ACADÉMIE
DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus
Mécanique

Bastien Andrieu, Bruno Maugars and Eric Quémerais

Dynamic load-balanced point location algorithm for data mapping

Volume 354 (2026), p. 53-70

Online since: 3 February 2026

https://doi.org/10.5802/crmeca.335

The Comptes Rendus. Mécanique *are a member of the Mersenne Center for open scientific publishing*
www.centre-mersenne.org — e-ISSN : 1873-7234

ACADÉMIE DES SCIENCES
INSTITUT DE FRANCE

Comptes Rendus. Mécanique
2026, Vol. 354, p. 53-70
https://doi.org/10.5802/crmeca.335

Research article

Dynamic load-balanced point location algorithm for data mapping

Bastien Andrieu [a,*], **Bruno Maugars** [b] **and Eric Quémerais** [a]

[a] ONERA/DMPE, Université Paris-Saclay, 92320 Châtillon, France

[b] ONERA/DAAA, Université Paris-Saclay, 92320 Châtillon, France

E-mails: bastien.andrieu@onera.fr, bruno.maugars@onera.fr, eric.quemerais@onera.fr

Abstract. Data mapping between geometric domains with non-matching discretizations is an essential step in multi-component numerical simulation workflows. This paper presents a novel point location algorithm, designed for transferring data from unstructured meshes to point clouds, in a massively parallel distributed environment. Special emphasis is placed on load balancing, which is paramount for making the most of computing resources and achieve optimal performance. In general, the geometric entities of interest are unevenly distributed in the input frame provided by the calling codes. The algorithm therefore aims to rapidly prune the search space using a series of parallel preconditioning techniques, while redistributing data equitably across all processes at each step. Exact point-in-cell location is then computed in an embarrassingly parallel, well-balanced frame. All data movements performed throughout the point location algorithm are transparent to the calling codes, as the resulting geometric and parallel mappings are returned in the same frame as the input data. These mappings enable data transfer via spatial interpolation and optimized process-to-process communications. A weak scaling study is carried out in three scenarios representative of the variety of real-life applications. Comparison with a state-of-the-art algorithm shows that the new algorithm performs better overall, with speed-ups of up to a factor of 10 on 4,800 CPU cores.

Keywords. High Performance Computing (HPC), Message Passing Interface (MPI), dynamic load balancing, computational geometry.

Note. Article submitted by invitation.

Manuscript received 15 January 2025, revised 5 October 2025, accepted 20 October 2025, online since 3 February 2026.

1. Introduction

In many scientific and engineering applications, numerical simulations require transferring data between arbitrarily discretized domains. Such applications include code coupling for multi-physics simulations [1,2], solution transfer following adaptive remeshing [3], and Lagrangian particle tracking [4]. Depending on the different numerical methods, these spatial discretizations usually consist of meshes or point clouds.

Data transfer from a *source* (donor) domain to a *target* (receiver) domain breaks down to two main steps. First, a mapping between the source and target degrees-of-freedom (DoFs) must be computed. When treating the target DoFs as points, the task consists in identifying which cell of the source mesh contains each of these points. Such points typically correspond to mesh

* Corresponding author

nodes or quadrature points in the Finite Element and Discontinuous Galerkin methods, or cell centers in the Finite Volume method. Second, the source-to-target mapping is used to interpolate and transfer data. The location step is the most computationally intense, but only needs to be performed once at initialization if both source and target geometries remain static during the simulation. However, data transfer between domains with dynamic discretizations requires repeated location computations, making the performance of this operation crucial. Ideally, the time needed for data mapping should be within the same order of magnitude as the time required for a single iteration of a computational code. Naively testing all possible pairs of cells and points results in quadratic complexity, which is prohibitively expensive in practical applications. It is therefore essential to devise efficient preconditioning techniques that eliminate all the irrelevant pairs.

The growing demand for high-fidelity simulations requires an ever-increasing number of degrees-of-freedom, and simulations exceeding a billion DoFs are becoming more prevalent [5]. Such simulations require so much memory and computing power that they can only be achieved on massively parallel distributed-memory systems, using parallel programming models such as the Message Passing Interface (MPI) [6]. In this context, the domains between which data is to be transferred are decomposed into partitions distributed across multiple processes.

This parallel context poses an additional challenge. The domain decompositions are tailored for the specific needs of each computational code and are therefore not optimal for the location problem. Since the source and target partitions are unlikely to align, extra communication is necessary to compute point location and exchange interpolated data. To prevent communication latency from becoming a bottleneck, the preconditioning stage must also minimize unnecessary communications. Besides, the geometric entities of interest may be poorly distributed in the decompositions provided as input to the location algorithm. For instance, when transferring data through the common surface between two adjacent volume domains, only a fraction of processes hold a portion of this surface in their partition. If no action is taken to redistribute the workload, this imbalance can severely degrade performance.

The distribution and amount of input data can vary significantly across different data mapping applications. Ensuring good performance regardless of this variability is challenging and developing a single algorithm that runs efficiently in all situations remains an open problem.

The rest of this paper is structured as follows. Section 2 gives a brief overview of existing work on data transfer for massively parallel simulations. A novel parallel point location algorithm tailored for data transfer between distributed meshes and point clouds is then described in Section 3. Finally, a performance study of this algorithm in several test cases representative of the variety of real applications is reported in Section 4, along with comparisons with another high-performance algorithm.

2. Related work

The *precise Code Interaction Coupling Environment* (preCICE) library [7] is a state-of-the-art open-source coupling library which features multiple data transfer methods. The underlying location algorithm, recently improved by Totounferoush et al. [8], can be broken down into two major steps. The first step consists in comparing the axis-aligned bounding boxes (AABBs) of the mesh partitions owned by each process, in order to establish the pairs of processes from the two coupled codes that will need to communicate. The AABBs are initially collected by a master process for each coupled code. They are then transmitted to the other master process and subsequently broadcast to all the remaining processes involved in the computation. In the second step, source mesh partitions are exchanged to the corresponding target processes using a process-to-process communication pattern. Each process then locates its own target points

within the received source mesh partitions. This is performed efficiently using an R-Tree data structure. While showing great improvement over their previous algorithm, some limitations still remain. Most of the total execution time is spent exchanging and comparing the AABBs in the first step and communicating source mesh partitions to the target processes. The load imbalance issue is also not addressed. In fact, the performance studies shown assume meshes uniformly distributed on all processes, which is unlikely in real-life surface coupling applications.

