Overset Grid Assembler and Flow Solver with Adaptive Spatial Load Balancing

Orhan Shibliyev

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mechanical Engineering

Eastern Mediterranean University
December 2021
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Mechanical Engineering.

Prof. Dr. Hasan Hacışevki
Chair, Department of Mechanical Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Mechanical Engineering.

Prof. Dr. İbrahim Sezai
Supervisor

Examining Committee

1. Prof. Dr. Hasan U. Akay
2. Prof. Dr. Uğur Atikol
3. Prof. Dr. Hasan Hacışevki
4. Prof. Dr. Mehmet Ş. Kavsaoğlu
5. Prof. Dr. İbrahim Sezai

ABSTRACT

In the present study, a parallel unsteady and coupled flow solver is developed to solve fluid flow around relatively moving components using a system of multiple unstructured meshes overlapping each other in a parallel computing environment. The use of multiple overlapping meshes is also referred as overset mesh methodology, which is convenient in solving fluid flow problems involving moving components such as flow around helicopters and wind farms.

Traditional single grid generation around all the components of a system is time consuming. Also, quality of the resultant single grid is usually unsatisfactory for critical regions of flow such as boundary layers and bodies in close proximity. Additionally, in unsteady flow simulations, excessive mesh stretching causes the solution accuracy to diminish significantly. Overset mesh methodology allows each component mesh to be generated independently with desired local properties. In this thesis, an overset grid assembler is developed to establish connectivity across component meshes in a parallel computing environment, where all meshes are partitioned into multiple mesh-blocks and processed on multiple cores. The cells are classified into 1) field cells on which the discretized Euler equations are solved, 2) receptors which interpolate data from (donor) field cells and 3) hole cells which are excluded from the flow solution due to overlapping invalid regions of space such as holes. Alternating Digital Tree and stencil walking are implemented to reduce the time spent on the overset mesh connectivity. Hole map is used to identify hole cells and integrated to the mesh connectivity algorithm in order to cut holes exactly.

Unlike traditional mesh partitioning where each partition contains similar number of cells, component meshes are partitioned spatially so that overlapping mesh-blocks reside in the same partitions. Spatial partitioning is performed using an octree to which mesh-blocks are registered. The octree is refined adaptively until octree-bins can be distributed to processors evenly. Load balancing is repeated whenever load imbalance exceeds a predefined threshold.

Validity of the developed code is tested on several test cases including the case of complex flow around a generic helicopter configuration in near hover condition and evaluated in terms of rotor-fuselage interaction, load balance, scalability and memory usage.

Even though load (re-)balancing was found to be the most time consuming task, it was shown that frequent load balancing reduced total simulation time considerably. The time saved with load rebalancing was 13% which added up periodically for every quarter rotation.

Speed-up results for combination of tasks (hole cut, donor search and overlap minimization) in the present work were compared with Suggar++ [1] which provided speed-up results for up to 8 processors. It was observed that present speed-up results showed linear behaviour compared to non-linear speed-up in Suggar++. Additionally, higher speed-up was obtained compared with Suggar++.

Keywords: Computational fluid dynamics; numerical algorithms; overset grid methodology; load balancing

ÖZ

Bu çalışmada, paralel bir hesaplama ortamında birbiriyle örtüşen çoklu yapılandırılmamış ağlardan oluşan bir sistem kullanarak, nispeten hareket eden bileşenlerin etrafındaki sıvı akışını çözmek için bir paralel kararsız ve birleştirilmiş akış çözücü geliştirilmiştir. Helikopter ve rüzgar santrali gibi hareketli bileşenleri içeren sıvı akışı problemlerinin çözümünde çoklu ağların kullanımı örtüşen sayısal ağ yöntemi olarak da adlandırılır.

Bir sistemin tüm bileşenleri etrafında geleneksel tek sayısal ağ üretimi, zaman alan bir işlemdir dolayısıyle pratik değildir. Ayrıca, sonuçta ortaya çıkan tek sayısal ağın kalitesi, sınır katmanı ve yakın cisimler gibi kritik akış bölgeleri için genellikle yetersizdir. Ek olarak, hareketli sayısal ağlı kararsız akış simülasyonlarında, sonraki zaman adımlarında aşırı sayısal ağ gerilimi, çözüm doğruluğunun önemli ölçüde azalmasına neden olur. Örtüşen sayısal ağ yöntemi, her bir bileşen sayısal ağın istenen yerel özelliklerle bağımsız olarak oluşturulmasına izin verir. Bu çalışmada, tüm sayısal ağların birden çok sayısal ağ bloğuna bölündüğü ve birden çok çekirdek üzerinde işlendiği paralel bir hesaplama ortamında bileşen sayısal ağları arasında bağlantı kurmak için bir örtüşen sayısal ağ kurucusu geliştirilmiştir. Hücreler, 1) ayrıklaştırılmış Euler denklemlerinin çözüldüğü alan hücreleri, 2) (donör) alan hücrelerinden gelen verileri enterpolasyon yapan reseptörler ve 3) geçersiz bölgeler ile kesişmesi nedeniyle akış çözümünden dışlanan delik hücreleri olarak sınıflandırılır. Örtüşen sayısal ağ kurumumda harcanan zamanı azaltmak için Alternating Digital Tree ve stencil walking algoritmaları uygunlanmıştır. Delik hücrelerini tanımlamak için delik haritası kullanılmıştır ve delikleri tam olarak kesmek için donör aramasına entegre edilmiştir.

Her bölümün benzer sayıda hücre içerdiği geleneksel sayısal ağ bölümlemesinden farklı olarak, bileşen sayısal ağları uzamsal olarak bölümlere ayrılır, böylece örtüşen sayısal ağ blokları aynı bölümlerde bulunur. Uzamsal bölümleme, sayısal ağ bloklarının kaydedildiği bir octree kullanılarak gerçekleştirilmiştir. Octree, octree-kutuları işlemcilere eşit olarak dağıtılıncaya kadar uyarlanabilir şekilde bölünmüştür. Yük dengesizliği önceden tanımlanmış bir eşiği aştığında yük dengeleme tekrarlanmıştır.

Geliştirilen kodun geçerliliği, genel bir helikopter konfigürasyonu etrafındaki karmaşık akış durumunda, havada asılı kalma durumunda test edilip, rotor-gövde etkileşimi, yük dengesi, ölçeklenebilirlik ve bellek kullanımı açısından değerlendirilmiştir.

Anahtar Kelimeler: Hesaplamalı akışkanlar dinamiği; sayısal algoritmalar; örtüşen sayısal ağ yöntemi; yük dengelenmesi

ACKNOWLEDGMENTS

I would like to thank to my supervisor Prof. Dr. İbrahim Sezai for his continuous support through my PhD study. It has been an invaluable opportunity to work with him and source of motivation for future studies.

I am thankful to my family for their patience and every kind of support. This work would not be completed without them.

I would like to thank my committee members Prof. Dr. Uğur Atikol, Prof. Dr. Hasan Hacışevki, Prof. Dr. Hasan U. Akay and Prof. Dr. Mehmet Ş. Kavsaoğlu for their contributions and insightful reviews. Further, I appreciate the support by TUBITAK ULAKBIM for computational resources.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ϕ	Azimuth angle
α	Shaft angle
μ	Advance ratio
σ	Rotor solidity
ω	Angular blade speed
τ	Pseudo time
Φ	Slope limiter
ρ_∞	Freestream density
θ_c	Collective pitch at the root of a blade
$\theta_{c,75}$	Collective pitch at 75% of a blade
θ_t	Twist angle
A	Rotor disc area
A_1	Lateral cyclic pitch
B_1	Longitudinal cyclic pitch
C_A	Axial force coefficient
C_l	Roll coefficient
C_m	Pitch coefficient
C_N	Normal force coefficient
C_n	Yaw coefficient
C_p	Pressure coefficient
C_T	Thrust coefficient
C_Y	Lateral force coefficient

p_∞	Freestream pressure
p_d	Modified dynamic pressure
R	Rotor radius
T	Thrust
t	Real time
V_∞	Freestream velocity
V_t	Blade tip velocity
AABB	Axis-aligned bounding box
ADT	Alternating Digital Tree
BTL	Byte Transfer Layer
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrichs-Lewy condition
HLL	Harten, Lax and van Leer, Riemann solver
HLLC	HLL Riemann solver including contact discontinuity
MPI	Message Passing Interface
OBB	Oriented bounding box
OGA	Overset Grid Assembly
OMM	Overset Mesh Methodology
SMA	Shared Memory Architecture

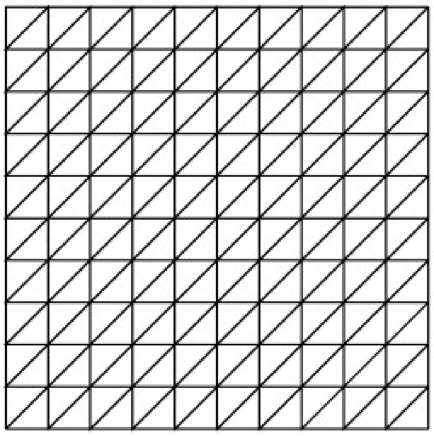
Chapter 1

INTRODUCTION

Numerical simulations, if performed in conjuction with experimental flow analysis, can considerably reduce the number of experiments, saving both time and expenses. Simulations can also be useful in situations when experimental environment is cumbersome to setup or open to hazards.

On the other hand, serious challenges are present in execution of simulations. First, the mathematical domain which is continous in nature needs to be discretized into discrete domain in the process called meshing. Structured grids (see Figure 1.1) that organize grid elements in a structured order are particularly effective in adjusting the alignment of elements according to flow characteristics. For example, the aspect ratio of elements is higher in wall-surface direction in boundary layer flows. Figure 1.2 shows a hybrid mesh with structured mesh in the boundary layer surrounded by an unstructured mesh. Also, from computational point of view, structured grids conserve degree of accuracy of derivatives to an extent after discretization. Gradient reconstruction, for example, has a higher accuracy on structured grids compared to unstructured grids.

With increasing geometric complexity, meshing algorithms may generate unsatisfactory mesh that does not fit well to the characteristics of local flow-field. In most complex cases, generation of structured grid is not possible. In these cases, generation of


(a) Structured mesh.

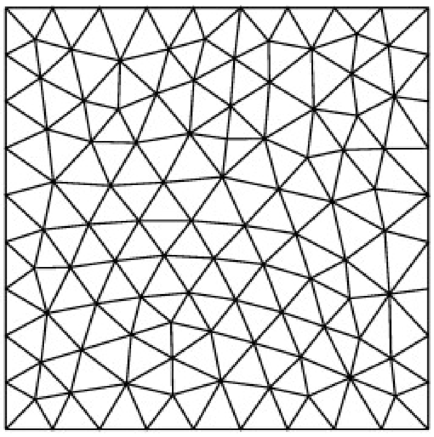

(b) Unstructured mesh.

Figure 1.1: Mesh types as displayed in Reference [2].

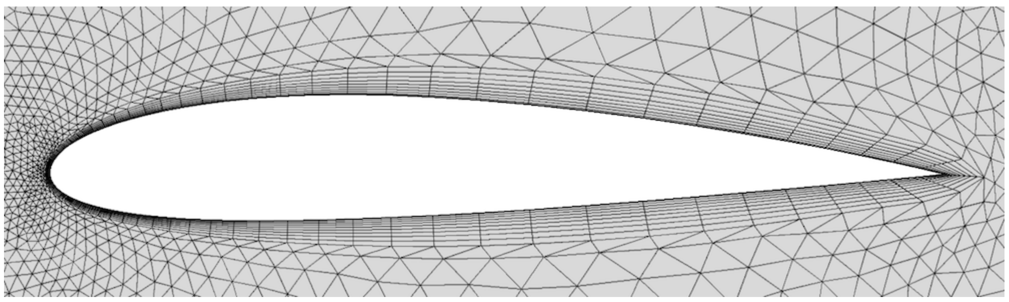


Figure 1.2: Hybrid mesh with structured mesh in the boundary layer as displayed in Reference [3].

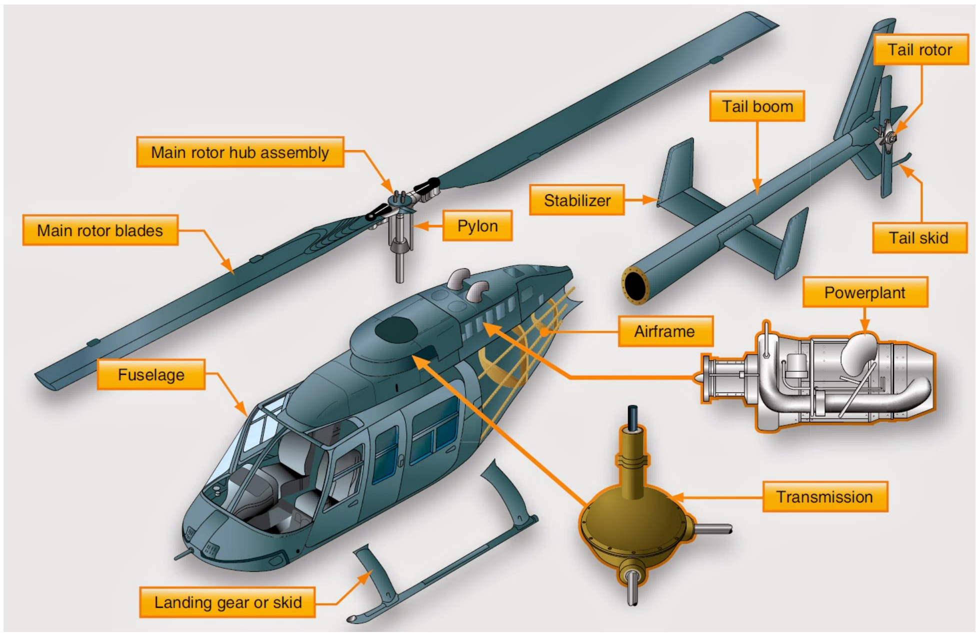


Figure 1.3: Components of a helicopter configuration as displayed in Reference [4].

unstructured grids with algorithms such as Delaunay triangulation and Advancing Front requires less effort. However, without structured element-to-element connectivity information, accuracy of temporal and spatial discretizations degrade.

Regardless of mesh type, structured or unstructured, there are configurations where the generated grid is bound to perform poorly. A typical case is unsteady simulation of flow involving multiple relatively moving components. For instance, a helicopter configuration [5], as show in Figure 1.3 has components such as fuselage, main and tail rotor blades and hub that are rotating in relative motion. As the components move relative to each other, the mesh need to be regenerated in every time step. If mesh regeration is avoided and the initial single grid is used, solution accuracy, unless proper mesh deformation technique is implemented, would degrade due to excessive stretching after many time steps.

It is possible to avoid mesh regeneration and use structured grids with a technique called Overset Mesh Methodology (OMM) which was first introduced in Reference [6] as Chimera grid technique. Regardless of the number and geometric complexity of components, OMM allows component meshes to be generated only once and independent from other components with desired mesh topology.

Since each mesh is generated independently from other meshes, there is no cell-to-cell connectivity between the meshes. As a result, it is necessary to identify overlapping cells to interpolate flow variables between meshes. In general, cells are classified into different types which are mainly

- the field cells on which flow solution is computed;
- the receptor cells which interpolate flow variables from field cells across overlapping mesh cells;
- the hole cells which are excluded from flow solution due to overlapping invalid regions of domain such as holes.

Identification of the cell types is the core task of assembler and is called donor search. Figure 1.4 shows several cell types of structured Cartesian and unstructured overset mesh after donor search. In each time step, the assembler identifies and transfers the cell types to the developed parallel flow solver which, in turn, solves the Euler equations on the field cells. As the component meshes are free to move, donor search has to be repeated after relocation of component meshes.

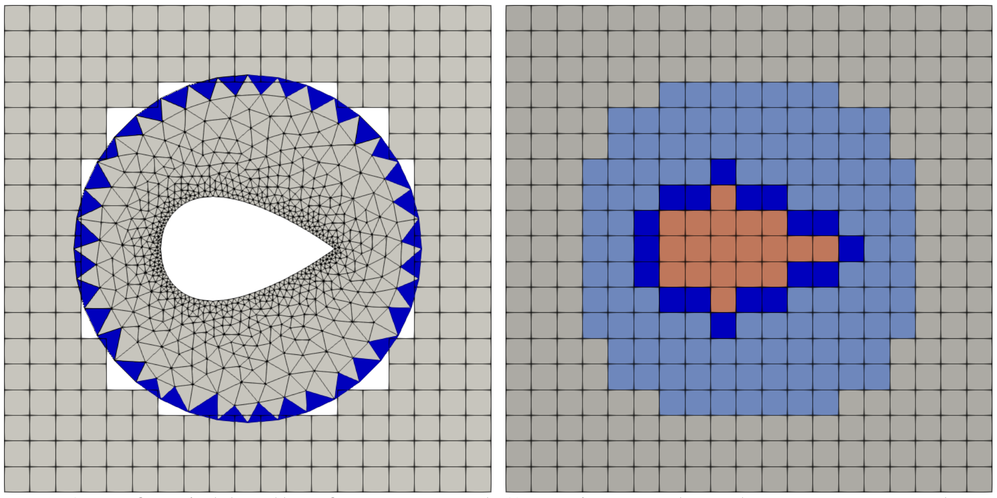


Figure 1.4: Left: Field cells of a structured Cartesian mesh and an unstructured overset mesh after donor search. Right: The structured mesh after donor search. Gray: Field cell, Orange: Hole cell, Light blue: Receptor, Blue: Mandatory receptor.

1.1 Motivation

Contrary to traditional single grid flow solvers for which work-load is characterized by only the number of cells, in overset mesh methodology, additionally spatial location of cells contribute to the work load. Cells which are overlapped by other cells cost more time compared to the non-overlapped cells since the overlapped cells may involve in data interpolation. It is, therefore, natural to gather the overlapping cells to the same partitions/processors termed as Spatial Partitioning/Decomposition [1]. However, spatial decomposition is not sufficient for load balancing, per se.

In Reference [1], mesh-system is spatially partitioned into pre-defined volumes, such as co-axial cylinders and Cartesian mesh depending on the geometry of the problem. Pre-defined shapes causes the spatial partitioning to be problem-dependent. Moreover, in Reference [1], spatial partitioning is not followed by load balancing. In order to avoid problem-dependent spatial partitioning, in this study, an octree is utilized for spatial partitioning. The octree is work-load adaptive, that is, the octree is successively refined at the most-loaded bins until balanced work-load distribution is achieved.

Cells are mapped to the octree based on intersection of bounding boxes of cells and octree-bins. Ideal partitioning is computed after each iteration by weighted graph partitioner, METIS [7]. Weight is defined in terms of number of cells which, unlike traditional load balancing, can belong to different meshes. The weight definition is what makes adaptive spatial partitioning novel and different from traditional partitioning and non-adaptive spatial partitioning.

