Journal of Computational Physics 376 (2019) 585–596

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A massively-parallel, unstructured overset method for mesh connectivity

Wyatt James Horne, Krishnan Mahesh [*]

University of Minnesota, Department of Aerospace Engineering and Mechanics, Minneapolis 55414, USA

A R T I C L E   I N F O

Article history:
Received 4 June 2018
Accepted 29 September 2018
Available online 4 October 2018

Keywords:
Overset
Unstructured
O(hundred thousand) meshes
Connectivity
Dynamic
Parallel

A B S T R A C T

We present a method that dynamically and efficiently performs connectivity calculations between many ($O(10^5)$) moving, unstructured overset meshes in parallel. In order to connect overset meshes, elements exterior to the solution domain must be removed from the simulation. In regions with many overlapping meshes, elements must be selectively removed to reduce redundancy while maintaining a solution over the entire domain. Around masked regions interpolation partner pairing is required between meshes to provide boundary conditions. For general unstructured meshes, these steps involve challenging computational geometry calculations which must be efficient and automatic. For many moving meshes each step must be massively parallelized and scalable to large numbers of computational cores. To establish communication patterns a parallelized master/slave algorithm is used which minimizes global communication and storage. To remove elements a parallel 'Forest Fire' flood-fill algorithm is used to set a masking variable. For interpolation partner pairing, and other necessary searches, k-dimensional tree data structures (k-d trees) are extensively used. Often in a calculation, the connectivity between overset meshes remains largely the same between time steps. The temporal coherence of the various objects in the connectivity calculation is directly used to only update necessary information with time, resulting in substantial cost savings. Details of the different algorithms are presented. Resulting connectivity and timings are shown for complex geometries. Parallel scaling is demonstrated for 100,000 spherical particles within a channel up to 492,000 processors.

1. Introduction

Performing computational simulations of many, moving bodies in a fluid presents significant challenges. Moving boundary conditions must be accurately imposed for the solution of the fluid flow. The motion of boundaries is generally not known *a. priori.*, and hence the methods must be dynamic in time for general motion. For a mesh based method with a single, body-fitted mesh this requires re-meshing a domain dynamically as bodies move. This procedure can be prohibitively expensive when many bodies are present. Boundary conditions can also be imposed in a non-conforming manner on the mesh as is the case for Immersed Boundary Methods (IBM) [1–6].

In IBM a force is added to the governing equation to impose non-conforming boundary conditions. This type of method is relatively efficient and is generally easier to implement compared to re-meshing. The method has been used successfully

[*] Corresponding author.
E-mail address: kmahesh@umn.edu (K. Mahesh).

https://doi.org/10.1016/j.jcp.2018.09.053
0021-9991/© 2018 Elsevier Inc. All rights reserved.

to study problems with many moving bodies such as particle-resolved direct-numerical simulations (PR-DNS). Challenging problems such as surface-resolved sediment bed erosion with many particles have been successfully investigated [7]. One disadvantage of IBM is the necessity of high resolution in the vicinity of moving boundaries, especially for high Reynolds number (Re) [6,8]. High mesh resolution must be placed in locations near moving boundaries, which is often unknown, or the mesh must be adaptively refined near the boundaries of bodies which has similar disadvantages to a re-meshing method. Additionally the calculation of forces and other quantities on the moving bodies requires a reconstruction operation where flow field quantities must be accurately recreated along the surface of bodies. This reconstruction is often conducted at Lagrangian points along surfaces. The placement of these points along the surface is a non-trivial selection where the nearby, possibly non-uniform, mesh resolution must be reflected in the placement of the points. In simulations of PR-DNS several moving surfaces can be present within a single element depending on the resolution of the fixed mesh. This can create a non-trivial constraint on the fluid flow such that none of the moving boundary conditions within such elements can be imposed accurately.

If multiple meshes are used, it is possible to use meshes which conform to moving bodies while providing adequate resolution near surfaces. Such is the case in overset methods where meshes are directly attached to surfaces and allowed to overlap arbitrarily [9,10]. A key challenge of an overset method is connecting the solutions between meshes robustly and efficiently throughout a domain. Elements within meshes can be located exterior to the solution domain and must be removed from the simulation. Additionally, there may be regions in the domain where many meshes overlap and redundant solutions exist. For best efficiency, elements within these regions must be selectively removed to reduce the redundancy while still covering the entire domain. Boundaries remain at the edges of overset meshes and the regions of removed elements, which must be supplied boundary conditions interpolated from meshes. This requires finding elements which overlap these boundaries. Each of these steps must be performed dynamically as the meshes move.

Methods have been developed and software packages are available which have been shown to effectively establish overset connectivity for moving body problems [11–13]. In general the available packages are not suitable to simulate large-scale problems on large numbers of meshes. This is necessary for many-body problems where the number of unknowns in a simulation can be $O(10^9)$ and the number of moving meshes $O(10^5–10^6)$ as would be the case for PR-DNS of particle-laden turbulent channel flows using an overset method. Overset methods have not been applied to simulating large numbers of moving bodies. The main reason is the inherent difficulty in establishing connectivity when many meshes are present.

Establishing communication patterns and geometrical connections between meshes at large scales presents unique challenges. Communication patterns between partitioned meshes must be efficiently established dynamically as meshes move while minimizing global communication and storage, which could be crippling to a simulation given the large amounts of data and computational cores present. Collective communication is necessary within a mesh, and between meshes, requiring the creation of collective communication patterns which must be efficient and robust. Finding geometrical overlap between the different objects within meshes must be localized and algorithmically efficient even if many meshes are overlapping and geometries are complex. Updating connectivity as meshes move must be efficient even with large-scale motion that crosses many meshes.