The *Finite Volume Mesh* (FVM) library [9] features location and exchange capabilities, leaving the interpolation step to the calling codes for better genericity. The location algorithm follows essentially the same two steps as preCICE, albeit with noticeable strategic differences. First, the partition AABBs are exchanged via collective communications, thus avoiding the bottleneck inherent to the master-slave approach implemented in preCICE. Second, the location computation is performed on the source side, meaning each process sends its target points to its corresponding source processes. This approach reduces the amount of communication, since smaller messages are needed to exchange points instead of mesh partitions. In order to keep the memory requirements low, blocking send/receive communications are used and the received point clouds are located one at a time. The location step is then accelerated by storing the received points in a local octree. Fournier [10] notes that this strategy can lead to excessive serialization in worst-case scenarios. To solve this problem, a technique for ordering communicating processes by recursive subdivision is implemented in the *Parallel Location and Exchange* (PLE) library, which is derived from FVM. Fournier also points out that the first coarse-grained filter based on a single AABB per process can lead to the communication of a large number of potentially irrelevant points, due to numerous false positive detections. Assuming a uniformly distributed point cloud, the number of points received by each process is correlated to the volume of its AABB. Yet this volume highly depends on the shape of the partitions and can vary considerably from one process to another, resulting in significant load imbalance. Possible optimizations are proposed to address this issue, including the use of a distributed box-tree data structure, enabling finer filtering. Once calculated by the calling code, the interpolated data is finally exchanged following the same communication pattern as in the first stage of the location algorithm. All pairs of processes which partition AABBs intersect thus communicate, even if no points from one process have effectively been located in the mesh partition held by the other. Load imbalance and serialization can therefore compromise this step as well.

The *Data Transfer Kit* (DTK) library [11] addresses the load imbalance issue by creating a secondary *rendezvous* decomposition [12], well balanced for the location problem. Points and cells outside the domain bounded by the intersection of the global source and target AABBs are first discarded. Recursive coordinate bisection is then performed on the combined source and target geometries. The MPI communicator is also recursively split along the way, leaving in the end one partition per process, each containing geometrically close source cells and target points. The splitting procedure aims to balance the combined number of source cells and target points. Some rendezvous partitions can thus end up containing virtually only source cells and no target points, or vice versa. This worst-case scenario can occur if the level of refinement of both source and target discretizations differ significantly. Geometric location in the rendezvous decomposition is accelerated using geometric binning or a local *k*d-tree. The authors also propose to perform the interpolation step in a well-balanced fashion in the same rendezvous decomposition, at the expense of additional communications required to transfer the source field data from the initial decomposition to the rendezvous decomposition.

The algorithm presented in the next section aims to remedy the shortcomings mentioned above, by developing a more refined preconditioning strategy and ensuring good load balance at each critical step.

3. Point location algorithm

3.1. *Key concepts*

Before presenting this algorithm, the following paragraphs define the terms used in this paper and outline the key concepts at the core of our point location algorithm.

3.1.1. *Point location problem*

We focus on data transfer between unstructured meshes and point clouds, with interpolation schemes using stencils restricted to the source cell containing each target point. Other types of interpolation with wider stencils (e.g. k nearest neighbors, radial basis functions, etc.) would require a different preconditioning strategy, which will be studied in future work.

Given a set S of *source* cells and a set T of *target* points, point location consists in finding for each point in T the *host* cell from S in which it lies, if any.

The two sets S and T are initially distributed on P processes that form an arbitrary MPI communicator. All subsequent communications will occur within this communicator, which is provided as an input.

3.1.2. *Frames*

Dynamic load balancing is essential to achieve high performance on distributed-memory systems, and involves redistributing data across processes throughout the algorithm. In this paper, the different data distributions are called *frames*. If E designates a set of entities, the part of E held by process p in frame $\mathscr{F}$ will be denoted by $E_p^{\mathscr{F}}$. The size of this part is denoted by $|E_p^{\mathscr{F}}|$. The *input* frame will be denoted by $\mathscr{I}$. In order to remain as generic as possible, no particular assumption is made on this frame. Some parts may contain more entities than others, some may even be empty. Some entities may also be held by multiple processes, as in the case of ghost cells or mesh vertices located on boundaries between adjacent partitions.

3.1.3. *Global identifiers*

To keep track of an entity (a source cell, its bounding box or a target point) moving across different frames, our algorithm relies on *global* identifiers (IDs). Contrary to the *local* IDs used within each partition, global IDs are *frame-independent* and *unique*, meaning that if two entities on different processes share the same global ID they are in fact two instances of the same entity. This is essential for achieving reproducible results that do not depend on how data is distributed in the input frame. Such global numbering can either be provided as an input, or generated from any ordered data set.

3.1.4. *Dynamic load balancing framework*

Dynamic load balancing is traditionally achieved using graph-based partitioning methods since they generally yield simply connected partitions with minimal edge cut. While these properties are essential for most computational codes, they are of little interest for solving the point location problem in parallel. When partition connectedness is not an issue, parallel sorting algorithms provide a more cost-effective solution for redistributing the workload. Global IDs are also used extensively for this purpose in our algorithm. Entities are re-partitioned by assigning each process an equal-sized block of entities with contiguous global IDs. This is achieved by sorting entities globally in ascending order of IDs using parallel bucket sort. If entities have associated weights, a parallel bucket sampling algorithm[1] is used in order to devise blocks of equal weight prior to sorting.

[1] An implementation on the GPU of this algorithm is presented in [13].

3.1.5. *Subsets*

Some input data might not be relevant to the location problem, e.g. cells that are guaranteed to contain no points. Our algorithm aims to isolate the *subsets* of interest by quickly pruning these irrelevant entities. Building new global numberings restricted to such subsets provides better conditioning for the sorting algorithms used for dynamic load balancing. Communication graphs are associated to each subset in order to enable data movement between the different frames. To make this possible, an explicit link between the subset and parent global numberings are maintained throughout the different steps of our algorithm. Table 1 illustrates this concept.

Table 1. Illustration of a subset (second row) and its parent set (first row) distributed on three processes (each process is represented by a color). The subset is described in two different frames: $\mathscr{F}_1$ which is balanced with respect to the parent set, and $\mathscr{F}_2$ which is balanced with respect to the subset. Frame $\mathscr{F}_2$ is obtained by redistributing the subset entities in ascending order of parent global IDs. This order is preserved in the global numbering proper to the subset.

