AIAA SciTech Forum
9 - 13 January 2017, Grapevine, Texas
58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference

10.2514/6.2017-0357

An Efficient Parallel Overset Method for Aerodynamic Shape Optimization

Gaetan K.W. Kenway,*, Ney Secco,†, Joaquim R. R. A. Martins‡
Asitav Mishra,§, Karthik Duraisamy,¶
University of Michigan, Department of Aerospace Engineering, Ann Arbor, MI

Structured mesh computational fluid dynamic solvers are inherently faster than unstructured solvers, which is particularly advantageous for aerodynamic design optimization, where hundreds of flow solutions are required. However, generating body-fitted multiblock meshes for complex geometries is challenging and is a time consuming task. The overset mesh technique greatly reduces the manual effort required to generate meshes over complex geometries by overlapping a series of simpler meshes. However, generating the necessary connectivity information between meshes in a robust and computationally efficient manner remains a challenge. We address this challenge by developing an efficient parallel overset grid assembly technique based on implicit hole cutting that is fully automatic. The method is fully parallel and scales to hundreds of processors. Several optimizations of the Common Research Model wing-body-tail configuration are performed using the meshes generated by our technique. We compare the best drag reduction obtained from multiblock and overset meshes using two different artificial dissipation schemes. The smooth, highly orthogonal overset meshes produce better results than the multiblock meshes, by up to 3 drag counts. An application to rotorcraft design is also presented. The demonstrated meshing flexibility and accurate transonic solutions make the overset mesh technique ideally suited for aerodynamic shape optimization.

I. Introduction

Many of the problems of interest in aerospace applications have complex geometric domains, both in fixed wing and rotary wing applications [1, 2, 3]. In most practical applications, it is not possible to use a single structured mesh to adequately define the geometry of interest. For this reason, most structured mesh approaches use the multiblock method, which combines many structured meshes in an unstructured manner to cover a more topologically complex region. While it is always technically possible to find a structured mesh topology to mesh a general domain or arbitrary complexity, it can be an extremely time consuming task to generate such a mesh, and there is no guarantee that the resulting mesh will be of high quality.

The difficulty with structured meshing can be greatly reduced by employing unstructured meshes [4], composed of polyhedral mesh elements (typically tetrahedrons, hexahedrons, and prisms). In general, a much higher degree of automation can be employed, reducing the burden on the end user. However, generating high quality prismatic boundary layer mesh as is normally required for high Reynolds number flow remains a difficult task. In addition, it is well recognized that the computational efficiency of unstructured techniques is lower than what can be achieved with structured meshes, primarily because of the higher memory usage and indirect memory addressing of unstructured meshes.

The overset approach to mesh generation—also known as chimera—can eliminate many of the limitations of multiblock meshes while retaining the overall computational efficiency of structured meshes. Traditionally, there have been two primary use cases for overset mesh techniques: 1) The simulation of bodies in relative motion and 2) Meshing simplification for structured grids. A typical application of the first use case is the simulation of the flow around a complete helicopter or wind turbine. Both configurations have a rotor that is in relative motion with respect to the remainder of the geometry. For these types of applications, both structured and unstructured mesh techniques have been employed [1, 5, 6].

The second use case, easing the mesh generation burden for structured meshes, is our primary motivation for using the overset mesh technique. An example of a complex configuration—the wing-body-nacelle-pylon configuration used for sixth Drag Prediction Workshop (DPW6)—is shown in Figure 1.

*Research Investigator, AIAA Member
†Ph.D. Candidate, AIAA Student Member
‡Professor, AIAA Senior Member
§Post Doctoral Research Fellow
¶Assistant Professor, AIAA Member

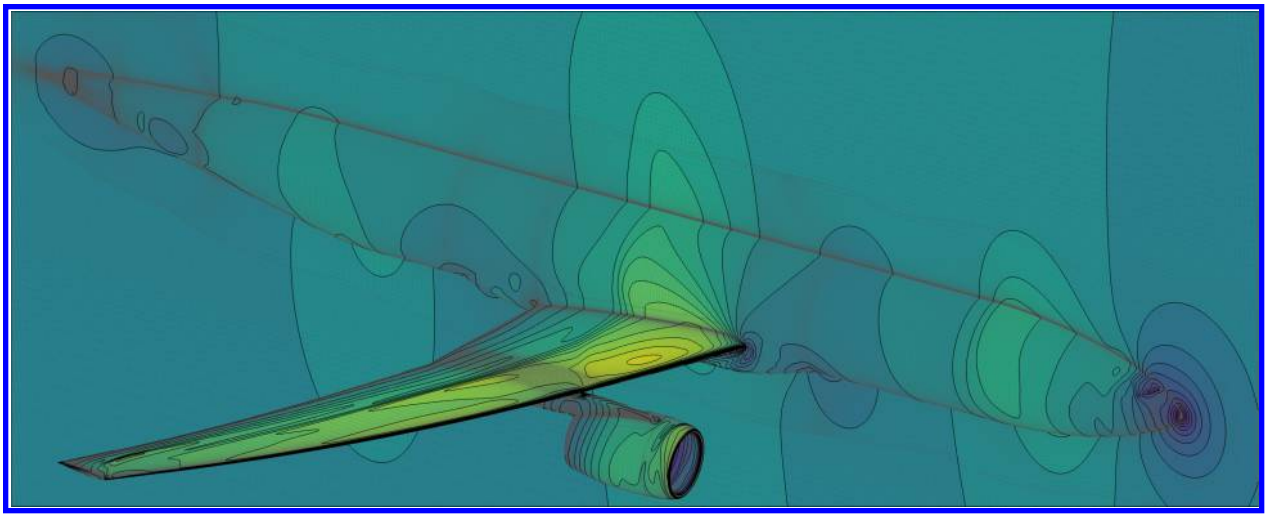


Figure 1: Complex wing-body-nacelle-pylon configuration from the sixth Drag Prediction Workshop.

This paper is organized as follows. First, we outline the overset grid assembly process, describing in detail the iterative process necessary for the implicit hole cutting (IHC) method. We then present a strong scaling study of the grid assembly algorithm using up to 3,960 cores. Finally, optimization results are presented for a transonic aircraft configuration using both the multiblock and overset mesh approach. A glossary of some commonly used overset terminology is listed in Appendix A for readers unfamiliar with the overset mesh technique.

1. Overview of the Overset Approach

The general concept of the overset approach is shown in Figure 2, which depicts two overlapping meshes: A red O-mesh around a cylinder and a black Cartesian background mesh. The outer two layers of the red mesh—denoted with squares—interpolate solution data from the background mesh. The reverse operation must also take place: The background mesh must receive information from the red near-field mesh. These cells are marked with black circles, and represent an artificial interpolation boundary: The background mesh cells that overlap the cylinder are blanked (i.e., removed) from the computation. The remaining jagged boundary supplies the necessary boundary condition information for the solution of the field equations inside the background mesh. Similarly, for the near-field mesh, the outer boundary receives information from the background and the inner boundary is a physical boundary condition. All interpolated or fringe cells must then find a donor cell on the overlapping mesh from which to interpolate the solution information.

In this simple example with just two overlapping meshes, it is straightforward to see where the fringe boundaries should be located, and which cells must be blanked and removed from the computation. However, the ability for a user to manually determine to what extent multiple overlapping meshes intersect and to identify blanked cells, is very quickly diminished with increasing mesh complexity. Consider the example shown in Figure 3. This example is similar to the case in Figure 2, except for a protrusion that is added to the cylinder, and the addition of another cylinder. In the next section we describe in detail an algorithm capable of handling complex overset meshes in an automatic and robust manner.

II. Implicit Hole cutting method

The basic ideas behind the IHC algorithm were first described by Lee and Baeder [7]. The main idea is that given the choice between two or more cells occupying the same physical space, we should select the cell with the best quality. The simplest criteria for cell quality is the cell volume: Smaller cells have lower truncation error and generally resolve the underlying physics in a CFD solution more accurately. Therefore, when given a choice to choose between two or more cells, we choose the one with the lowest cell volume.

The key observation made by Lee and Baeder is that cells close to the surface of a body of interest are generally smaller than the cells on the background mesh. This is especially true for meshes suitable for RANS solutions, where very small off-wall spacing is required to accurate resolve the viscous sub-layer. Since these cells have extremely

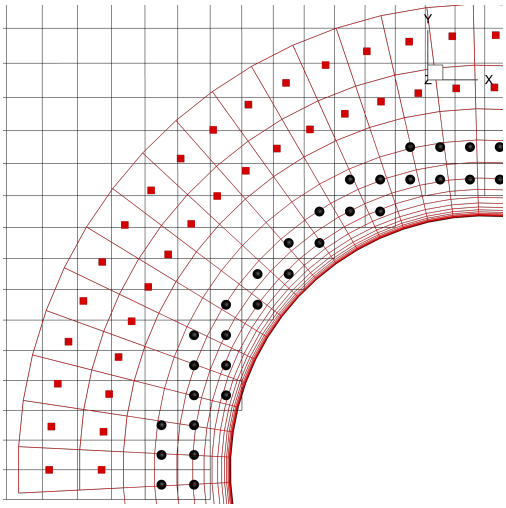


Figure 2: Overset approach: O-mesh around a cylinder (blue) with a Cartesian background mesh (back). Symbols mark cells which interpolate field information from the other mesh.

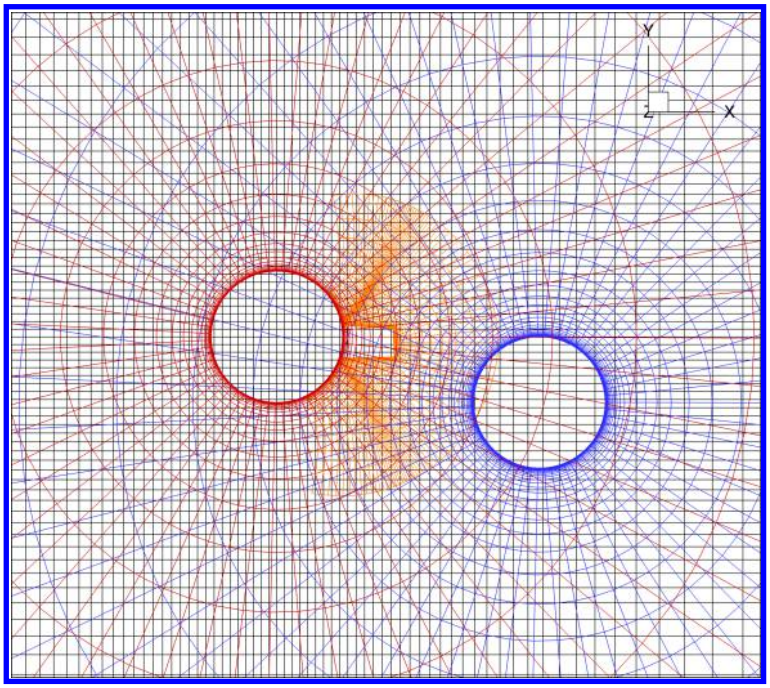


Figure 3: Manually determining which cells must be blanked becomes increasingly difficult as the number of overlapping meshes increases.

small volumes, background meshes or other near-field meshes will be automatically or "implicitly" cut by the wall. The result of this simplest IHC algorithm applied to the configuration in Figure 3 is shown in Figure 4.

If we look at the right cylinder, we can see how the background mesh (black) is interrupted by the cylinder mesh, and the background mesh cells that lie inside the body are isolated from the remainder of the compute cells. We have "implicitly" cut the background mesh without ever specifying where the body surface is actually located. Lee and Bader make the argument that background mesh cells remaining as compute cells can safely remain in the simulation provided they do not contaminate the "real" solution. This is true only if no "inside" cells use "outside" cells in their stencil and vice-versa. For the example, in Figure 4, this is not always the case: There is at least 1 cell gap between the inside and outside parts, but not the two layers required for a second order finite volume scheme to ensure no solution

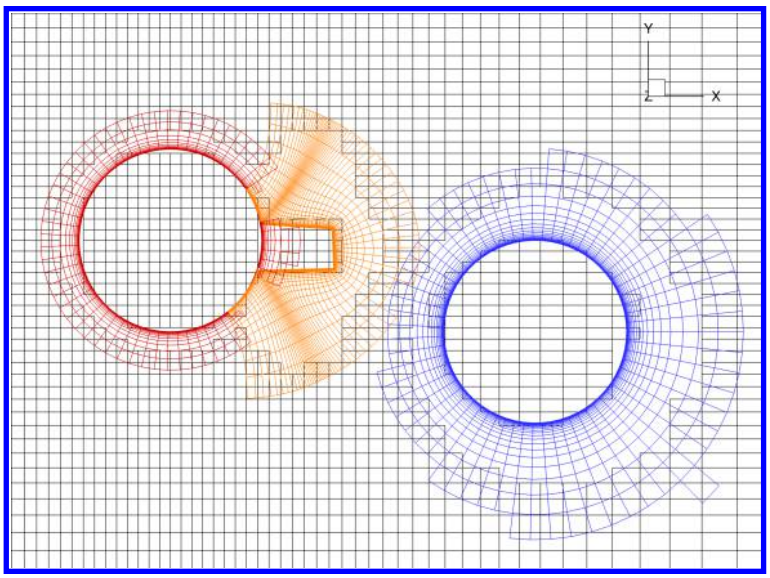


Figure 4: Basic implicit hole cut method. Only compute cells are shown. Note that the background mesh cells remain inside the body.

contamination.

Further issues arise for the left body. In this case, we have two overlapping surface meshes (red and pink). Again, in this case, the section of the red mesh inside the body is isolated from the remaining computational domain. The same is true for the background mesh. However, here we have an additional complication: The internal cut section cylinder mesh (red) contains un-blanked solid wall cells that will be included in the surface integration. This results in incorrect integrated force quantities, such as lift and drag, which are of primary interest in aerodynamic analysis.

These examples highlight the key challenge of all overset grid assembly techniques: How to identify and remove cells that lie inside the body? This challenge can be further complicated by meshes with surface-surface overlap that prevents the immediate identification of a topologically closed body. For cases without surface-surface overlap, an alternative identification algorithm is presented in Appendix B.