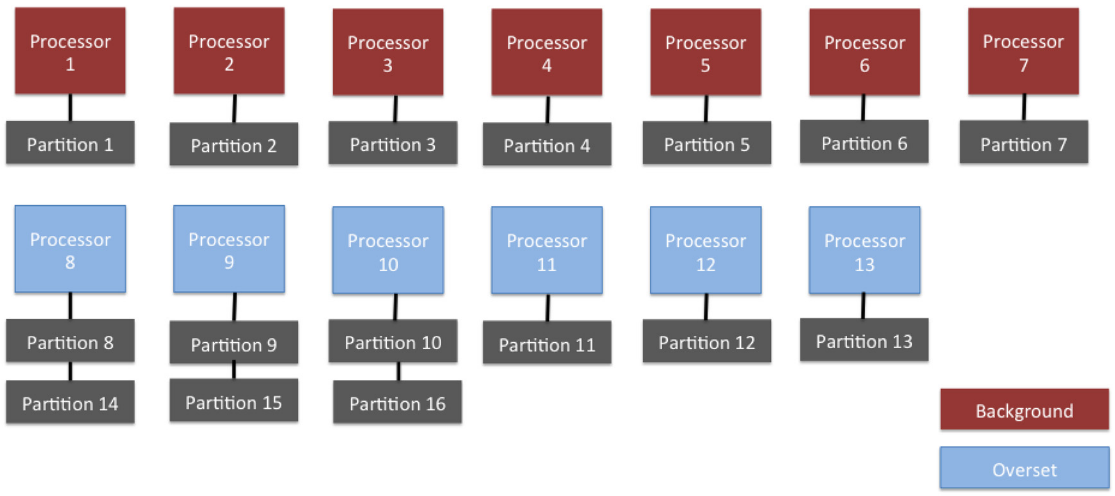
In this paper, we present an overset method which can utilize large numbers of meshes distributed over large numbers of computational cores. Details of the communication strategy are first presented in section 2. The removal of elements within the domain is then shown in section 3. Interpolation partner pairing is presented in section 4. Results from connectivity calculations are shown as relevant to each section. Scaling is then demonstrated for 100,000 spherical particles in turbulent channel flow in section 5.

2. Communication

Consider an example case of several objects in a channel as shown in Fig. 1. Each mesh, including the body meshes and background domain, is first individually partitioned to a set number of partitions. Processors are then assigned to be background or overset processors. Background mesh partitions, which are the channel mesh partitions in this case, are assigned to background processors such that there is one partition per processor. Overset partitions, which are the body mesh partitions in this case, are then assigned to overset processors in a cyclical fashion as shown in Fig. 1. Note that multiple partitions are allowed on the same overset processor and overset meshes can be duplicated as desired. To control the balance of the computational load, the number of partitions can be adjusted for each mesh individually and the number of overall processors can be adjusted. Even with movement, the required communication between partitions on the same mesh will not change throughout a simulation. By first partitioning each mesh the communication within a mesh is treated as static thus alleviating cost.

Partitions on different meshes which geometrically overlap must communicate mesh connectivity information such as the presence of solid boundaries or interpolation pairings. This requires that partitions must determine other overlapping partitions dynamically with time. It is desirable for best scaling that little to no global information about the different partitions is stored. Additionally, it is desirable that costly $O(N_{part}^2)$, where N_{part} is the number of partitions, calculations be avoided as much as possible. To avoid both of these undesirable properties we employ a parallel master/slave method which uses Cartesian spatial-partitioning to dynamically determine partition overlap.

This calculation is effectively the same as detecting collisions between different partitions. Spatial partitioning has been used to reduce the cost of collision detection successfully in the graphics community [14]. A Cartesian mesh is created,



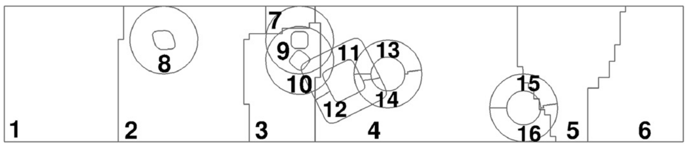


Fig. 1. Partitioned background and overset meshes in channel with partition numbering and resulting processor assignment color coded by processor type for 13 processors and 16 partitions. Numbering begins on the background channel mesh and continues onto the overset mesh partitions. Note the cyclical processor assignment is only applied to the overset partitions.

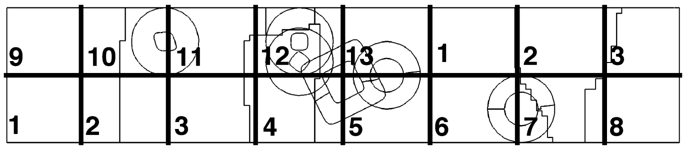


Fig. 2. Communication Cartesian mesh for overset meshes in channel numbered by assigned processor.

spanning all objects present in the calculation. The objects are binned into the different cells of the Cartesian mesh. Detailed collision calculations are then conducted only between objects that share a cell. This effectively culls the number of detailed collision calculations required, thus alleviating the cost.

For the present method, we begin by constructing a Cartesian mesh over the entire domain as shown in Fig. 2. This requires the calculation and storage of the Cartesian mesh spacing and size, a total of 6 global numbers. Each cell in the Cartesian mesh is assigned to a processor, which can be a background or overset processor, which will determine all overlap within the cell. The uniform size of the cells is set such that each processor has approximately one cell assigned to it. In most cases it is not possible to have one cell per processor thus multiple cells per processor are allowed as necessary. The assignment of processors to Cartesian cells is done in a cyclical fashion as depicted in the figure. By using this assignment the Cartesian mesh cell numbering (i, j, k) directly maps to a unique processor that can be determined by all processors with only knowledge of the Cartesian mesh size and spacing.

Determining if an object belongs to a Cartesian cell is done through an axis-aligned bounding box (AABB) comparison. Axis-aligned boxes which complete encompass the maximum and minimum points of an object are created for each object and are examined for overlap using

$$Box1_{i,max} \geq Box2_{i,min}, Box1_{i,min} \leq Box2_{i,max}. \tag{1}$$

Here, $Box1_i$ and $Box2_i$ refers to the maximum or minimum point in the specified i direction for each AABB respectively. If all of the conditions are met then there is overlap between the two AABB. In this comparison, an AABB is created over the object which is then compared to each cell for overlap.

Since a Cartesian mesh is used for the spatial partitioning, the cells themselves are AABB. Since the details of the Cartesian mesh are known, the range of cells over cell numbering (i, j, k) can be directly determined from a given partition's AABB. Since there is a direct map from (i, j, k) to an assigned processor, it is straightforward to determine not only the Cartesian cell within which a partition lies, but also the processor which has been assigned to that cell. For each partition,