	Frame $\mathscr{F}_1$												Frame $\mathscr{F}_2$					
Global ID in parent set	3	7	1	9	2	6	10	12	5	8	4	11	4	6	7	8	10	11
Global ID in subset	–	3	–	–	–	2	5	–	–	4	1	6	1	2	3	4	5	6

3.2. *Algorithm outline*

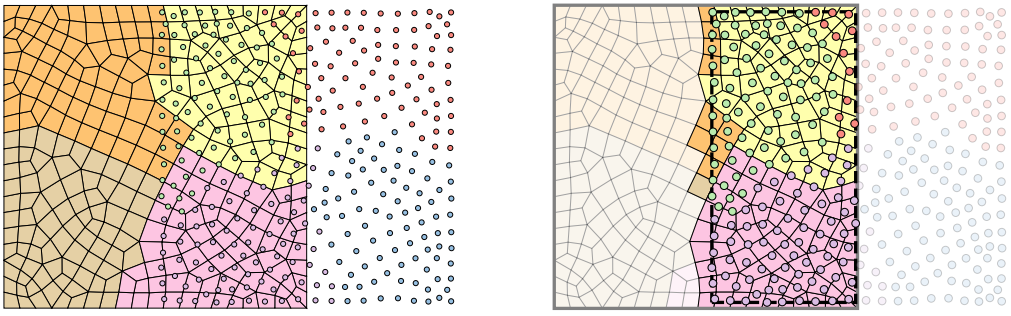
The point location algorithm presented in this paper is structured in the following five main steps. A first coarse-grained filter is followed by a second, finer preconditioning step based on fast point-in-box tests. This search for candidate cell-point pairs is accelerated using a distributed tree structure, which allows to redistribute evenly the location workload. The third step consists in computing exact point-in-cell location for all candidate pairs. Potential conflicts are then resolved in a fourth step, in order to retain at most one host cell per point. Last, the source-to-target process-to-process communication channels are established for subsequent exchanges of interpolated data.

These five steps are detailed in the next sections and illustrated with a basic, two-dimensional example.

3.3. *Coarse filtering*

The first step in our point location algorithm consists in a coarse filtering similar to the one proposed by Plimpton et al. [12]. The aim is to quickly eliminate cells and points that are clearly irrelevant to the location problem. Each process computes the AABB of the source cells in its partition. A collective reduction operation is then performed to obtain the global AABB of S, denoted by $\underline{S}$ (shown in solid gray line in Figure 1(b)). Let T' denote the subset of target points located inside $\underline{S}$. These are the only points that can possibly be located inside a source cell. The global AABB $\underline{T'}$ of T' is then computed in a similar way (shown in dashed black line in Figure 1(b)). Let S' denote the subset of cells whose AABB intersects $\underline{T'}$. These are the only cells that can possibly contain target points. Cells (resp. points) not in S' (resp. T') will no longer be considered in the remainder of the algorithm. This step proves useful when the source and target geometries overlap only partially (such as in the example of Figure 1), yet induces very little overhead if they overlap completely, as will be highlighted in Section 4.

At this stage, each process holds a (possibly unequal) part of each subset S' and T', respectively denoted by $S'^{\mathscr{I}}_p$ and $T'^{\mathscr{I}}_p$. In the example shown in Figure 1, S' is distributed over almost only two processes.



(a) Whole partitions.

(b) Subsets S' and T' in the same frame. The solid gray and dashed black rectangles depict the global AABBs of S and T', respectively.

Figure 1. Source mesh and target point cloud partitions in the input frame $\mathscr{I}$ (each color represents a process).

3.4. *Search for candidate pairs*

The search for candidate cell-point pairs relies on inexpensive point-in-box tests. However, the naive strategy that consists in checking all possible pairs local to each process becomes prohibitively expensive for large numbers of cells and points. Besides, at this stage, subsets S' and T' are arbitrarily distributed in the input frame, and their respective parts generally do not align.

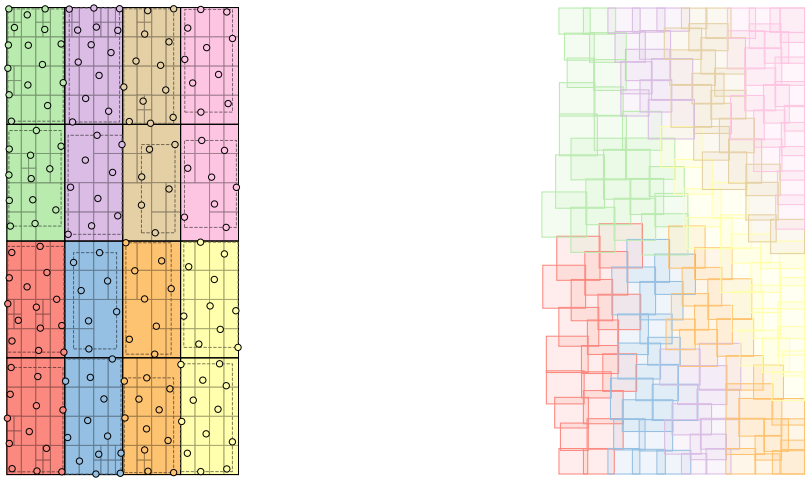
Therefore, each process must identify to which other processes each of its points or boxes should be sent. This filtering is also based on AABB comparisons. To address the issues raised in Section 2, our algorithm relies on multiple boxes per process. This more refined filter helps reducing the amount of data exchanged in this step.

Sundar et al. [14] present a method for constructing distributed linear octrees that naturally yields such boxes. This octree structure can also be leveraged to speed up the second stage of the candidate search, local to each process.

In principle, the distributed box-tree proposed by Fournier [10] is quite similar to this structure. Both are constructed following a bottom-up approach, ensuring a good load balance using re-partitioning based on the Morton space-filling curve [15]. However, whereas the box-tree requires fine-tuning of four parameters with complex combined effects, the octree proposed by Sundar is governed by just two parameters: the maximum depth of the tree and the maximum number of points contained in each of its leaf nodes.

3.4.1. *Octree construction*

The algorithm for constructing the distributed octree [14] consists in the following main steps. Target points are first sorted globally in ascending Morton order and redistributed evenly to obtain a new frame $\mathscr{O}$. In this frame, all partitions $\{T'^{\mathscr{O}}_p\}$ contain an equal amount of points. This first step is further detailed in Section 3.4.3. The points are then sorted locally and converted into octree leaves at maximum depth. Then, a minimal linear octree is constructed in parallel by filling in the gaps between successive leaves by empty octants. Finally, the octree is coarsened so as to partition the point cloud into as few coarse *blocks* as possible while maintaining good load balance. From each of these coarse blocks stems a finer sub-tree local only to the process owning that block. This algorithm ensures that the number of blocks per process is between 1



(a) Distributed quadtree and the associated target partitions in frame $\mathscr{O}$. Coarse blocks (solid black rectangles) and their effective AABBs (dashed rectangles) are shown, along with the sub-trees local to each process.

(b) AABBs of the source cells in the Morton SFC-based balanced frame $\mathscr{B}$.

Figure 2. SFC-based frames for target and source entities.

and 8. These sub-trees share a common ancestry, thus forming a complete, distributed octree. Such an octree (in fact a two-dimensional quadtree) is represented in Figure 2(a).