The main steps of our IHC algorithm are summarized in Algorithm 1, and each step in the process will be further explained in detail. We assume that the input to the IHC algorithm is a set of N computational blocks distributed over P processors, where $N \geq P$. That is, each processor contains 1 or more computational blocks. The distribution is determined by a partitioning algorithm that tries to ensure each processor has roughly the same number of cells.

We now explain the main steps of the algorithm by using Figure 3 as an example. Even though this is a 2D example, it highlights most of the issues of the IHC method that appear in more complex 3D applications.

The first step in the algorithm is the identification of cluster IDs. We define a cluster as a collection of one or more point-patched computational blocks connected in an arbitrary fashion.

The main task before starting the donor search operation, is to determine the *overlap matrix*. The overlap matrix is a $N \times N$ logical matrix, where N is the number of computational blocks. In general this matrix will be sparse and it is important to store it as such. Since the number of blocks (N) must be at least as large as the number of processors (P), the amount of memory to store the overlap matrix increases with P^2, which is undesirable. When a sparse matrix is used, the memory scales only with a constant times the number of processors.

To compute the overlap matrix, we first compute the Cartesian coordinate-aligned bounding boxes for each block. The bounding boxes are communicated to all processors. Then each local block is checked against the bounding boxes of all blocks that belong to a different cluster. We have assume that all blocks within the same cluster do not overlap, as this can slightly reduce the number of nonzeros entries in the overlap matrix.

Any overlap of the block bounding boxes is treated as a potential overlap, and registers a nonzero entry in the overlap matrix. The overlap matrix is conservative: A `True` entry implies that two blocks *may* physically overlap; a `False` entry guarantees two blocks do not overlap. If a given pair of blocks overlap, two entries are generated in the overlap matrix, and thus the overlap matrix must be symmetric. However, a nonzero entry at (row, col) has a different meaning than the entry at (col, row). The rows represent the donors, while the columns represents the search coordinates. For example, a nonzero entry at (3, 2) means "search for donors in block 3 using the search coordinates

Algorithm 1 Implicit hole cutting

```
Determine cluster IDs
Compute Cartesian-aligned domain bounding boxes
Determine global overlap matrix
Load balance
Determine overlap comm pattern, and its transpose
Build walls for each cluster
Compute wall-point for each cell
Flag forced receivers
Build volume ADTs
Assemble search points
Communicate ADTs/Search Points
Perform surface correction
Perform assigned searches
Communicate donor information using transpose overlap comm pattern
Initialize status array
Determine wall donors
while iblank is changing do
    Re-initialize status array
    Implicit hole cut
    Flood interior cells
    Determine donors
    Flag forced receivers
    Flag invalid donors
    Irregular cell correction
end while
Fringe reduction
Final comm structures
```

from block 2". The basic overlap matrix the cylinder problem is shown below in Figure 5. Note that the diagonal will always be empty due to the fact that a given block cannot overlap itself.

Block	1	2	3	4
1		x	x	x
2	x		x	x
3	x	x		x
4	x	x	x	

Figure 5: Overlap matrix for cylinder problem

One of the main difficulties in producing highly-scalable IHC overset assembly methods is that the problem is inherently poorly load-balanced: Some blocks intersect more blocks than others and thus require more searches than others. Therefore, a desirable partitioning from a solver's perspective (equal number of cells on each processor) is not ideal for overset connectivity.

To investigate the effects of load-balancing, we run the two-cylinder using 8 processors, which requires block splitting to ensure each processor has a compute block (Figure 6). The background mesh is partitioned into 5 blocks and the protrusion mesh is divided into two blocks for a total of 9 blocks.

The global overlap matrix for the partitioned mesh is given in Figure 7.

Once the global overlap matrix is established, we know which pairs of meshes must be searched to find potential donors. The next task is to determine how to distribute the required searches to each of the available processors. This is a crucial step as the amount of work each processor has to perform, and thus the time required, may vary considerably.

There are three main ways of distributing the computational work specified in a given overlap matrix for a given block distribution:

1. Communicate only the search points (columns)

2. Communicate only the donor cells (rows)

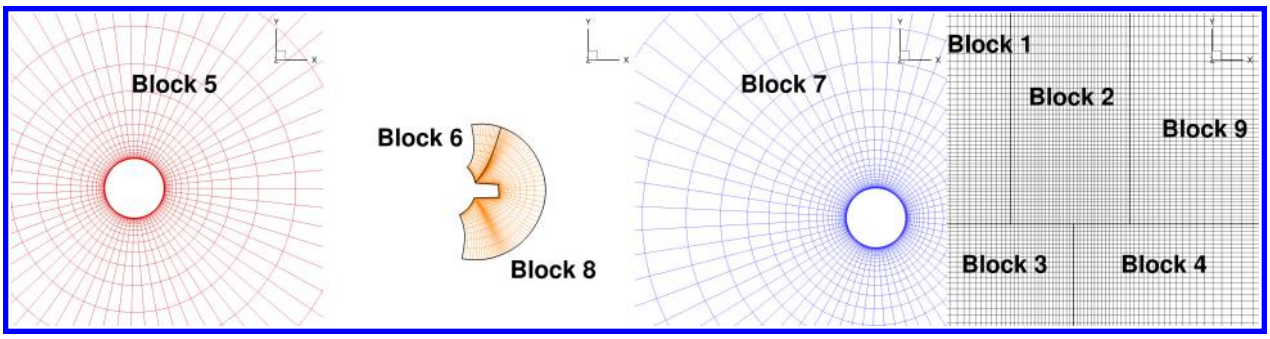


Figure 6: Block partitioning using 8 processors.

Processor	Block	1	2	3	4	5	6	7	8	9
0	1					x		x		
1	2					x	x	x	x	
2	3					x		x		
3	4					x		x	x	
4	5	x	x	x	x		x	x	x	x
4	6		x			x		x		
5	7	x	x	x	x	x	x		x	x
6	8		x		x	x		x		x
7	9					x		x	x	

Figure 7: Overlap matrix for partitioned cylinder problem

3. Communicate both search points and donor cells (rows and columns)

The problem with either of the first two methods can be seen readily in the above example: The overlap matrix often contains relatively dense rows and columns, and thus the processors containing these blocks may have to do much more work than the other processors, leading to poor load balancing. To remedy this problem, we adopt the third method of work distribution: Both the donor cells and the search points may be communicated to any processor assigned to perform the search.

The next step is to determine an appropriate block distribution. The load balancing algorithm's goal is to distribute an equal amount of "work" to each available processors such that the total amount of time required to perform searches is a minimum. The first time the overset assembly is executed, it is not possible to accurately estimate the exact amount of wall time any particular set of searches will require. The reason for this is that the time required to perform m searches on a particular block depends on how many of the search points actually find donors. If two blocks fully overlap and all m search points find donors, it takes much longer than if only a very small number of search points find donors. Since this information is not known until the search is actually completed, the first time the algorithm is called, we use the total number of search points as the measure of work on which to perform the load balancing. Then, during the actual search, we record the actual amount of wall time required to perform the search. The next time the overset assembly is required, we redo the load balancing based on the actual compute time. This method universally improves the efficiency of the algorithm.

The result of the load-balancing algorithm using the number of search points as the work unit is shown in Figure 8. The nonzero entry of the matrix refers to the processor that has been assigned the task.

When the algorithm runs a second time, the load balancing has now changed, based on the actual computation time of the first call. The resulting load balancing is shown in Figure 9. Further results of the effects of load balancing are shown in Section III.

Once the load balancing algorithm has assigned each set of search points to a processor, we determine which donor cells and which search points must be communicated to the processors assigned to perform the search. The transpose or reverse of this communication pattern is also assembled, as this is required when sending the donor information back to the process owning the search points.

An alternating digital tree (ADTree) data structure is then generated for each computational block. Note that the

	Cost	5.52	5.52	5.54	5.32	4.50	1.00	6.00	5.57	5.52
Processor	Block	1	2	3	4	5	6	7	8	9
0	1					0		0		
1	2					0	0	0	1	
2	3					1		1		
3	4					1		1	2	
4	5	2	2	2	3		3	3	3	3
4	6		4			4		4		
5	7	4	4	5	5	5	5		5	6
6	8		6		6	6		6		7
7	9					7		7	7	

Figure 8: Assigned processor after load balancing based on search-point criteria.

	Cost	5.52	5.52	5.54	5.32	4.50	1.00	6.00	5.57	5.52
Processor	Block	1	2	3	4	5	6	7	8	9
0	1					0		0		
1	2					0	0	0	0	
2	3					1		1		
3	4					1		1	1	
4	5	1	2	2	2		3	3	4	1
4	6		4			4		4		
5	7	4	5	5	5	6	5		6	6
6	8		7		7	7		7		7
7	9					7		7	7	

Figure 9: Assigned processor after time-based load balancing.

ADTree is build from the *dual mesh*, including a single level of halo cells as shown in Figure 10. The reason for this is that the dual mesh formed by just the real cells on a block would leave gaps at the block-to-block boundaries. This means there is a layer of dual mesh cells that are duplicated on each of the two mesh blocks sharing a face. More duplications may occur at block edges and block corners. It is therefore possible for a search point to find identical donors on two or more different ADTrees. This apparent inconsistency does not, however, pose a problem due to the use of halo cells, since all potential donors would supply identical interpolated values.

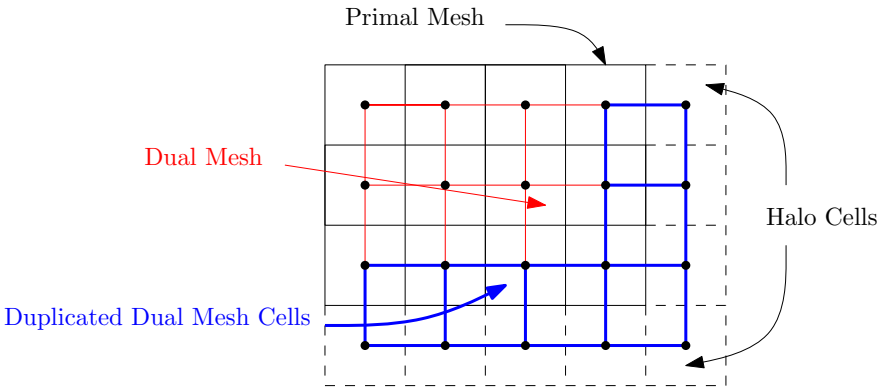


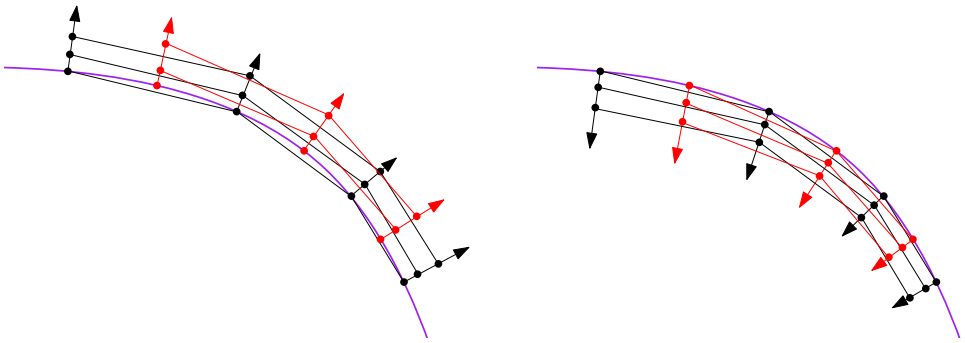
Figure 10: ADTree formed from the dual mesh cells.

The ADTree uses the concept of recursively grouped coordinate-aligned bounding boxes to very quickly determine the one or more cells for which a point could potentially lie inside. Once the list of potential cells is accumulated, a Newton search is performed on each potential donor to determine the u, v, w coordinates of the search point inside the donor cell's coordinate system. This procedure is fast and robust, usually requiring only 3 to 5 Newton steps to converge to machine precision. The ADTree permits fast spatial searches with complexity $\mathcal{O}(\log n)$ for n volume cells.

The ADTree data structure can be serialized such that it can be easily communicated to required processors. We must also assemble the search points. Since we are using a cell-centered solver, the search points are the cell centers of all real cells. Search points are treated separately from the ADTrees since there is much less memory required to store the search points than a serialzed ADTree.

Once each processor has assembled the ADTrees and search points, they are communicated with the processors that require them. This communication is overlapped with the actual searches such that the donor searches may be performed immediately upon receipt of the required ADTree and search points.

The core of the implicit hole cutting procedure is the searching of potential donors for all real cells. There is however, a slight complication that arises when two meshes with viscous off-wall spacing overlap on a common wall surface. This occurs even if the primal nodes of each surface mesh lie exactly on the true, continuous, underlying surface geometry. The discrete nature of the meshes may result in an inability to find the appropriate donor. Schematics of the two possible situations are shown in Figures 11a and 11b.



(a) Convex geometry (b) Concave geometry

Figure 11: Surface-surface overlap projection.

Suppose that the red mesh attempts to find donor cells on the black mesh. In the convex case, the search point

is offset inside the first black cell. If it found a donor cell, however, the parametric coordinate would correspond to the mid-cell height as opposed to the wall itself. This results in the incorrect transfer of solution information. For the concave case, once again, suppose the red mesh points attempts to find donors on the black mesh. In this case, the first mesh point lies *outside* the discrete representation of the black mesh geometry, resulting in an inability to find a valid donor. The solution to this issue is to project the desired search point onto the surface definition of the block we are searching for. This is effective in correcting for both convex and concave cases.

In an effort to reduce the required number of searches required for surface correction, we pre-compute the closest wall cell-center, x_{wall} for every block that contains a wall-type boundary condition. A point-based k-d tree is used for this search. Then, during the donor search, only a small number of surface cell centers must be projected onto the donor surface mesh. After all the surface wall points are projected, an offset vector, o, is computed for each (volume) search point according to the following equation:

$$d = |||x_p - x_{\text{wall}}|||o = max(1 - (d/d^*) * *3) * \Delta \tag{1}$$

where x_p is the search point, Δ is the vector projecting x_{wall} onto the donor wall mesh, and d^* is a user-supplied attenuation distance. Note that this example demonstrates surface correction effect using the primal mesh. In practice, the dual surface mesh must be used for surface-surface correction.