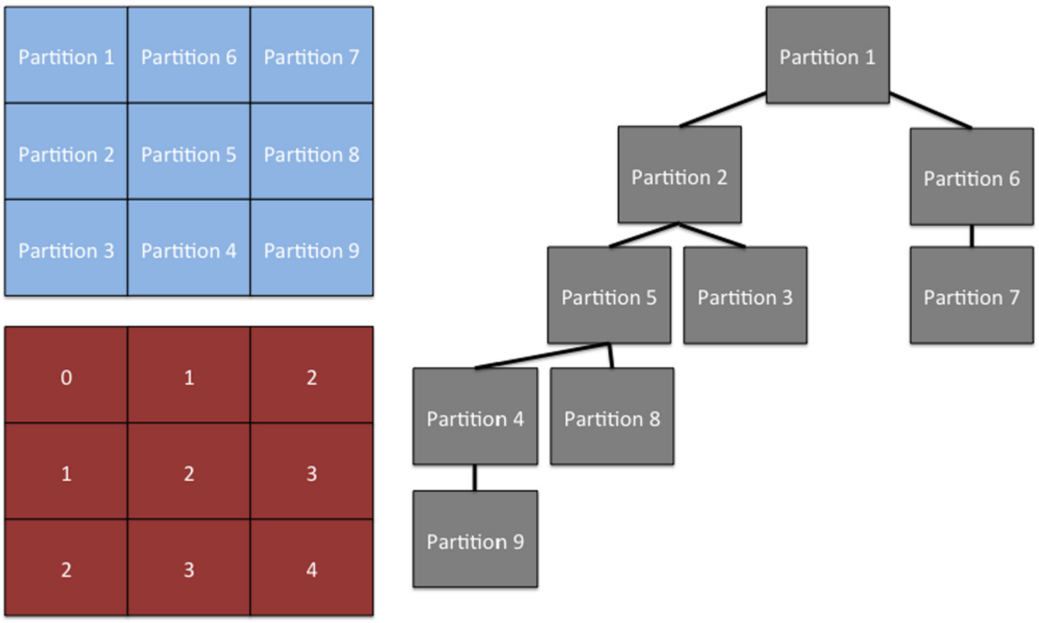


Fig. 3. Constructing binary communication pattern for a Cartesian partitioning over 9 partitions. (🟦) indicates partitioning, (🟥) indicates N_j, and (⬜) indicates the resulting tree. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

the range of (i, j, k) on the Cartesian mesh is calculated and the meta-information of the partition's geometry is communicated to the Cartesian cell processor. This information includes the AABB around the partition which is used to determine partition overlap. The communication from partitions to Cartesian cell processors is one-way such that a Cartesian processor may receive many messages or none at all, depending on the partitions present. The Cartesian processors process the meta-information received and sends back the resulting overlapping partitions to the processor assigned to each partition present within their respective cells.

Within a calculation it is desirable to conduct collective communication within a mesh as well as between overlapping partitions on different meshes. Using the Message Passing Interface (MPI) one could create a communicator object for each mesh and between meshes to give access to useful MPI routines. This is not currently a feasible strategy when many meshes are present due to limitations of speed and memory when dynamically creating the required communicator objects. For example at this time of writing, a typical Intel compiler is limited to approximately 16,000 communicators. Thus, this strategy on an Intel compiler would effectively limit a simulation to approximately 16,000 meshes. To overcome this limitation, a tree communication data structure is created for each mesh which stores all necessary information to perform collective operations akin to MPI routines such as MPI_ALLREDUCE.

Tree construction for an example Cartesian partitioned mesh is illustrated in Fig. 3. To create the tree structure a processor within a mesh is chosen to be the head of the tree for the mesh. A graph is created in parallel starting from the head processor using a flood-fill algorithm. An integer (N_j) starting with a value of 0 is passed starting from the head processor to neighboring partitions. As partitions receive N_j its value is increased by unity and sent to neighboring partitions. This is repeated until all partitions have received the integer message. The resulting value of N_j represents the number of partitions a given partition is away from the head partition. To construct the tree structure, each partition compares its local N_j to values of the neighboring partitions. If a neighboring partition has a smaller value of N_j relative to a given partition, the neighbor partition is considered to be on a higher branch of the tree. Conversely, if N_j is lower on a neighboring partition it is considered to be on a lower branch of the tree. Neighboring partitions with the same value of N_j are considered to be on the same branch of the structure. The relative locations of neighboring partitions on the tree are stored within each partition. Redundant tree information is sometimes found where a given processor is connected to several branches. In such cases all but one connecting branch is removed by pruning branches with large N_j. Note that only N_j is necessary to determine the pattern. Partitions can be general unstructured shapes or on different meshes and a tree can be created using this methodology.

To use the resulting tree for an operation like MPI_ALLREDUCE, messages are started at the bottom of the tree, reducing the value at each branch. The final reduced value is calculated at the chosen head processor and sent back down the tree. If the head processor is chosen appropriately such that a balanced tree is found, the resulting operation is expected to be similar in cost to the logarithmic cost of MPI_ALLREDUCE [15].

In addition to collective communication it is necessary to communicate arbitrary lists of data which are possibly dynamic with time between overlapping partitions. This is necessary for the cutting and interpolation procedures which will be discussed in the next sections. To perform such communication, a stacking strategy is used. For a given partition, each

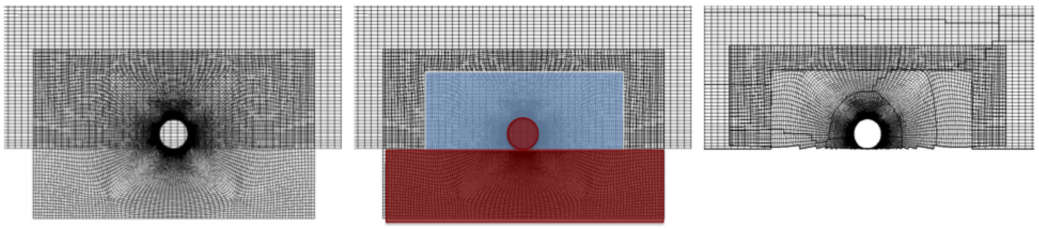


Fig. 4. Spherical particle near wall overset example before and after cutting using an angled rectangular box primitive shape. Blue (🟦) regions indicate a region where overlap must be reduced. Red (🟥) regions indicate elements that lie within solid boundaries. Note that elements on the background channel mesh within the particle and (🟦) are removed and elements on the particle mesh below the channel wall are removed.

overlapping partition is assigned a send and receive data stack to be used for communication. As data is found during a calculation which must be communicated to other partitions it is added to its the corresponding partition's send stack. As desired, the data within the stacks can be synchronized such that the data from all send stacks is placed within the receive stack of the corresponding processor and partition. Once the stacks are synchronized the receive data stacks can be accessed as needed for further calculation. The synchronization is conducted using non-blocking communication such that synchronization can be started while other calculations are performed and then finished when needed. This is utilized whenever possible to mask the cost of communication for best parallel efficiency.