In traditional load balancing, cells are permanently assigned to partitions. Therefore, after mesh motion, partitions move with the associated cells together. In spatial decomposition, however, partitions remain stationary during displacement of cells. As an analogy, traditional load balancing is Lagrangian while load balancing with spatial decomposition is Eulerian.

The only study, according to the knowledge of the present author, that implemented load balancing with traditional partitioning is Reference [8]. In that study, load balance is restored after mesh motion by examining task durations in previous time steps. Excess loads are exchanged between processors temporarily and once the remote processors processed the loads received from the host processors, the processed loads are returned back to the host processors. In other words, the load distribution is corrected at the end of a time step. In the current study, load distribution is aimed to be predicted in the beginning of a time step. Compared to Reference [8], the current load balancer is predictive rather than corrective.

Permanent load balance is impossible in moving mesh simulations and load balancer has to dynamically recompute load distribution irrespective of the method of partitioning. Another merit of the current of partitioning method compared to Reference [8] is that as overlapping cells always reside in the same partitions/processors, loads do not need to be sent from remote processors back to the host processors. The present method however, has the extra burden of relocating cells to proper partitions/processors after mesh motion.

1.2 Scope and objectives

This study aims to contribute to the field of numerical solution techniques by improving *spatially* partitioned overset mesh technique pioneered by Reference [1] and enhancing the technique with load balancing capability. Due to its simplicity, overwhelmingly, traditional partitioning techniques are used for overset meshes. It is aimed to contribute to the literature by evaluating parallel performance of the developed overset mesh solver which is specificially spatially partitioned and load-balanced. It is useful to add the capability of traditional partitioning in order to compare against the spatial partitioning technique. However, considering the the amount of time investment, traditional partitioning technique is out of the scope of this study.

Another objective of the study is to apply the practical knowledge and experience gained from this study to well-established open-source solvers which do not have spatial partitioning capability. The overset mesh solver developed in this study is mostly self-sufficient and important modules such as computational geometry and generic data exchange are developed in-house. As internal implementation details are under-documented in the literature, the present study provides an insight to the overset mesh solvers.

1.3 Dissertation outline

In this thesis, chapters and their contents are organized as shown below.

Chapter 2	Discusses history and past research about overset mesh methodology in terms of donor search techniques, parallelization and load balancing.
Chapter 3	Covers overset mesh methodology and discretization of the Euler equations, flux reconstruction and the gradient limiter. It also covers parallelization, partitioning, load balancing and donor search techniques in detail.
Chapter 4	Presents CAD model of the test case of generic helicopter configuration. Then, results for validation of the overset grid assembler are presented. Computational data is compared against experimental data. Subsequently, this chapter presents parallel performance results in terms of dominant tasks and speed-up of parallel algorithm. Finally, the conclusions are drawn based on the findings.
Chapter 5	Summarizes the methodology and results of the dissertation. Conclusions are drawn based on the findings.

Chapter 2

LITERATURE REVIEW

2.1 Parallel communication

Initial overset mesh packages executed sequentially [9–14]. With the ever increasing demand for more computing resources to solve realistic problems, packages started to run on multi-core computers.

Parallel computations can be performed with Central Processing Units, CPUs, or General Purpose Graphical Processing Units, GPGPUs. Both CPUs and GPUs consist of processing units called cores. The number of cores in a CPU may go up to 64 cores without hyperthreading (AMD EPYC 7002 Series) and 128 processors with hyperthreading (Ampere Altra Max). Whereas, a GPGPU may contain a few thousand cores such as GeForce GTX TITAN Z containing 5760 cores. Although a GPGPU contains a two-order of magnitude more number of cores, the GPGPU cores perform slower than CPU cores. GPGPU is useful for highly parallel programs, thanks to high amount of cores. Overset mesh methodology has been successfully implemented on GPGPU devices in numerous studies [15–17].

Another important metric besides the number of cores is the memory management. At present, there are two different memory management types: shared and distributed. Figure 2.1 shows a typical Shared Memory Architecture (SMA) configuration. Data

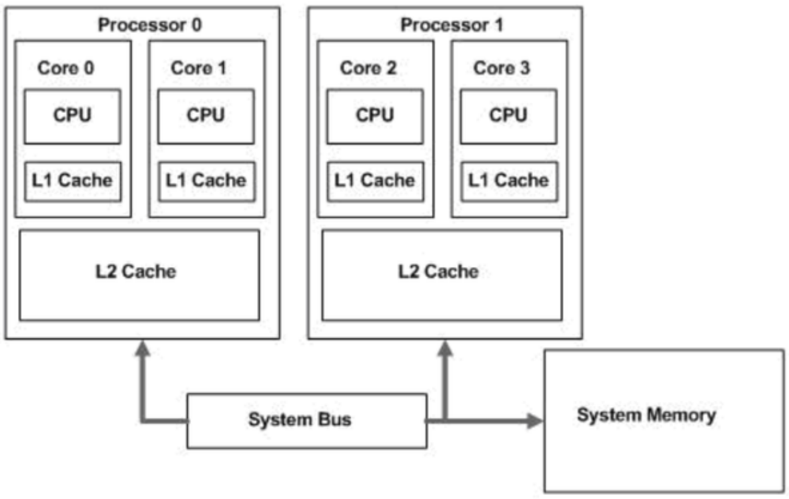


Figure 2.1: Shared memory architecture as displayed in Reference [18].

access rate from a core to L1 cache, L2 cache and RAM (system or main memory in other context) decreases in descending order. Therefore, it is advantageous to store the frequently used variables in memory locations as close as possible. Compilers can optimize this procedure although, explicit management by developer is also possible.

Since RAM is shared by all the cores, data read/write may not be concurrent. If no careful measures are taken, data race conditions may happen causing the overwrite of data during a read operation. When cores attempt to read/write data, locks/mutex are used in order to avoid race conditions.

Shared memory machines has a limited maximum number of cores and additional cores cannot be added since the circuitry is designed for a specific heat dissipation rate. In this case, multiple shared memory machines are inter-connected physically via cables with either Ethernet or InfiniBand protocols to form a cluster or wireless which is referred as cloud computing. Data bandwith rate is higher in physical connections than wireless. Each shared memory machine in the cluster is called a node. There is no upper limit on the number of nodes in the cluster. Node-to-node data access rate is slower than that of in-node as transfer of data across nodes is bound to protocol dictated by Ethernet/InfiniBand.

Message Passing Interface (MPI) [19] is the de facto library for communication across nodes as well in-node cores in distributed memory arrangement systems. MPI as its name suggests, is merely an interface and it has several implementations such as OpenMPI [20], MPICH [21] and Intel MPI [22]. MPI is independent of programming languages, therefore, can be used with any language. Data transfer is controlled by user explicitly with point-to-point and collective communication routines. In addition to point-to-point communication that requires both the sender and the receiver to be involved in data transfer, one-sided communication which involves only the sender is also available. MPI is especially useful in transfer of data structures which contains standard type members such as integer and float; for example, an array of integers and floats. One of the biggest difficulties, since MPI is not aware of programming language, is the transfer of user-defined data structures. The layout of user-defined data structure such as struct/class have to be described to MPI in terms of bytes. Extra care should be spent if the data structure contain members which are in different location of heap memory. This difficulty is absent in programming languages which have embedded parallelization capabilities such as Chapel [23], Julia [24], Coarray Fortran [25]. Another way of avoiding explicit description of data layouts is language specific (C++) libraries such as Boost MPI which use Boost Serialization to serialize/deserialize data for communication at the cost of time.

Research institutions usually own clusters to be managed by specific personnel. The burden of installation of packages such as operating system, compilers, libraries, resource management software such as SLURM [26] and also maintenance of disfunctioning hardware such as user/server nodes and cables are transferred from the users to maintenance personnel. Another alternative, although not as popular, is commercial vendors such as Google Cloud, Microsoft Azure and Amazon Web Services (AWS). Service can be obtained as pay to go or via subscription for a certain time. In this case, the users are responsible from the maintenance of their own cluster. Opensource projects such as ElastiCluster [27] greatly reduce the time needed to setup cluster nodes, however, all the capabilities of an in-house cluster still cannot be attained.

2.2 Parallel Overset Mesh Methodology

Computational solution of Partial Differential Equations (PDE) requires the PDE to be transformed to discrete PDE which is solved on discrete points in computational domain. Finite Volume method discretizes spatial domain into control volumes or cells. The collection of cells is called a mesh. Discrete PDE is solved on the cells where solution is of interest.

Parallelism in CFD applications is usually achieved via data parallelism, especially, control volume parallelism. Mesh is partitioned/decomposed into chunks containing approximately equal number of cells and each chunk is allocated to a processor. The load of a partition is based on the number of cells the partition contains, hence the name, cell-based partitioning. Figure 2.2 shows partitions of an airfoil mesh with cell-based partitioning.

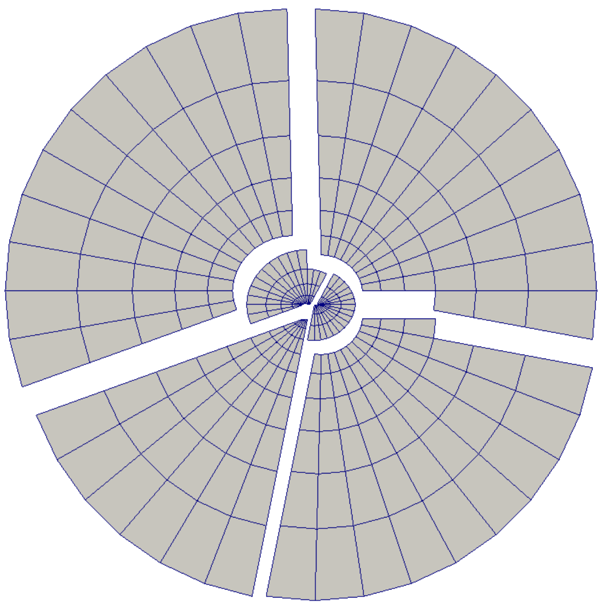


Figure 2.2: Partitions of an airfoil mesh with cell-based partitioning.

In the early works [28–30], the number of partitions were equal to the number of processors which is called coarse grain parallelism [31]. In subsequent works [8, 10, 32–48], the number of partitions exceeded the number of processors which is called fine grain parallelism [49]. When a single mesh is used for the whole computational domain, cell-based partitioning is effective in distribution of work-load among processors as each cell usually has the same computational cost. For example, most of the time, the same governing equations are solved on all cells. Since cell-based partitioning ignores spatial locality, overlapping mesh-blocks can reside in different processors and as a result, overlapping mesh-blocks have to be exchanged for donor search and processed information need to be sent back to the host processor. In most of the parallel overset mesh studies, predominantly, an implementation of MPI is used for distributed parallel computing. However, shared memory implementations with OpenMP are also available [50].

In Reference [1], cells which can belong to different meshes were distributed to processors based on their spatial location. The main difference of spatial partitioning from cell-based partitioning is that cells which occupy the same region of space are partitioned together and partitions are not optimized to have approximately equal number of cells. Spatial domain was subdivided into an auxiliary Cartesian or cylindrical mesh to which cells are registered. Finally, cells are relocated to processors which own associated bins. In Reference [1], this is called Spatial Decomposition Volume. In both cell-based and spatial partitioning, at some point, overlapping cells are exchanged to reside in the same processors. However, in the case of cell-based partitioning the remote cells are sent back to their host processor after donor search, while, this step is not required for spatial partitioning. Therefore, spatial partitioning saves time by skipping one way of transfer. It is possible to use two different partitioning layouts for the assembler and the solver. For example, in Reference [51], the assembler was partitioned spatially while the solver was partitioned based on the number of cells. However, dual partitioning duplicates memory usage which is a valuable resource in high performance computing.

2.3 Load balance

In overset mesh methodology, computational cost of each cell may vary greatly depending on the type of cell. Discrete PDE are solved on field cells, flow variables are interpolated between donor and receptor cells and hole cells do not participate in any task, therefore, have no computational cost. Assemblers that are partitioned with cell-based partitioning for which, each cell is assumed to have the same computational cost, are bound to result in high load-imbalance. In previous works, cell-based partitioning has been used commonly [8, 10, 28–30, 32–48], however, only few of them [8, 52] attempted to fix load-imbalance.

In Reference [8,52] a cell-based partitioning is used and high load-imbalance is reported. Subsequently, a dynamic load-balance algorithm is implemented to alleviate load-imbalance. In Reference [52], resultant partitioning layout is also applied to the flow solver causing huge imbalance for the solver. In Reference [8] partitioning layout never changes or implicitly changes temporarily. In addition, domain connectivity was assumed to be the most time consuming task and was correlated with the number of 'query points' for which the type (field, receptor, etc.) is sought for. Query points are distributed among processors iteratively until loads in each processor drops under a threshold. It was reported that total assembly time is reduced from 2.1 seconds to 0.5 seconds. However, no relative temporal load-imbalance, that is, the ratio of slowest to the fastest processor time was reported. This metric, which is provided in the results section of the dissertation, is important in order to judge correlation of connectivity task to the number of points.

2.4 Donor search

Donor search is basically comprised of determination of overlapping cells. In this context, the centroid of a cell is called a query point and is subjected to a search operation to find the cell (donor) which contains the query point. A primitive way of donor search is stencil walking/jumping [10]. With this algorithm, a starting or seed cell is chosen and the cell (target) which contains the query point is reached by walking from cell to cell assuming that cell-to-cell connectivity is already established. Proximity of the seed to the target determines the cost of the algorithm. The choice of seed can be improved by registering the spatial domain to auxiliary Cartesian meshes or inverse maps. Each sub-block of the auxiliary mesh may contain one or more cells [8, 53]. Another alternative of inverse maps is Alternating Digital Tree [54] (ADT) which is a type of binary tree with alternating axes. As inverse map, ADT can be used to improve seed selection, however, it can also be used as standalone [1, 8, 40, 55], that is, does not need to be integrated to the donor search.

2.5 Hole cutting

Hole cutting which is for identification of hole cells is executed in two ways: Direct [1, 56] and approximate [8, 10, 37, 53, 57–59].

In direct hole cutting, the cells of overset grid that overlap boundaries of a hole are determined. Subsequently, the interior cells are identified with flood/fill algorithm. Direct hole cutting determines hole cells exactly and no approximation is involved. However, it is more time consuming compared to approximate hole cutting and also prone to geometric errors.

In approximate hole cutting, simple geometries, mostly rectangles, are used to represent holes. The method remains *approximate* if no further operation follows. The method or specifically 'hole mapping' can be exact if it is integrated to donor search operation [8]. In Reference [8], an initial axis-aligned bounding box which contains the hole geometry is adaptively refined until each sub-blocks contains either none or only one kind of boundary element such as outer or hole boundary element. The cost of hole mapping is proportional to the proximity of outer and hole boundaries, that is, more refinements/resolution is needed when hole and outer boundaries are in close proximity. For refinements typically quadtree and octree are used for two- and three-dimensional geometries. Reference [60] improves hole mapping by minimizing the overlap.

Initial input mesh-system consisting of all the background and component meshes, is usually partitioned with unweighted graph partitioning which produces partitions containing approximately equal number of cells based on cell connectivity. With overset mesh methodology, in the absence of cell-to-cell connectivity, overset meshes require interpolation of flow variables across overlapping mesh cells. Mesh cells are partitioned spatially so that the ones which occupy the same region reside in the same partition therefore, communication time for interpolation of variables is diminished. Although spatial partitioning reduces total run time by localizing interpolation of flow variables, it does not alleviate load imbalance. Spatial partitioning is performed adaptively by refining the most loaded bin of an octree until balanced partitions are obtained.

Chapter 3

METHODOLOGY

3.1 Overview

The overset mesh solver is composed of successive stages as shown in Figure 3.1. The solver takes a partitioned mesh-system, which contains arbitrarily overlapping background and component meshes, as an input. The input mesh-system is already partitioned and can be processed in parallel. However, the current partitioning layout causes severe load imbalance for the overset mesh solver unlike a single-mesh solver. The mesh-system is re-partitioned and load-balanced spatially and distributed to the processors. At this stage, the processors contain overlapping and disconnected (no cell-to-cell connectivity) mesh-cells belonging to different meshes. Even the cells that belong to same meshes are disconnected due to being partitioned in parallel environment. The assembler identifies the cells to be used as links between disconnected meshes (donor-receptor pairs) and also partitions (ghost cells). The assembler also determines the cells to be excluded from the mesh system due to overlapping invalid regions of spatial domain. The flow solver solves the discretized Euler equations on the meshes based on the information provided by the assembler. For example, the discretized equations are not solved on receptor cells which only interpolate data from their donors. If the problem to be solved is a moving body problem, positions of moving meshes are updated based on total force and moment on respective component unless the path of the component is prescribed. Since the cells are partitioned spatially, after

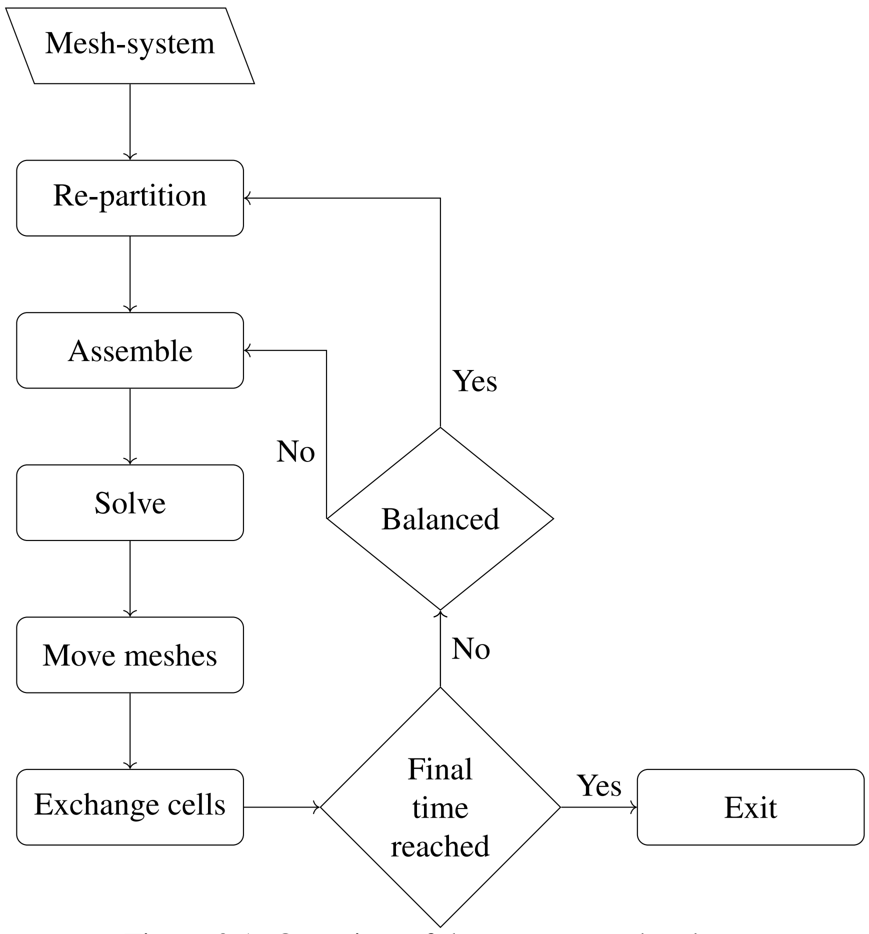


Figure 3.1: Overview of the overset mesh solver.

mesh motion, moving cells are relocated to new partitions/processors. The overset mesh solver exists if the final time is reached in an unsteady problem otherwise the mesh-system is re-partitioned if the re-computed load imbalance exceeds the prescribed threshold.