Once the offset vector is computed for each search point, potential donors can be found for each search point. Once all donor searches are completed, all potential donors for the search points are returned to the originating processor using the transpose communication structure of the overlap matrix.

Before the iterative phase of the algorithm is started, it is necessary to identify the cells that intersect solid walls. This is accomplished by recording the potential donors for all cells immediately next to walls and tagging the donors of these cells as wall donors.

The next phase of the algorithm is to determine the state of every cell in the mesh: Compute cell, receiver cell or blanked cell. This procedure has to be iterative for reasons that will become clear shortly. For this section of the IHC algorithm, it is useful to have a single "status" variable encoding the various potential attributes of every real and halo cell in the mesh. The following list the possible logical attributes of a cell a given cell in the mesh:

- isDonor
- isCompute
- isFloodSeed
- isFlooded
- isWallDonor
- isReceiver

The first step of the iterative procedure is the actual core of the IHC algorithm. For each real cell, its own cell volume is compared against all other potential donors. A few slight modifications are necessary to the basic idea. Firstly, on the second and subsequent iterations, flooded cells and flood seeds do not require donors and are thus skipped. Secondly, forced receiver cells, have their volume set to an arbitrary large number. This ensures any potential donor is be accepted, even if the donor cell volume is larger than the cell's own volume. The core IHC algorithm in detailed in Algorithm 2.

Returning to the two cylinder example, we can now look at the state of each cell after the receivers have been identified. (Figure 12).

Notice how the cells background mesh have been implicitly hole cut by the body with compute cells remaining inside. The next step is to remove the compute cells remaining inside the body. This procedure is known as flooding. The basic idea is to remove all simply connected regions inside the body by recursively flagging cells that have been identified as wall-donors, that is cells, that intersect solid walls. The flooding procedure is initialized from these seeds. The pseudo code for the flooding algorithm is given in Algorithm 3.

The flooding algorithm recursively floods all compute cells until an interpolated cell is encountered. It is therefore critical that the interior compute cells are fully surrounded by interpolated cells or the flood may "leak" and flood all blocks in the cluster.

After the flooding procedure, the state of the 4 meshes is now shown in Figure 13.

All the former compute cells on the background mesh have now been successfully flooded. This completes the necessary body hole cut for this mesh. However, on the two cylinder meshes, while the flood seeds surrounding the

Algorithm 2 Core IHC algorithm

```
for Each real cell do
    if IsFlooded or IsFloodSeed then
        skip
    end if
    curVolume = volume
    if isForcedReceiver then
        curVolume = Large
    end if
    for i = 1,nDonors do
        if DonorVolume[i] < curVolume then
            curVolume = donorVolume[i]
            set isReceiver=True
            set isCompute =False
        end if
    end for
end for
```

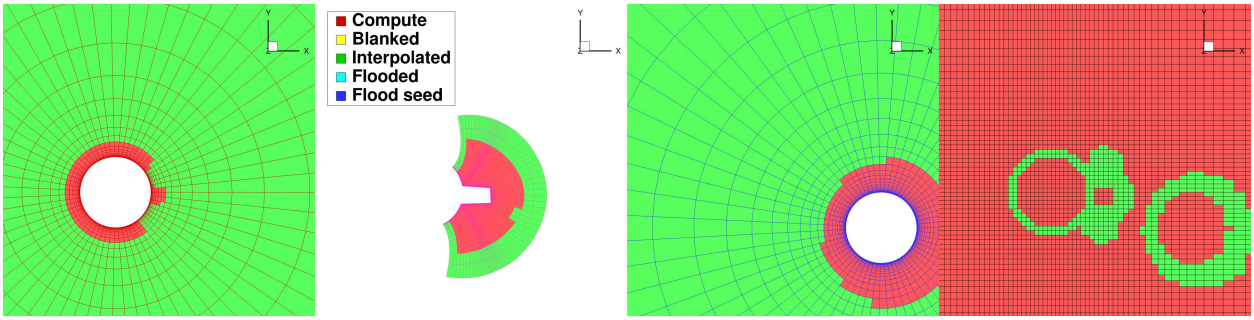


Figure 12: State of each cell after implicit hole cut.

Algorithm 3 Flood all interior cells.

```
LoopIter=1
loop
    nChanged=0
    if LoopIter=1 then
        Add all floodSeeds to stack
        Set Cells as floodSeed
    else
        Add seeds from halo cells to stack
    end if
    while Stack is not empty do
        Pop cell from stack
        if isCompute and not isReceiver then
            nChanged = nChanged + 1
            Set isFlooded=True
            Add 6 nearest neighbors to stack
        end if
    end while
    Exchange halos
    LoopIter = LoopIter + 1
    if nChanged = 0 then
        exit loop
    end if
end loop
```

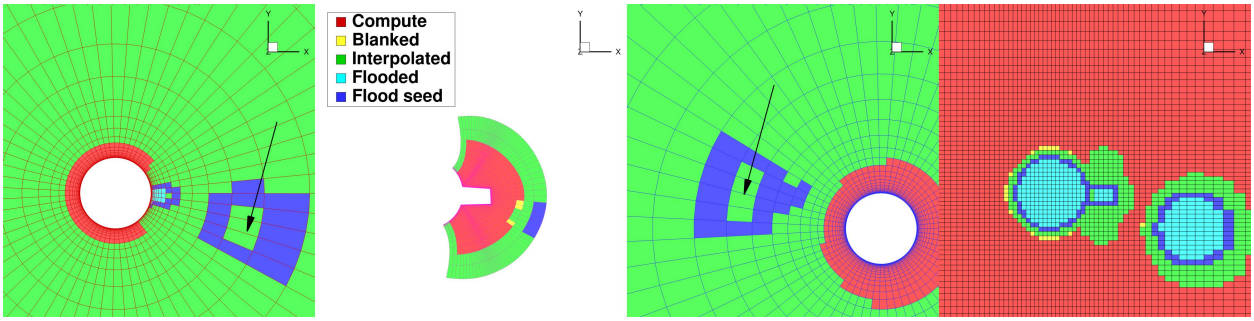


Figure 13: State of after flooding on first iteration. The arrows on the first and third figures show the interpolated cells that remain inside solid bodies.

solid bodies have been correctly identified, the interior cells remain as interpolated. The reason for this is that those interior cells have found donors on the background mesh. However, their donor cells have now been flooded and are no longer available as donors. This data dependency is the reason for the iteration loop: As cells become flooded, they are no longer valid donors and the interpolated cells using them as donors must find either a different donor or become a compute cell.

After the each cell determines if it is a receiver, the next step is to notify the cells supplying information to the receiver, that the cell is actually donor. This step requires another communication, and sets the `isDonor` flag on the required cells.

The next two steps are used to prepare for the next iteration by updating information that has potentially changed due to the flooding procedure. The first is flagging the forced receivers. In this step, cells surrounding flooded cells, flood seeds or the overset boundary condition are flagged. This is necessary to ensure compute cells do not use any blanked cells in their computational stencil. The computational stencil for the RANS equations in ADflow is shown in Figure 14. The second step is to identify donors that are no longer valid as a result of the flooding. The volume of the invalid donors is set to an arbitrary large value ensuring that the donor is never accepted.

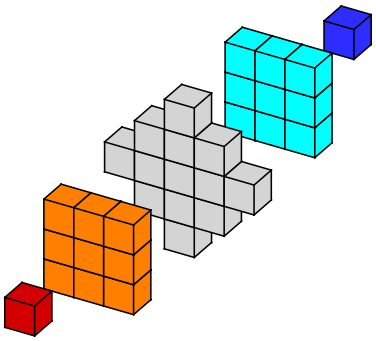


Figure 14: Computational stencil for the RANS equations.

The algorithmic step is known is irregular cell correction. It will generally happen during the search process that some cells will be identified both as a donor and a receiver, which is referred to as an irregular cell. Such a situation could result in mutually dependence interpolation stencils between block pairs and is thus not allowed. Irregular cells are corrected by converting the receiver to a compute cell.

Finally, at the end of the iteration, the status of every cell is checked against the status at the start of the loop and if no cell has changed status, the loop is broken. Several iterations of this loop are usually required to iteratively determine the status of each cell.

Returning now to the cylinder example, on the second iteration, the cylinder mesh cells inside the opposing cylinder cannot find donors since the flooded background mesh cells are now invalid. Thus they remain as compute cells and are subsequently flooded. The state of the cells at this point is shown in Figure 15.

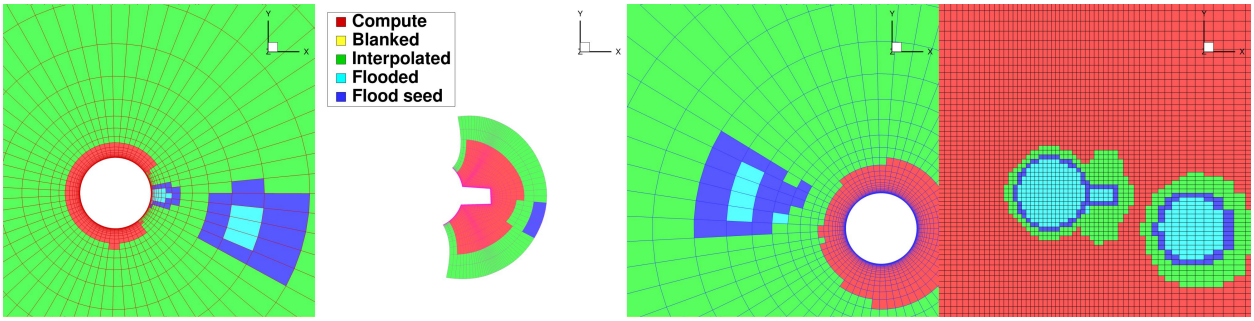


Figure 15: State of after flooding on second iteration.

For this simple example, no further changes in the cell status is needed. An additional iteration of the algorithm is required to confirm that there are no further changes. For more complex configuration with many more overlapping meshes, 5 or more iterations may be necessary to fully determine the state of every cell.

After the iterative section is completed, two final operations are performed: Fringe reduction and overset communication structures.

The goal of fringe reduction is to eliminate any interpolated cells that do not influence a compute cell. This reduces the amount of data that must be communicated at each iteration of flow solver, lowering the computational cost per iteration. The result of the fringe reduction operation is shown in Figure 16.

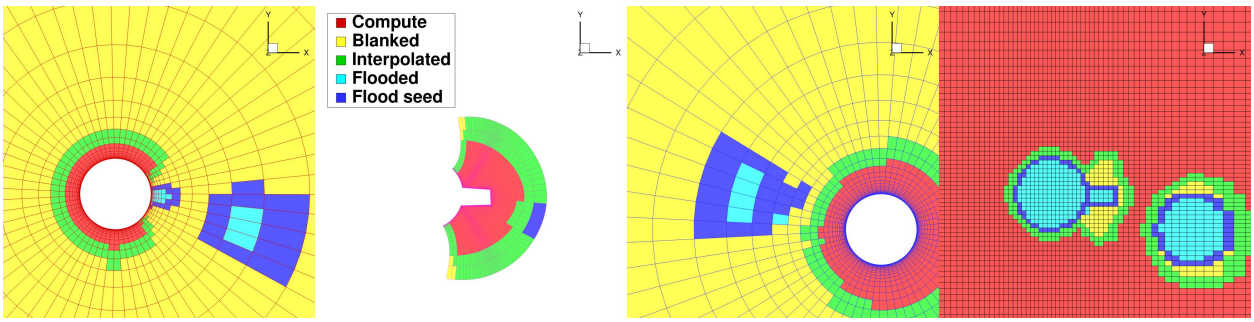


Figure 16: Final hole cut after fringe reduction.

In this case the total number of fringe cells from approximately 1,800 to 800. The final set of overlapping meshes, showing only the compute cells is given in Figure 17.

The final operation of the IHC algorithm is the generation of the final optimized communication data structure to be used during the CFD solution. The final composite overset mesh after successful completion of the IHC algorithm is given in Figure 17.

III. Overset Connectivity Scaling

In this section we examine the strong scaling behavior of the overset connectivity algorithm. The test case for the study is the wing-body Common Research Model configuration used for DPW6 (shown in Figure 18). The mesh contains approximately 14 million cells arranged in eight blocks: five near-field meshes and three hierarchically refined Cartesian background meshes.

The time required for the connectivity algorithm was recorded for processor counts ranging from 1 to 3,960. For each partitioning, three assemblies were perform: Without any load balancing (communicate only search points), load balancing based on the number of searches and finally load balancing based on time. The speed up for each of these methods are shown in Figure 19. A number of interesting trends are revealed. Firstly without load balancing, the maximum speedup observed is approximately 28. The primary reason for this speedup is due to the fact that as the computational blocks are split, dense rows (requiring a single set of donors to search for all other search points) in the overlap matrix becomes multiple, sparser rows. Therefore the maximum amount of work any one processor must

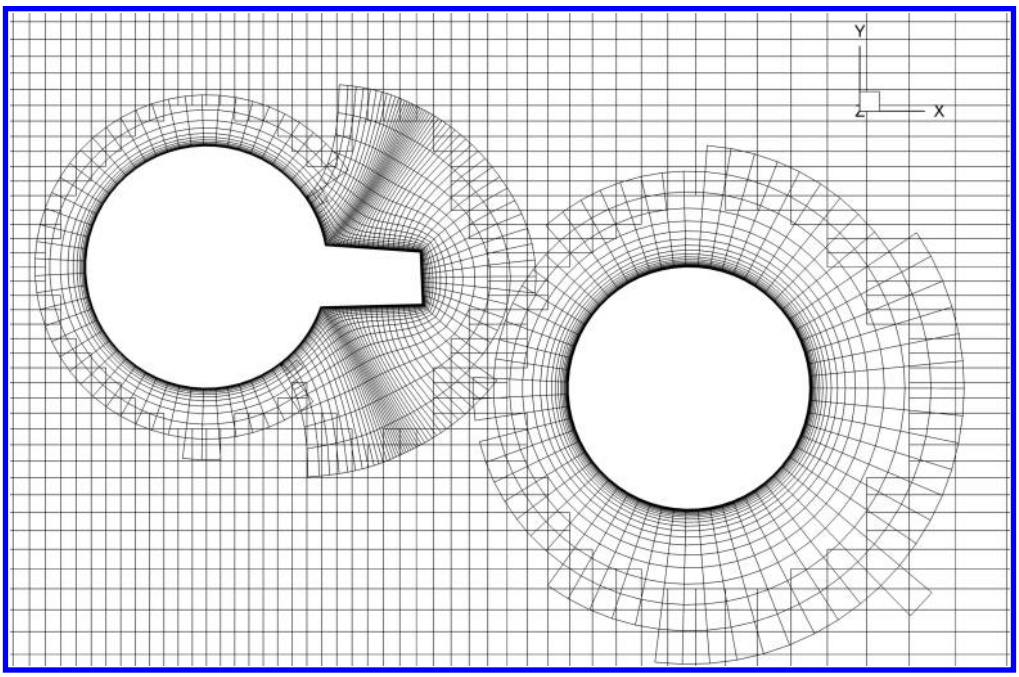


Figure 17: Final overset connectivity; only compute cells are shown.

perform decreases, but not as rapidly as the number of processors is increased. This limits the speed up, especially at the higher processor counts. In general, the search-point based load balancing performs better than the no load balancing case, with a couple of exceptions. There are very pronounced drops in performance for processor counts of 4, 24 and 128. This is due to small number of processors performing a disproportionate number of the donor searches. However, once the searches are re-partitioned based on the actual search times, the performance improves in all cases. The strong scaling limit for this case is between 1,980 and 3,960 processors. The overall speedup for these cases is 178.