3. Cutting

When several meshes with solid bodies are in close proximity, elements reside within the solid bodies and must be removed from a simulation. An example is shown in Fig. 4 for a spherical particle near a wall where elements are found in the wall and within the particle. In regions where meshes overlap redundant solutions are present as shown in the blue region in the figure. It is desirable to remove elements within these regions to improve efficiency. To remove elements a masking variable is used (ψ) which decouples the solution of the motion of fluid and solid bodies from removed elements when set to a value of 0. The process of removing elements by setting ψ will be referred to as cutting, similar to previous overset studies [10].

To reduce overlap between meshes, volumes are first constructed for each overset mesh which encompass the different solid bodies on the overset mesh while being contained by the mesh. Elements on other overlapping meshes inside these volumes are selectively removed from the simulation ensuring that elements are present throughout the domain. Simple primitive volumes are selected, such as spheres or angled rectangular boxes, when possible, to simplify and optimize calculations of overlap. For complex geometries it is not always feasible to use such primitive volumes. For such cases, solid faces along a body are projected outwards and the resulting projected body is used. In many cases, cutting elements within the primitive or projected volumes is all that is required. This is true when all of the elements that lie within solid bodies also lie within the primitive volumes. One example where this occurs is a 5-bladed propeller attached to a hull where an overset mesh is used for the propeller and a fixed background mesh is used for the hull. The result of cutting using a rectangular, spherical and projected volume for such a case is shown in Fig. 5.

Cutting of primitive volumes and of solid bodies is performed using a parallel flood-fill method. Fig. 6 depicts the flood-fill process for two spheres. To begin the method, elements which intersect the surfaces of cutting volumes must be marked. To do this, AABB of elements are first compared to AABB of cutting volumes using Eq. (1). This calculation is only done between partitions which overlap as found by the communication step outlined in the previous section. For each element found to satisfy the relation, the nearest faces along cutting volumes are found. To determine if points inside of elements lie within the cutting volume, the signed, normal distance to nearby faces is calculated in index notation as

$$d_{plane} = (x_{f,i} - x_{p,i})n_{f,i} \tag{2}$$

where d_{plane} is the normal distance from a plane coincident to the face, $x_{f,i}$ is a point along the face, $x_{p,i}$ is the test point and $n_{f,i}$ is the face normal according to index i. As a convention, face normals along cutting surfaces are defined to point into the body as shown in Fig. 8. Thus if $d_{plane} \leq 0$ the point lies within the body. In numerical experiments it has been found that typically only the closest face out of all nearby cutting volumes is necessary to accurately cut using Eq. (2). In general, several of the closest faces can be assessed for a more robust result.

If an element is found to intersect a nearby face according to Eq. (1) and has at least one node which is inside a solid body according to d_{plane} then it is masked from the solution. These elements mark the edges of regions on meshes which must be filled with masked values. Elements inside these regions must then be marked as starting points for the flood-fill algorithm. Neighboring elements to those that have been masked are assessed for overlap using the same searching procedure as before. If the neighboring element is found to lie within the body according to d_{plane} of its nodes it is marked as an internal flood element. To perform the flood we utilize a 'Forest Fire' flood-fill algorithm which avoids recursion through the use of a queueing system. The algorithm is presented in pseudocode in Proc. (2). The procedure is performed

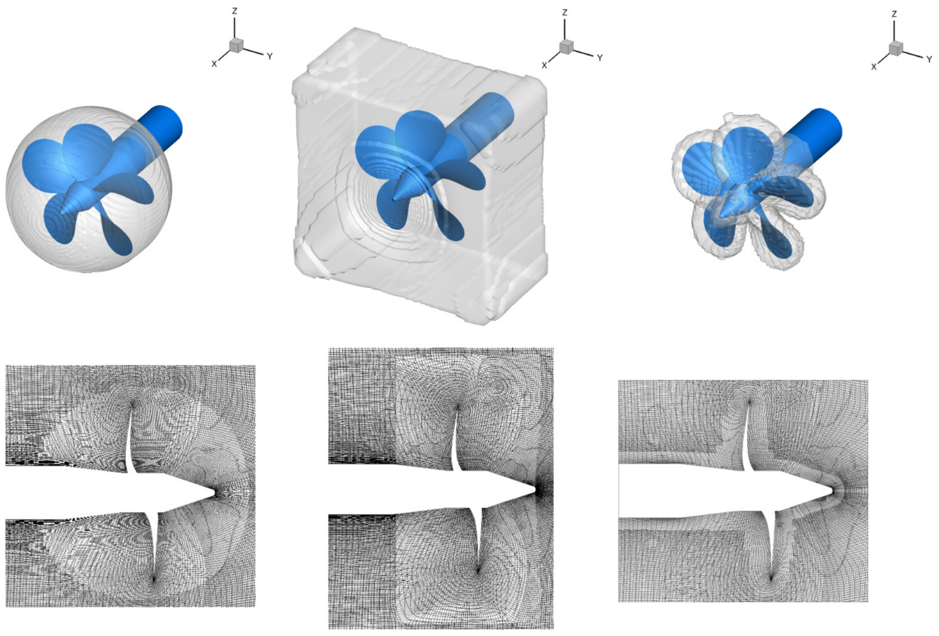


Fig. 5. Primitive volume cutting shown for a propeller attached to a hull. Elements removed and resulting cut using spherical, rectangular and projected primitive volumes are shown respectively. The two meshes contain 9 million elements total and the case is partitioned over 424 processors.