3.4.2. *Octree query*

Once octree construction is complete, our algorithm proceeds as follows. The coarse blocks are gathered on all processes so that each process can determine to which other processes it should send the source AABBs it owns. A given source AABB is sent to a process if it intersects at least one of the coarse blocks owned by this process. In order to further reduce the amount of false positive detections, each block is reduced to the AABB of the points it contains. Such AABBs are shown as dashed rectangles in Figure 2(a).

As the number of processes increases, the memory and CPU cost of storing and querying these blocks can become problematic. CPU overhead can be amortized by storing the blocks in a tree structure, thus enabling queries in $O\big(|S'_p|\log(P)\big)$ time. This operation is trivial, since the blocks are natively obtained from a coarse octree. The shared coarse tree associated to the example in Figure 2(a) is represented in Figure 3. Besides, a single instance of this coarse tree can be stored on each compute node, taking advantage of shared memory. On typical supercomputers with tens of cores per compute node, the memory footprint of the coarse tree can thus be reduced by at least an order of magnitude.

Querying the coarse tree thus provides a connection between each source AABB and its processes of interest. This connection is used to exchange the boxes through collective MPI communications. As a result, each process receives a partition $S'^{\mathscr{O}}_p$ of source AABBs in the octree frame $\mathscr{O}$. The part of the octree local to this process is then inspected in order to find candidate points for each of these boxes.

Despite the acceleration provided by the tree structure, querying the coarse blocks can become a critical step if the partitions $S'^{\mathscr{I}}_p$ are unbalanced. This is typically the case in the example

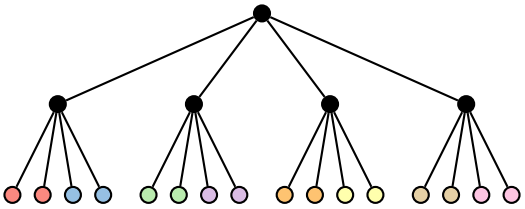


Figure 3. Schematic view of the shared coarse tree. Each leaf corresponds to a coarse block, colored by its owner process.

shown in Figure 1(b). Moreover, an arbitrary distribution of boxes can result in a highly dense communication graph, significantly increasing the latency of collective communications.

To address these issues, the source AABBs are redistributed before querying the shared coarse tree. A good heuristic is to apply the same sorting and redistribution as performed on the points in the first step of octree construction. The new, balanced frame thus obtained is denoted by $\mathscr{B}$. As shown in Figure 2(b), the source partitions $\{S_p'^{\mathscr{B}}\}$ are mostly aligned with the target partitions $\{T_p'^{\mathscr{O}}\}$. The next paragraph briefly describes how this re-partitioning is achieved.

3.4.3. *SFC-based re-partitioning*

Given an arbitrarily distributed set of geometric entities (i.e. points or boxes), the goal is to find a new, well-balanced distribution that maximizes data locality. Space filling curves (SFC) have been used extensively for this purpose [16,17].

SFCs map continuously points in three-dimensional space to one-dimensional space. These curves have inherent multi-resolution properties due to their iterative construction, making them attractive for constructing linear tree structures. Setting a maximum resolution reduces space to a finite number of voxels. The cartesian coordinates of a point can then be encoded as a nonnegative integer, corresponding to its voxel index, dictated by the traversal order of the SFC. The Morton SFC (or "Z-order curve") [15] provides a straightforward encoding that can be very efficiently computed using bit shifts. The Morton ordering is equivalent to that obtained by traversing an octree depth first.

Subsets S' and T' are re-partitioned independently, but in the same way. The strategy described in Section 3.1.4 is followed, except that Morton codes are used instead of arbitrary global IDs. These Morton codes are computed by encoding either the target point coordinates or the coordinates of the source AABB centers. The subset global IDs thus obtained correspond to the order in the globally sorted array of Morton codes.

The re-partitioning induces a new balanced frame for each subset S' and T', to which data must be communicated from the input frame. The coordinates of the target points are communicated, along with only the AABBs of the source cells. The search for candidate pairs indeed requires no additional information such as mesh connectivity or vertex coordinates. To obtain the final source-to-target mapping in the input frame, as discussed in Section 3.7, it is essential to maintain an explicit link between subset entities and their corresponding parent sets. The global IDs of entities in the parent sets S and T are therefore communicated to the new frames as well.

Since S' and T' are re-partitioned independently from each other, the exchange of point data can be overlapped by computations for finding the balanced distribution of source cells, using non-blocking collective communications. Moreover, the exchange of cell data can be overlapped by the construction of the distributed octree.

3.5. *Exact point-in-cell location*

Once the octree query is complete, a connection is obtained between the source cells and their candidate points. This connection is represented in the form of a graph that is distributed in the octree frame $\mathscr{O}$.

The next step consists in computing the exact location of these candidate points for each cell. Weights for subsequent interpolation of source data at target points must also be calculated. Applications based on the Finite Element method usually rely on shape function interpolation. To this end, the isoparametric coordinates of each candidate point are determined using Newton–Raphson iteration. When dealing with ill-conditioned shape functions (e.g. highly curved high-order elements), this iteration may fail to converge. In this case a second, more robust technique is used, which consists in recursively subdividing the cell after a first decomposition into simplices. When dealing with meshes composed of arbitrary polygonal or polyhedral cells, a first inclusion test determines whether each candidate point lies inside or outside the cells. In the first case, mean value coordinates [18,19] are calculated to serve as interpolation weights. In the second case, the nearest projection onto the cell's faces is computed, along with the distance from this projection.

These exact location computations could be carried out in the frame $\mathscr{O}$. However, this frame is not guaranteed to be well balanced. Some processes may indeed hold more cells than others, or cells with more candidate points. Load balance in this stage is critical since computing exact location is expensive. In addition, these computations require a full description of the source cells, i.e. mesh connectivity along with vertex coordinates, whereas only AABBs are available at this point in frame $\mathscr{O}$. This additional information is only available in the input frame $\mathscr{I}$.

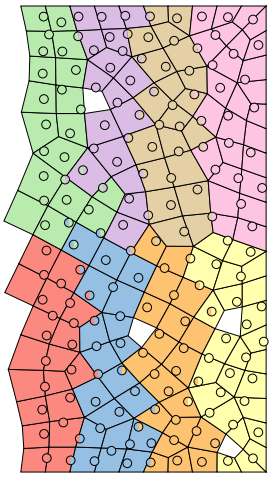
The choice is thus made to create a new, better balanced rendezvous frame in which exact location will be computed. In order to keep data movements to a minimum, the rendezvous frame is created such that each cell is owned by a single process. Points may in turn be duplicated on multiple processes. The number of candidate points in the AABB of each cell gives a simple yet effective estimation of the workload associated to that cell. This way, cells with no candidate points can simply be discarded. The different cell types could also be taken in consideration to further refine this estimate.