To put the overset connectivity times in perspective, we compare the time required for overset assembly (with time-based load balancing) with the time required for a costly pre-processing step, the wall distance computation, and the core operation of one of the flow solution algorithms, a single single 5-stage Runge–Kutta iteration. The times are given in Table 1.

Table 1: Overset set assembly time compared to wall distance time and RK iteration.

Processors	Overset assembly (s)	Wall distance (s)	RK Iteration (s)	RK iterations/Overset assembly
32	10.62	69.98	3.06	3.47
256	2.44	18.67	.50	4.88

The overset assembly takes significantly less time that the wall distance computation, although this exact computation could be speed up considerably by relaxing the requirement of exact wall distance for cells outside the boundary layer region. A more important comparison is with the Runge–Kutta iteration. Using 256 processors, a typical processor count for this grid, the overset assembly requires less time than 5 cycles.

IV. Optimization on Overset Grids

The aerodynamic shape optimizations are performed with the MDO of Aircraft Configuration with High Fidelity (MACH) framework. This framework was developed to perform aerostructural design optimization studies of aircraft configurations [8, 9], and integrates modules for CFD, structural analysis, geometry parametrization, and numerical optimization. MACH has been used extensively for both aerodynamic shape [10, 11, 12, 13, 14] and aerostructural design optimization [15, 16, 17, 18] of aircraft and also hydrofoils [19]. Here we use only the aerodynamic capabilities

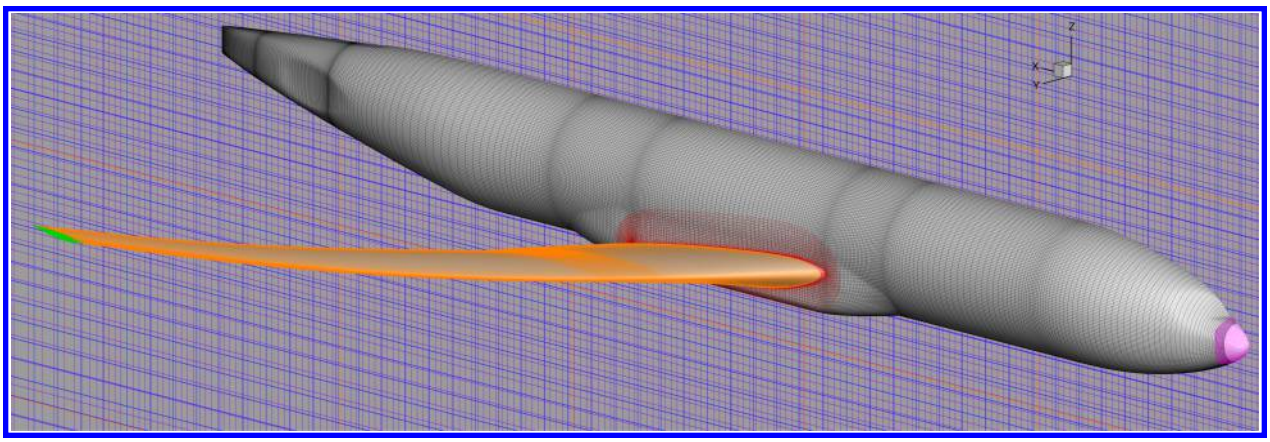


Figure 18: Wing-body mesh used for scaling study.

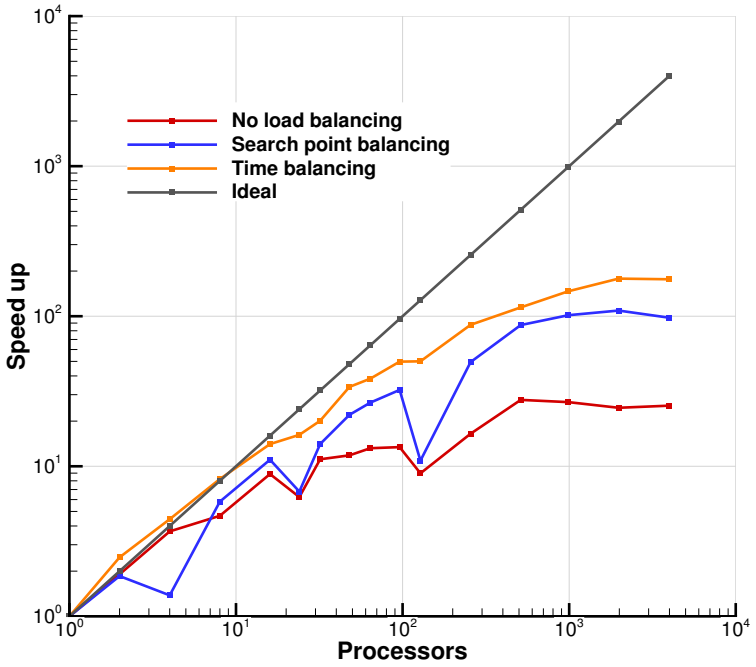


Figure 19: Strong scaling speedup for the IHC overset connectivity algorithm.

of MACH, which are described in the remainder of this section.

This is a lift-constrained single point drag minimization. A much more extensive optimization study of this configuration was performed by the authors [20]. The goal here is to examine the effect of using overset meshes on a well-known optimization problem.

1. Modifications to ADflow for Overset

The IHC approach described in Section II was implemented directly in the flow solver ADflow. ADflow, solves the RANS equations with a finite volume method in either steady, unsteady or time spectral modes [21, 22]. The discretization scheme uses central fluxes with artificial dissipation (scalar or matrix [23]) and the Spalart–Allmaras turbulence model [24]. A fully-coupled Newton–Krylov method is used to solve the mean flow and turbulence equations simultaneously.

One of the major advantages of using overset meshes with structured meshes is that once the overset connectivity information is established, very little modification to the underlying algorithms is required. The summary of the

modifications required for ADflow are the following:

1. Residuals are multiplied by `max(iblank(i,j,k), 0)`. This ensure all non-compute cells do not participate in the solution.
2. Addition of the communication routine for exchanging interpolated solution information.
3. For the implicit solution methods, an identity block must be added for non-compute cells to eliminate the zero-diagonal.
4. Linearization of the of interpolation stencil for preconditioning matrix.
5. Zipper mesh implementation.

All of the above modifications except for the zipper mesh required minimal modifications to the existing multiblock code. When overlapping surfaces are present in the geometry, integration of forces and moments over the solid boundary is no longer accurate due to double-counting the overlapping cell area. There are two main methods for dealing with this issue: the weighted panel approach [25] and the zipper mesh approach [26]. We implemented the zipper approach following the approach described by Chan [26].

ADflow contains a discrete adjoint method implemented using a combination of reverse mode automatic differentiation and analytic methods for the efficient computation of the gradients of functions of interest. Lyu *et al.* [27] describe the CFD adjoint implementation in more detail. As with the solution of the flow equations, little additional implementation is necessary to use overset meshes in an aerodynamic shape optimization. The main modifications are the following:

1. Addition of the *transpose* overset communication routine.
2. Linearization of the zipper surface integration routines.

For the optimizations presented below, we use the *frozen connectivity* approach, where the entire mesh is deformed, including the background mesh, using the mesh deformation algorithm described below. Since the same deformation field is applied uniformly, there is no need to update the overset connectivity. The approach ensures smooth functions for gradient-based optimization.

2. Geometric Parametrization

We use an free-form deformation (FFD) volume approach [28] that we implemented [29] and have used extensively in the past for aerodynamic [30, 31, 10, 11, 13, 32] and aerostructural optimization studies [8, 15, 9, 16]. The FFD approach can be visualized as embedding the spatial coordinates defining a geometry inside a flexible volume. The parametric locations corresponding to the baseline geometry are found using a Newton search algorithm. Once the baseline geometry has been embedded, perturbations made to the FFD volume propagate within the embedded geometry via the evaluation of the nodes at their parametric locations.

3. Mesh Movement

The FFD approach used to parameterize the geometry applies deformations only to the surface mesh, that is, the part of the volume mesh that lies on the physical surface. A separate procedure is then required to propagate surface perturbations to the remainder of the volume mesh. The mesh movement algorithm used in this work is an efficient analytic inverse distance method similar to the one described by Luke *et al.* [33]. Updating the mesh for a new configuration is fast, typically requiring less than 0.1% of the CFD solution time. Sensitivities required for the adjoint method are provided by a combination of reverse-mode automatic differentiation and analytic methods. Due to the unstructured nature of the mesh deformation algorithm, no additional modifications are necessary for use with overset grids.

4. Optimization Algorithm

The high computational cost of RANS-based optimization demands an optimization algorithm that calls the function evaluations as few times as possible. We use SNOPT (sparse nonlinear optimizer) [34] through the Python interface pyOpt [35]. SNOPT is a gradient-based optimizer that implements a sequential quadratic programming method; it is capable of solving large-scale nonlinear optimization problems with thousands of constraints and design variables. SNOPT uses an augmented Lagrangian merit function, and the Hessian of the Lagrangian is approximated using a quasi-Newton method. We have successfully employed the SNOPT algorithm to solve a wide variety of aerodynamic and aerostructural optimization problems [9, 16, 12, 11, 30].

5. Baseline Geometry

The baseline geometry is defined the Aerodynamic Design Optimization Discussion Group (ADODG) Case 5. The geometry is the Common Research Model "Wing-Body-Tail $i_H = 0$" configuration previously used in the 4^{th} Drag Prediction Workshop [36]. It is representative of a twin-aisle long range transport aircraft. The main reference parameters for the CRM are listed in Table 2.

Quantity	Value
Reference area	$594\,720.0$ in^2
Reference chord	275.8 in
Moment reference	$(1325.90\ ,\ 0,\ 177.95)$ in
Reynolds number ($M = 0.85$)	43×10^6

Table 2: Reference quantities for CRM full configuration [36].

6. Computational Meshes

We generated a series of multiblock and overset meshes for the CRM wing-body-tail configuration. The multiblock meshes were created completely using the meshing software ICEMCFD. The surface meshes for the overset grids were generated using ICEMCFD and the volume meshes extruded using an in-house hyperbolic mesh marching algorithm. Each set of grids contains two families: the "1 series" and the "0.5 series". Each "0.5 series" mesh has approximately 2.8 times more cells than the corresponding coarser "1 series" mesh below it, and has approximately 2.8 times fewer cells than the next finer "1 series" mesh above it. The two coarsest grids—L2 and L1.5—are used for optimization, while the finer grids–L1 and L0.5—are used only for post-optimization verification purposes. The mesh metrics are summarized in Table 3. A comparison between the surface mesh resolution of the multiblock and overset meshes are shown in Fig. 20. Note that the overset meshes are more refined than the corresponding multiblock mesh. There are two reasons for this. Firstly, it becomes increasingly difficult to generate very coarse overset meshes due to the overlap requirements. For this reason, the coarsest overset mesh is more resolved than the L2 multiblock mesh. Secondly, the overset meshes have a higher surface resolution as it possible to resolve areas of interest with the increased number of nodes propagating throughout the entire mesh.

Mesh type	Mesh level	Chord-wise Cells	Span-wise cells	$y^+_{\max}$	Total cells	Baseline scalar C_D (counts)	Baseline matrix C_D (counts)
Multiblock	L0.5	224	144	~ 0.5	14 233 600	235.28	231.50
	L1	168	108	~ 0.7	5 921 536	242.40	235.27
	L1.5	112	72	~ 1.1	1 779 200	270.24	251.53
	L2	84	54	~ 1.7	740 192	300.12	273.27
Overset	L1	260	154	~ 0.7	9 822 874	229.15	227.13
	L1.5	184	110	~ 1.2	3 629 280	232.46	228.19
	L2	130	77	~ 1.8	1 232 948	239.35	230.39

Table 3: Mesh characteristics and corresponding trimmed drag coefficient for baseline configuration.

7. Optimization Problem Statement

A series of optimizations were performed on both overset and multiblock meshes using both scalar and matrix dissipation. The goal is to investigate the effects of the mesh type and artificial dissipation scheme on the optimized designs.