3.2 Spatial partitioning

In overset mesh technique, component meshes are generated independently, therefore, the component meshes can potentially overlap and have no cell-to-cell connectivity. In order to represent a region of space uniquely and transmit flow variables across component meshes, overlapping cells must be identified. As mentioned earlier, identification and classification of overlapping cells is called donor search.

Table 3.1: Mesh-files to be read by processors.

Processor 1	Processor 2
`fuselage_1.vtk`	`fuselage_2.vtk`
`blade_A_1.vtk`	`blade_A_2.vtk`
`blade_B_1.vtk`	`blade_B_2.vtk`
`blade_C_1.vtk`	`blade_C_2.vtk`
`blade_D_1.vtk`	`blade_D_2.vtk`

In a parallel computing environment, data or particularly mesh-system (all the component meshes) need to be decomposed into chunks of meshes called mesh-blocks to be distributed to processors. Figure 3.3 shows four mesh-blocks of a rotor blade mesh. In order to simplify visualization, mesh-blocks are displaced in Figure 3.3. A processor may store multiple mesh-blocks belonging to different meshes. Figure 3.2 shows the flowchart of the decomposition/partitioning algorithm that is used in this study. There are three main processes of partitioning: 1) initial un-weighted graph partitioning, 2) geometric partitioning and 3) weighted graph partitioning.

Un-weighted graph partitioning starts with decomposition of each component mesh into n mesh-blocks, where, n is the number of processors. If there are m component meshes, the total number of mesh-blocks, hence, mesh-files is $n \times m$. Each processor reads the corresponding mesh-file. For example, suppose that there are a total of two processors and five components: Fuselage and four blades. The fuselage mesh is decomposed into two mesh-blocks and named as `fuselage_1.vtk` and `fuselage_2.vtk`. The other component meshes would be decomposed and the respective mesh-files would be named similarly. Processors would read mesh-files as shown in Table 3.1.

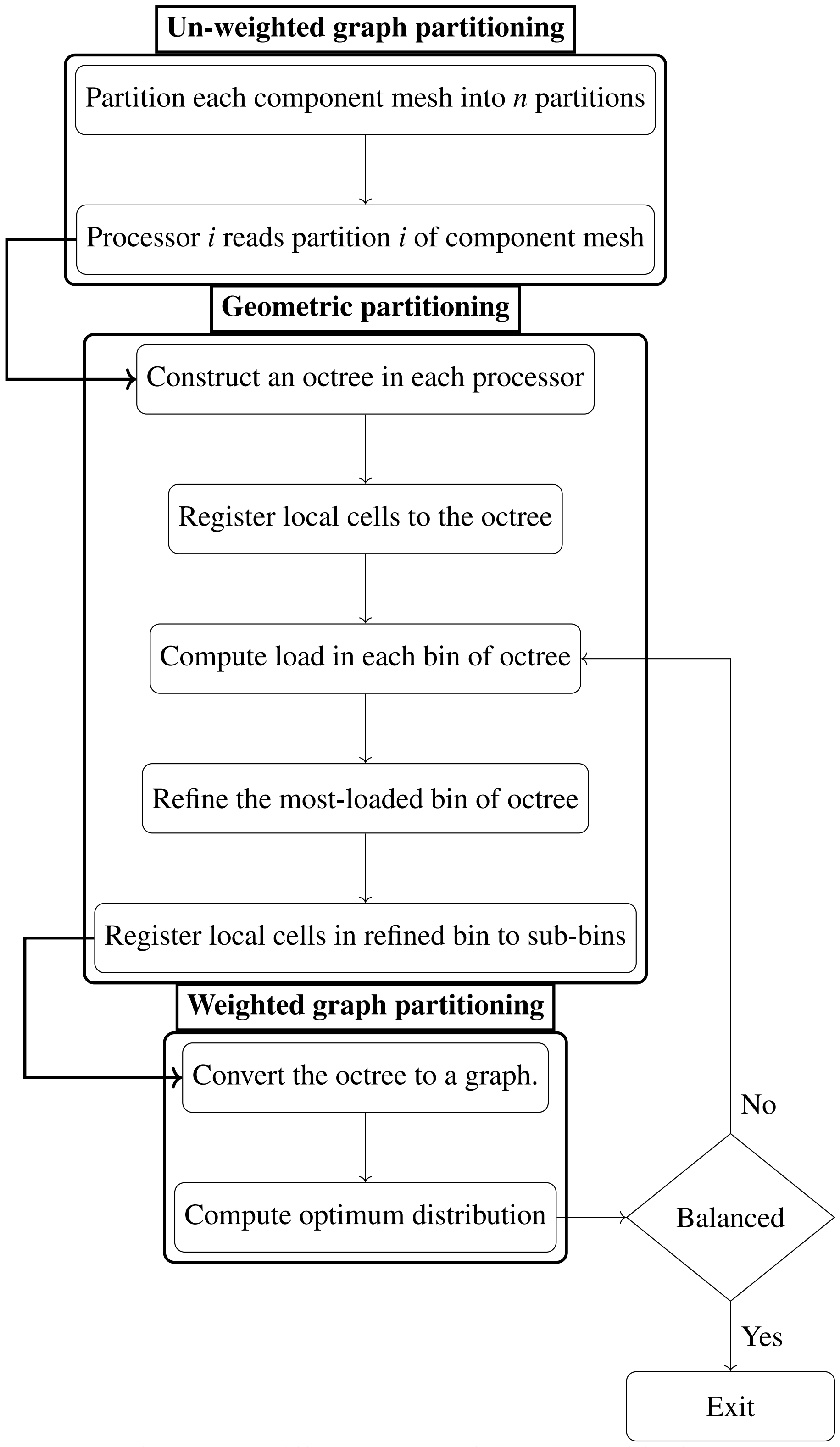

Un-weighted graph partitioning
Partition each component mesh into n partitions
Processor i reads partition i of component mesh
Geometric partitioning
Construct an octree in each processor
Register local cells to the octree
Compute load in each bin of octree
Refine the most-loaded bin of octree
Register local cells in refined bin to sub-bins
Weighted graph partitioning
Convert the octree to a graph.
Compute optimum distribution
Balanced
No
Yes
Exit

Figure 3.2: Different stages of domain partitioning.

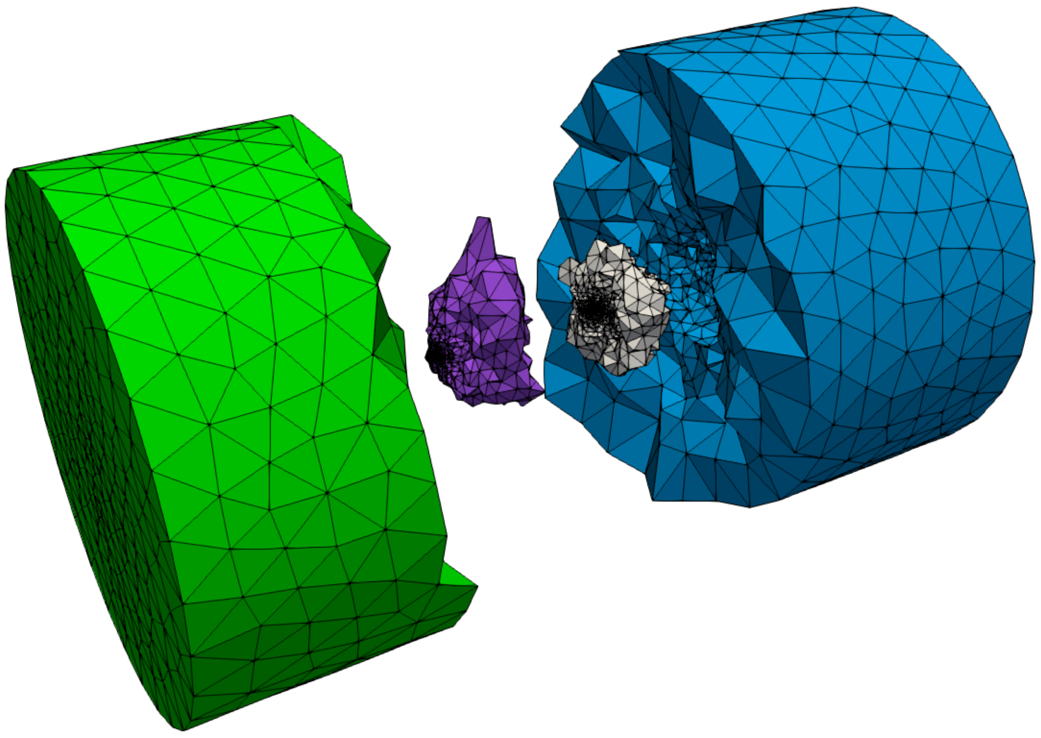
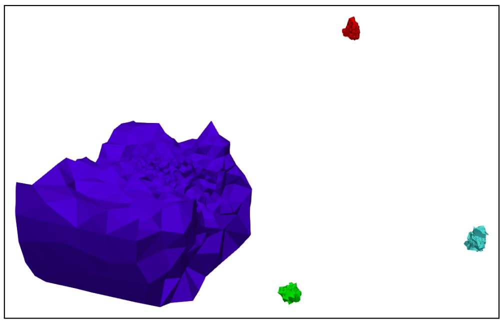


Figure 3.3: Four mesh-blocks of a rotor blade mesh.

Mesh decomposition is performed by the graph partitioner, METIS [7]. METIS decomposes each component mesh separately and, by default, allocates approximately equal number of cells in mesh-blocks belonging to a mesh. For example, the number of cells in `fuselage_1` and `fuselage_2` are approximately equal. As each component mesh is decomposed separately, the number of cells in `blade_A_1` and `blade_B_1` are different. Allocating equal number of cells in mesh-blocks is a result of un-weighted or uni-weighted graph partitioning/decomposition of METIS. It is possible to assign different weights to cells. Cells with similar weights would be gravitated to be partitioned together. In the case of un-weighted graph partitioning, each cell has the same weight. Even decomposition of meshes is suitable for applications for which execution time of tasks are expected to be a function of number of cells. Typical applications are flow solvers involving iterative solution of the same discretized


(a) This partition contains mesh-blocks that are not spatially overlapping.

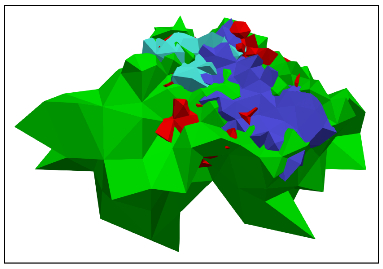

(b) Spatial partitioning contains mesh-blocks are that are spatially overlapping.

Figure 3.4: Mesh-blocks in a processor before and after spatial partitioning.

governing equations on all cells. In other words, for flow solvers, unless different tasks are performed by the cells, the time-cost of all the cells are identical. In overset mesh assembly, however, the aim is to determine overlapping cells, therefore, non-overlapped cells cost no time compared to overlapped cells.

As mentioned, for donor search, the overlapping cells which, at this stage, reside in different processors, must be transferred to the correct processors to be stored together. For this purpose, the mesh-system is re-partitioned after un-weighted partitioning based on spatial location of cells. This kind of partitioning is termed as Spatial Partitioning [1]. Figure 3.4 shows two partitions before and after spatial partitioning. In Figure 3.4a, mesh-blocks belonging to four different component meshes are assigned to the partition based on the number of cells. In Figure 3.4b mesh-blocks that belong to four different component meshes are assigned to the spatial partition based on their spatial locations, therefore, the mesh-blocks are overlapping.

In spatial partitioning, as the mesh-system is non-stationary, different mesh-cells arrive to and leave the partitions the processors are assigned with. On the other hand, in traditional approach, mesh-cells are permanently assigned to processors regardless of motion of mesh-system. One of the merits of the traditional approach is that cell-to-cell connectivity does not need to be maintained since there is no disconnection/reconnection of moving cells. In addition, no time is spent for re-partitioning of mesh-system spatially. However, the overlapping mesh-blocks are still required to be brought together regardless of the approach followed. The traditional approach solves this problem by transferring overlapping mesh-blocks and sending processed information back to the host processor. Downside of this approach is that the communication has to be repeated twice in every time step. In spatial partitioning, time is saved by eliminating the need for sending processed information back to the host processor since overlapping mesh-blocks reside in the same partitions.

In Reference [1], mesh-system is decomposed into pre-determined volumes based on expected path of the component meshes. As an example, rotor blade meshes are decomposed into cylindrical volumes as shown in Figure 3.5. Cylindrical volume is an appropriate choice for the rotor blade problem since the cells do not move across cylindrical partitions during rotor revolution, therefore, the cells do not need to be transferred across partitions/processors to be in correct spatial partitions. However, the choice of pre-determined volumes depends on the path of the components, therefore, spatial partitioning is problem-dependent.

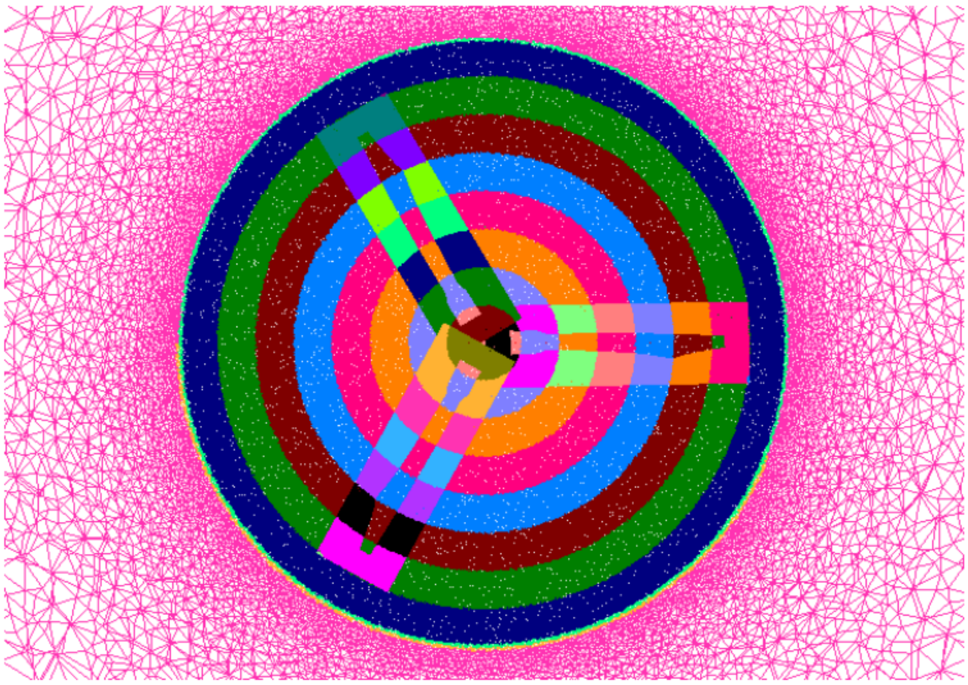


Figure 3.5: Spatial partitioning of rotor blade meshes with co-axial cylindrical volumes [1].

3.3 Load balancing

Regardless of decomposition volume, the total number of cells in spatial partitions are expected to greatly vary as some of the partitions would have more overlapping cells at a particular time. Even if partitions are somehow made to have equal number of cells, in subsequent time steps, after relative motion of components, the ideal decomposition becomes imbalanced. In Reference [1], no further load balancing is attempted after spatial partitioning.

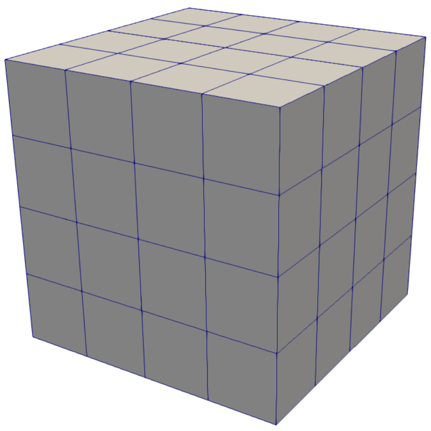
Distribution of n spatial partitions to n processors would result in high load imbalance. However, possibility of even distribution is more likely if the number of spatial partitions is higher than the number of processors. It is possible to increase the resolution of data structure, for example, co-axial radial mesh, in order to have more partitions than number of processors. However, uniform increase of resolution is demanding in memory storage. In this study, an adaptive approach is adopted and resolution of only the most-loaded volumes are increased.

3.3.1 Space partitioning data structure

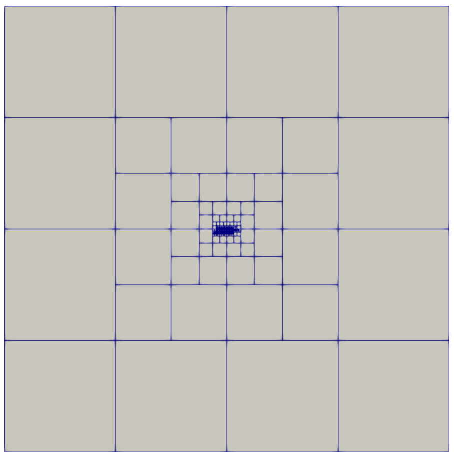
Initially, a 2x2x2 Cartesian mesh is constructed in each processor. Figure 3.6a shows the Cartesian mesh in the process of refinement. Each sub-block of the Cartesian mesh is called a bin. Bins of the Cartesian mesh is divided into eight bins (two bins in each coordinate axis). After sub-divisions the data structure is not a Cartesian mesh any more but called an octree. The dimensions of the octree is the same as the dimensions of the mesh-system. As processors contain only mesh-blocks of components meshes, processors are unaware of the dimensions of the mesh-system. Dimensions of the octree are obtained by gathering and merging bounding volumes of the local mesh blocks.

There several ways of representing a three-dimensional geometry with a bounding volume as shown in Figure 3.7. The cost of constructing the bounding volume increases and the accuracy of representation decreases from left to right in Figure 3.7. For example, Oriented Bounding Box (OBB) requires determination of principal axes and is more time consuming to construct compared to spherical bounding volume which requires only a centroid and a radius. For the case of Axis Aligned Bounding Box (AABB), the principal axes are aligned with coordinate axes, hence, no time is needed to be spent for determination of the principal axes. Since the octree and its bins are also axis-aligned, AABB is the ideal choice of bounding volume.