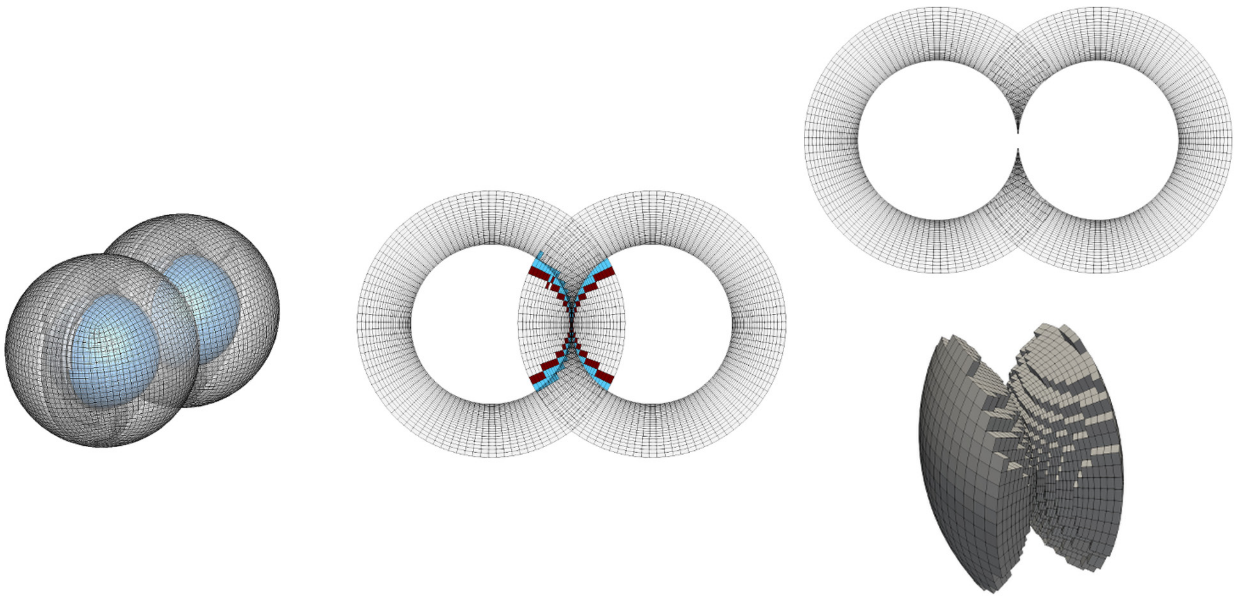


Fig. 6. Flood fill surface cutting for two touching spheres. Both meshes, surface cut and flood-fill starting elements and final cut mesh are shown respectively. In the center figure elements cut along surfaces are marked with blue (🟦) and flood-fill starting elements are shown in red (🟥) along a center slice through both spheres. In the last figure, the final removed volumes from both meshes are shown in addition to the final mesh along a center slice through both spheres.

starting at internal flood elements and iteratively at elements along inter-processor boundaries which have been marked by neighboring processors.

The outlined cutting method requires searching for neighboring nodes and face centroids. To perform all point searches k-dimensional, or k-d, tree data structures are extensively used. Initially created by Bentley [16], k-d trees are spatial partitioned binary trees in multiple dimensions where an arbitrary list of points can be organized based on splitting planes which pass through median points at each branch of the structure. A diagram showing the resulting k-d tree for a short list of points in 2D is shown in Fig. 7. Note that while only 1 point is included per node in the shown structure, it possible in general to have several nearby points within a single node. Nominally, k-d trees cost $O(N_p log(N_p))$ to construct and

Algorithm 1 Cutting Psuedocode.

```
1:  procedure CUT VOLUMES (PARTITION $\gamma$)
2:      for all overlapping partitions $\alpha$ do
3:          if cutting volumes present on $\gamma$ then
4:              send faces $\delta$ on volumes which overlap $\alpha$
5:          if faces received from $\alpha$ then
6:              for Elems overlapping volume AABB do
7:                  if Elem overlaps closest $\delta$ then
8:                      set $\psi = 0$
9:      for Elems with $\psi == 0$ do
10:         for neighbors of Elem do
11:             if neighbor also overlaps closest $\delta$ then
12:                 Start ‘Forest Fire’ flood-fill
13:     Return
```

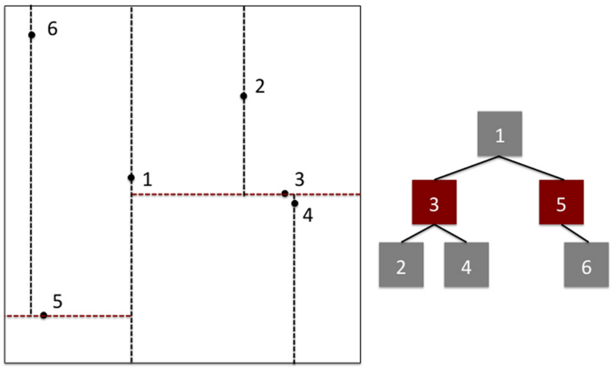


Fig. 7. K-dimensional tree structure for 6 points in 2D. Color indicates direction of each split within tree.

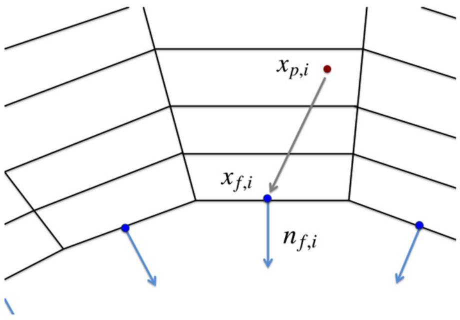


Fig. 8. Evaluating point intersection with general surface. Since nearest $n_{f,i}$ and $x_{f,i} - x_{p,i}$ are in the same direction, $d_{plane} > 0$ and point is outside of body.

$O(log(N_p))$ to search for N_{near} nearest points from a list of N_p points. This is a substantial cost savings when compared to an exhaustive search which is $O(N_{near}N_p)$. Commonly in overset assembly a list of points must be searched many times. Thus a k-d tree can be constructed once for many searches yielding more efficiency. This is the case for cutting where a partition searches from the same list of overlapping solid boundary faces for each overlapping element.

K-d tree data structures are similar to alternating digital tree structures (ADT) used by other overset assembly methods [17,18]. ADT allow for the storage and evaluation of finite objects in addition to points at the nodes within the tree structure. This allows for direct evaluation of geometric intersection between objects using the tree when performing a search. One drawback is the considerable cost of creation of the ADT. Additionally, the cost of searching and required storage of an ADT is expected to be higher than a k-d tree structure due to the finite volume information within the tree. In the present method, detailed geometric overlap calculations between finite objects are generally simplified to point searches allowing for extensive use of k-d trees instead of ADT. This is expected to yield improved computational efficiency which is required to perform connectivity calculations between many overset meshes.