The optimization problem corresponds to Case 5.1 as defined by the ADODG [37]. The optimization problem

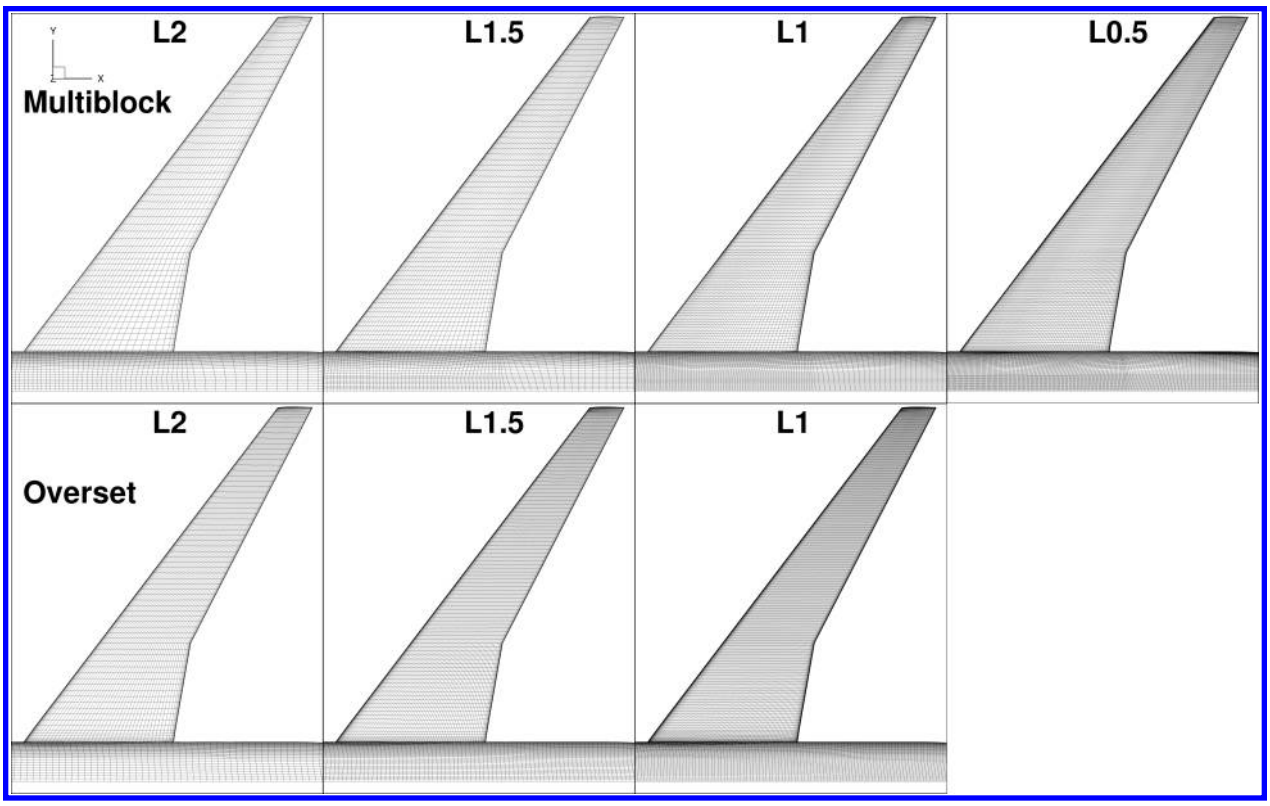


Figure 20: Spatial resolution for each mesh size.

statement can be written as:

$$\begin{array}{rll} \text{minimize} & C_D & \text{Quantity} \\ \text{with respect to} & \text{Wing cross sectional shape} & 240 \\ & \text{Wing twist} & 9 \\ & \text{Angle of attack } (\alpha) & 1 \\ & \text{Tail rotation angle } (\eta) & 1 \\ \text{subject to} & C_L - C_L^* = 0.0 & 1 \\ & C_{M_y} = 0.0 & 1 \\ & t_j \geq t_{j_{\text{CRM}}} & 750 \end{array} \tag{2}$$

The lift and moment coefficient constraints ensure that the aircraft is trimmed which can be achieved by the appropriate combination of angle of attack and tail rotation angle. The thickness t_j is computed at 750 points arranged in a 25×30 regular grid in the chordwise and spanwise directions, respectively. These thicknesses are constrained to be greater or equal than the original thicknesses of the CRM geometry at the corresponding points. Since it is desirable in transonic flow to make the wing as thin as possible [14], these constraints ensure that the wing does not become too thin, which would result in a significant structural weight increase. The consequence of the thickness constraints is that only changes in the wing camber are available to the optimizer. The single operation condition is given in Table 4.

Table 4: Operating condition for optimizations

Case	Mach	C_L	Re
5.1	0.85	0.500	43.00×10^6

The ADODG specification for Case 5 requires the parameterization not to modify the planform, and any shape modification must be made only in the vertical direction. Additionally, twist rotation is permitted for the wing, as well as a solid-body rotation of the horizontal tail for trimming the aircraft. A graphical description of the design variables

used in the optimizations is given in Figure 21.

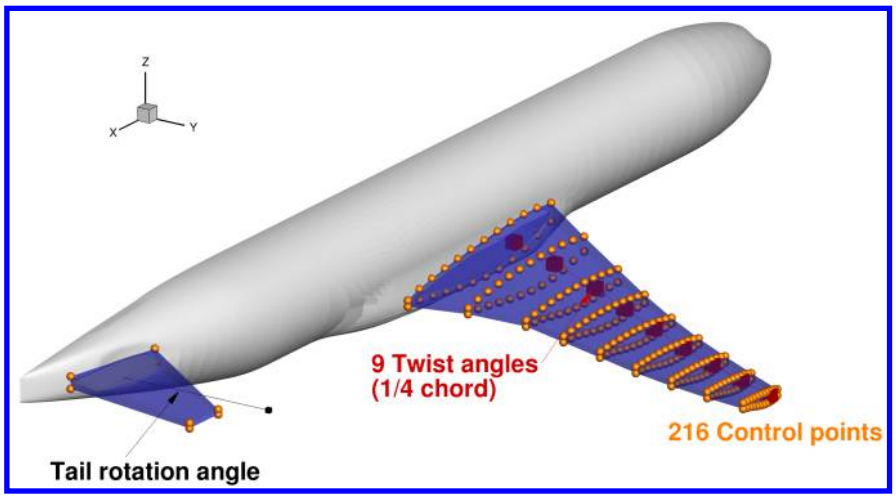


Figure 21: CRM configuration showing the shape, twist, and tail rotation design variables.

8. Optimization Results

To reduce the overall computational cost of performing the optimizations, we employ the multilevel optimization approach described previously by the authors [11, 14]. Optimizations are first carried out on the coarsest mesh (L2), and the resulting optimum design becomes the starting design for the next finer mesh (L1.5). In this work, only the first two grid levels are used for optimization.

Figure 22 shows the evolution of the SNOPT merit function and optimality for the two mesh types and two discretization types. The merit function is the value of the augmented Lagrangian given by SNOPT, which becomes the same as the objective function value once all the constraints are satisfied towards the end of the optimizations. The optimality is the residual of the Karush–Kuhn–Tucker (KKT) optimality conditions, and it is thus a measure of how well the optimization has converged [34].

Between one and two orders of magnitude reduction in the optimality condition is obtained for each of the coarse and fine grid optimizations. The absolute drag values vary considerably: The initial drag value for the multiblock mesh with scalar dissipation is 300 counts, while the initial value with matrix dissipation is approximately 230. Despite the wide difference in absolute drag values, the L2 and L1.5 optimizations for all cases show similar behavior: A 7-9 drag count reduction on the coarse mesh followed by a much smaller improvement on the finer mesh. This behavior highlights the merit of the multilevel approach as most of the potential improvement can be obtained by using a coarser mesh with lower computational cost.

Due to the large differences in absolute values in between the various meshes, it is beneficial to perform a grid refinement study to estimate what the drag would be with zero-mesh spacing. We performed drag convergence studies on both the baseline and optimized designs with all four combinations of mesh type and artificial dissipation. These results are given in Figure 23. Note that for each case, the configuration has the same C_L and $C_M = 0.0$ is enforced by rotating the tail.

The extrapolated drag value for the baseline solution using the 4 different grid-dissipation combinations, all converge to values within 1 count of each other. There is however, a extremely large variation in the slope of these lines, highlighting the detrimental effect of the non-smooth multiblock grid as and the highly dissipative nature of the scalar artificial dissipation scheme. All of the optimized designs show drag reduction compared to the baseline design. The multiblock-scalar optimization showed the smallest improvement, followed by the multiblock-matrix optimization and the two overset optimizations yielded almost identical improvements.

To further better understand the differences in each of the optimized designs, we performed a cross validation study. A grid second grid convergence study for the multiblock-scalar, multiblock-matrix and overset-scalar cases were run using the overset-mesh with matrix dissipation. The Richardson extrapolation curves for each of the designs as well as the original extrapolation curves are shown in Figure 24.

The Richardson extrapolation curves for each case are remarkably similar, differing by approximately 0.2 counts. This provides further confidence that the true solution to the PDE with no numerical error is very close to the Richard-

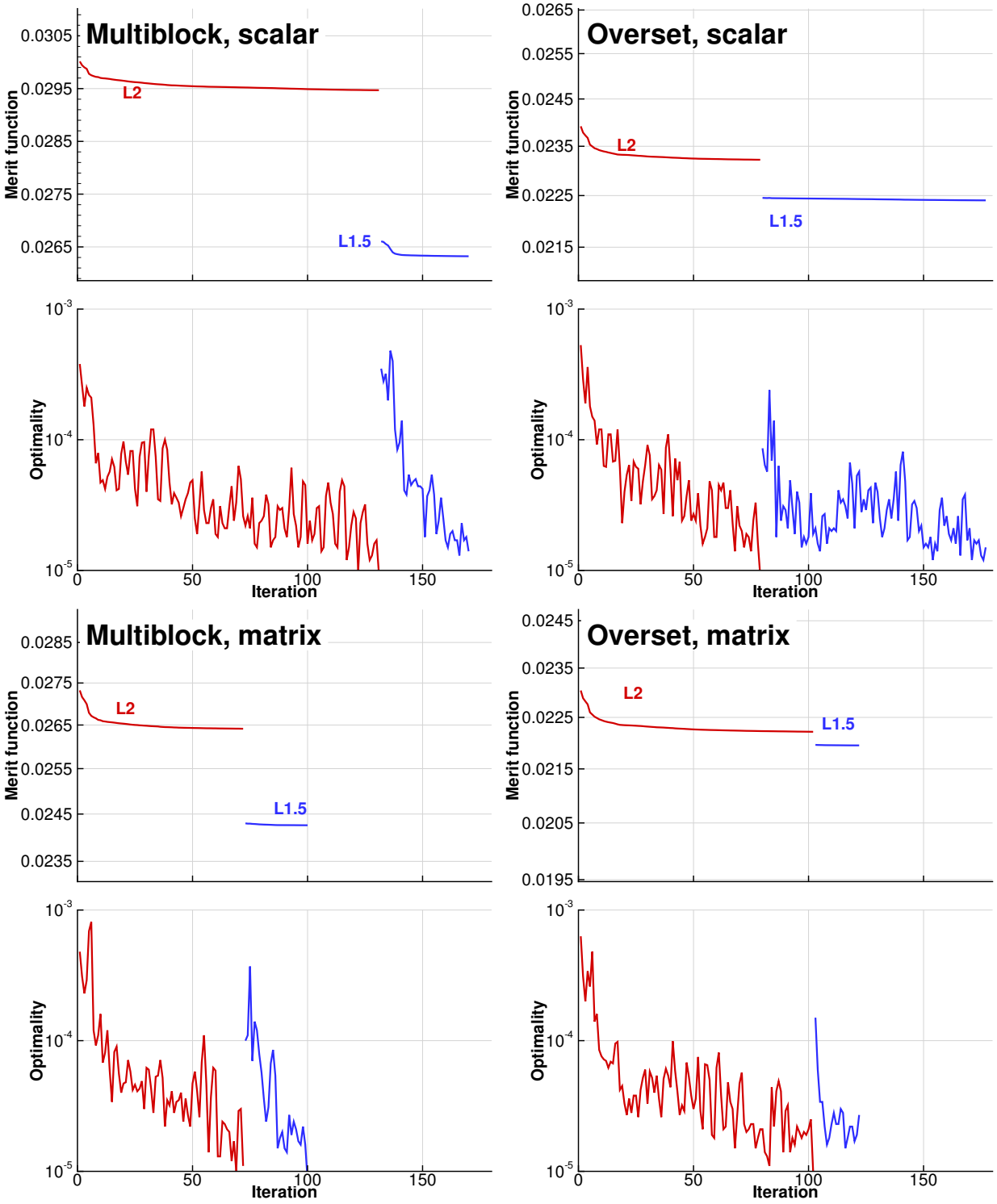


Figure 22: Merit function and optimality evolution for each optimization case.

son extrapolated values. It is now clear that the two designs obtained from the overset meshes exhibit better convergence characteristics than the multiblock-optimized designs. In fact, the overset-scalar optimized design has a slightly lower extrapolated drag value that the overset-matrix design. This is most likely due to the fact that this optimization executed more design iterations on the L1.5 mesh than the corresponding overset-matrix solution (Figure 22).