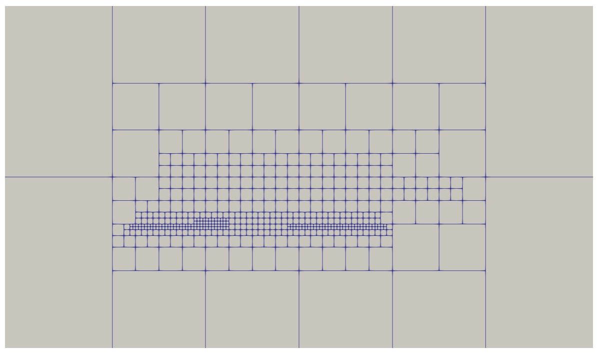
Processors register cells to octree-bins based on intersection of AABB of cells and bins.



(a) Computational domain in the process of adaptive spatial partitioning.



(b) Slice view shows refinements in the most loaded regions.



(c) Close up view of the slice.

Figure 3.6: Octree space partitioning.

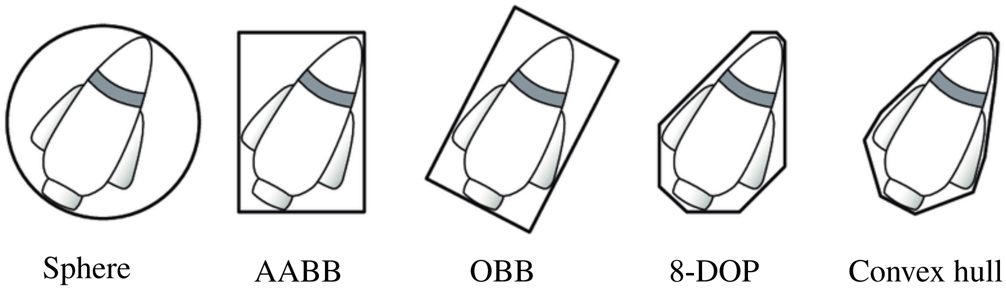


Figure 3.7: Different types of bounding volumes as displayed in Reference [61].

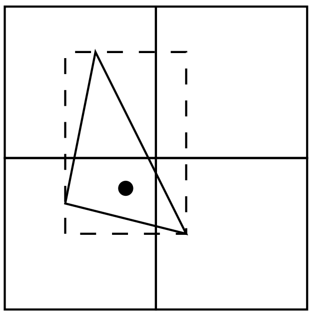


Figure 3.8: Registration of a cell with triangle shape to a quad-tree based on intersections of AABBs of the cell and bins of the quad-tree.

Figure 3.8 shows registration of a triangle-shaped cell to the octree (only some portion of the octree is shown). The cell is registered to the octree based on intersection of AABB (shown with dashed rectangle) and the bins. Since the AABB intersects all bins in the figure, the cell is assigned to all the bins, although the triangle does not intersect all the bins. Registration of a cell to an incorrect bin due to AABB intersection test is preferable compared to cost of using expensive polygonal intersection tests. Intersection tests based on AABBs cause cells to be duplicated in different octree-bins. In order avoid redundant operations, such as solution of governing equations on duplicate cells, the spatial partition which contains the centroid of the cell (shown as a filled circle in Figure 3.8) is assigned as the 'owner' of the cell and the cell is tagged as a 'resident' of the spatial partition. In flow solution, for example, the governing equations are solved only on the resident cells.

3.3.2 Load calculation

The load in each bin is calculated by summing up the number of cells. However, it is possible to calculate load differently depending on the objective of specific operation. From assembler point of view, the aim is to determine overlapping cells, hence, the load of a bin is zero unless there are at least two mesh-blocks in the bin. On the other hand, for flow solver, load of a bin is proportional to the number of cells irrespective of parent meshes. As shown in the results section, solver is more time-consuming than assembler, therefore, bin loads are calculated based on cost definition of flow solver.

Load distribution in the octree is communicated in all processors in order to identify the the most loaded bin of the octree synchronously. Each processors communicates the local octree and the loads in bins are summed up to obtain the global load distribution. The most loaded bin is refined to a 2x2x2 sub-bin and all cells in the parent bin are registered to the sub-bin. After the refinement, the geometry of the octree still remains identical in all processors as the most loaded bin to be refined is the same in all processors. However, since processors possess different mesh-blocks, identical bins have different loads in different processors.

3.3.3 Weighted graph partitioning

The bins of octree are input to METIS as graph-vertices. Figure 3.9 shows a two-dimensional version of octree (for visualization purpose) and its graph representation. Connectivity of bins are provided to METIS in order to obtain connected graph which is needed for contiguous partitions. A connection between two bins is made if the bins share a common face. For example, in Figure 3.9, bins 1 and 3 are not connected as they only share a vertex, not a face.

The load of each bin corresponds to the weight of each graph-vertex. `METIS_PartGraphKway` function of METIS computes a *k*-way partitioning where *k* is the number of processors. If no partitioning with less than 10% imbalance is found, the algorithm returns back to octree refinement as shown in Figure 3.2. The *k*-way

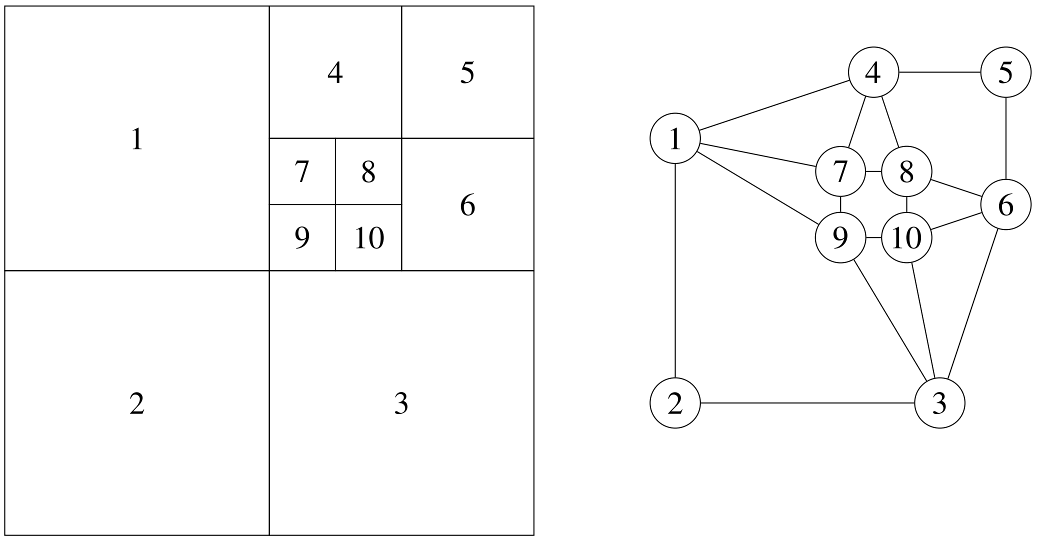


Figure 3.9: A quad-tree and its graph representation.

partitioning algorithm of METIS disregards the partitioning layout computed in previous refinement stage. After every refinement, the partitioning layout is recomputed from scratch. The algorithm, which makes use of the partitioning layout of previous iterations, can be significantly reduce the partitioning time.

Once a satisfactory partitioning is obtained after all refinements, each partition (containing multiple bins) are distributed to respective processors. Finally, the mesh-blocks corresponding to the same parent meshes in partitions are merged in order to reduce intra-processor communication. Note that, ParMETIS, which is the parallel version of METIS, cannot be used at this stage since distribution of octree bins to processors in unknown.

3.4 Data transfer

MPI is a library independent of any programming language. Although this feature allows MPI to be used by a programming language, it also complicates communication of data types. Communication of native types such as `int` and `double` is relatively

```
struct S
{
    int i[3];
    double d[5];
};
```

Figure 3.10: A simple user-define data type.

```
data_type[0]=MPI_INT;
data_type[1]=MPI_DOUBLE;

block_length[0]=3;
block_length[1]=5;

displacement[0]=0;
displacement[1]=12;

MPI_Type_create_struct(2, block_length,
                       displacement, data_type, &S);
MPI_Type_commit(&S);
```

Figure 3.11: Description of user-defined data type to MPI.

much easier than communication of derived or user-defined data types. Derived data type which is usually expressed in a container, specifically `class` or `struct` form in C++, need to be described to MPI byte-by-byte with correct relative positions of members. Figure 3.10 shows a simple user-defined data type and Figure 3.11 shows the description of the user-defined data to MPI in C++ programming language. Description of more complicated user-defined data types is cumbersome and error prone. In a worse scenario, when a user-defined data structure contains a mixture of members stored in stack and heap memories, MPI either fails to execute the communication or require time consuming work-arounds.

Boost MPI which is a library in C++ extends the capabilities of MPI by allowing users to work in C++ environment but, especially simplifies data transfer by using Boost

Serialization library underneath. The serialization library serializes any data in binary/ASCII/XML and other formats without forcing users to expose data types manually. Temporarily serialized data is deserialized in receiver processor as a background process. The main shortcomings of the serialization library is slower data transfer and higher memory usage. However, at initial stages of development, Boost libraries speed-up the development and at the same time, reduces code complexity.

The template `class Exchanger` as shown in Figure 3.12 is specialized/implemented for communication of different data types such as communication of spatial partitions after load balancing or communication of cell types after assembly. Note that specializations are required to inherit from `class Exchanger` and implement the virtual `prepare_storage` function.

The data to be sent are grouped into groups of `struct Group` by the destination rank so that the data can be sent once for each destination. Figure 3.13 shows the function that sends and receives data. Note that for a specific group, another send operation would not begin until the previous one is completed. However, meanwhile receive operation can be processed, thus, the send communication is non-blocking. This method is more efficient than sending and receiving all data for once. It is possible that all send and receive operations to be completed after this function. However, usually, more receive operations are left over. The receive operation shown in Figure 3.14 is repeated until no more receive requests left. The whole send/recv process can be seen in Figure 3.15.

The main drawback of the method is that once the communication is completed, the class instance is deallocated after the class instance gets out of scope. Although this routine is recommended in C++ programming, in the case of MPI it is problematic. The buffers that are used for sending/receiving data need to be reused. Otherwise, as explained in Reference [62], even though class instance is deallocated, the chunk of memory used by the buffers cannot be reused. As a result, memory usage of the developed assembler and the solver keep increasing until they get out of memory. The serialization/deserizalition library which is used for data transfer is also used for saving/restoring data files to restart the simulation. Reusage of the buffers is one of the future works of the study.

```
template<typename T> using ArrCon = std::vector<T>;
template<typename T>
class Exchanger
{
    protected:

    struct Group
    {
        int dest_rank_;
        int dest_tag_;
        std::deque<T> data_;

        void add(const T& data);

        Group();
    };

    public:

    Exchanger();
    void exchange();
    const ArrCon<T>& arrival() const;
    ArrCon<T>& arrival();
    const std::deque<Group>& group() const;

    protected:

    void send_recv();
    void add();
    std::deque<Group>& group();
    ArrCon<T> arrival_;

    private:

    const boost::mpi::communicator* comm_;
    std::vector<int> global_nrecv_;
    int nrecv_;
    std::deque<Group> group_;
    int total_nrecv_;

    void prepare();
    void inform_receivers();
    void send();
    void recv();
    virtual void prepare_storage() = 0;
};
```

Figure 3.12: The template for data transfer. The arguments of the functions are missing for simplification.

```
template<typename T>
void Exchanger<T>::send()
{
    for (Group& gr: group_)
    {
        boost::mpi::request req =
        comm.isend(gr.dest_rank_, gr.dest_tag_, gr.data_);

        while (true)
        {
            auto status = req.test();

            if (status)
            {
                break;
            }

            recv(rank);
        }
    }
}
```

Figure 3.13: The generic send function. The arguments of the functions are missing for simplification.

```
void Exchanger<T>::recv()
{
    if (nrecv_ == total_nrecv_) {
        return;
    }

    auto status =
    comm_->iprobe(
    boost::mpi::any_source, boost::mpi::any_tag);

    if (!status) {
        return;
    }

    int nincoming = global_nrecv_[status->source()];

    std::vector<T> temp;
    comm_->recv(status->source(), status->tag(), temp);
    nrecv_ += nincoming;
    std::copy(temp.begin(), temp.end(),
    std::back_inserter(arrival_));
}
```

Figure 3.14: The generic receive function. The arguments of the functions are missing for simplification.

```
template<typename T>
void Exchanger<T>::send()
{
    nrecv_ = 0;

    arrival_.clear();

    send(*comm_, verbose, rank, "", profiler);

    while (nrecv_ != total_nrecv_) {
        recv(verbose, rank, "", profiler);
    }

    comm_->barrier();
}
```

Figure 3.15: The generic send/recv function. The arguments of the functions are missing for simplification.

3.5 Donor search

Donor search categorizes all mesh-cells into certain types to be fed to flow solver which invokes different algorithms based on the type of cells. Definitions of cell types are listed below.

- Field or computational cell on which governing equations are solved.
- Receptor cell on which governing equations are not solved, but flow variables are interpolated from a (donor) field cell. Since component meshes are generated independently and there is no cell-to-cell connectivity across component meshes, flow variables need to be interpolated from donors to receptors across meshes.
- Mandatory receptor cell which is located next to an intergrid boundary submerged inside another mesh. Since boundary condition cannot be applied intrinsically, it is mandatory to interpolate flow variables.
- Hole cell on which neither governing equations are not solved nor, flow variables are interpolated.

- Orphan cell is a mandatory receptor cell whose donor cell is also a receptor cell.

  It can be avoided by converting the type of donor cell from receptor to field cell.

Several cell types of structured Cartesian and unstructured overset mesh after donor search are shown in Figure 1.4. As explained in the following sections, in the case of multiple overlapping cells coexisting in a region of space, the cell which has the smallest volume is chosen to be the (donor) field cell and the remaining cells are assigned to be receptor cells. This is why the Cartesian mesh, in Figure 1.4, being a coarser mesh, do not have any field cells.

Regardless of order of interpolation between donors and receptors, interface fluxes will not be conserved in whole mesh-system due to non-conforming faces of intergrid cells. Conservation of fluxes can be maintained by regenerating mesh in intergrid region so that intergrid faces of mesh-blocks conforms perfectly [63, 64] or by using auxiliary mesh in the intergrid region [65]. In this work, neither of the flux-conservation methods are implemented and conservative variables are interpolated resulting in non-conservative interface fluxes.

In the following sections, details about identification of cell types are explained. In Sections 3.5.1 and 3.5.2, identification of the hole cells and the remaining cell types, respectively, are explained.

3.5.1 Hole cutting

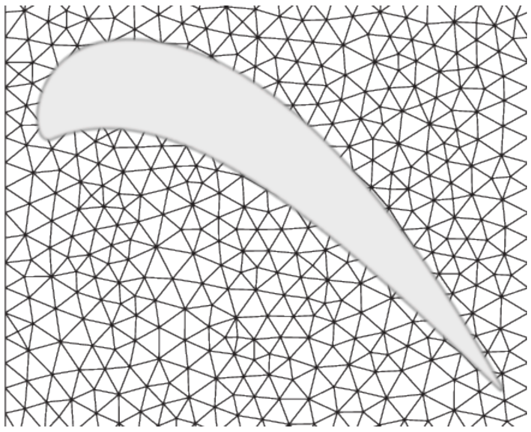
Components are non-porous and closed volumes through which fluid cannot flow. In the context of hole cutting, these volumes are defined as holes.

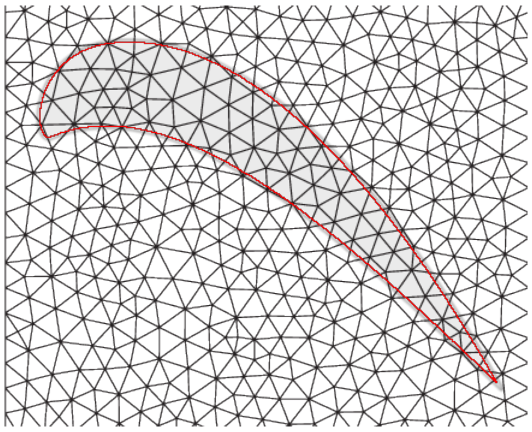
Hole cutting task is for determining hole cells, that is, the cells which overlap one or more holes in another mesh. There are broadly two approachs to hole cutting: Exact and approximate cutting.

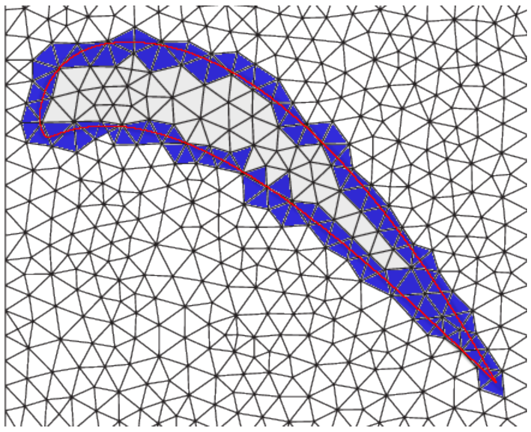
Direct cutting which is an exact cutting method identifies hole cells in two stages. Consider an unstructured mesh and a hole as shown in Figure 3.16a. The cells of the unstructured mesh that intersect the hole are to be developed. The hole is represented by the boundary faces (shown in red in Figure 3.16b) and AABB of the hole. In parallel overset grid assembly, usually, only fragments of entire hole boundary are present in processors. In this case, processors communicate AABBs of fragments of holes to set up the AABB of the entire hole boundary.

First, geometric search operations are performed to identify the cells of the unstructured mesh (shown in blue in Figure 3.16c that overlap the boundary of the hole. These blue cells are also called boundary profile. In order to speed up geometric search, a data structure, such as binary or quad tree, which stores unstructured cells, can be used. Second, the remaining cells that are interior to the hole (shown in green in Figure 3.16d are identified with one of the variants of flood/fill algorithm [66].

The flood/fill algorithm in direct cutting starts with a seed cell which is known to be interior to the boundary profile. The algorithm is not robust as the unstructured mesh cells inside the hole profile may be multiply connected, that is, some of the interior cells may be separated by the hole profile. In this case, multiple interior seeds are required. Once an interior seed is found, the seed tags the neighbor (untagged, white) cells as hole cells. The tagging operation continues recursively until no more neighbor untagged cell is left. If recursive tagging spils out of the AABB of the hole, all the cells are untagged


(a) A hole on an unstructured mesh.


(b) The cells on and inside the red hole boundary are to be identified.


(c) The blue cells indicate the cell which overlap the hole boundary.