4. Interpolation partner pairing

Boundaries remain around cut regions which require boundary conditions from the solutions of overlapping meshes. For the example of a spherical particle near a wall, interpolation boundaries are depicted in Fig. 9. Elements adjacent to these

Algorithm 2 'Forest Fire' Flood-Fill.

```
1: procedure FLOOD-FILL (ELEMS)
2:     create Q, an empty queue of Elems
3:     add Elem at end of Q
4:     while Q is not empty do
5:         set p equal to the first member of Q
6:         Loop through neighbor Elems of p
7:         if $\psi_{nbr}$ == 1 then
8:             set neighbor $\psi = 0$
9:             add neighbor to end of Q
    Return
```

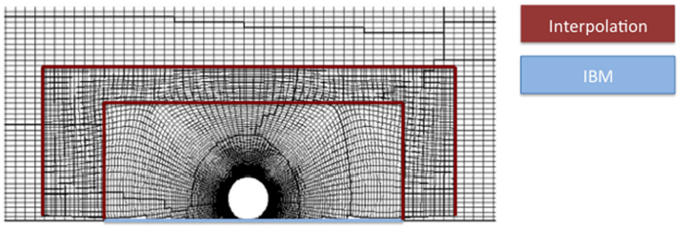


Fig. 9. Spherical particle near a wall with marked interpolation and IBM boundaries.

boundaries are treated as ghost cells with values set through interpolation by overlapping meshes. To construct interpolation stencils all elements and their neighbors from different meshes which overlap each ghost cell must be found.

To conduct the search all ghost cells are communicated to neighboring, overlapping partitions as determined by the communication process and an AABB comparison. K-d trees of element centroids are constructed on partitions which receive ghost cells. For each ghost cell an N-nearest neighbor search is conducted using the k-d tree. Elements must not be cut or themselves ghost cells to be considered for interpolation. Every found element from the search which overlaps the ghost cell and valid neighboring elements are sent back to the original partition. Note that all overlapping partitions of a ghost cell perform the search such that interpolation elements from several meshes can be used as desired. The connectivity between elements is preserved during this process such that stencils can reasonably constructed for each ghost cell. N is set to the maximum allowed number of elements which may be used by a given ghost cell for interpolation. This allows control of the efficiency of the interpolation process by limiting searches when ghost cells and overlapping elements are vastly different in size.

In certain circumstances there are no overlapping elements available for interpolation. An example of this is a spherical particle approaching a wall as shown in Fig. 9 where elements on the particle mesh along the bottom wall do not have overlapping elements for interpolation. In every case where this occurs the surface of a solid body is near the ghost cells. IBM is used in these cases to impose the boundary condition of the solid surface onto the non-aligning unstructured mesh. In order to reconstruct the solid surface boundary condition the geometry of nearest face and connectivity information is needed. All of this information is available during the cutting process which produced the IBM interpolation boundary. Instead of performing a potentially costly search of nearby solid boundary faces to each ghost cell, the necessary information is stored in cut elements during the cutting process outlined in the previous section.

As will be shown in the next section interpolation partner pairing is the most expensive calculation in the assembly process by a large margin. This is especially apparent when elements between meshes are vastly different in size as is the case in the previously shown propeller case in Fig. 5 where interpolating elements are, in places, found to be 1/20 in volume to their corresponding ghost cells. One strategy that can be used to alleviate the cost is to use the temporal coherence of elements within the meshes. This type of strategy has been used effectively in work on efficient collision detection [14,19]. As meshes move, elements typically move relatively smoothly such that most ghost cells overlap the same elements from time step to time step. It is faster to verify that the interpolation elements of a ghost cell are still valid rather than performing another costly search. Thus for interpolation partner pairing ghost cells first verify existing interpolation connectivity. If the interpolating elements no longer adequately overlap the ghost cell or if any interpolating element is cut or becomes a ghost cell the connectivity is marked as invalid. Any invalid interpolation connections are replaced by performing an interpolation search as outlined above. If an existing interpolation connection is still valid interpolation weights are updated to reflect any potential movement and nothing else is adjusted.

Fig. 10 shows the time savings for the previously shown propeller case rotating for 100 time steps. In the figure t_c is the time for full overset assembly, including communication and cutting, accounting for temporal coherence and t_f is the time for full assembly recalculated fully every time step. On average $t_f = 1.6$ secs, $t_c = 0.7$ secs such that the overall savings are approximately 46% for the case described. The savings are case dependent and vary based on the movement of elements along interpolation boundaries with time. If the rotation rate is increased, less savings are found with the absolute maximum cost being approximately t_f. If no motion occurs the interpolation presents far less computational cost compared to the cutting and communication procedures. For the propeller example the cost of assembly is approximately 0.2 secs with no motion. Even though the savings reported here are specific to the propeller example, savings are expected for most

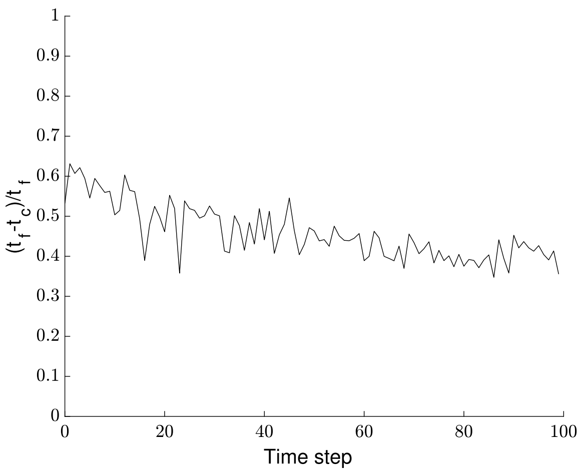


Fig. 10. Overset assembly cost savings using temporal coherence. t_c is the time each step for full overset assembly accounting for temporal coherence. t_f is the full assembly cost recalculating the full assembly. The average values of $t_c = 0.7$ secs, $t_f = 1.6$ secs. These timings were acquired using Quartz (Intel Xeon 2.1 Ghz) at Lawrence Livermore National Laboratory (LLNL).

general moving body problems so long as elements along interpolation boundaries do not move more than one element length between time steps.

The timings found for the propeller are found to be competitive to similar cases as reported in other overset assembly work. In particular, in the work of Roget and Sitaraman [12] the timings for overset assembly of unstructured meshes of a helicopter fuselage and a 4-bladed rotor, termed the HART-II case in the work, are reported. While the geometries are not the same, the overall number of elements per processor, speed of the machines used to acquire the timings, and geometrical features are similar to the propeller. It is found in the work that the overall cost of assembly is approximately 1.3 seconds. This is similar in cost to the timings found for the propeller without including temporal coherence.