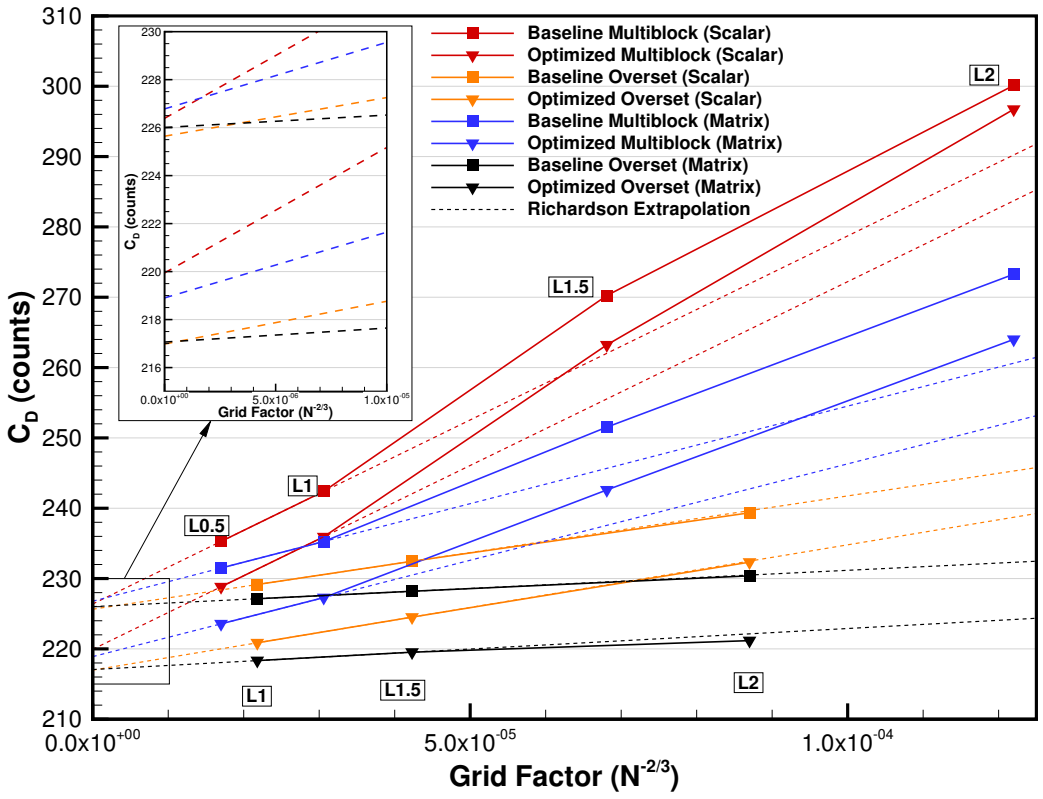


Figure 23: Grid convergence study for baseline and optimized configurations.

Taken together, these results demonstrate the need for high quality meshes and accurate numerical methods for optimization. The increased flexibility in overset mesh generation allows for the generation of high quality structured meshes in terms of orthogonality and smoothness.

For the multiblock meshes, the choice of numerical scheme had a measurable impact on the optimized design, resulting in approximately 1 drag count difference. However, for the overset meshes, the choice of numerical scheme had little impact, with both optimizations yielding designs with essentially identical performance.

A comparison of the surface C_p distributions for each of the designs is shown in Figure 25. The left column shows the optimized design as analyzed with the finest grid for its own grid type and solution method. These solutions correspond to the finest grid solutions in Figure 23. These solutions are compared with the finest cross-checked solutions from Figure 24.

Compared to the baseline design given in the top row, all designs were able to significantly reduce the shock strength yielding drag reduction. However, a closer examination shows that both multiblock solutions have weak shocks on the cross-checked solutions that are not present on overset solutions. The overset-scalar solution however, shows very little difference when analyzed with the matrix scheme, maintaining its shock free solution.

Figure 26 shows four cross sectional shapes as well as the corresponding C_p distributions. The z scale is enlarged to show the small variations in shape that distinguish the different designs. The $2y/b = 0.67$ section clearly shows the weak shock on the two multiblock meshes that is not present on the overset mesh optimized designs.

V. Hovering Helicopter Rotor

The overset mesh implementation as well as the flow and adjoint solver is further evaluated in the context of optimization of a helicopter rotor. The hovering rotor set up of Caradonna and Tung [38] is considered in the present work as a benchmark case. The rotor considered is a two-bladed rigid rectangular untwisted rotor composed of naca0012 airfoil sections. The rotor radius is 1.143m with an aspect ratio of 6. The rotor has a precone angle of $0.5°$. The rotational speed of the rotor corresponds to a tip Mach number $M_{tip} = 0.439$ and rated thrust coefficient value of $C_T = 0.00459$. The tip Reynolds number is 1.96 million.

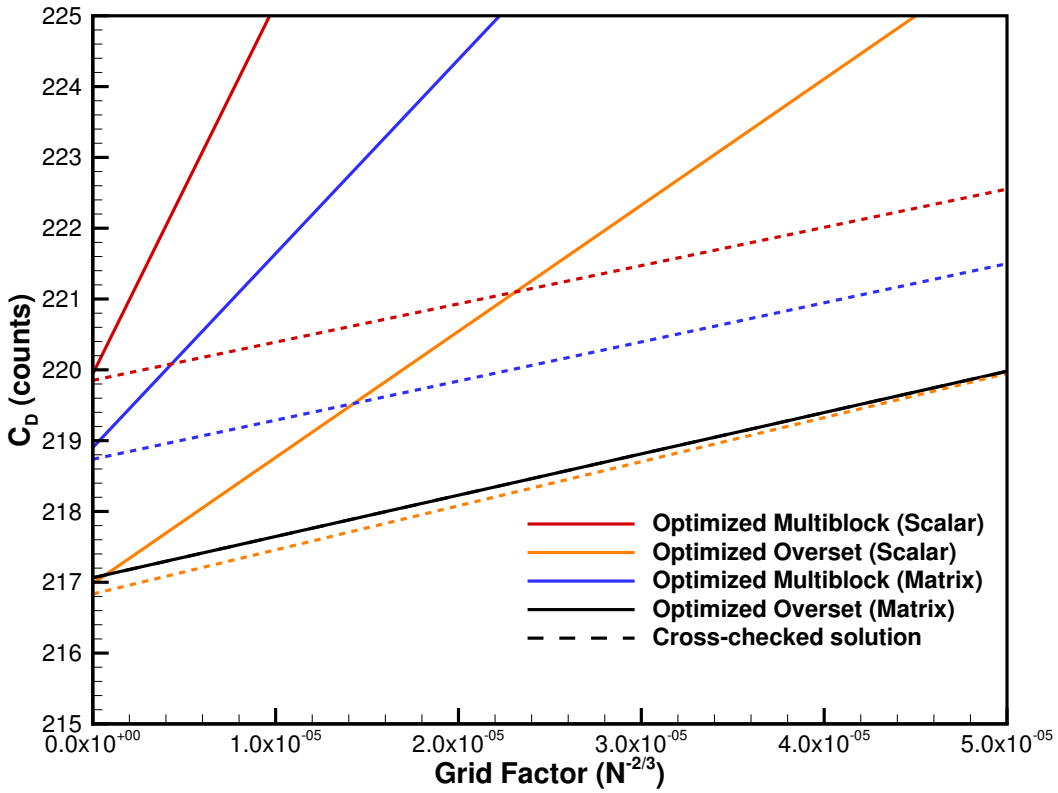


Figure 24: Optimized design cross-checked using overset mesh with matrix-dissipation.

Figure 27 shows the computational domain with overset connectivity. The computational domain consists of multiple mesh domains; a structured blade mesh for the near field flow solutions, a Cartesian background to resolve the rotor wake and another Cartesian peg mesh at the rotation axis to resolve the root flow field. For performance validation, a fine mesh with a total 10.64 million cells is used. However, in the interest of computationally efficient optimization evaluations, a coarser mesh with 1.33 million cells (every other cell in the fine mesh) is considered.

Figure 28 validates the predicted surface pressure against the experimental values at several spanwise stations of the rotor. Table 5 compares the fine and coarse mesh discretizations for the thrust and torque coefficients.

Table 5: Rotor Thrust Verification and Validation. Experimental $C_T = 0.00459$.

Mesh cells	C_T	C_Q
1.33×10^6	0.00423	0.00071
10.64×10^6	0.00455	0.00056

1. Towards Rotor Optimization

A constrained optimization problem is considered using the hovering rotor set up described above. The objective of the optimization is to minimize C_Q with rotor twist (ξ) as design variables, with a target thrust constraint. The constrained minimization problem can be stated as:

$$\min_{\xi} \quad C_Q(\xi) \tag{3}$$

$$\text{Subject to :} \quad C_T(\xi) - C_T^* = 0.0 \tag{4}$$

$$\text{With bounds :} \quad -15^\circ \leq \xi \leq 15^\circ$$

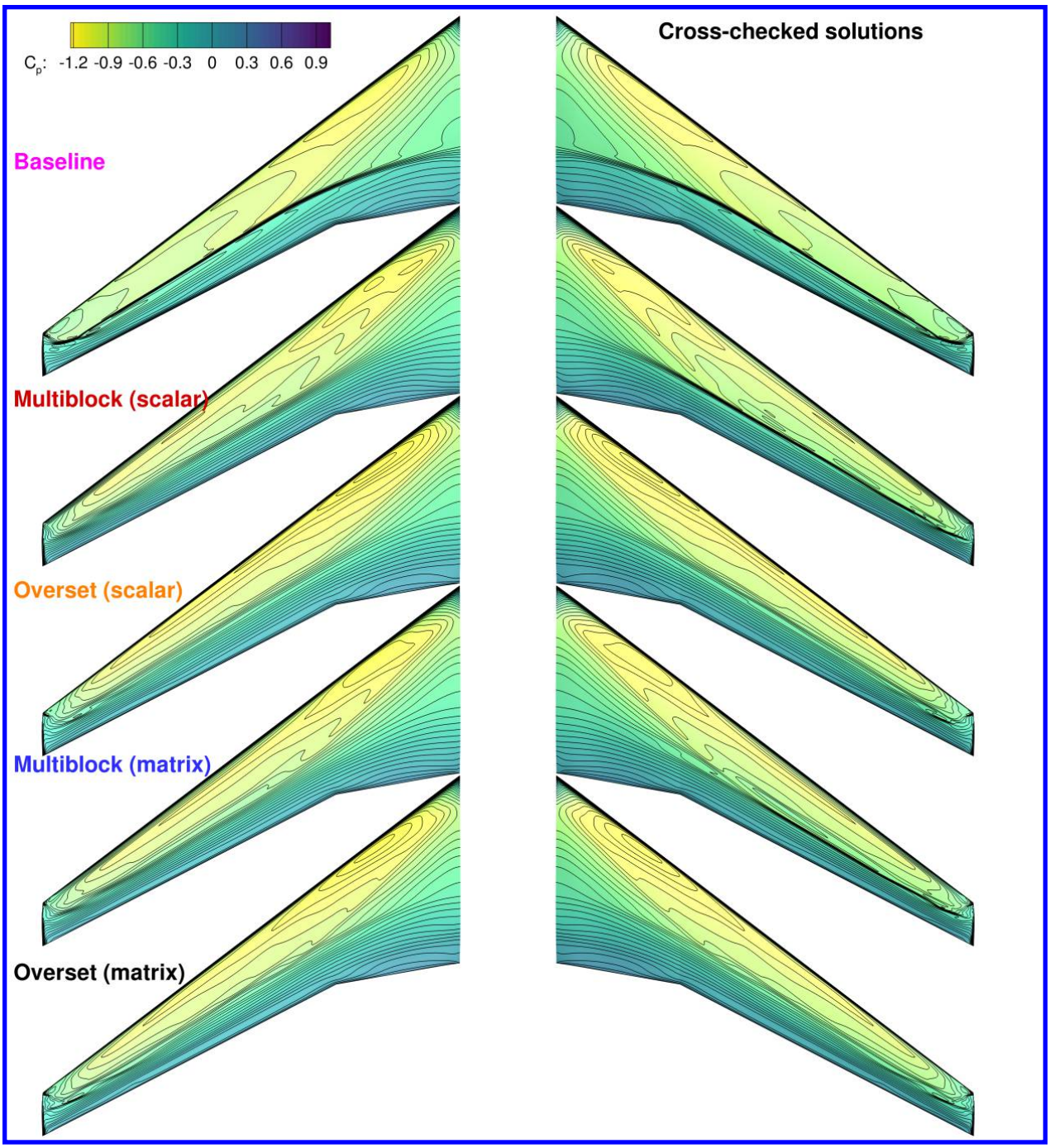


Figure 25: Wing surface C_p comparison with finest mesh (left). Cross checked solutions using L1 overset mesh with matrix dissipation.

where, C_T^* is the target thrust value of the rotor, 0.00459. Geometric parameterization for the rotor is achieved using a FFD volume consisting of 8 spanwise stations – see Figure 29a. For the present optimization problem, the root sections are held fixed to achieve relative twist along the span compared to the baseline untwisted blade. Figure 29b shows the baseline Cp contours. The tip region shows high suction peak values near leading edge upper surface, resulting in larger thrust contributions. The tip region also experiences significant compressibility effects. Figure 30 shows the torque as well as the thrust surface normal sensitivities from the first design iteration. The sensitivities of torque near leading edge and trailing edge suggest a general pitch down, as expected from ideal rotor theory. Also, the maximum

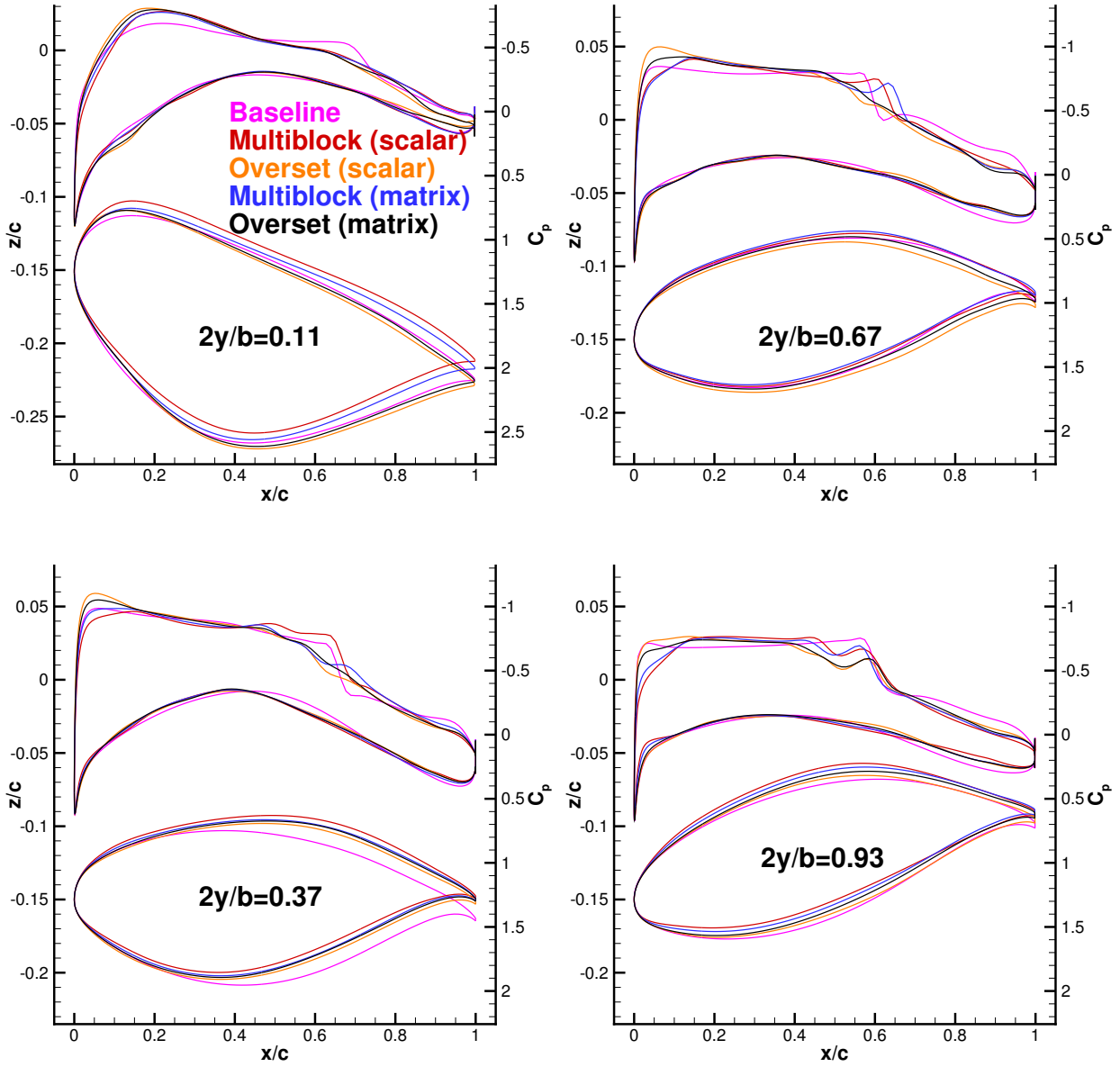


Figure 26: Optimized design cross-checked using overset mesh with matrix-dissipation.

twist sensitivities exist at the outboard stations. The thrust sensitivities near the leading and trailing edge regions show opposite trends compared to the twist sensitivities.