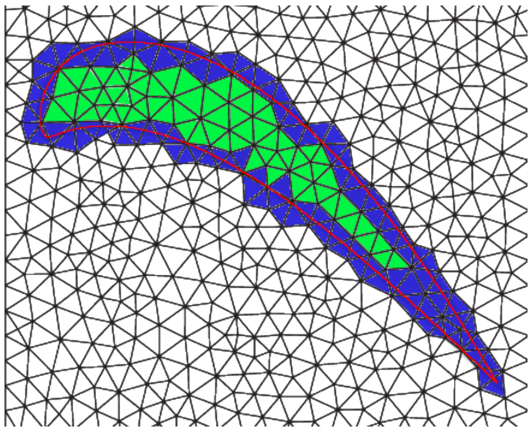

(d) Green interior cells are identified with flood/fill algorithm.

Figure 3.16: Direct cutting.

until the root of the search tree. Usage of AABB for stopping the recursive search is required in case the seed happens to be exterior to the hole profile.

Approximate methods, on the other hand, relies on simple geometric shapes in order to identify the hole cells. One of the popular approximate structure is hole map which is a Cartesian mesh. In Figure 3.17a, a green hole boundary and a red outer boundary, are shown. First, a Cartesian mesh is set on the hole. Initial resolution of the Cartesian mesh is optional and even an AABB (Cartesian mesh with only one bin) is sufficient to represent the hole. The aim of hole mapping method is to refine the Cartesian mesh until no bins of the Cartesian mesh intersects both hole and outer boundaries. Figure 3.17b shows an intermediate stage of hole mapping. Note that, some of the bins still intersects both hole and outer boundaries. Figure 3.17c shows the final resolution of the Cartesian mesh. Next, in Figure 3.17d, the hole boundary is approximated with (green) bins of the Cartesian mesh. Comparison of Figure 3.16c and Figure 3.17d shows that in the former, the hole boundary is represented with unstructured mesh cells exactly whereas in the latter, the hole boundaries are approximated with the bins of the Cartesian mesh.

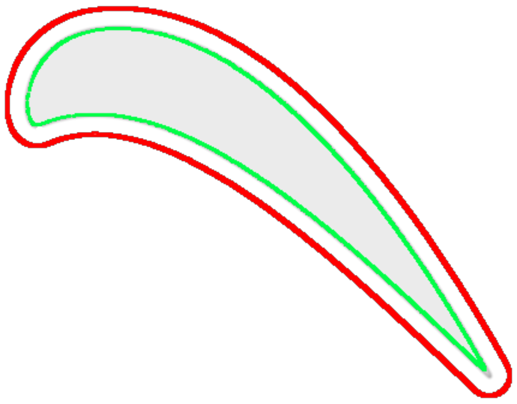
Similar to direct cutting, the bins interior to the hole are identified with flood/fill algorithm as shown in Figure 3.17e, however, in this case, flood algorithm is more robust due to working on simpler geometric shapes (quadrangles in hole map versus polygons in unstructured grid, in two-dimensional space). For the same reason, non-robust flood/fill algorithm using interior seeds are avoided and scan-line flood/fill algorithm is implemented similar to Reference [8]. Scan-line algorithm traverse each row/line in the hole map and tags bins located between already tagged bins.

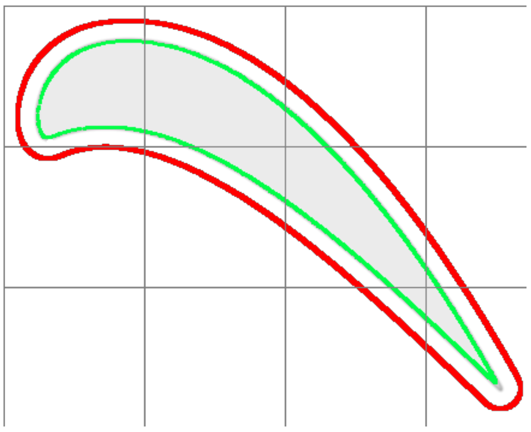
The final Cartesian mesh or the hole map is used in order to identify the hole cells. If centroid of an unstructured mesh cell such as in Figure 3.16 is contained by any green bin of the hole map, the cell is tagged as a hole cell. It is possible that an unstructured mesh cell intersects a green bin of the hole map but the centroid of the cell is exterior to any green bin of the hole map. In this case, the unstructured mesh cell would be, falsely, tagged as a non-hole cell as the hole mapping method is approximate. However, if the hole cutting is used in conjunction with the core algorithm (explained in Section 3.5.2), hole cells are identified correctly.

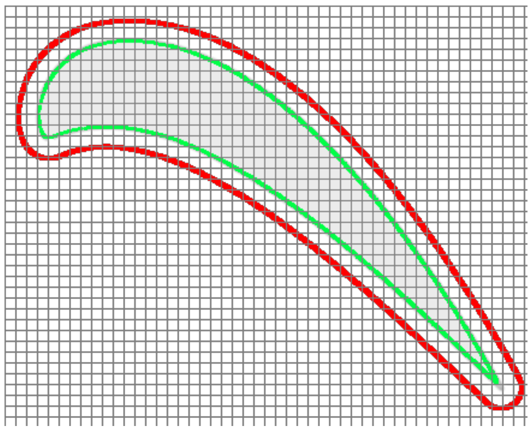
3.5.2 Core algorithm

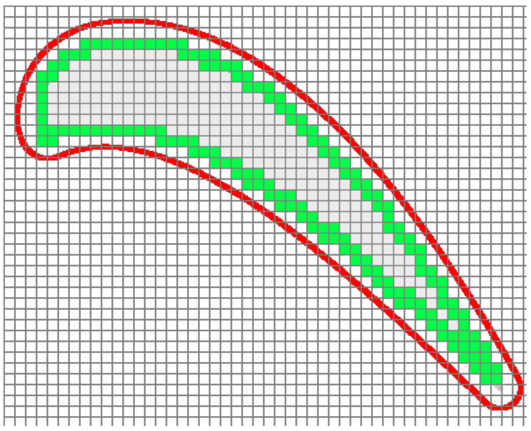
In the first time step, donor search is performed with Alternating Digital Tree (ADT) [54] which is a space partitioning binary tree. ADT splits spatial domain into portions with axis-aligned cutting planes, recursively. The axis of cutting plane alternates in cyclic order in accordance with the levels in ADT. In three dimensional space, there are three cutting planes with cutting axes parallel to x-, y-, z-axis. The position of cutting plane along the cutting axis depends on the number of dimensions required to define an object to be inserted. For example, if a point (zero-dimensional) is to be inserted to an ADT, positions of cutting planes would be (x, y, z) coordinates of the point along the respective cutting axes. Moreover, if the inserted object is a rectangle, positions of cutting planes would be the six (x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}) dimensions of the rectangle in the respective orientations of the cutting plane. In general, objects to be inserted to ADT are n-orthotopes or hyperrectangles. In the example above, point and rectangle are 1- and 2-orthotopes. In this work, the objects are AABB, which is 3-orthotope, of mesh-cells. Therefore, both the domain and objects to be inserted are six dimensional.

For visualization of insertion operation into ADT, Figure 3.18 demonstrates insertion of points into ADT in two dimensional space. In the root level, point A is the median in x-axis and it is associated with the whole domain, Ω_A. Domain Ω_A is divided by a plane with cutting axis parallel to the x-axis at point A. Next, cutting axis is changed to be the y-axis. Point B which is one of the points in the left portion of Ω_A and is the median in y-axis is assigned to Ω_B which is the left portion of Ω_A. Point C is inserted similar


(a) Red outer boundary and green hole boundary.


(b) Some of the bins intersect both outer and hole boundaries.


(c) Hole map is refined until none of the bins intersect both outer and hole boundaries.


(d) Bins which intersect the hole boundary are identified as hole profile.

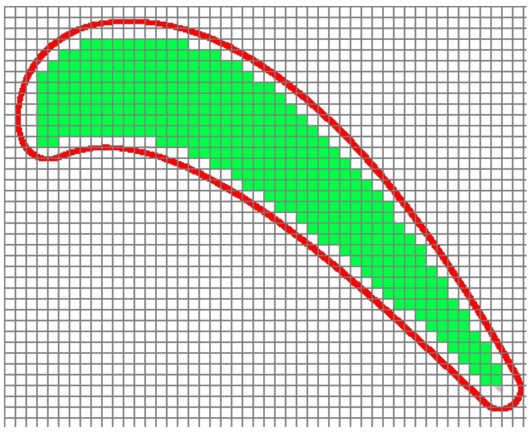

(e) Interior cells are identified with scan-line flood/fill algorithm.

Figure 3.17: Hole mapping.

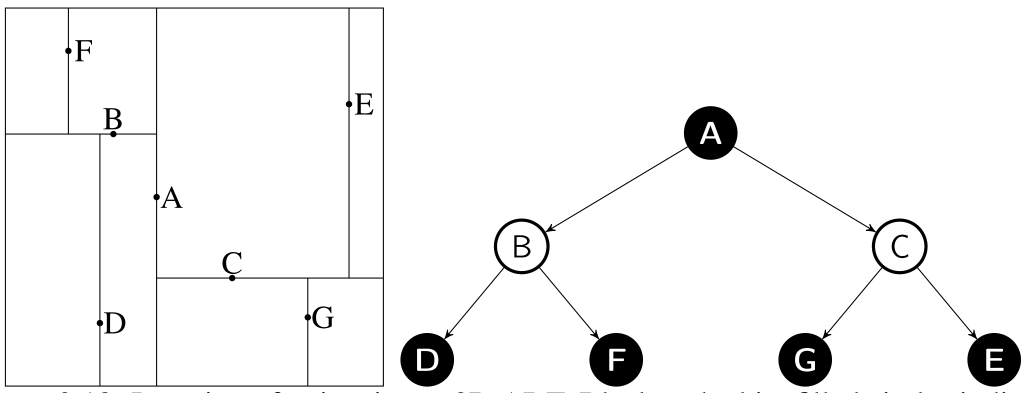
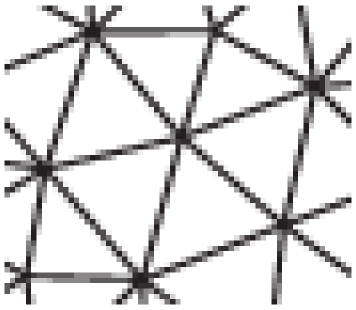


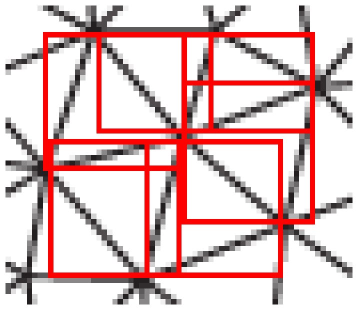
Figure 3.18: Insertion of points into a 2D ADT. Black and white filled circles indicate partitioning of domain in x- and y-axis.

to Point B except that it belongs to the right portion of Ω_A. The rest of the points are inserted to sub-branches of the tree in the same fashion.

Higher-dimensional objects such as polygons are inserted to ADT with their AABBs. Figure 3.19a shows some of the cells of unstructured mesh. AABB of each cell, as shown in Figure 3.19b, is inserted to ADT by one of their six components, (x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max}) depending on the current cutting axis. Note that, unlike cell/polygon of the unstructed mesh, the AABBs are overlapping.

Search operation on ADT is similar to insertion operation. Given a query point, as shown in Figure 3.19c, AABBs which overlap the query point are returned. If the query point is outside the hypercube of the ADT, search operation returns an empty array meaning that none of AABBs in the tree overlaps the query point. Assuming that the query point is inside the hypercube of the tree, the search operates recursively by always starting from the root node. First, it is tested whether the query point is contained by the AABB in the current node. If so, the AABB is added to the array which is to be returned after the recursive search operation is completed. Regardless of containment


(a) Some of the cells of an unstructured grid.


(b) Red AABBs around the cells.

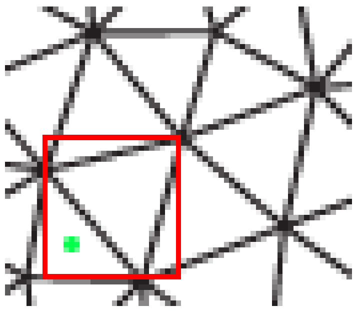

(c) Green query point is inside the AABB but
outside the polygon of the cell.

Figure 3.19: Donor search with ADT of AABBs.

of the query point by the AABB, the query point is redirected to one or both portions of the current node by testing the dimension of the query point against the cutting axis associated with the current level.

Note that the geometric shape of mesh-cells are polyhedra which are not axis-aligned in an unstructured grid. However, ADT accepts only axis-aligned objects to be inserted. Therefore, given a mesh-block, M_i, AABB and associated ID of each mesh-cell is inserted to the respective ADT T_i. Once an ADT tree is constructed for every mesh-block, candidate donor cell for each mesh-cell in M_i is searched out by providing the centroid (query point) of the cell as input to all trees T_j where, $j \neq i$. As the output of search operation, each tree T_j returns an array of IDs corresponding to the mesh-cells in

M_j. Although the query point intersects one of the AABBs in the returned array such as in the case of Figure 3.19c, the query point may not be contained by the polygon of the associated cell. Therefore, the query point is tested for containment by each polyhedron corresponding to the returned ID. The mesh-cell which contains the query point is added to the list of candidate donors. The procedure is shown in Figure 3.20.

```
foreach Mesh-block $M_i$ do
    foreach Tree $T_j$ where $j \neq i$ do
        foreach Mesh-cell $c_i$ in $M_i$ do
            Query-point qp = centroid of $c_i$;
            array[$\text{ID}_1, \cdots, \text{ID}_n$] = $T_j$.search(qp);
            foreach ID in array do
                if $M_j$.cell(ID).polyhedron contains qp then
                    Add $M_j$.cell(ID) to the list of candidate donors of $c_i$;
                end
            end
        end
    end
end
```

Figure 3.20: Donor search using ADT

In the first time step, ADT is used for donor search. In subsequent time steps, donor search is performed with stencil walk algorithm if a cell had a donor in previous time step, otherwise ADT is used as in the first time step. Referring to Figure 3.21, stencil walk algorithm starts with choosing a starting or seed cell (shaded) and walking towards the query point (marked with a cross) by using cell-to-cell connectivity. First, a search line is drawn between the centroid of the seed cell and the query point. The search line is tested for intersection by the faces of the seed cell. The current cell (which was initially the seed cell) is updated to be the neighbor cell which shares the intersected

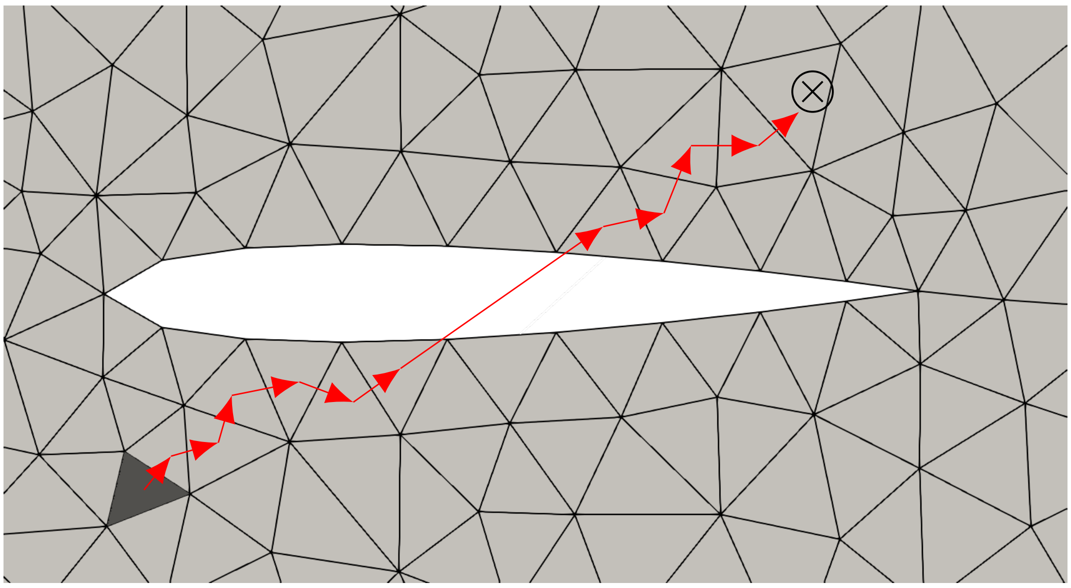


Figure 3.21: Stencil walk algorithm starts from seed cell (shaded cell) and using cell-to-cell connectivity walk towards the cell which contains the target (shown as a cross in circle).

face. When the current cell is near a boundary, and therefore, has no neighbor which shares the intersected face, all other faces of that boundary are tested for intersection by the search line. If an intersected boundary face found, the current cell is updated to be the interior cell of that boundary. If no other boundary face is found, then the algorithm stops indicating that no cell containing the query point is found. When none of the faces of the current cell intersects the search line, the current cell is added to the list of candidate donor cells.

The reason of using stencil walk algorithm in the subsequent time steps is the absence of seed cells in the first time step. For receptors, the seeds are chosen to be the centroid of donor cells. For other cells types, seeds are not available, therefore, as mentioned, ADT is used for donor search. Since stencil walk algorithm involves costly geometric intersections tests, high number of walks is avoided by not picking random seed cells and using ADTs whenever proper seed cells are not available.

After donor search, if no candidate donor cell is found for a mesh-cell, the query point is either outside of M_j or inside a hole (if exists) in M_j. If M_j has a hole, and if the query point is inside any of the bins of hole map of M_j, then the mesh-cell is labelled as hole cell. Once a mesh-cell is labelled as a hole cell, its type cannot be changed regardless of interaction of the mesh-cell with other mesh-blocks. Otherwise, if the mesh-cell is not a hole cell, it is labelled as field cell.

If a candidate donor cell is found and the volume of the query cell is larger than that of the candidate donor cell, then the mesh-cell is labelled as receptor cell. A receptor cell may have multiple candidate donor cells. In this case, the candidate donor cell which has the smallest volume is assigned as the donor of the mesh-cell. If the mesh-cell has a smaller volume than a candidate donor cell, the mesh-cell is labelled as field cell. The reason for choosing the cell with smaller volume is to represent the domain with the highest resolution possible. Another option is to choose the cell closest to respective boundary. This kind of selection is beneficial in preserving boundary layer cells.

The cells which have a hole neighbor or are located next to an intergrid boundary are labelled as mandatory receptor cells. These receptor cells serve as intergrid boundary conditions.

Volume of intersection between mesh-blocks can be reduced in order to save time by avoiding redundant interpolation of variables from donors to receptors. Minimization of overlapped region is performed by excluding receptors having no field neighbor from flow solution.

Once donor search is completed, if the assembler and the solver have different partitioning layouts, the assembler maps cell types and donor IDs to the layout of the solver. In the case of sharing of the same partitioning layout, no further operation is required.

