Technical report. University of Minnesota Digital Conservancy; 1997. https://hdl.handle.net/11299/215346
16. Pellegrini F. Scotch and libScotch 5.1 user's guide; 2008. HAL Id: hal-00410327.
17. Campbell PM, Carmona EA, Walker DW. Hierarchical domain decomposition with unitary load balancing for electromagnetic particle-in-cell codes. Proceedings of the Fifth Distributed Memory Computing Conference; 1990:943–950. doi:10.1109/DMCC.1990.556303
18. Sauget M, Latu G. Dynamic load balancing for PIC code using Eulerian/Lagrangian partitioning. arXiv preprint arXiv:1706.08362, 2011.
19. Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs. *J Parallel Distrib Comput*. 1998;48:96-129. doi:10.1006/jpdc.1997.1404
20. Willebeek-LeMair MH. Strategies for dynamic load balancing on highly parallel computers. *IEEE Trans Parallel Distrib Syst*. 1993;4:979-993. doi:10.1109/71.243526
21. Subramanian R, Scherson ID. An analysis of diffusive load balancing. Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures; 1994:220–225. doi:10.1145/181014.181361
22. Cybenko G. Dynamic load balancing for distributed memory multiprocessors. *J Parallel Distrib Comput*. 1989;7:279-301. doi:10.1016/0743-7315(89)90021-X
23. Arndt H. *Load balancing auf Parallelrechnern mit Hilfe endlicher Dimension-Exchange-Verfahren*, Dissertation; 2003. https://www.yumpu.com/s/tVy7oGHZhSFgb9m1
24. Hosseini SH, Litow B, Malkawi M, Mcpherson J, Vairavan K. Analysis of a graph coloring based distributed load balancing algorithm. *J Parallel Distrib Comput*. 1990;10:160-166. doi:10.1016/0743-7315(90)90025-K
25. Xu CZ, Lau FCM. Analysis of the generalized dimension exchange method for dynamic load balancing. *J Parallel Distrib Comput*. 1992;16:385-393. doi:10.1016/0743-7315(92)90021-E
26. Moureau V, Domingo P, Vervisch L. Design of a massively parallel CFD code for complex geometries. *Comptes Rendus Méc*. 2011;339:141-148. doi:10.1016/j.crme.2010.12.001
27. Malandain M, Maheu N, Moureau V. Optimization of the deflated conjugate gradient algorithm for the solving of elliptic equations on massively parallel machines. *J Comput Phys*. 2013;238:32-47. doi:10.1016/j.jcp.2012.11.046
28. Verdier A, Marrero SJ, Vandel A, et al. Local extinction mechanisms analysis of spray jet flame using high speed diagnostics. *Combust Flame*. 2018;193:440-452. doi:10.1016/j.combustflame.2018.03.032
29. Larabi H, Lartigue G, Moureau V. LES study of an n-heptane/air turbulent spray jet flame. Proceedings of the 2018 AIAA Aerospace Sciences Meeting; 2018. doi:10.2514/6.2018-0500
30. Malbois P. *Analyse expérimentale par diagnostics lasers du mélange kérosène/air et de la combustion swirlée pauvre prémélangée, haute-pression issue d'un injecteur Low-NOx*. Thesis; 2017.
31. Salaün E. *Etude de la formation de NO dans les chambres de combustion haute pression kérosène/air par fluorescence induite par laser*. Thesis; 2017.
32. Domingo-Alvarez P, Benard P, Lartigue G, Moureau V, Grisch F. Impact of spray droplet distribution on the performances of a kerosene lean/premixed injector. *Flow Turbul Combust*. 2020;104:421-450. doi:10.1007/s10494-019-00073-5
33. Abkarian M, Mendez S, Xue N, Yang F, Stone HA. Speech can produce jet-like transport relevant to asymptomatic spreading of virus. *Proc Natl Acad Sci USA*. 2020;117(41):25237-25245. doi:10.1073/pnas.2012156117
34. Occigen supercomputer hardware. https://www.cines.fr/en/supercomputing-2/hardwares/the-supercomputer-occigen/configuration/
35. Dapogny C, Dobrzynski C, Frey P. Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. *J Comput Phys*. 2014;262:358-378. doi:10.1016/j.jcp.2014.01.005
36. Balarac G, Basile F, Benard P, et al. Tetrahedral remeshing in the context of large-scale numerical simulation and high performance computing. *Math In Action*. 2021;11:129-164.
37. Benard P, Balarac G, Moureau V, et al. Mesh adaptation for large-eddy simulations in complex geometries. *Int J Numer Methods Fluids*. 2015;81:719-740. doi:10.1002/fld.4204

How to cite this article: Stock A, Lartigue G, Moureau V. Diffusive orthogonal load balancing for Euler–Lagrange simulations. *Int J Numer Meth Fluids*. 2023;95(8):1220-1239. doi: 10.1002/fld.5191