VI. Conclusion

We presented a fully automatic implicit hole cutting approach for overset grid connectivity of multiblock meshes. One of the main issues limiting the scalability of the overset grid connectivity methods is the performing appropriate load balancing. A unique time-based load balancing method when combined with the ability to fully communicate ADTrees and search points results in far better speedups that can be obtained without load balancing. A strong scaling study with a 14M cell mesh showed a maximum speedup of 178 on 1,980 cores.

The use of the overset mesh technique was demonstrated on a representative aerodynamic shape optimization of a transonic wing-body-tail configuration. Four sets of optimizations were performed, investigating the effect of mesh

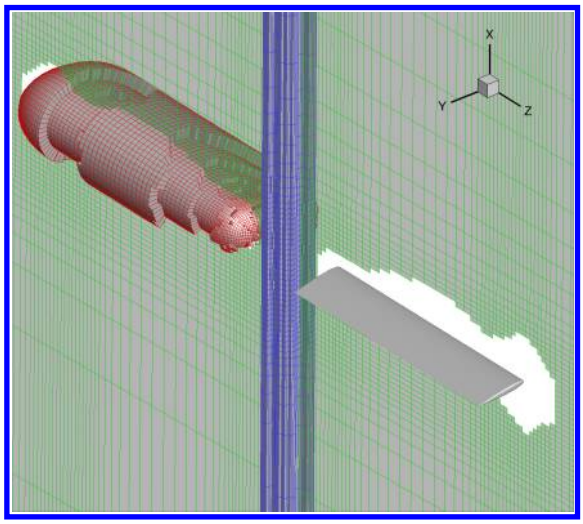
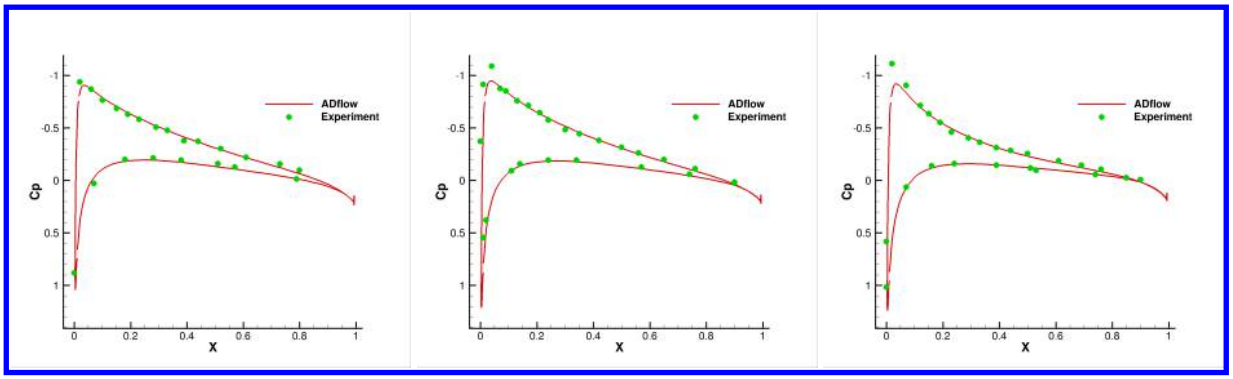
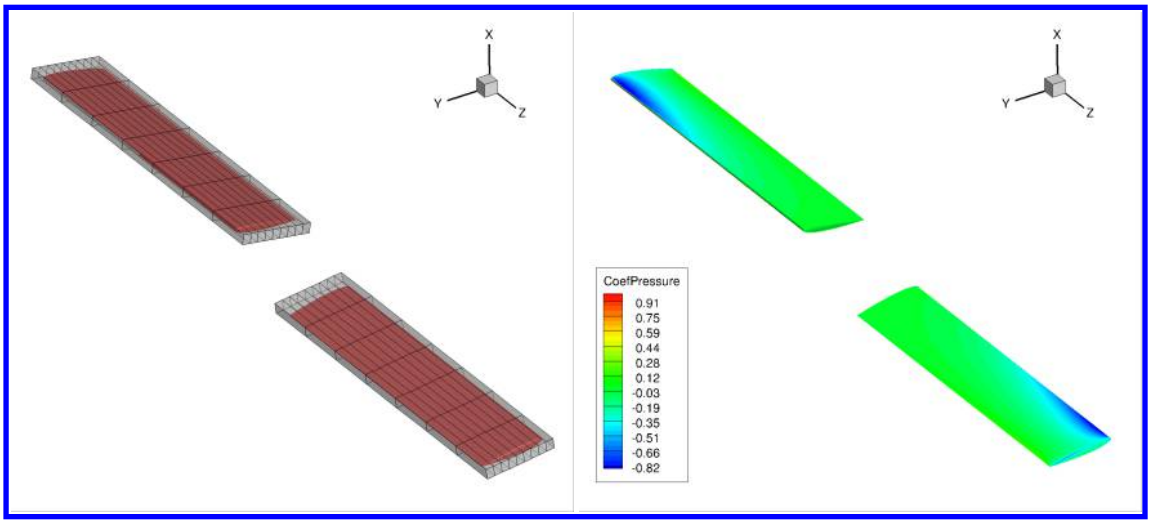


Figure 27: Overset mesh system for Caradonna and Tung rotor [38] at 8° collective pitch



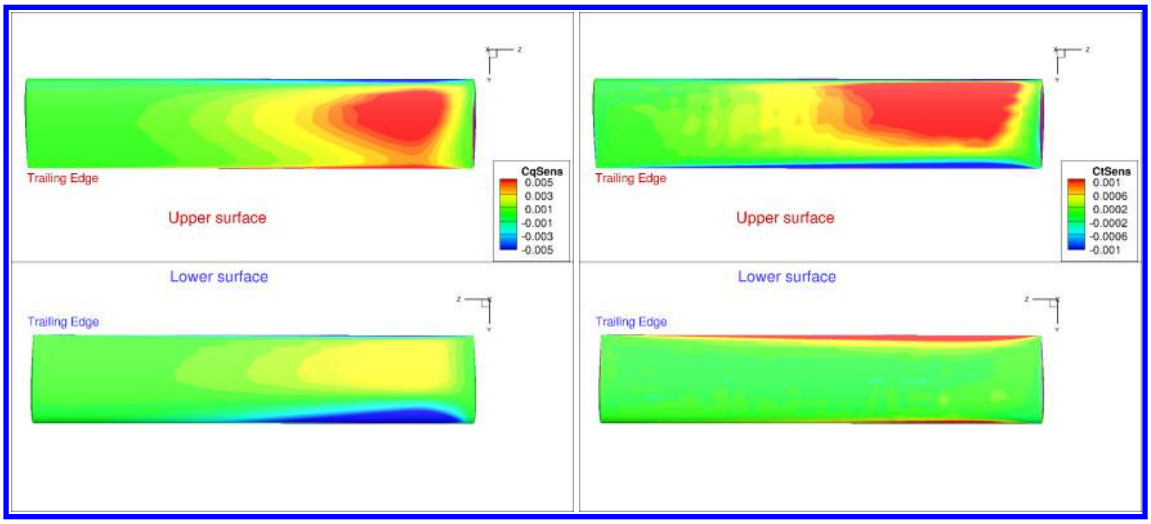
(a) Cp at $r/R = 0.68$ (b) Cp at $r/R = 0.80$ (c) Cp at $r/R = 0.96$

Figure 28: Cp comparison on Caradonna and Tung rotor at 8° collective



(a) FFD Volume (b) Cp contours of baseline geometry

Figure 29: FFD box and surface pressure contours of baseline geometry



(a) C_Q sensitivity (b) C_T sensitivity

Figure 30: Twist: Optimization convergence and torque surface sensitivity contours

type (multiblock and overset), as well as type of artificial dissipation (scalar or matrix). Optimizations performed with overset-meshes produced optimal drag values that were nearly independent of type of artificial dissipation. In contrast the optimized drag value for the multiblock matrix dissipation combination was approximately one count higher, while the scalar-based optimziation value was approximately 3 counts higher.

The results highlight the requirement of performing optimization on sufficiently resolved spatial discretization, as well the importance of smooth and nearly orthogonal meshes. The ability to generate quickly and easily generate high quality component-based meshes for the overset grid technique is highly desirable for aerodynamic shape optimization.

References

[1] Mishra, A., *Coupled CFD/CSD Prediction of the Effects of Leading Edge Slat on Rotor Performance*, Ph. d. thesis, University of Maryland, 2012.

[2] Duraisamy, K., Sitaraman, J., and Baeder, J., "High Resolution Wake Capturing Methodology for Accurate Simulation of Rotor Aerodynamics," *61st Annual Forum of the American Helicopter Society*, Dallas, Texas, June 8–11 2005.

[3] Sitaraman, J., Potsdam, M., Jayaraman, B., Datta, A., Wissink, A., Mavriplis, D., and Saberi, H., "Capability Enhancements in Version 3 of the Helios High Fidelity Rotorcraft Simulation Code," *49th AIAA Aerospace Sciences Meeting and Exhibit*, Orlando, FL, January 4–7 2011, AIAA Paper 2011-1123.

[4] Mavriplis, D. J., "Unstructured Grid Techniques," *Annual Review of Fluid Mechanics*, Vol. 29, 1997, pp. 473–514.

[5] Lakshminarayan, V. K., *Computational Investigation of Microscale Coaxial Rotor Aerodynamics in Hover*, Ph. d. thesis, University of Maryland, 2009.

[6] Landmann, B. and Montagnac, M., "A highly automated parallel Chimera method for overset grids based on the implicit hole cutting technique," *Journal for Numerical Methods in Fluids*, Vol. 66, 2011, pp. 778–804.

[7] Lee, Y. and Baeder, J. D., "Implicit Hole Cutting — A New Approach to Overset Grid Connectivity," *16th AIAA Computational Fluid Dynamics Conference*, Washington, DC, June 2003.

[8] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Scalable Parallel Approach for High-Fidelity Steady-State Aeroelastic Analysis and Derivative Computations," *AIAA Journal*, Vol. 52, No. 5, May 2014, pp. 935–951. doi:10.2514/1.J052255.

[9] Kenway, G. K. W. and Martins, J. R. R. A., "Multipoint High-Fidelity Aerostructural Optimization of a Transport Aircraft Configuration," *Journal of Aircraft*, Vol. 51, No. 1, January 2014, pp. 144–160. doi:10.2514/1.C032150.

[10] Lyu, Z. and Martins, J. R. R. A., "Aerodynamic Design Optimization Studies of a Blended-Wing-Body Aircraft," *Journal of Aircraft*, Vol. 51, No. 5, September 2014, pp. 1604–1617. doi:10.2514/1.C032491.

[11] Lyu, Z., Kenway, G. K., and Martins, J. R. R. A., "Aerodynamic Shape Optimization Investigations of the Common Research Model Wing Benchmark," *AIAA Journal*, Vol. 53, No. 4, April 2015, pp. 968–985. doi:10.2514/1.J053318.

[12] Lyu, Z. and Martins, J. R. R. A., "Aerodynamic Shape Optimization of an Adaptive Morphing Trailing Edge Wing," *Journal of Aircraft*, Vol. 52, No. 6, November 2015, pp. 1951–1970. doi:10.2514/1.C033116.

[13] Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration," *Journal of Aircraft*, Vol. 53, No. 1, January 2016, pp. 276–293. doi:10.2514/1.C033328.

[14] Kenway, G. K. W. and Martins, J. R. R. A., "Multipoint Aerodynamic Shape Optimization Investigations of the Common Research Model Wing," *AIAA Journal*, Vol. 54, No. 1, January 2016, pp. 113–128. doi:10.2514/1.J054154.

[15] Liem, R., Kenway, G. K. W., and Martins, J. R. R. A., "Multimission Aircraft Fuel Burn Minimization via Multipoint Aerostructural Optimization," *AIAA Journal*, Vol. 53, No. 1, January 2015, pp. 104–122. doi:10.2514/1.J052940.

[16] Kenway, G. K. W., Kennedy, G. J., and Martins, J. R. R. A., "Aerostructural Optimization of the Common Research Model Configuration," *15th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Atlanta, GA, June 2014, AIAA 2014-3274.