3.5.3 Interpolation scheme

For overset mesh methodology, several interpolation schemes are possible to interpolate flow variables from donor cells to receptor cells [67]. The first possibility is the first order inverse distance scheme that uses weighted average of multiple donor cells in order to find interpolated function as shown in Equation (3.1).

$$\mathbf{U}_r = \frac{\sum \omega \mathbf{U}_d}{\sum \omega} \tag{3.1}$$

where, $\mathbf{U}_r$ and $\mathbf{U}_d$ are conservative variables for the receptor and donor cells, respectively. The weight for each donor cell is defined as the inverse distance between the centroids of the donor cell and the receptor cell as shown in Equation (3.2). The interpolated function is guaranteed to be bounded by the flow variables of donor cells, therefore, no further use of limiters is required. In Equation (3.2), $\overrightarrow{x_r}$ - $\overrightarrow{x_d}$ is the distance vector from the receptor cell centroid to the donor cell centroid.

$$\omega = \frac{1}{\left\| \overrightarrow{x_r} - \overrightarrow{x_d} \right\|}. \tag{3.2}$$

The scheme used in this work is a second order scheme that involves the gradient of flow variables ($\nabla\mathbf{U}_d$) at a single donor cell as shown in Equation (3.3). Compared to the inverse distance scheme, the interpolated value can be greater than extrema of flow variables at donor cells. In order to limit the interpolated flow variables to the extrema of donor cells, a limiter Φ is required. Details of the limiter function are explained in Section 3.6.5.

$$\mathbf{U}_r = \mathbf{U}_d + \Phi \nabla \mathbf{U}_d \cdot (\overrightarrow{x_r} - \overrightarrow{x_d}) \tag{3.3}$$

3.6 Mathematical formulation

3.6.1 Overview

The Euler equations model fluid flow problems when inertial or convective force dominates the flow and the effect of viscosity through solid-fluid and fluid-fluid is negligible. They have the advantage of admitting discontinuous solutions such as shock wave. In the case of Navier-Stokes equations, shock capturing is cumbersome as shock wave has a definite thickness, therefore, the validity of continuum assumptions becomes questionable. In absense of viscous forces, solution of governing equations simplifies without sacrificing important features of flow. In the case of rotor-fuselage interaction, the Euler equations are satisfactory for prediction of pressure estimations. However, the Euler equations have drawbacks in capturing rotor wake structure.

Figure 3.22 shows the flowchart of the flow solver. The algorithm operates within each physical time step. If the numerical formulation is implicit or dual-time step approach is used, multiple non-linear iterations are needed to solve the non-linear system of equations. In the beginning of the algorithm, receptors interpolate variables from their donors. If the numerical formulation is implicit, a linear system of equations is solved to compute the change in conserved variables. In the case of explicit solver, the change in conserved variables is computed explicitly without solving the linear system of equations. Face fluxes are computed with different methods: Roe's method and HLLC as explained in Sections 3.6.3 and 3.6.4. In the algorithm, global residual corresponds to the maximum of residuals among all processors. As the mesh-system is partitioned, ghost cells in remote partitions are updated after solution of flow variables.

3.6.2 Governing equations

Differential form of the time dependent and three dimensional Euler equations [68] is

$$\mathbf{U}_t + \mathbf{F}_x + \mathbf{G}_y + \mathbf{H}_z = 0 \tag{3.4}$$

where, **U** is vector of conserved variables

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix} \tag{3.5}$$

and **F**, **G** and **H** are vectors of fluxes in x-, y- and z- directions, respectively.

$$\mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{bmatrix} \tag{3.6}$$

In equations Equation (3.5) and Equation (3.6), ρ is density, u, v and w are velocity components in x-, y- and z- directions, respectively, E is total energy and p is static pressure.

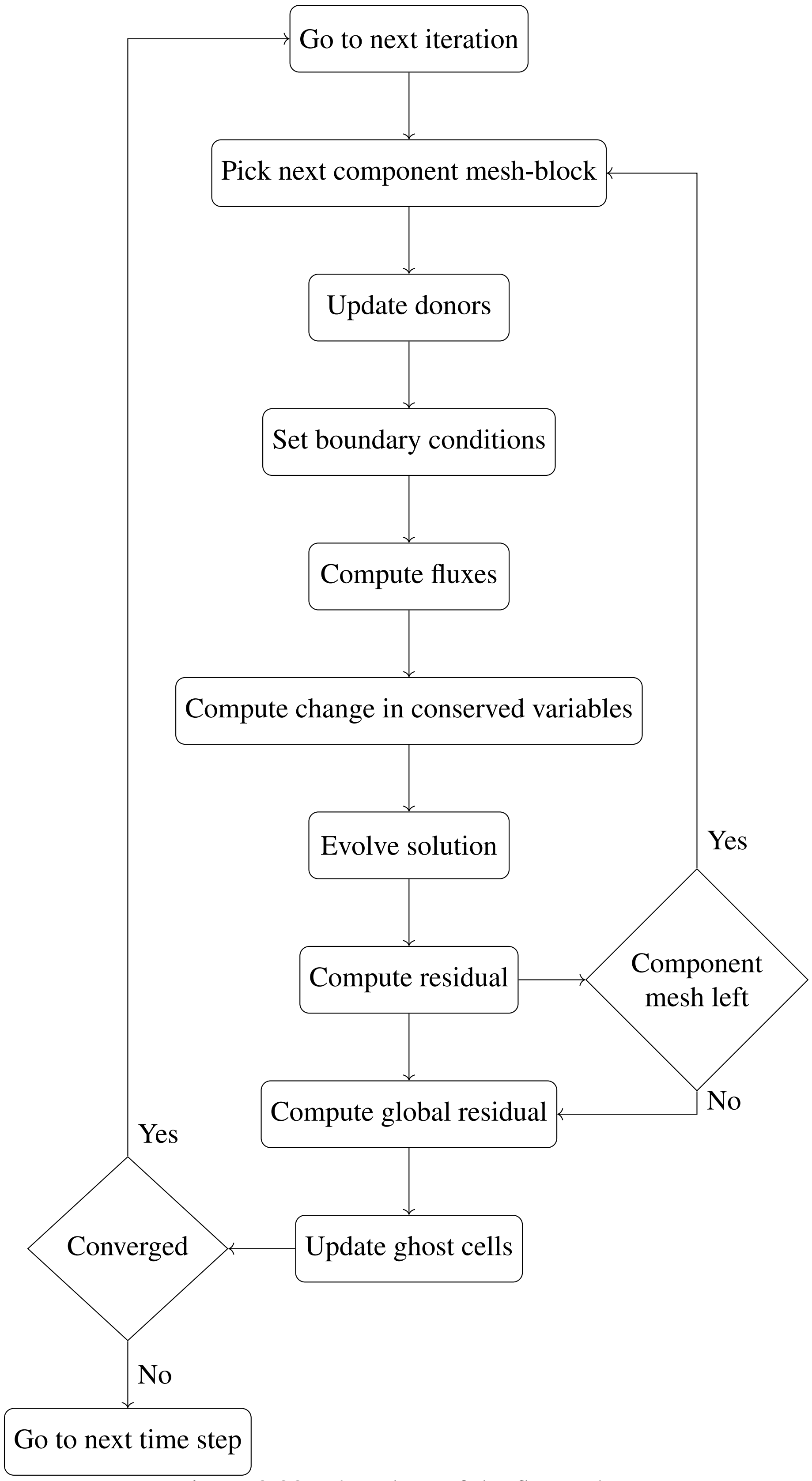


Figure 3.22: Flowchart of the flow solver.

In order to admit discontinuous solutions such as shocks and contact discontinuity, integral form of Euler equations is considered as follows:

$$\frac{\partial}{\partial t}\int_V \mathbf{U}\,\mathrm{d}V + \int_A \mathscr{H}\cdot\mathbf{n}\,\mathrm{d}A = 0 \tag{3.7}$$

where, V is a control volume, A is area of one of the faces of V, $\mathbf{n}$ is the outward unit vector normal to the area A and $\mathscr{H}$ is tensor of fluxes $\mathscr{H} = (\mathbf{F}, \mathbf{G}, \mathbf{H})$.

Conservative variables $\mathbf{U}$ in Equation (3.7) are assumed to be constant throughout the unit volume V. This assumption causes a loss of spatial accuracy inversely proportional to the unit volume. This kind of formulation is called Finite Volume Formulation. In this context, the unit volume corresponds to a finite volume or a cell. Moreover, cell-centered discretization is used, that is, conservative variables are stored at the centroid of a cell, as shown in Figure 3.23. Orientation of fluxes $\mathscr{H}$ depends on the solution and may, in fact, be in opposite direction. Alternative of cell-centered discretization is node-centered discretization which stores variables at cell vertices. Both discretizations have merits and drawbacks explained in Reference [69]. Mainly, node-centered discretization is more suitable to high order spatial discretizations. In this work, second order spatial accuracy is targeted, therefore, cell-centered discretization is appropriate. Finally, the finite volume formulation is used on unstructured mesh with no limitation to geometry of cells. Mathematical formulation can be applied to any polyhedral geometry such as tetrahedron and hexadedron.

The term $\mathscr{H}\cdot\mathbf{n}\,\mathrm{d}A$ represents flux component normal to the face A, and is found by rotating fluxes $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ to face-normal direction with the rotation matrix

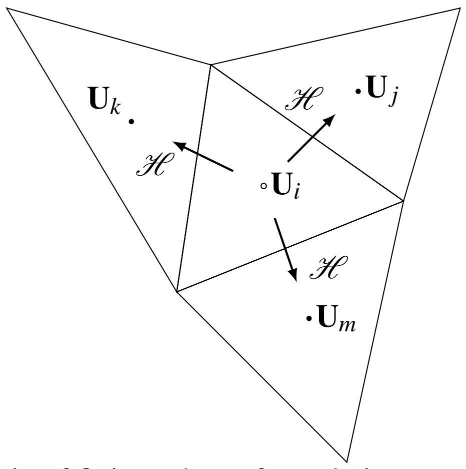


Figure 3.23: Schematic of finite volume formulation on an unstructured mesh.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta^{(y)}\cos\theta^{(z)} & \cos\theta^{(y)}\sin\theta^{(z)} & \sin\theta^{(y)} & 0 \\ 0 & -\sin\theta^{(z)} & \cos\theta^{(z)} & 0 & 0 \\ 0 & -\sin\theta^{(y)}\cos\theta^{(z)} & -\sin\theta^{(y)}\sin\theta^{(z)} & \cos\theta^{(y)} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

where, $\theta^{(y)}$ and $\theta^{(z)}$ are found from components of unit normal vector $[n_x, n_y, n_z]^T$ or explicitly

$$\begin{aligned} n_x &= \cos\theta^{(y)}\cos\theta^{(z)} \\ n_y &= \cos\theta^{(y)}\sin\theta^{(z)} \\ n_z &= \sin\theta^{(y)} \end{aligned} \tag{3.9}$$

such that

$$\begin{aligned} \theta^{(z)} &= \text{atan2}(n_y, n_x) \\ \theta^{(y)} &= \sin^{-1}(n_z) \end{aligned} \tag{3.10}$$

atan2 is another version of $\tan^{-1}$ and by taking two arguments instead of one, it returns a unique value unlike $\tan^{-1}$.

Rotational invariance of Euler equations reads

$$\mathscr{H} \cdot \mathbf{n} = \mathbf{T}^{-1}\mathbf{F}(\mathbf{TU}) \tag{3.11}$$

After the rotation, the variables ρ and E in $\mathbf{U}$ are unaffected by the rotation however, the velocity components in rotated frame of reference have become

$$\hat{u} = \cos\theta^{(y)}\cos\theta^{(z)}u + \cos\theta^{(y)}\sin\theta^{(z)}v + \sin\theta^{(y)}w \tag{3.12}$$

$$\hat{v} = -\sin\theta^{(z)}u + \cos\theta^{(z)}v \tag{3.13}$$

$$\hat{w} = -\sin\theta^{(y)}\cos\theta^{(z)}u + -\sin\theta^{(y)}\sin\theta^{(z)}v + \cos\theta^{(y)}w \tag{3.14}$$

In Finite Volume formulation, $\mathbf{U}$ is uniform throughout the control volume, therefore, the first term of Equation (3.7) can be simplified such that

$$V\frac{\partial \mathbf{U}}{\partial t} + \int_A \mathbf{T}^{-1}\mathbf{F}(\mathbf{TU})\,\mathrm{d}A = 0 \tag{3.15}$$

It is also possible to rewrite Equation (3.15) as follows

$$\mathbf{T}^{-1}\left[V\frac{\partial(\mathbf{TU})}{\partial t} + \int_A \mathbf{F}(\mathbf{TU})\,\mathrm{d}A\right] = 0 \tag{3.16}$$

Defining rotated conserved variables as $\hat{\mathbf{U}} = [\rho, \hat{u}, \hat{v}, \hat{w}, E]^T \equiv \mathbf{TU}$ and dropping $\mathbf{T}^{-1}$ in Equation (3.16) leads to

$$V\frac{\partial \hat{\mathbf{U}}}{\partial t} + \int_A \mathbf{F}(\hat{\mathbf{U}})\,\mathrm{d}A = 0 \tag{3.17}$$

Discretizing the second term as summation of flux through each face of a control volume:

$$V\frac{\partial \hat{\mathbf{U}}}{\partial t} + \sum_{f}^{\text{nface}} \mathbf{F}(\hat{\mathbf{U}})A_f = 0 \tag{3.18}$$

Returning back to the differential form:

$$\hat{\mathbf{U}}_t + \mathbf{F}(\hat{\mathbf{U}})_x = 0 \tag{3.19}$$

Equation (3.19) is called x-split Riemann problem. It is solved at faces of control volumes with initial values

$$\hat{\mathbf{U}}(x,t=0) = \begin{cases} \hat{\mathbf{U}}_L & \text{if} \quad x < 0, \\ \hat{\mathbf{U}}_R & \text{if} \quad x > 0, \end{cases} \tag{3.20}$$

where, $\hat{\mathbf{U}}_L$ and $\hat{\mathbf{U}}_R$ are data states at left and right cells of an arbitrary face. x in Equation (3.20) is the coordinate in face-normal direction and its origin is on the face in local frame of reference. In the rest of the chapter, the cap notation, $\hat{()}$, is dropped for simplicity.

In quasi-linear form, with chain rule:

$$\mathbf{U}_t + \mathbf{A}\mathbf{U}_x = 0 \tag{3.21}$$

where, $\mathbf{A}$ is the Jacobian of the flux $\mathbf{F}$. Equation (3.21) is three-dimensional hyperbolic partial differential equations; have five real eigenvalues with corresponding five right eigenvectors. The eigenvalues are

$$\lambda_1 = u - a, \lambda_2 = \lambda_3 = \lambda_4 = u, \lambda_5 = u + a \tag{3.22}$$

and the corresponding right eigenvectors which satisfies $\mathbf{A}\mathbf{K}^{(i)} = \lambda_i \mathbf{K}^{(i)}$ are

$$K^{(1)} = \begin{bmatrix} 1 \\ u-a \\ v \\ w \\ H-ua \end{bmatrix} \quad K^{(2)} = \begin{bmatrix} 1 \\ u \\ v \\ w \\ k \end{bmatrix} \quad K^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ v \end{bmatrix} \quad K^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ w \end{bmatrix} \quad K^{(5)} = \begin{bmatrix} 1 \\ u+a \\ v \\ w \\ H+ua \end{bmatrix} \tag{3.23}$$

Each characteristic field (eigenvalue and associated eigenvector) corresponds to a particular wave structure. The characteristic fields are shown in the t-x diagram in Figure 3.24. In the diagram, the region between the left and the right data states or between the λ_1 and λ_5 characteristic fields is called the star region. For Euler equations with Equations of State as closure conditions, there are three possible wave structures: Shock wave, contact discontinuity, rarefaction or expansive wave, and shear wave. $K^{(1)}$ and $K^{(5)}$ correspond to either shock wave or rarefaction wave across which primitive variables change, discontinuously and smoothly, respectively. Waves that belong to λ_2 family of characteristics are all coincident and correspond to contact discontinuity across which pressure and normal velocity are constant and density changes discontinuously. In addition, in three-dimensional space, tangential velocities v (λ_3) and w (λ_4) also change discontinuously across shear waves that are coincident to contact discontinuity. In general, solution at an arbitrary space and time is obtained by superposition of local waves that emanate from each face. In Finite Volume formulation, as solution is uniform throughout a control volume, solution inside a control volume is found by simply adding fluxes through the cell faces.

In order to determine the variables which change across waves, Generalized Riemann Invariants [70] is used

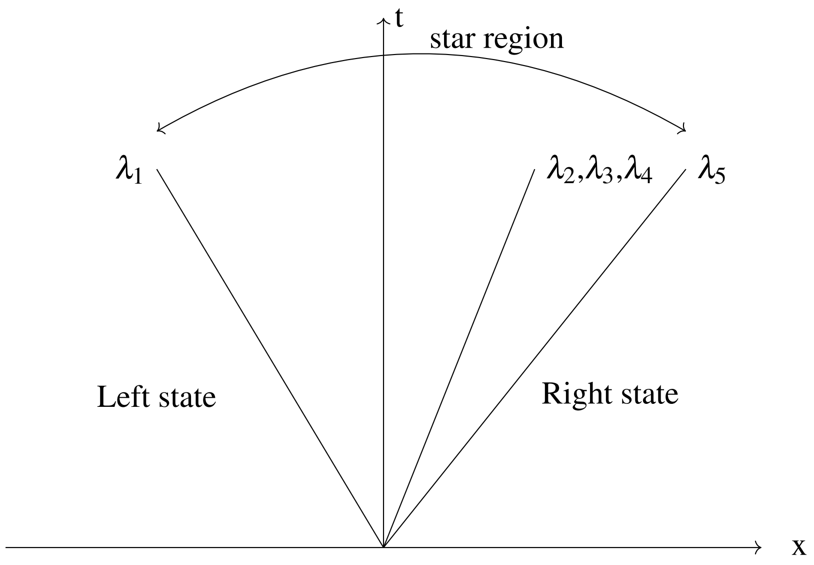


Figure 3.24: t-x diagram.

$$\frac{\mathrm{d}U_1}{K_1^{(i)}} = \frac{\mathrm{d}U_2}{K_2^{(i)}} = \frac{\mathrm{d}U_3}{K_3^{(i)}} = \frac{\mathrm{d}U_4}{K_4^{(i)}} = \frac{\mathrm{d}U_5}{K_5^{(i)}} \tag{3.24}$$

where, the subscripts in Equation (3.24) corresponds to the components of vector of conserved variables or the associated right eigenvector and superscript, i indicates the λ_i-wave family.

Split Riemann problem Equation (3.19) is similar to one-dimensional Riemann problem for Euler equations except the additional shear waves, thus, simplifying the application of Riemann solver. Equation (3.19) is shown in order to show similarity with one-dimensional case and also to show wave possible wave structures, and for solution of Euler equations integral form will be used instead of differential form.

In overset mesh methodology, each (component or background) mesh is independent from other meshes, and also can move relative to other meshes. Therefore, the mesh-system is not fixed in space but also does not move with material or fluid.