This rendezvous frame is obtained using the approach described in Section 3.1.4 by sorting cells in ascending order of the subset global IDs derived from the SFC-based re-partitioning. The data locality provided by this global numbering reduces duplication of mesh entities (e.g. faces, vertices) since adjacent cells are likely to be owned by the same process in the rendezvous frame, as indicated in Figure 4(a).

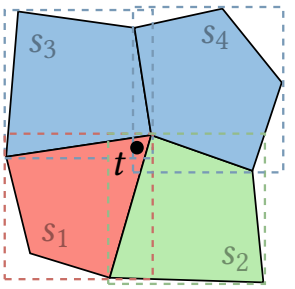
3.6. *Conflicts resolution*

Each target point may be contained in the AABB of multiple source cells. Besides, these cells may reside on different processes in the rendezvous frame, and therefore so do the target points. In the end, each point must be mapped to at most one host cell. Possible conflicts are resolved by filtering the location data computed in the previous step in order for each point to keep only the best match among its associated candidate cells. This filtering is performed in two stages. First, the candidates local to each process are sorted in ascending order of signed distance.[2] The nearest candidate in that process is then kept, while the others are discarded. The second stage is similar, except that this time the sorting is carried out on the best candidates identified by all the processes in the previous stage. If a point is located exactly between two cells, the conflict is resolved by selecting the cell with lower global ID. This way, the location algorithm is guaranteed

[2] A negative distance indicates that the point is contained inside the cell.



(a) Partitions of candidate cells and points in the rendezvous frame $\mathscr{R}$. Points duplicated on multiple processes and are shown in multiple colors.



(b) Location conflict for a point with multiple candidate cells (colored by owning process in frame $\mathscr{R}$).

Figure 4. Exact point-in-cell location steps.

to yield reproducible results, that do not depend on the number of processes. In order to balance the workload, this second stage is performed in a new frame $\mathscr{C}$.

The approach described in Section 3.1.4 is used once again to devise such a balanced frame. Distance values of the cells that pass the first filter are exchanged from the rendezvous frame $\mathscr{R}$ to the conflict-resolution frame $\mathscr{C}$. The result of the second filter is then sent back to frame $\mathscr{R}$ so that each cell can discard candidate points for which it is not the best match.

In the situation depicted in Figure 4(b), a target point t is associated to four candidate cells, distributed on three processes in frame $\mathscr{R}$ (indicated by colors). On the blue process, the first filter retains the nearest cell, namely s_3. Distances from s_1, s_2 and s_3 are then communicated to the process holding t in frame $\mathscr{C}$. The second filter finally retains s_1.

3.7. *Return to the input frame*

At this stage, all false positive detections have been eliminated. The remaining cell-point pairs form the actual mapping used for transferring data between the source and target domains. However, this data is usually distributed in the same frame as the input source and target partitions, namely frame $\mathscr{I}$. This data could be first communicated from frame $\mathscr{I}$ to the rendezvous frame $\mathscr{R}$, in which interpolation would be performed, as suggested by Plimpton et al. [12]. Nevertheless, frame $\mathscr{R}$ may no longer be well balanced since the connection between cells and points has just been pruned. Besides, additional application-specific information may be required for spatial interpolation, and would need to be communicated to frame $\mathscr{R}$ as well. Experience shows that the interpolation step is about two orders of magnitude less expensive than the location step. Possible load imbalance due to poor distribution in the input frame therefore does not compromise performance. Sticking to frame $\mathscr{I}$ for interpolation is thus a sensible choice. The source-to-target mapping is thus transferred from the rendezvous frame to the input frame.

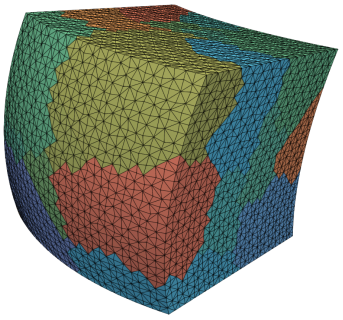
A process-to-process communication pattern is finally established between the source and target partitions in this input frame, to enable subsequent exchanges of interpolated data.

4. Performance results

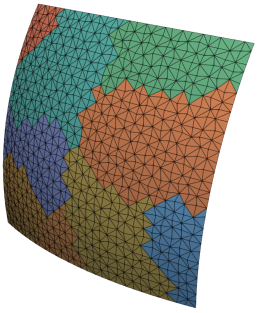
The point location algorithm presented in this paper has been implemented in the open-source CWIPI coupling library [20,21], along with high-level wrappings to MPI primitives to carry out the transfer of interpolated data using non-blocking send/receive communications, based on the previously described communication pattern. CWIPI formerly relied on the FVM library for data mapping, using the algorithm mentioned in Section 2. The tests presented in this section were carried out in version 1.1.0 of CWIPI [21] which incorporates both algorithms, thereby enabling a direct comparison.

4.1. *Test cases description*

The following setup is considered to study and compare the performance of the two point location algorithms. Starting from a cartesian grid, an unstructured source mesh is obtained by decomposing the hexahedral cells into tetrahedra. To break the alignment with the cartesian axes, the geometry is deformed and a slight random perturbation is applied to the vertex coordinates, as shown in Figure 5.



(a) Volume source mesh used in the full and partial volume overlap scenarios.



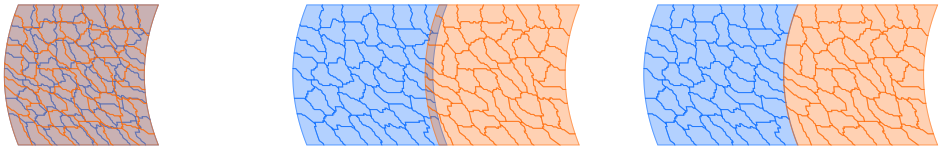
(b) Surface mesh extraction used in the surface overlap scenario.

Figure 5. Meshes used in the tests (colors indicate the different partitions).

A second mesh is generated using the same procedure, and its cell centers are employed as the target point cloud. Grids with slightly different numbers of points are used so that the two meshes do not match exactly. The source and target meshes are partitioned using the PT-Scotch library [22] on two separate MPI communicators, each with the same number of processes. The communicators are then merged by CWIPI into a single COMM_WORLD communicator. Since the partitioning method used is not deterministic, the sensitivity of both algorithms to input distributions is examined by executing each test case multiple times. A weak scaling study is carried out, with roughly $2.5 \cdot 10^5$ cells (resp. points) in each source (resp. target) partition. First, a source-to-target mapping is constructed using the point location algorithms. Then, a field evaluated on the source mesh is interpolated and transferred to the target points using either the communication scheme proposed by FVM or the one described in the end of Section 3.7. Interpolation here consists simply in assigning each target point the field value of its host cell, as the study focuses primarily on the performance of the parallel algorithm (the

use of a more refined scheme such as shape-function interpolation would have no impact on overall performance, since the interpolation weights are calculated regardless). The code for reproducing the test cases (as well as the input meshes, which are automatically generated at runtime) is available online (see Section 5 for the details).