[17] Brooks, T. R., Kennedy, G. J., and Martins, J. R. R. A., "High-fidelity Aerostructural Optimization of a High Aspect Ratio Tow-steered Wing," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, January 2016. doi:10.2514/6.2016-1179.

[18] Burdette, D. A., Kenway, G. K., and Martins, J. R. R. A., "Performance Evaluation of a Morphing Trailing Edge Using Multipoint Aerostructural Design Optimization," *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, American Institute of Aeronautics and Astronautics, January 2016. doi:10.2514/6.2016-0159.

[19] Garg, N., Kenway, G. K. W., Lyu, Z., Martins, J. R. R. A., and Young, Y. L., "High-fidelity Hydrodynamic Shape Optimization of a 3-D Hydrofoil," *Journal of Ship Research*, Vol. 59, No. 4, December 2015, pp. 209–226. doi:10.5957/JOSR.59.4.150046.

[20] Kenway, G. K. and Martins, J. R. R. A., "Aerodynamic Shape Optimization of the CRM Configuration Including Buffet-Onset Conditions," *54th AIAA Aerospace Sciences Meeting*, American Institute of Aeronautics and Astronautics, San Diego, CA, January 2016. doi:10.2514/6.2016-1294.

[21] van der Weide, E., Kalitzin, G., Schluter, J., and Alonso, J. J., "Unsteady Turbomachinery Computations Using Massively Parallel Platforms," *Proceedings of the 44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, 2006, AIAA 2006-0421.

[22] Mader, C. A., Martins, J. R. R. A., Alonso, J. J., and van der Weide, E., "ADjoint: An Approach for the Rapid Development of Discrete Adjoint Solvers," *AIAA Journal*, Vol. 46, No. 4, April 2008, pp. 863–873. doi:10.2514/1.29123.

[23] Swanson, R. C., Radespiel, R., and Turkel, E., "Comparison of several dissipation algorithms for central difference schemes," *in AIAA 13th Computational Fluid Dynamics Conference, AIAA, 1997 (AIAA Paper*, pp. 97–1945.

[24] Spalart, P. and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting and Exhibit*, 1992. doi:10.2514/6.1992-439.

[25] Boger, D. A. and Dreyer, J. J., "Prediction of Hydrodynamic Forces and Moments for Underwater Vehicles Using Overset Grids," *44th AIAA Aerospace Sciences Meeting and Exhibit*, Reno, Nevada, Januaray 2006, AIAA Paper 2006-1148.

[26] Chan, W. M., "Enhancements to the Hybrid Mesh Approach to Surface Loads Integration on Overset Structured Grids," *19th AIAA Computational Fluid Dynamics Conference*, San Antonio, Texas, June 2009, AIAA Paper 2009-3990.

[27] Lyu, Z., Kenway, G. K., Paige, C., and Martins, J. R. R. A., "Automatic Differentiation Adjoint of the Reynolds-Averaged Navier–Stokes Equations with a Turbulence Model," *21st AIAA Computational Fluid Dynamics Conference*, San Diego, CA, Jul. 2013. doi:10.2514/6.2013-2581.

[28] Sederberg, T. W. and Parry, S. R., "Free-form Deformation of Solid Geometric Models," *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, Aug. 1986, pp. 151–160. doi:10.1145/15886.15903.

[29] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, Sept. 2010. doi:10.2514/6.2010-9231, AIAA 2010-9231.

[30] Mader, C. A. and Martins, J. R. R. A., "Stability-Constrained Aerodynamic Shape Optimization of Flying Wings," *Journal of Aircraft*, Vol. 50, No. 5, September 2013, pp. 1431–1449. doi:10.2514/1.C031956.

[31] Mader, C. A. and Martins, J. R. R. A., "Computing Stability Derivatives and their Gradients for Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 52, No. 11, November 2014, pp. 2533–2546. doi:10.2514/1.J052922.

[32] Kenway, G. K. W. and Martins, J. R. R. A., "Multipoint Aerodynamic Shape Optimization Investigations of the Common Research Model Wing," *Proceedings of the AIAA Science and Technology Forum and Exposition (SciTech)*, Kissimmee, FL, January 2015. doi:10.2514/6.2015-0264.

[33] Luke, E., Collins, E., and Blades, E., "A Fast Mesh Deformation Method Using Explicit Interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, Jan. 2012, pp. 586–601. doi:10.1016/j.jcp.2011.09.021.

[34] Gill, P. E., Murray, W., and Saunders, M. A., "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal of Optimization*, Vol. 12, No. 4, 2002, pp. 979–1006. doi:10.1137/S1052623499350013.

[35] Perez, R. E., Jansen, P. W., and Martins, J. R. R. A., "pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization," *Structural and Multidisciplinary Optimization*, Vol. 45, No. 1, January 2012, pp. 101–118. doi:10.1007/s00158-011-0666-3.

[36] Vassberg, J. C., Tinoco, E. N., Mani, M., Rider, B., Zickuhr, T., Levy, D. W., Brodersen, O. P., Eisfeld, B., Crippa, S., Wahls, R. A., Morrison, J. H., Mavriplis, D. J., and Murayama, M., "Summary of the Fourth AIAA Computational Fluid Dynamics Drag Prediction Workshop," *Journal of Aircraft*, Vol. 51, No. 4, jul 2014, pp. 1070–1089. doi:10.2514/1.c032418.

[37] Kenway, G. K. W. and Martins, J. R. R. A., "AIAA ADODG Case 5: CRM Wing-Body-Tail Optimization at Flight Reynolds Number," Tech. rep., AIAA, May 2015.

[38] Caradonna, F. X. and Tung, C., "Experimental and Analytical Studies of a Model Helicopter Rotor in Hover," Tech. Rep. NASA-TM-81232, September 1981.

A. Overset Glossary of Terms

Real Cell Any cell that is present in the mesh description supplied from the mesh generation program. The number of real cells is taken as the overall mesh size. See Figure 31.

Halo Cell Cells that pad the exterior of the real cells. Halo cells may be one of two types:

Boundary Halos Halo cells "behind" physical boundary conditions. These are used to impose boundary conditions on the PDE.

Neighbor Halos Halo cells behind a block face that is connected in a cell-matched fashion to another block. Neighbor halos are duplicates of real cells on a different block.

The number of halo cells in a given mesh depends on the block distribution and how many times the original blocks are split for purposes of load-balancing. All real cells are given a unique index as schematically outlined in Figure 31 and neighbor halos have access to this index. Boundary halos have a index value of -1 indicating they are not real cells.

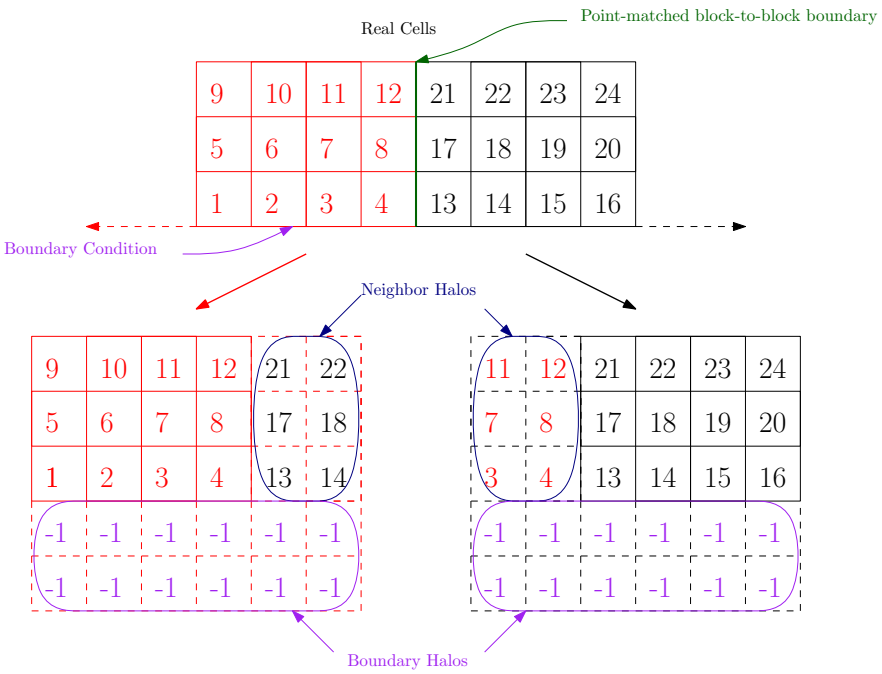


Figure 31: Illustration of real, neighbor halo and boundary halo cells.

Compute Cell (`iBlank=1`) These are cells remaining after the hole cutting procedure. These cells are the only ones that provide solution unknowns to the underlying PDE. In other words, the equations are solved only on these cells.

Interpolated Cell (`iBlank=-1`) Interpolated cells do not participate in the solution of the flow equations. These cells receive interpolated solution information from compute cells. These cells may constitute part of a compute cell's computational stencil. By construction, the residual in these cells is set to zero. Interpolated cells are also sometimes referred to fringe cells or receiver cells.

Blanked Cell (`iBlank=0`) These cells are computed removed from the computation. The state variables are not updated and do not receive interpolated values from other overlapping meshes. The residual in these cells is set to zero by construction.

Flood Seed Cell (`iBlank=-3`) These cells have been identified as a potential donor to a cell adjacent to a wall (Wall Donor) that itself is not next to a wall. An alternative definition would be cells that intersect solid walls. These form the starting location for flooding process. For simulation purposes, these cells are treated as blanked cells.

Flooded Cell (`iBlank=-2`) These are former compute cells inside the solid body that have been converted to holes due to the flooding procedure. Like flood seed cells, they are treated as holes in the simulation.

Explicitly Blanked Cell (`iBlank=-4`) These cells are explicitly removed from the simulation by a user provided function. The most common usage of explicit blanking is to remove cells "behind" a symmetry plane.

Irregular Cell A cell that is both a flagged both as a donor and a receiver.

Forced Receiver These are cells that must be able to find a donor cells. The cell's quality measure is ignored and the best potential donor is chosen. Two layers of cells surrounding holes must be marked as forced receivers as well as the last two layers of cells before on overset boundary condition.

Donor Cell Any cell that participates in the interpolation stencil of a fringe cell. All donor cells must also be compute cells.

Orphan Cell A cell that required to find to a donor (forced receiver) but was unable to do so.

Invalid Donor These are cells that are not available to be used as donors for other cells. All flooded cells, flood seed cells and forced receivers are considered invalid donors.

Cluster A collection of one or more point-patched computational blocks. It must be possible to stencil-walk from any cell on a cluster to any other cell on a cluster. All multiblock meshes without overset boundaries have precisely 1 cluster. Meshes with overset boundaries must have at least two distinct clusters.

Block A set of cells that have a logical i, j, k ordering. It is the basic building block of multiblock meshes.

Background Mesh A computation mesh, usually Cartesian that overlaps one or more near-field meshes. Background meshes usually will not contain solid-wall boundary conditions.

Near-field Mesh A computational mesh that contains wall boundary conditions. Near-field meshes are usually body-fitted to match the geometric definition of the body of interest.

B. Explicit Interior Hole Cut

For meshes without overlapping surfaces, an explicit hole cutting algorithm may be employed to effectively remove all interior cells before performing the implicit hole cutting algorithm. This eliminates the need for interior flooding and removes the iterative status determination loop.

The input to the algorithm is a closed, oriented surface with all surface normals "out" or into the field. The basic premise of the algorithm is to determine the closest physical location on the boundary for every cell center. A vector, v from this surface point to the location in question can then be formed. If the dot product of v with surface normal is positive, the cell is outside the boundary. If the dot product is negative, the cell must be inside the boundary. While the algorithm is exceedingly simple, there is a salient feature that requires further clarification. The correctness of the algorithm depends on a continuous surface normal being available. Consider for example, a sharp acute corner shown in Figure 32. Each of two surfaces adjoining the corner have surface normals as indicated. If we consider a point located at P_3, it is clear the closest point to the geometry is the convex corner. However, since this point belongs to two elements, there are two potential normals that could be used. The dot product with the normal on the upper surface is positive indicating the cell is outside, but the dot product is negative for the normal from the lower surface.

The solution to this ambiguity is to treat surface normals as a nodal property and not an surface element property. To accomplish this, a unique set of surface nodes are determined and a unit normal for each node is determined by averaging the normals from each node's surrounding elements. With the unique node based surface normals identified, the problem in Figure 32 poses no issue: both the upper and lower elements have the same normal information and either element will correctly classify the dot product. It is also worth noting that with the node-based normals, we can interpolate the normals to any point on an element to get a better estimate of the local surface normal.

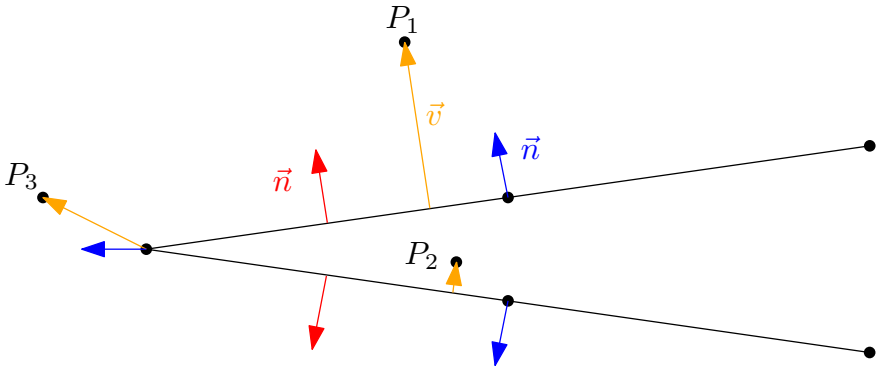


Figure 32: Closest point algorithm at a sharp convex corner.

This algorithm is exceptionally robust and we have not found any exceptional corner cases that break the fundamental algorithm. The computational cost of the algorithm depends on the efficiency of the nearest point search algorithm. For RANS simulations with turbulence models requiring wall distances, the interior hole cutting algorithm can be combined with the wall distance computation for very little computational effort.