Moving mesh velocities are incorporated to Euler equations with Arbitrary Lagrangian-Eulerian (ALE) so that meshes move with predetermined velocities. Mesh velocity components $[b_x, b_y, b_z]^T$ in rotated frame of reference are substracted from Eulerian velocity components as follows:

$$\begin{aligned} u &= u - b_x \\ v &= v - b_y \\ w &= w - b_z \end{aligned} \tag{3.25}$$

Exact solution of Riemann problem for Euler equations require numerical solution of an implicit equation and no analytical solution is possible. From computational point of view, considering that Riemann problem is to be solved for each face and in every time step, exact solution of Riemann problem is impractical, therefore, fluxes are computed approximately. In general, there are two ways to approximately solve Riemann problem: Evaluating flux function with an approximate state as in Primitive Variable Riemann Solvers (PVRS) and Two-Rarefaction Riemann Solver (TRRS) and approximating flux directly as in Roe family of solvers [71–76], Osher family of solvers, HLL (Harten, Lax and van Leer) family of solvers [76–80]. Roe Riemann solvers are known to produce rarefaction shock which is an unphysical solution violating the entropy condition unless appropriate entropy fix is applied. One of the advantages of HLL family of solvers is positivity preservation, that is, density and dependent properties such as internal energy and speed of sound is always positive. Of HLL family of solvers, HLLC (C stands for contact discontinuity) [80] which is a three-wave model corresponding to three characteristic fields. The one-dimensional Euler equations consist of three characteristic fields: Rarefaction, shock and contact discontinuity. Pure HLL solver which is based on two-wave model ignores contact discontinuity, therefore, it is an incomplete Riemann solver for Euler equations. HLLC, on the other hand, being a three-wave model is a complete Riemann solver for the Euler equations. In three-dimensional space, note that, the contact discontinuity has multiplicity of three and out of five characteristics only three are distinct. Hence, HLLC is also a complete Riemann solver for the Euler equations in three-dimensional space.

3.6.3 Flux approximation with Roe

With Roe Riemann solver, face flux is approximated by replacing the Jacobian in Equation (3.21) with approximate Roe Jacobian calculated with Roe-averaged variables.

$$\mathbf{F} = \left[\frac{\mathbf{F}_L + \mathbf{F}_R}{2} + \frac{\widetilde{\mathbf{A}}(\mathbf{U}_L - \mathbf{U}_R)}{2}\right] A_f \tag{3.26}$$

where, $\widetilde{\mathbf{A}}$ is the Roe Jacobian

$$\widetilde{\mathbf{A}} = \widetilde{\mathbf{K}} |\widetilde{\lambda}| \widetilde{\mathbf{K}}^{-1} \tag{3.27}$$

and $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{K}}^{-1}$ are the right and the left eigenvectors and $|\widetilde{\lambda}|$ are the absolute eigenvalues

$$|\widetilde{\lambda}| = \begin{bmatrix} |\widetilde{u} - \widetilde{a}| & 0 & 0 & 0 & 0 \\ 0 & |\widetilde{u}| & 0 & 0 & 0 \\ 0 & 0 & |\widetilde{u}| & 0 & 0 \\ 0 & 0 & 0 & |\widetilde{u}| & 0 \\ 0 & 0 & 0 & 0 & |\widetilde{u} + \widetilde{a}| \end{bmatrix} \tag{3.28}$$

$$\widetilde{\mathbf{K}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ \widetilde{u}-\widetilde{a} & \widetilde{u} & 0 & 0 & \widetilde{u}+\widetilde{a} \\ \widetilde{v} & \widetilde{v} & 1 & 0 & \widetilde{v} \\ \widetilde{w} & \widetilde{w} & 0 & 1 & \widetilde{w} \\ \widetilde{H}-\widetilde{a}\widetilde{u} & \widetilde{k} & \widetilde{v} & \widetilde{w} & \widetilde{H}+\widetilde{a}\widetilde{u} \end{bmatrix} \tag{3.29}$$

$$\widetilde{\mathbf{K}}^{-1} = \begin{bmatrix} \widetilde{H}+\widetilde{a}(\widetilde{u}-\widetilde{a})/\gamma^* & -(\widetilde{u}+\widetilde{a}/\gamma^*) & -\widetilde{v} & -\widetilde{w} & 1 \\ -2\widetilde{H}+4\widetilde{a}^2/\gamma^* & 2\widetilde{u} & 2\widetilde{v} & 3\widetilde{w} & -2 \\ -2\widetilde{v}\widetilde{a}^2/\gamma^* & 0 & 2\widetilde{a}^2/\gamma^* & 0 & 0 \\ -2\widetilde{w}\widetilde{a}^2/\gamma^* & 0 & 0 & 2\widetilde{a}^2/\gamma^* & 0 \\ \widetilde{H}-\widetilde{a}(\widetilde{u}+\widetilde{a})/\gamma^* & -\widetilde{u}+\widetilde{a}/\gamma^* & -\widetilde{v} & -\widetilde{w} & 1 \end{bmatrix} \tag{3.30}$$

calculated with approximate Roe variables

$$\begin{aligned} \widetilde{\rho} &= \sqrt{\frac{\rho_R}{\rho_L}} \\ \widetilde{u} &= \frac{u_L+\widetilde{\rho}u_R}{1+\widetilde{\rho}} \\ \widetilde{v} &= \frac{v_L+\widetilde{\rho}v_R}{1+\widetilde{\rho}} \\ \widetilde{w} &= \frac{w_L+\widetilde{\rho}w_R}{1+\widetilde{\rho}} \\ \widetilde{H} &= \frac{H_L+\widetilde{\rho}H_R}{1+\widetilde{\rho}} \\ \widetilde{k} &= \frac{\widetilde{u}^2+\widetilde{v}^2+\widetilde{w}^2}{2} \\ \widetilde{a} &= \sqrt{(\gamma-1)(\widetilde{H}-\widetilde{k})}. \end{aligned} \tag{3.31}$$

In Equation (3.31), $\widetilde{\rho}$, $\widetilde{u}$, $\widetilde{H}$, $\widetilde{k}$, $\widetilde{a}$ are the Roe-averaged density, normal velocity, total enthalpy, kinetic energy and speed of sound and $\widetilde{v}$ and $\widetilde{w}$ are the Roe-averaged tangential velocities.

3.6.4 Flux approximation with HLLC

The HLLC solver, in addition to HLL solver which only considers two-wave model, accounts for contact discontinuity and shear waves in order to completely solve Riemann problem for Euler equations.

The differential form of Euler equations in face-normal coordinate frame in Equation (3.32) is valid on control volume centered at cell centroid. Equation (3.32) is also valid on a face such that

$$\mathbf{U}_t + \mathbf{F}_x = 0 \tag{3.32}$$

where, x is face-normal coordinate and $x = 0$ at a face. The conservation law in integral form:

$$\frac{d}{dt}\int_{x_L}^{x_R} \mathrm{d}x = \mathbf{F}(\mathbf{U}(x_L,t)) - \mathbf{F}(\mathbf{U}(x_R,t)) \tag{3.33}$$

Considering a wave which can be shock, contact or head of rarefaction wave at position $s(t)$ which is located between x_L and x_R, the left side of equation above becomes:

$$\frac{d}{dt}\int_{x_L}^{s(t)} \mathrm{d}x + \frac{d}{dt}\int_{s(t)}^{x_R} \mathrm{d}x = \mathbf{F}(\mathbf{U}(x_L,t)) - \mathbf{F}(\mathbf{U}(x_R,t)) \tag{3.34}$$

Using relation:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\int_{\xi_1(\alpha)}^{\xi_2(\alpha)} f(\xi,\alpha)\,\mathrm{d}\xi = \int_{\xi_1(\alpha)}^{\xi_2(\alpha)} \frac{\partial f}{\partial \alpha}\,\mathrm{d}\xi + f(\xi_2,\alpha)\frac{\mathrm{d}\xi_2}{\mathrm{d}\alpha} - f(\xi_1,\alpha)\frac{\mathrm{d}\xi_1}{\mathrm{d}\alpha} \tag{3.35}$$

for the above equation:

$$S[\mathbf{U}(s_L,t) - \mathbf{U}(s_R,t)] + \int_{x_L}^{s(t)} \mathbf{U}_t(x,t)\,\mathrm{d}x + \int_{s(t)}^{x_R} \mathbf{U}_t(x,t)\,\mathrm{d}x = \mathbf{F}(\mathbf{U}(x_L,t)) - \mathbf{F}(\mathbf{U}(x_R,t)) \tag{3.36}$$

where, S is the speed of the discontinuity and $\mathbf{U}(s_L,t)$ and $\mathbf{U}(s_R,t)$ are the vector of conservative variables when approaching the discontinuity from left and right, respectively. Shrinking the control volume up to wave or approaching the wave from x_L and x_R vanishes the integral terms:

$$S[\mathbf{U}(s_L,t)-\mathbf{U}(s_R,t)] = \mathbf{F}(\mathbf{U}(x_L,t)) - \mathbf{F}(\mathbf{U}(x_R,t)) \tag{3.37}$$

Equation (3.37) can be expressed simply as

$$\Delta\mathbf{F} = S\Delta\mathbf{U} \tag{3.38}$$

which is called the Rankine-Hugoniot relation. Applying the Rankine-Hugoniot relation to the fastest waves, that is, except the middle wave:

$$\mathbf{F}_{*K} = \mathbf{F}_K + S_K(\mathbf{U}_{*K} - \mathbf{U}_K) \tag{3.39}$$

where, superscript $*$ corresponds to the star region in Figure 3.24, $K = L$ for the left and $K = R$ for the right wave.

Pressure and normal velocity across middle wave (contact discontinuity or shear wave) are constant and tangential velocities jump:

$$\begin{aligned} p_{*L} = p_{*R} &= p_* \\ u_{*L} = u_{*R} = u_* &= S_* \\ v_{*L} &= v_L \\ v_{*R} &= v_R \end{aligned} \tag{3.40}$$

From tangential momentum equation (3rd or 4th rows) in Equation (3.39):

$$\rho_* = \rho_K \frac{u_K - S_K}{S_* - S_K} \tag{3.41}$$

From momentum equation in normal direction (2nd row in the system of equations) in Equation (3.39) and replacing ρ_{*L} with Equation (3.41):

$$p_* = p_K + \rho_K (S_K - u_K)(S_* - u_K) \tag{3.42}$$

From energy equation (5th row) in Equation (3.39):

$$E_* = \rho_K \frac{S_K - u_K}{S_K - S_*} \left[\frac{E_K}{\rho_K} + (S_* - u_K) \left[S_* + \frac{p_K}{\rho_K (S_L - u_K)} \right] \right] \tag{3.43}$$

Setting above equations:

$$S_* = \frac{p_R - p_L + \rho_L u_L (S_L - u_L) - \rho_R u_R (S_R - u_R)}{\rho_L (S_L - u_L) - \rho_R (S_R - u_R)} \tag{3.44}$$

As a result, $\mathbf{U}_{*K}$ is:

$$\mathbf{U}_{*K} = \rho_K \left(\frac{S_K - u_K}{S_K - S_*} \right) \begin{bmatrix} 1 \\ S_* \\ v_K \\ w_K \\ \frac{E_K}{\rho_K} + (S_* - u_K) \left[S_* + \frac{p_K}{\rho_K (S_K - u_K)} \right] \end{bmatrix} \tag{3.45}$$

Now everything is known in Rankine-Hugoinot equation.

$$\mathbf{F}^{hllc}_{i+\frac{1}{2}} = \begin{cases} \mathbf{F}_L & \text{if} \quad 0 \le S_L, \\ \mathbf{F}_{*L} & \text{if} \quad S_L \le 0 \le S_*, \\ \mathbf{F}_{*R} & \text{if} \quad S_* \le 0 \le S_R, \\ \mathbf{F}_R & \text{if} \quad 0 \ge S_R \end{cases} \tag{3.46}$$

There are several ways for the estimation of the wave speeds such as 1) directly obtaining from data states, 2) using Roe-averaged quantities and 3) solving pressure-based equation. Direct methods are impractical for real problems. In this work, pressure-based estimation [80] is followed. S_L and S_R are estimated as:

$$S_L = u_L - a_L q_L \tag{3.47}$$

$$S_R = u_R + a_R q_R \tag{3.48}$$

$$q_K = \begin{cases} 1 & \text{if} \quad p_* \le p_K \\ \left[1 + \frac{\gamma+1}{2\gamma}(p_*/p_K - 1)\right]^{1/2} & \text{if} \quad p_* \ge p_K \end{cases} \tag{3.49}$$

where, $K = L$ for the left state and $K = R$ for the right state. Pressure in the star region is approximated with pressure derived with Primitive Variable Riemann Solvers (PVRS) [81]:

$$p_{pvrs} = \frac{1}{2}(p_L + p_R) - \frac{1}{2}(u_R - u_L)\bar{\rho}\bar{a} \tag{3.50}$$

where,

$$\bar{\rho} = \frac{1}{2}(\rho_L + \rho_R) \tag{3.51}$$

$$\bar{a} = \frac{1}{2}(a_L + a_R) \tag{3.52}$$

where, a_L and a_R are speed of sound at the left and the right states.

3.6.5 Spatial discretization

Left and right states at faces are computed with MUSCL (Monotonic Upstream-centered Scheme for Conservation Laws) scheme [82]. Second order spatial accuracy is obtained by linear extrapolation of cell-averaged primitive variables to cell faces.

The gradients are found with least-squares optimization [83]. For cell i, value of a conservative variable u at neighbor j is found with

$$u_j = u_i + \frac{\partial u_i}{\partial x}\Delta x + \frac{\partial u_i}{\partial y}\Delta y + \frac{\partial u_i}{\partial z}\Delta z \tag{3.53}$$

where, u_j is the conservative variable at neighbor j and u_i is the conservative variable at the centroid of the cell i. If the same equation is written for each neighbor of the cell i, the result is a system of algebraic equations $Ax = b$. Each matrix and vector in the system of algebraic equations are shown in Equation (3.54). Note that the coefficient matrix A has size of $N \times 3$, where N is the number of neighbors. Considering that the minimum of neighbors is four for a three-dimensional cell and therefore, $N > 3$, the linear system is over-determined.

$$A = \begin{bmatrix} x_1 - x_i & y_1 - y_i & z_1 - z_i \\ \vdots & \vdots & \vdots \\ x_j - x_i & y_j - y_i & z_j - z_i \\ \vdots & \vdots & \vdots \\ x_N - x_i & y_N - y_i & z_N - z_i \end{bmatrix} x = \begin{bmatrix} \frac{\partial u_i}{\partial x} \\ \frac{\partial u_i}{\partial y} \\ \frac{\partial u_i}{\partial z} \end{bmatrix} b = \begin{bmatrix} u_1 - u_i \\ \vdots \\ u_j - u_i \\ \vdots \\ u_N - u_i \end{bmatrix} \tag{3.54}$$

In matrix A, the number of rows is higher than the number of columns, therefore, A is not invertible. The over-determined system cannot be solved but can be approximated by minimizing least-squares problem:

$$\sum(b-\hat{b})^2 = \sum(b-A\hat{x})^2 \tag{3.55}$$

where, $\hat{b}$ is the approximate right-hand side vector and $\hat{x}$ is the approximate solution. After solution of least-squares problem, the gradients are as shown in Equation (3.56) [84].

$$\hat{x} = \begin{bmatrix} \frac{\partial u_i}{\partial x} \\ \frac{\partial u_i}{\partial y} \\ \frac{\partial u_i}{\partial z} \end{bmatrix} = \sum_j^N \begin{bmatrix} \alpha_{ij,1} - \frac{r_{12}}{r_{11}}\alpha_{ij,2} + \Psi\alpha_{ij,3} \\ \alpha_{ij,2} - \frac{r_{23}}{r_{22}}\alpha_{ij,3} \\ \alpha_{ij,3} \end{bmatrix} (u_j - u_i) \tag{3.56}$$

Defining $\Delta(\cdot)_{ij} = (\cdot)_j - (\cdot)_i$, coefficients in Equation (3.56) are shown in Equation (3.57).

$$\begin{gathered} \alpha_{ij,1} = \frac{\Delta x_{ij}}{r_{11}^2}, \quad \alpha_{ij,2} = \frac{1}{r_{22}^2}\left(\Delta y_{ij} - \Delta x_{ij}\frac{r_{12}}{r_{11}}\right) \\ \Psi = \frac{r_{12}r_{23} - r_{13}r_{22}}{r_{11}r_{22}} \\ \alpha_{ij,3} = \frac{1}{r_{33}^2}\left(\Delta z_{ij}^2 - \frac{r_{23}}{r_{22}}\Delta y_{ij} + \Psi\Delta x_{ij}\right) \\ r_{11} = \sqrt{\sum_j^N \Delta x_{ij}^2}, \quad r_{12} = \frac{1}{r_{11}}\sum_j^N \Delta x_{ij}\Delta y_{ij} \\ r_{13} = \frac{1}{r_{11}}\sum_j^N \Delta x_{ij}\Delta z_{ij}, \quad r_{22} = \sqrt{\sum_j^N \Delta y_{ij}^2 - r_{12}^2} \\ r_{23} = \frac{1}{r_{22}}\left(\sum_j^N \Delta y_{ij}\Delta z_{ij} - r_{12}r_{13}\right) \\ r_{33} = \sqrt{\sum_j^N \Delta z_{ij}^2 - (r_{13}^2 + r_{23}^2)} \end{gathered} \tag{3.57}$$

After the gradients are calculated, conservative variables at a face can be constructed from the conservative variables at cell centroid with

$$\mathbf{U}_i(\overrightarrow{r} - \overrightarrow{r_i}) = \mathbf{U}_i + \Phi \nabla \mathbf{U}_i \cdot (\overrightarrow{r} - \overrightarrow{r_i}) \tag{3.58}$$

by setting the position vector, $\overrightarrow{r}$ to the position of face centroid such as $\overrightarrow{r} = \overrightarrow{r_f}$. Φ is either the Barth-Jespersen (BJ) limiter [85] which is known to be satisfactory for shock capturing or the Venkatakrishnan limiter [86] which is a differentiable unlike Barth-Jespersen limiter. In order to calculate Φ, first the minimum $\Delta\mathbf{U}_i^{\min} = \min(\mathbf{U}_j - \mathbf{U}_i)$ and maximum $\Delta\mathbf{U}_i^{\max} = \max(\mathbf{U}_j - \mathbf{U}_i)$ differences in $\mathbf{U}$ between the cell and the neighbors are found. Subsequently, the unlimited values are found at each neighbor as $\mathbf{U}_{ij} = \mathbf{U}_i(\overrightarrow{r_j} - \overrightarrow{r_i})$. Then, the limiter Φ is calculated with

$$\Phi_{ij} = \begin{cases} \min\left(1, \frac{\Delta\mathbf{U}_i^{\max}}{\mathbf{U}_{ij} - \mathbf{U}_i}\right) & \text{if} \quad \mathbf{U}_{ij} - \mathbf{U}_i > 0 \\ \min\left(1, \frac{\Delta\mathbf{U}_i^{\min}}{\mathbf{U}_{ij} - \mathbf{U}_i}\right) & \text{if} \quad \mathbf{U}_{ij} - \mathbf{U}_i < 0 \\ 1 & \text{if} \quad \mathbf{U}_{ij} - \mathbf{U}_i = 0 \end{cases} \tag{3.59}$$

Finally, Φ is found as $\Phi = \min(\Phi_{ij})$.