Different scenarios representative of the wide range of real-life applications are realized by offsetting the meshes relative to each other in the x-direction, as represented in Figure 6. For each of the three configurations considered, execution times for the location and interpolation steps are reported for both algorithms. Minimum, mean and maximum times over the multiple executions of each test are reported. Finally, a detailed breakdown of the location time is presented for the new algorithm.



(a) Full volume overlap. **(b)** Partial volume overlap. **(c)** Surface overlap.

Figure 6. Schematic view of the three scenarios. The partitioned source and target domains are shown in blue and orange, respectively.

4.2. *Hardware description*

All the tests presented in this paper were carried out on the same supercomputer. Each of its compute nodes is composed of two 2.4 GHz Intel Xeon 6240R (Cascade Lake) processors, each one containing 24 cores, totalling 48 cores per node. The nodes are interconnected by an Intel OmniPath 100 GB/s low-latency network. All experiments were run using one MPI rank per core.

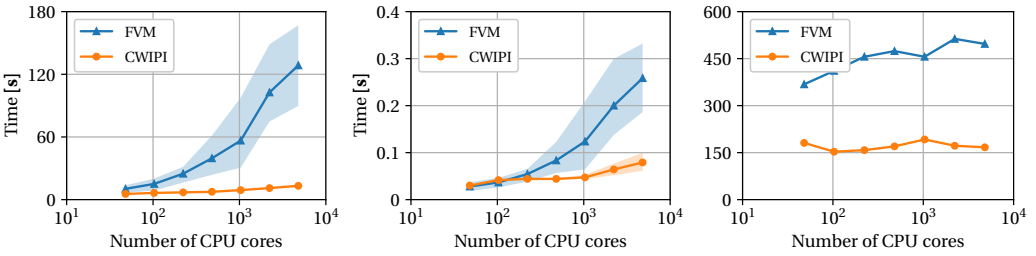
4.3. *Performance analysis*

4.3.1. *Full volume overlap*

In the first test case, the source and target geometries overlap completely. This scenario arises in situations such as solution transfer before restarting a simulation on a different mesh. In this case, the coarse filtering step described in Section 3.3 reveals useless, since the global source and target AABBs are essentially identical. However, it induces very little overhead since it accounts for less than 1 % of the total execution time. This step is therefore omitted in the breakdown given in Figure 8.

Although the source mesh is evenly distributed, the volume of the partition AABBs can vary considerably between processes. In the worst case, a volume ratio of 300 is observed between the largest and smallest AABBs. The number of points to be located by each process in the FVM algorithm is equally unbalanced, as anticipated in Section 2. Furthermore, Figure 7 shows that the execution time of the FVM algorithm varies significantly between different runs, indicating a high sensitivity to the input partitioning. In contrast, the new algorithm delivers the same execution time regardless of the partitioning, and exhibits greater parallel efficiency. This improvement is explained by the finer preconditioning (Section 3.4) and the redistribution of the location workload (Section 3.5). A maximum load imbalance of 10 % is observed in the rendezvous frame.

Figure 7(b) demonstrates the benefits of optimizing the communication graph used to exchange the interpolated data. The strategy described in Section 3.7 proves reasonable, considering the small proportion of execution time devoted to the "return to input frame" step, as shown in Figure 8. Figure 7(c) also confirms that the interpolation step is significantly cheaper than point location (by about two orders of magnitude).



(a) Point location.

(b) Interpolation and exchange.

(c) Ratio point location / interpolation and exchange.

Figure 7. Execution times for the FVM and CWIPI algorithms (full volume overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

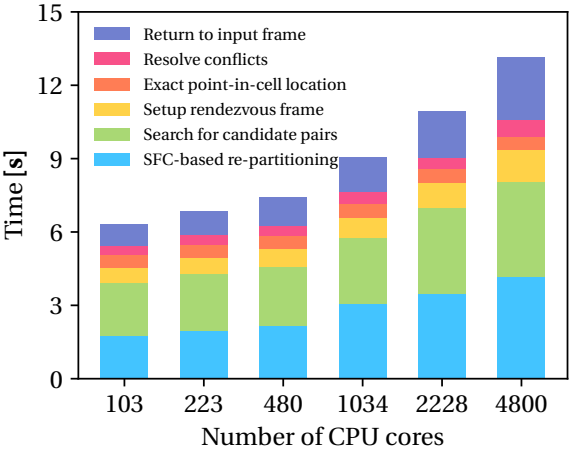


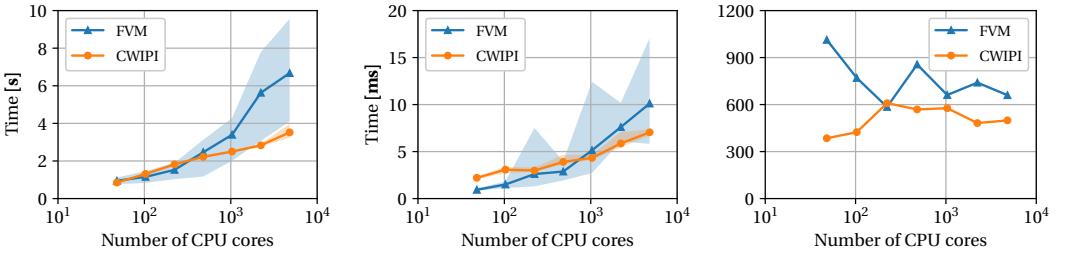
Figure 8. Breakdown of average execution time for the new point location algorithm (full volume overlap).

In applications with dynamic geometries, point location must be performed frequently. Higher frequency generally translates into greater fidelity and numerical robustness. The CPU cost of data mapping should ideally be comparable to that of one iteration of a computational code, which is on the order of 1 µs per cell for state-of-the art computational fluid dynamics solvers. In comparison, the CPU cost of the new point location algorithm ranges from 20 µs to 50 µs per target point in this test. Dynamic simulations with large meshes thus remain challenging, and optimizations are necessary to overcome the data mapping bottleneck. As indicated in Figure 8, these optimizations should focus primarily on the preconditioning stage.

4.3.2. *Partial volume overlap*

In the second test case, the target point cloud is shifted so that it overlaps only a thin layer of the source mesh (a few cells thick). This scenario mimics applications such as the multi-component simulation presented in [23], in which data mapping is used to integrate the different moving parts of a full aircraft engine into a single unsteady large-eddy simulation.