Algorithm 3 Interpolation Pairing Psuedocode.

```
 1: procedure INTERPOLATION PAIRING (PARTITION $\gamma$,N)
 2:     for all overlapping partitions $\alpha$ do
 3:         if ghost cells on $\gamma$ then
 4:             send ghost cells which overlap $\alpha$
 5:         if ghost cells received from $\alpha$ then
 6:             if Elem k-d tree not created then
 7:                 create Elem k-d tree
 8:             for each ghost cell do
 9:                 n-neighbor search (k-d tree, N nbrs)
10:                 send back overlapping Elems + Nbrs
11:     for ghost cells on $\gamma$ do
12:         if Elems received for cell then
13:             calculate interpolation weights
14:         else
15:             calculate IBM weights
16:     Return
```

5. 100,000 spherical particles in a channel

In order to demonstrate the method's capability to perform overset assembly when many overset meshes are present, such as is the case for PR-DNS, a strong scaling study is performed for 100,000 spherical particles in a channel. A single, unstructured spherical particle overset mesh is duplicated and distributed randomly across a channel mesh which has dimensions selected to match DNS studies of turbulent channel flow at friction Reynolds number, $Re_\tau = 180$ [20]. Information about the overset mesh and channel is listed in Table 1. In the table L_x is the length of the channel in the specified direction with N_x as the number of elements in the corresponding direction. Δ_{bg} is the uniform element spacing in the channel, D_p is the diameter of the particle, Δ_e is the element spacing at the edge of the overset, and Δ_s is the spacing near the surface of the particle on each overset mesh. The distribution of particles in the channel and the overset mesh is depicted in Fig. 11. The streamwise and spanwise boundary conditions of the channel are periodic. The vertical boundaries are chosen to be walls. Calculating overset connectivity through periodic boundaries requires establishing communication patterns and overlap between geometries that may be far apart within the domain. A linear mapping which maps points and AABB across planar periodic boundaries is used at each step of the method to produce connectivity with meshes across

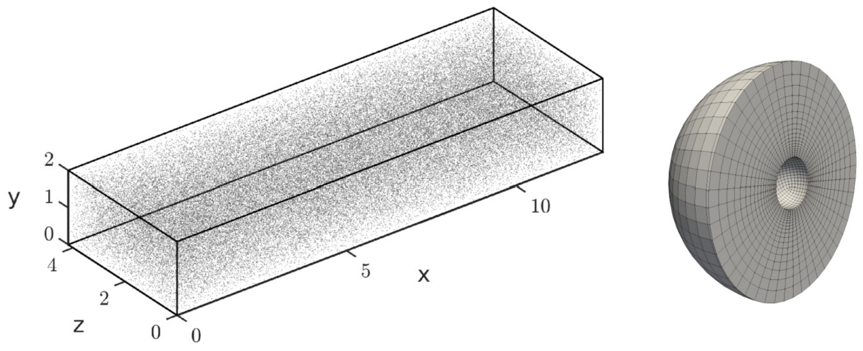


Fig. 11. Particle distribution within channel and particle overset mesh. Each point in the distribution is a particle mesh at actual scale.

Table 1
Channel and overset mesh information.

$L_x \times L_y \times L_z$	$N_x \times N_y \times N_z$	D_p/Δ_{bg}
$4\pi \times 2 \times 4/3\pi$	$2514 \times 400 \times 838$	5

N_{elem}	D_p/Δ_e	D_p/Δ_s	D_p
15543	5	40	0.013

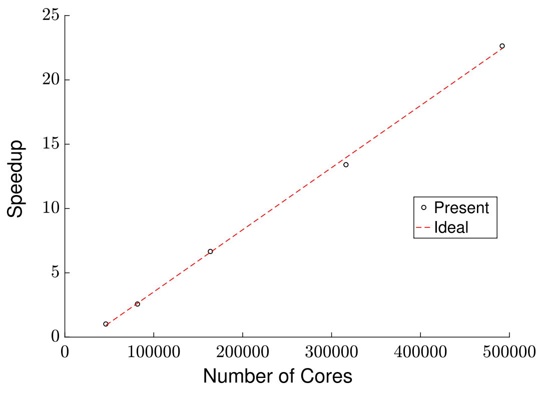


Fig. 12. Strong scaling for 100,000 spherical particles in a channel.

periodic boundaries. Details of the mapping and the necessary step-by-step periodic adjustments are not the focus of the present study and thus are omitted.

In total there are approximately 2.8 billion elements in the case. The channel spacing is chosen to be uniform with size $\Delta_{bg} = D_p/5$. On the overset mesh near the particle surface the mesh spacing is chosen to be nearly uniform with $\Delta_s = D_p/40$ and to grow at a rate of 5% away from the particle surface towards the edge of the overset mesh. The overset mesh is 15,543 elements in size. The channel background mesh is 850 million elements in size.

For the simulations the particles are set to move at a fixed velocity equal to the average flow velocity within a channel according the log profile given by

$$u_p^+ = \frac{1}{k} ln(y^+) + C^+ \tag{3}$$

where $u_{p,+}$ is the dimensional particle velocity normalized by friction velocity, and y^+ is the vertical location of the particle relative to the nearest wall in plus units, $k = 0.41$ is the Von Kármán constant, and $C^+ = 5.0$. Timings using MPI_WTIME() are taken over 20 time steps of particle motion using $dt = 0.001$. All of these simulations were conducted on MIRA at Argonne National Laboratory (ANL). The number of processors for the study ranged from 46384 to 492192 processors. Meshes were repartitioned for each simulation for best load-balancing. The processor load varied from 55,000 elements per processor to 5,000 elements per processor. Several partitions were required per overset processor for the heavier loadings with a maximum of 7 partitions per processor being required for 46384 processors. The total number of partitions ranged from 300,000–500,000 depending on the re-partitioning.

The resulting strong scaling is shown in Fig. 12 and the corresponding average timings are shown in Fig. 13. Over the range investigated excellent scaling is found when compared to ideal linear scaling. The cutting and interpolation pairing processes are both found to individually scale. Establishing communication patterns is found to be a relatively cheap oper-

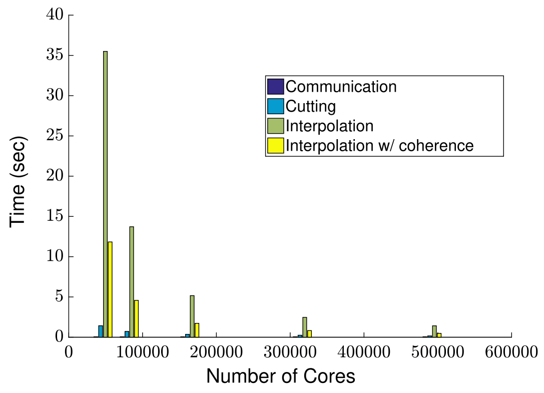


Fig. 13. Overset assembly timings for 100,000 spherical particles in a channel.

ation over all cases with a cost of O(10) milliseconds. Interpolation partner pairing is the most expensive operation. Using temporal coherence is found to reduce the cost of the interpolation pairing for this case by approximately 70% over all loadings investigated.

In strong scaling it is often found that increased communication costs result in less than ideal scaling when large numbers of cores are present. In the current study the number of partitions, the partition sizes, number of interpolation elements per processor, and cut elements per processor all decrease with increases in number of cores. Using the current method this overall results in a lessening of the required overset communication with increased core count which likely yields the found scaling. The use of temporal coherence is found to provide substantial cost savings for this case. It is possible that improvements could be made by utilizing further past connectivity information to extrapolate more accurately which ghost cells will need new interpolation elements for a given time step.

Conclusions

The method presented in this work was created to connect many, moving overset meshes in an efficient and scalable manner. Using strategies from work on collision detection a parallel master/slave spatial partitioning method was created. Dynamic collective and point-to-point communication is required between overlapping mesh partitions. For collection communication a binary tree communication strategy was employed which avoids expensive creation of MPI objects while retaining comparable computational cost when used for reduction operations. Point-to-point communication was handled using an asynchronous stacking strategy. Establishing communication patterns and performing the necessary communication for overset assembly was demonstrated to be efficient and scalable up to O(10^5) cores and partitions, even when several partitions are present on each core.

Within an overset calculation it is necessary to remove elements within solid boundaries and to remove elements in regions of overlap to reduce solution redundancy. Where possible elements are removed from primitive volumes, such as spheres or angled rectangular boxes, instead of the more complex underlying geometry. Relatively limited amounts of information is required to be communicated to determine overlap using such primitive volumes and overlap calculations are typically simpler and more efficient. A flood-fill algorithm is used where elements along the surfaces of primitive volumes and solid boundaries are first removed and elements within the volumes are then removed using a flooding algorithm. The algorithm used ensures that detailed calculations of overlap are only conducted locally along the surfaces of cutting volumes rather than throughout the domain yielding reasonable computational cost and scalability. The cutting strategy was demonstrated to be efficient for complex geometries such as a 5-bladed, marine propeller attached to a hull. When many meshes are present the strategy was found to be low in computational cost and to also reasonably scale.

To connect solutions throughout the domain interpolation is conducted between overlapping meshes. This requires interpolation partner pairing between overlapping elements on different meshes. Primitive volume overlap comparisons, such as AABB, are used to narrow required searches. K-d trees are extensively used to conduct detailed searches for nearby elements. The temporal coherence of element motion is used to retain interpolation pairing connectivity as meshes move rather than performing potentially expensive and unnecessary searches. While the interpolation pairing procedure is the most expensive operation, it was found to reliably scale to large numbers of cores when many meshes were present. The use of temporal coherence was found to provide substantial overall cost savings for interpolation pairing and the overset connectivity process, which was found to be approximately 50–70% for the cases presented in this work.

Acknowledgements

This work is supported by the DOE under DE-NA0002373 and NSF under Grant CBET-1510154. Computational resources provided by Argonne National Laboratory were extensively used in this work as part of the Predictive Science Academic

Alliance Program (PSAAP2) through the Department of Energy. In particular, the machines MIRA and Cooley were used to produce the large scaling results presented. Sreevatsa Anantharamu is gratefully acknowledged for providing the large channel mesh required for the scaling study. Praveen Kumar and Thomas Kroll are acknowledged for creating the propeller and hull meshes used to demonstrate the cutting process.

References

[1] Y. Tseng, J. Ferziger, A ghost-cell immersed boundary method for flow in complex geometry, J. Comput. Phys. 192 (2003).
[2] C. Peskin, The fluid dynamics of heart valves: experimental, theoretical and computational methods, Annu. Rev. Fluid Mech. 14 (1981).
[3] R. Mittal, G. Iaccarino, Immersed boundary methods, Annu. Rev. Fluid Mech. 37 (2005).
[4] F. Sotiropolous, X. Yang, Immersed boundary methods for simulating fluid-structure interaction, Prog. Aerosp. Sci. 65 (2014).
[5] L. Zhu, C. Peskin, Interaction of two filaments in a flowing soap fill, Phys. Fluids 15 (2003).
[6] G. Iaccarino, R. Verzicco, Immersed boundary technique for turbulent flow simulations, Appl. Mech. Rev. 56 (2003) 331–347.
[7] A. Kidanemariam, M. Uhlmann, Interface-resolved direct numerical simulation of the erosion of a sediment bed sheared by laminar channel flow, Int. J. Multiph. Flow 67 (2014) 174–188.
[8] F.S.X. Yang, Immersed boundary methods for simulating fluid-structure interaction, Prog. Aerosp. Sci. 65 (2014) 1–21.
[9] Z. Wang, V. Parthasarathy, A fully automated chimera methodology for multiple moving body problems, Int. J. Numer. Methods Fluids 33 (2000) 919–938.
[10] R. Meakin, N. Suhs, Unsteady aerodynamic simulation of multiple bodies in relative motion, AIAA Pap. 98 (1989) 1996.
[11] P. Buning, T. Pulliam, Cartesian off-body adaption for viscous time-accurate flow simulations, in: Proceedings of 20th AIAA Computational Fluids Dynamics Conference, 2011.
[12] B. Roget, J. Sitaraman, Robust and efficient overset grid assembly for partitioned unstructured meshes, J. Comput. Phys. 260 (2014) 1–24.
[13] R. Noack, D. Boger, R. Kunz, P. Carrica, Suggar++: An improved general overset grid assembly capability, in: Proceedings of 19th AIAA Computational Fluids Dynamics Conference, 2009.
[14] C. Ericson, Real-Time Collision Detection, CRC Press, 2004.
[15] R. Thakur, R. Rabenseinfer, W. Gropp, Optimization of collective communication operations in mpich, Int. J. High Perform. Comput. Appl. 19 (2005) 49–66.
[16] J. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (1975) 509.
[17] S. Rogers, N. Suhs, W. Dietz, Pegasus 5: an automated pre-processor for overset-grid cfd, AIAA Pap. 41 (2003) 1037–1045.
[18] J. Bonet, J. Peraire, An alternating digital tree (adt) algorithm for 3d geometric searching and intersection problems, Int. J. Numer. Methods Eng. 31 (1991) 1–17.
[19] M. Lin, Efficient Collision Detection For Animation and Robotics, PhD thesis, University of California, Berkeley, 1993.
[20] R.D. Moser, J. Kim, N.N. Mansour, Direct numerical simulation of turbulent channel flow up to $Re_\tau = 590$, Phys. Fluids 11 (1999) 943–945.