Full source and target geometries are provided to the location algorithms, letting them filter out irrelevant points and cells. Figure 9 shows that variations in the input partitioning have a strong impact on the performance of the FVM algorithm, as in the first test case. Besides, most processes are idle during the location step, since the AABB of their source mesh partition intersect no target partition AABB. This load imbalance results in suboptimal parallel efficiency.



(a) Point location. **(b)** Interpolation and exchange. **(c)** Ratio point location / interpolation and exchange.

Figure 9. Execution times for the FVM and CWIPI algorithms (partial volume overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

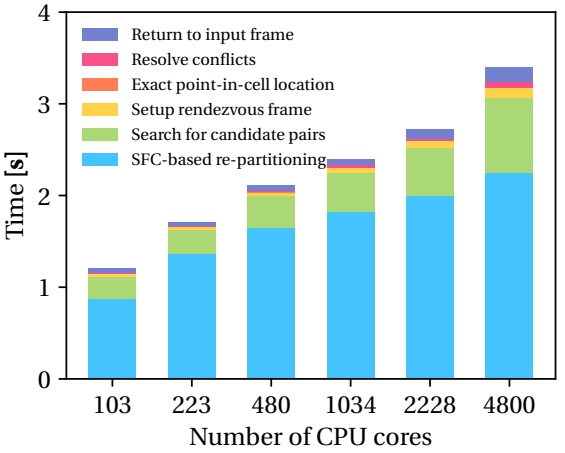


Figure 10. Breakdown of average execution time for the new point location algorithm (partial volume overlap).

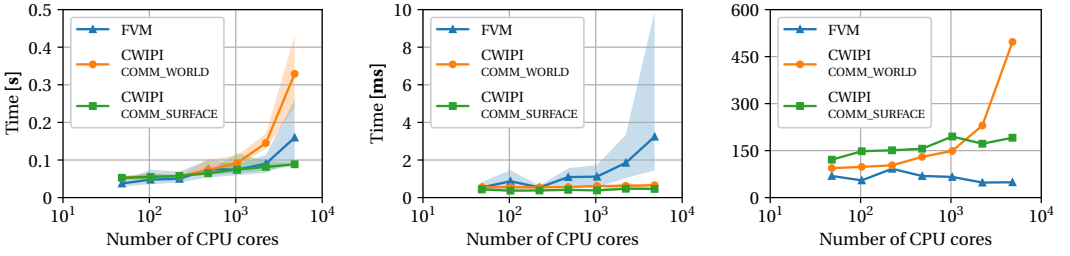
The new algorithm effectively eliminates up to 95 % of the source cells and target points in the first coarse filtering step, once again with minimal extra cost. Dynamic load balancing proves less effective than in the first test case, since the average workload per process is considerably reduced. In fact, the SFC-based re-partitioning step accounts for about 70 % of the total execution time, as reported in Figure 10. A more detailed analysis reveals that the bucket sampling algorithm is responsible for 60 % of the cost of this step. This algorithm relies on parameters

tuned using heuristics, which could be further optimized for our application. The communication graph used by the FVM algorithm to transfer the interpolated data is much sparser than in the first case. Consequently, both algorithms yield comparable execution times for this step. Still, the new point location algorithm performs better on average and remains much less sensitive to the input data distribution. In this second test, the location step takes around 10 µs per target point[3] with the new algorithm, which is still quite expensive compared to an iteration of fluid simulation.

4.3.3. *Surface overlap*

In the third test case, the target point cloud is shifted so that it overlaps only one boundary face of the source domain. This scenario is representative of surface couplings such as simulation of fluid-structure interaction [1]. In such applications, data transfer is typically used to apply specific boundary conditions. Only the surface geometric entities and their associated DoFs are therefore considered.

As in the other two test cases, volume meshes are first generated and partitioned. The surface of interest is then extracted without subsequent redistribution, as shown in Figure 5(b). Consequently, the source cells and target points are unevenly distributed in this input frame, since the surface is contained in only a fraction of the volume partitions. This fraction diminishes as the number of processes increases. Nevertheless, the deterioration in load imbalance is tempered by a decrease in the average number of cells and points per partition, which scales as $O(P^{-1/3})$. In this situation, communication latency becomes predominant. The new algorithm carries out numerous data movements and is therefore penalized.



(a) Point location. **(b)** Interpolation and exchange. **(c)** Ratio point location / interpolation and exchange.

Figure 11. Execution times for the old and new algorithms (surface overlap). Average times over all runs are represented by solid lines, while shaded areas illustrate the range of elapsed times across all runs.

A solution to improve parallel efficiency consists in using fewer processes so that the increased average workload compensates for the communication overhead. This strategy is tested by splitting the initial MPI communicator (here COMM_WORLD), to obtain the sub-communicator COMM_SURFACE restricted to processes holding a non-empty share of the surface in the input frame. Figures 11 and 12 demonstrate the effectiveness of this strategy. In multi-component simulations such as in [23], multiple simultaneous data transfers need to be carried out, in relatively small, distinct geometric regions. Assigning one sub-communicator to each data transfer task would allow for a more effective utilization of computational resources. Yet, this approach could be refined by determining an optimal communicator size based on an assessment of the total workload. Splitting the working communicator after the coarse filtering step could enable even further improvement. However, if data transfer is considered as part of a multi-component

[3] All the target points are considered, including the ones not contained in any cell.

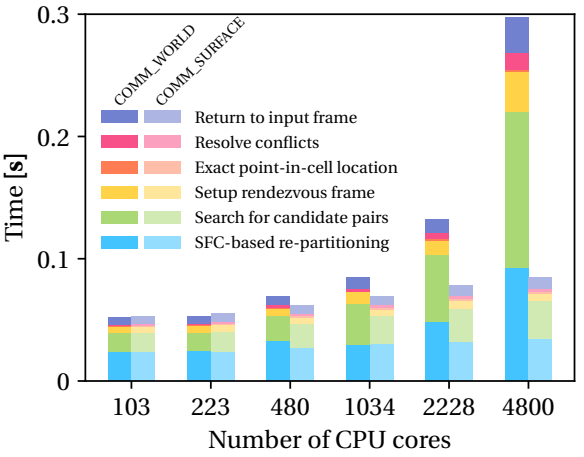


Figure 12. Breakdown of average execution time for the new point location algorithm (surface overlap), using either COMM_WORLD or COMM_SURFACE as the input MPI communicator.

simulation, performance remains acceptable (on the order of 1 µs per target point)[4] and the proposed optimizations become less significant.

5. Conclusion

In this paper a novel point location algorithm has been presented, designed for data mapping between distributed meshes and point clouds in a massively parallel environment. Comparisons with a state-of-the-art algorithm are favorable, with a speed-up of up to a factor of 10. This achievement is made possible by a more refined preconditioning strategy combined with dynamic load balancing.

Good scaling is observed for up to 1.2 billion cells and points on 4,800 CPU cores, proving that our algorithm can be integrated in a wide range of real-life, large-scale data transfer applications. In order to prepare for the exascale era, the performance analysis still needs to be carried out on larger supercomputers. For instance, the first and last step of the algorithm (see Section 3.4.3 and 3.7 respectively) rely heavily on collective (all-to-all) MPI communications which could become a bottleneck with a higher number of MPI ranks.

Work is also underway to harness the full potential of heterogeneous architectures and accelerate the preconditioning stage through CPU-GPGPU hybridization [13]. The expected speed-up should help overcome the current bottleneck experienced in dynamic simulations. Finally, the possible optimizations proposed in this paper to mitigate the communication overhead will also be explored in future work.

Acknowledgments

The manuscript was written through contributions of all authors.

[4] The whole *volume* mesh is considered.

Underlying data

The tests presented in Section 4 can be reproduced by running the Python script `python_perfo_location_octree.py` available in the `tests/` directory after cloning CWIPI's GitHub repository: https://github.com/onera/cwipi/blob/master/tests/python_perfo_location_octree.py.

Declaration of interests

The authors do not work for, advise, own shares in, or receive funds from any organization that could benefit from this article, and have declared no affiliations other than their research organizations.

References

[1] T. Fabbri, G. Balarac, V. Moureau and P. Benard, "Design of a high fidelity Fluid–Structure Interaction solver using LES on unstructured grid", *Comput. Fluids* **265** (2023), article no. 105963.

[2] G. Coria, J.-D. Parisse, J.-M. Lamet and N. Dellinger, "Modeling and simulation of chemical reactions at the surface of an ablative wall interacting with a hypersonic flow", 2022. Conference paper: 9th European Conference for Aeronautics and Aerospace Sciences (EUCASS-3AF).

[3] F. Alauzet, "A parallel matrix-free conservative solution interpolation on unstructured tetrahedral meshes", *Comput. Methods Appl. Mech. Eng.* **299** (2016), pp. 116–142.

[4] A. Palha, L. Manickathan, C. S. Ferreira and G. van Bussel, "A hybrid Eulerian-Lagrangian flow solver", 2015. Online at https://arxiv.org/abs/1505.03368.

[5] A. W. Cary, J. Chawner, E. P. Duque, W. Gropp, W. L. Kleb, R. M. Kolonay, E. Nielsen and B. Smith, "CFD Vision 2030 Road Map: Progress and Perspectives", in *AIAA AVIATION 2021 FORUM*, American Institute of Aeronautics and Astronautics, 2021.

[6] The MPI Forum, "MPI: a message passing interface", in *Proceedings of the 1993 ACM/IEEE conference on Supercomputing* (B. Borchers and D. Crawford, eds.), ACM Press, 1993, pp. 878–883.

[7] G. Chourdakis, K. Davis, B. Rodenberg, et al., "preCICE v2: A sustainable and user-friendly coupling library", *Open Res. Eur.* **2** (2022), article no. 51 (47 pages).

[8] A. Totounferoush, F. Simonis, B. Uekermann and M. Schulte, "Efficient and scalable initialization of partitioned coupled simulations with preCICE", *Algorithms* **14** (2021), no. 6, article no. 166 (17 pages).

[9] Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. G. Sunderland and J. C. Uribe, "Optimizing Code_Saturne computations on Petascale systems", *Comput. Fluids* **45** (2011), no. 1, pp. 103–108.

[10] Y. Fournier, "Massively parallel location and exchange tools for unstructured meshes", *Int. J. Comput. Fluid Dyn.* **34** (2020), no. 7–8, pp. 549–568.

[11] S. Slattery, P. Wilson and R. Pawlowski, "The Data Transfer Kit: A geometric rendezvous-based tool for multiphysics data transfer", in *International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering (M&C 2013)*, 2013, pp. 5–9.

[12] S. J. Plimpton, B. Hendrickson and J. R. Stewart, "A parallel rendezvous algorithm for interpolation between multiple grids", *J. Parallel Distrib. Comput.* **64** (2004), no. 2, pp. 266–276.

[13] R. Cazalbou, F. Duchaine, E. Quémerais, B. Andrieu, G. Staffelbach and B. Maugars, "Hybrid Multi-GPU Distributed Octrees Construction for Massively Parallel Code Coupling Applications", in *PASC '24 — Proceedings of the Platform for Advanced Scientific Computing Conference*, ACM Press: Zurich, Switzerland, 2024.

[14] H. Sundar, R. S. Sampath and G. Biros, "Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel", *SIAM J. Sci. Comput.* **30** (2008), no. 5, pp. 2675–2708.

[15] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*, IBM, 1966.

[16] S. Aluru and F. E. Sevilgen, "Parallel domain decomposition and load balancing using space-filling curves", in *Proceedings 4th International Conference on High-Performance Computing*, IEEE, 1997, pp. 230–235.

[17] R. Borrell, J. C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez and G. Houzeaux, "Parallel mesh partitioning based on space filling curves", *Comput. Fluids* **173** (2018), pp. 264–272.

[18] K. Hormann and M. S. Floater, "Mean value coordinates for arbitrary planar polygons", *ACM Trans. Graph.* **25** (2006), no. 4, pp. 1424–1441.

[19] T. Ju, S. Schaefer and J. Warren, "Mean Value Coordinates for Closed Triangular Meshes", *ACM Trans. Graph.* **24** (2005), no. 3, pp. 561–566.

[20] E. Quémerais, "La Bibliothèque de Couplage CWIPI — Coupling With Interpolation Parallel Interface", in *ONERA, le centre français de recherche aérospatiale*, 2013. Online at https://w3.onera.fr/cwipi/bibliotheque-couplage-cwipi (accessed on November 27, 2025).

[21] ONERA, *onera/cwipi: Library for coupling parallel scientific codes via MPI communications to perform multi-physics simulations*, 1.1.0. Online at https://github.com/onera/cwipi/tree/cwipi-1.1.0 (accessed on November 27, 2025).

[22] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering", *Parallel Comput.* **34** (2008), no. 6–8, pp. 318–331.

[23] C. P. Arroyo, J. Dombard, F. Duchaine, L. Gicquel, B. Martin, N. Odier and G. Staffelbach, "Towards the Large-Eddy Simulation of a Full Engine: Integration of a 360 Azimuthal Degrees Fan, Compressor and Combustion Chamber. Part I: Methodology and Initialisation", *J. Global Power Propul. Soc.* **2021** (2021), pp. 1–16.