3.6.6 Implicit formulation

In order to avoid constraining time steps, an implicit formulation is used. If the vector of conservative variables at new level, $\mathbf{U}^{n+1}$ appears on both sides of semi-discretized equations such as

$$V\frac{d\mathbf{U}}{dt} = \sum \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n+1})A_f \tag{3.60}$$

then the Euler equations need to be solved implicitly. The RHS of the equation above is linearized to get

$$\mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n+1})A_f = \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n})A_f + \mathbf{T}^{-1}\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U}^{n})}{\partial(\mathbf{T}\mathbf{U}^{n})}\Delta(\mathbf{T}\mathbf{U}^{n})A_f. \tag{3.61}$$

In the following evaluation of the Jacobian (the last term in Equation (3.61)), for simplicity, superscript $()^n$ is dropped. Derivative of flux function is evaluated with respect to the left and right states such that

$$\mathbf{T}^{-1}\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U})}{\partial(\mathbf{T}\mathbf{U})}\Delta(\mathbf{T}\mathbf{U}) = \mathbf{T}^{-1}\left[\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U}_L)}{\partial(\mathbf{T}\mathbf{U}_L)}\Delta(\mathbf{T}\mathbf{U}_L) + \frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U}_R)}{\partial(\mathbf{T}\mathbf{U}_R)}\Delta(\mathbf{T}\mathbf{U}_R)\right]. \tag{3.62}$$

Since Roe Jacobian $\widetilde{\mathbf{A}}$ has constant coefficients, derivative of $\widetilde{\mathbf{A}}$ with respect to conservative variables is zero. Substituting derivatives of Roe flux in Equation (3.26),

$$\mathbf{T}^{-1}\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U})}{\partial(\mathbf{T}\mathbf{U})}\Delta(\mathbf{T}\mathbf{U}) = \mathbf{T}^{-1}\left[\frac{\mathbf{A}_L + \widetilde{\mathbf{A}}}{2}\Delta(\mathbf{T}\mathbf{U}_L) + \frac{\mathbf{A}_R - \widetilde{\mathbf{A}}}{2}\Delta(\mathbf{T}\mathbf{U}_R)\right]. \tag{3.63}$$

Taking rotation matrix out in $\Delta(\mathbf{T}\mathbf{U}_L)$ and $\Delta(\mathbf{T}\mathbf{U}_R)$,

$$\mathbf{T}^{-1}\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U})}{\partial(\mathbf{T}\mathbf{U})}\Delta(\mathbf{T}\mathbf{U}) = \mathbf{T}^{-1}\left[\frac{\mathbf{A}_L + \widetilde{\mathbf{A}}}{2}\mathbf{T}\Delta\mathbf{U}_L + \frac{\mathbf{A}_R - \widetilde{\mathbf{A}}}{2}\mathbf{T}\Delta\mathbf{U}_R\right]. \tag{3.64}$$

Equation (3.61) becomes

$$\mathbf{T}^{-1}\frac{\partial \mathbf{F}(\mathbf{T}\mathbf{U})}{\partial(\mathbf{T}\mathbf{U})}\Delta(\mathbf{T}\mathbf{U})A_f = \mathbf{M}_L\Delta\mathbf{U}_L - \mathbf{M}_R\Delta\mathbf{U}_R \tag{3.65}$$

where,

$$\mathbf{M}_L = \left(\mathbf{T}^{-1}\frac{\mathbf{A}_L + \widetilde{\mathbf{A}}}{2}\mathbf{T}\right)\Delta\mathbf{U}_L A_f, \tag{3.66}$$

$$\mathbf{M}_R = -\left(\mathbf{T}^{-1}\frac{\mathbf{A}_R - \widetilde{\mathbf{A}}}{2}\mathbf{T}\right)\Delta\mathbf{U}_R A_f. \tag{3.67}$$

Substituting Equation (3.65) into Equation (3.61),

$$\mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n+1})A_f = \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n})A_f + \mathbf{M}_L\Delta\mathbf{U}_L - \mathbf{M}_R\Delta\mathbf{U}_R. \tag{3.68}$$

Substituting Equation (3.68) into the right hand side of Equation (3.60),

$$\sum_f^{N_f} \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n+1})A_f = \sum_f^{N_f} \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U}^{n})A_f + \sum_f^{N_f} \mathbf{M}_L \Delta \mathbf{U}_L^n - \sum_f^{N_f} \mathbf{M}_R \Delta \mathbf{U}_R^n. \tag{3.69}$$

Defining $\mathbf{R}(\mathbf{U}) = \sum_f^{N_f} \mathbf{T}^{-1}\mathbf{F}(\mathbf{T}\mathbf{U})A_f$,

$$\mathbf{R}(\mathbf{U}^{n+1}) = \mathbf{R}(\mathbf{U}^n) + \sum_f^{N_f} \mathbf{M}_L \Delta \mathbf{U}_L^n - \sum_f^{N_f} \mathbf{M}_R \Delta \mathbf{U}_R^n. \tag{3.70}$$

In general, for a cell i, Equation (3.70) can be written as

$$\mathbf{R}(\mathbf{U}_i^{n+1}) = \mathbf{R}(\mathbf{U}^n) + \sum_f^{N_f} \mathbf{M}_i \Delta \mathbf{U}_i^n + \sum_f^{N_f} \mathbf{M}_j \Delta \mathbf{U}_j^n \tag{3.71}$$

where, subscript $()_j$ represents a neighbor value and

$$\mathbf{M}_i = \begin{cases} \mathbf{M}_L & \text{if } i \text{ is on the left of the face } f \\ -\mathbf{M}_R & \text{if } i \text{ is on the right of the face } f \end{cases} \tag{3.72}$$

$$\mathbf{M}_j = \begin{cases} -\mathbf{M}_R & \text{if } i \text{ is on the left of the face } f \\ \mathbf{M}_L & \text{if } i \text{ is on the right of the face } f. \end{cases} \tag{3.73}$$

3.6.7 Temporal discretization

Consider a generic implicit temporal discretization

$$\alpha V \frac{\Delta \mathbf{U}_i}{\Delta t} - \mathbf{H} + \mathbf{R}(\mathbf{U}_i^{n+1}) = 0. \tag{3.74}$$

where, α is a coefficient depending on temporal discretization and $\mathbf{H}$ is the higher order terms of the temporal discretization. Substituting Equation (3.71) into Equation (3.74) and rearranging,

Table 3.2: The coefficient and higher order term for different temporal discretizations.

Implicit Scheme	α	$\mathbf{H}$
Euler	1	$\mathbf{0}^T$
Three-time level	$\frac{3}{2}$	$\frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^n - \mathbf{U}^{n-1})$

$$\left(\alpha\frac{V}{\Delta t} + \sum_f^{N_f}\mathbf{M}_i\right)\Delta\mathbf{U}_i = -\mathbf{R}(\mathbf{U}_i) + \sum_f^{N_f}\mathbf{M}_j\Delta\mathbf{U}_j + \mathbf{H}. \tag{3.75}$$

Consider three-time level backward Euler discretization

$$V\frac{3\mathbf{U}_i^{n+1} - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}}{2\Delta t} = -\mathbf{R}(\mathbf{U}_i^{n+1}). \tag{3.76}$$

Separation of the first and higher order terms in the temporal term yields

$$V\frac{3\mathbf{U}_i^{n+1} - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}}{2\Delta t} = \frac{3}{2}\frac{V}{\Delta t}\Delta\mathbf{U}_i - \frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}_i^n - \mathbf{U}_i^{n-1}). \tag{3.77}$$

In the form of Equation (3.75),

$$\left(\frac{3}{2}\frac{V}{\Delta t} + \mathbf{M}_i\right)\Delta\mathbf{U}_i = -\mathbf{R}(\mathbf{U}_i^n) + \sum_f^{N_f}\mathbf{M}_j\Delta\mathbf{U}_j + \frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^n - \mathbf{U}^{n-1}). \tag{3.78}$$

In the case of three-time-level backward Euler method, $\alpha = 3/2$ and $\mathbf{H} = \frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^n - \mathbf{U}^{n-1})$. Table 3.2 shows α and $\mathbf{H}$ for the first and second order Euler methods. If a steady state solution is sought, Δt is set to an infinitely high number.

3.6.8 Dual time step approach

For overset mesh simulation, time step needs to be as high as possible in order to reduce the number of time steps, hence, load rebalance. In order to lower the time step, dual time step (DTS) approach [87] is used. DTS solves the unsteady equation as if a steady equation by introducing a pseudo time step in addition to physical time step. Usage of local pseudo time steps speeds up convergence to a solution.

Consider three-time level backward Euler method with first and higher order terms separated in pseudo time

$$\frac{3}{2}\frac{V}{\Delta t}(\mathbf{U}^{n+1}-\mathbf{U}^{n})-\frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^{n}-\mathbf{U}^{n-1})=-\mathbf{R}(\mathbf{U}^{n+1}). \tag{3.79}$$

Denoting new time level as $s+1$ instead of $n+1$

$$\frac{3}{2}\frac{V}{\Delta t}(\mathbf{U}^{s+1}-\mathbf{U}^{n})-\frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^{n}-\mathbf{U}^{n-1})=-\mathbf{R}(\mathbf{U}^{s+1}). \tag{3.80}$$

Adding pseudo temporal term to the left hand side

$$\frac{V}{\Delta\tau}(\mathbf{U}^{s+1}-\mathbf{U}^{s})+\frac{3}{2}\frac{V}{\Delta t}(\mathbf{U}^{s+1}-\mathbf{U}^{n})-\frac{1}{2}\frac{V}{\Delta t}(\mathbf{U}^{n}-\mathbf{U}^{n-1})=-\mathbf{R}(\mathbf{U}^{s+1}) \tag{3.81}$$

where, $\Delta\tau$ is the pseudo local time step calculated as

$$\Delta\tau=\mathrm{CFL}\frac{V}{\sum_{f}^{\mathrm{nface}}\max(|\lambda|)A_f}. \tag{3.82}$$

and $\max(|\lambda|)$ is the maximum absolute eigenvalue among all faces of a cell.

In the beginning of a dual time step solution, the conservative variables at pseudo and physical times are equal, that is, $\mathbf{U}^{s}=\mathbf{U}^{n}$.

Substracting $\frac{3V}{2\Delta t}\mathbf{U}^{s}$ from both sides and rearranging

$$\left(\frac{V}{\Delta\tau}+\frac{3}{2}\frac{V}{\Delta t}\right)\Delta\mathbf{U}^{s}=-\mathbf{R}(\mathbf{U}^{s+1})-\frac{V}{2\Delta t}(3\mathbf{U}^{s}-4\mathbf{U}^{n}+\mathbf{U}^{n-1}). \tag{3.83}$$

where, $\Delta\mathbf{U}^{s}=\mathbf{U}^{s+1}-\mathbf{U}^{s}$. Equation (3.83) is solved to steady state that is $\Delta\mathbf{U}^{s}\to 0$.

When steady state is reached Equation (3.79) is recovered. The conservative variables at the new time level are updated as $U^{n+1} = U^{s+1}$.

3.6.9 Solution of linear system of equations

Combining Equation (3.78) and Equation (3.83)

$$\left(\frac{V}{\Delta\tau} + \frac{3}{2}\frac{V}{\Delta t} + \mathbf{M}_i\right)\Delta\mathbf{U}_i^s = -\mathbf{R}(\mathbf{U}_i^s) + \sum_f^{N_f}\mathbf{M}_j\Delta\mathbf{U}_j^s - \frac{V}{2\Delta t}(3\mathbf{U}_i^s - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}). \quad (3.84)$$

By defining

$$\mathbf{D}_i = \frac{V}{\Delta\tau} + \frac{3}{2}\frac{V}{\Delta t} + \mathbf{M}_i, \quad (3.85)$$

$$\mathbf{O}_j = -\sum_f^{N_f}\mathbf{M}_j, \quad (3.86)$$

$$\mathbf{b}_i = -\mathbf{R}(\mathbf{U}_i^s) - \frac{V}{2\Delta t}(3\mathbf{U}_i^s - 4\mathbf{U}_i^n + \mathbf{U}_i^{n-1}), \quad (3.87)$$

$$\mathbf{x} = \Delta\mathbf{U}^s \quad (3.88)$$

Equation (3.84) is written in $\mathbf{Ax} = \mathbf{b}$ form where,

$$\mathbf{A} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{O}_2 & \cdots & \mathbf{O}_n \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{O}_1 & \mathbf{O}_2 & \cdots & \mathbf{D}_n \end{bmatrix}. \quad (3.89)$$

Equation (3.89) is block matrix having 5×5 block at each entry. The linear system of equations is solved by AMGCL [88] with an algebraic multigrid solver which uses smoothed aggregation at coarsening, incomplete LU factorization at relaxation and GMRES as iterative solver.

After solution of the linear system of equations, the conservative variables at new pseudo time are updated as $\mathbf{U}^{s+1} = \mathbf{U}^s + \Delta\mathbf{U}^s$

Chapter 4

RESULTS AND DISCUSSION

4.1 Validation of the overset mesh solver

This section includes test cases to evaluate validity of both the assembler and the flow solver. Some of the settings are common in all test cases. First, the working fluid is assumed to be air which has the ratio of specific heats $\gamma = 1.4$ and the ideal gas equation is used to close the Euler equations. Second, the flow solver is developed to handle three-dimensional problems therefore, lower dimensional problems are solved in three-dimensional context by assigning empty boundary conditions to appropriate surfaces. Third, only the Roe's Riemann solver is used with the implicit formulation. Finally, tolerance of numerical error is set as 10^{-12}.

4.1.1 Shock tube

Shock tube problem is a one-dimensional problem useful in validating the accuracy of Riemann solvers. Two gases with different densities and pressures are separated by a discontinuous membrane (contact discontinuity) in middle of 1 unit-length tube as shown in Figure 4.1. Initial conditions are shown in Table 4.1. Both end of the tube are set to Dirichlet boundary condition. Flux through the remaining surfaces of the tube are ignored by setting them to empty boundary condition. This allows solution of one- and two-dimensional problems in three-dimensional framework. The number of cells is 100.

Table 4.1: Initial conditions for the shock tube.

Variable	Left	Right
Density	1.0	0.125
Pressure	1.0	0.1
Velocity	0.0	0.0

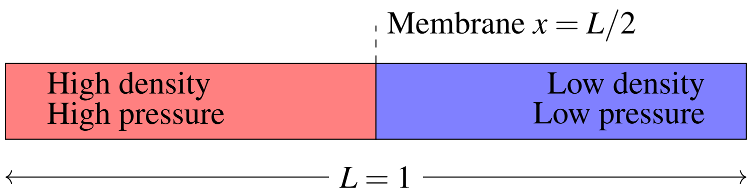


Figure 4.1: Shock tube.

At $t = 0$, the membrane disappears and three different types of waves emerge: A rarefaction wave that travels to the left, a shock wave that travels to the right and a contact discontinuity in-between. Figure 4.2 shows pressure profile at $t = 0.2$ seconds obtained with the HLLC Riemann solver. It is obvious that second order MUSCL scheme with Venkatakrishnan and Barth-Jespersen limiters produce more accurate results than first order upwind scheme. Setting the coefficient $K = 0.3$ for Venkatakrishnan limiter captures discontinuities better than Barth-Jespersen limiter however, in this case, the former is not TVD (Total Variation Diminishing), hence, produces overshoots.

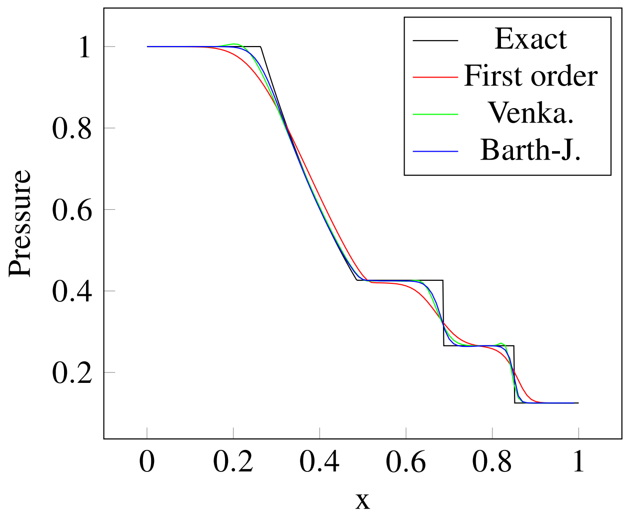


Figure 4.2: Pressure profile in the shock tube.

4.1.2 Steady transonic flow over airfoil

This is a two-dimensional problem featuring steady transonic flow past a NACA 0012 airfoil with chord length $c = 1$. Figure 4.3 shows the O-shaped unstructred mesh around the airfoil. The mesh contains 48468 triangular prismatic cells. The radius of outer boundary is 30 chord length. Free-stream pressure is set such that the speed of sound is unity. Airfoil is rotated $1.25°$ clock-wise to attain angle of attack. Flow conditions are shown in Table 4.2. Initial flow field is set to free-stream values.

The Euler equations are solved implicitly with backward Euler method. As the flow is steady, first order temporal discretization is preferred over the second order three-time level Euler method. MUSCL scheme with Venkatakrishnan limiter is used to obtain second order spatial accuracy. CFL number of set to 10.

Slip-wall boundary condition is applied to the airfoil surface. Normal velocity component at the wall is set to the opposite of that of the neighbor interior cell to prevent mass flux throught wall surfaces. At the farfield, Riemann invariant boundary condition [89] is preferred over Dirichlet boundary condition which is known to undesirably reflect transient flow back the computational domain. The remaining surfaces are set to empty boundary condition.

The same problem is solved with overset mesh approach. A cubic background mesh with sides 40 unit length is added to the mesh-system. Radius of the airfoil mesh is shrinked from 30 to 10 unit length. Figure 4.4 shows the background and the airfoil mesh. The farfield boundary condition on the outer overset mesh boundary is assigned to the background mesh surfaces and replaced with overset mesh boundary condition.

Figure 4.5 shows pressure coefficient on the airfoil. The present results are in good agreement with the inviscid reference data [90]. The maximum difference between the pressure coefficients of the single mesh method and Reference [90] is $\Delta C_p \approx 30.1\%$ occuring right after the shock on the upper surface of the airfoil. The single and overset mesh approaches are perfectly matching with the maximum difference in pressure coefficients $\Delta C_p \approx 10^{-3}\%$ among all points on the airfoil. The good agreement between the results of the single and overset mesh methods is expected since the outer overset mesh boundary is sufficiently away from the airfoil surface.

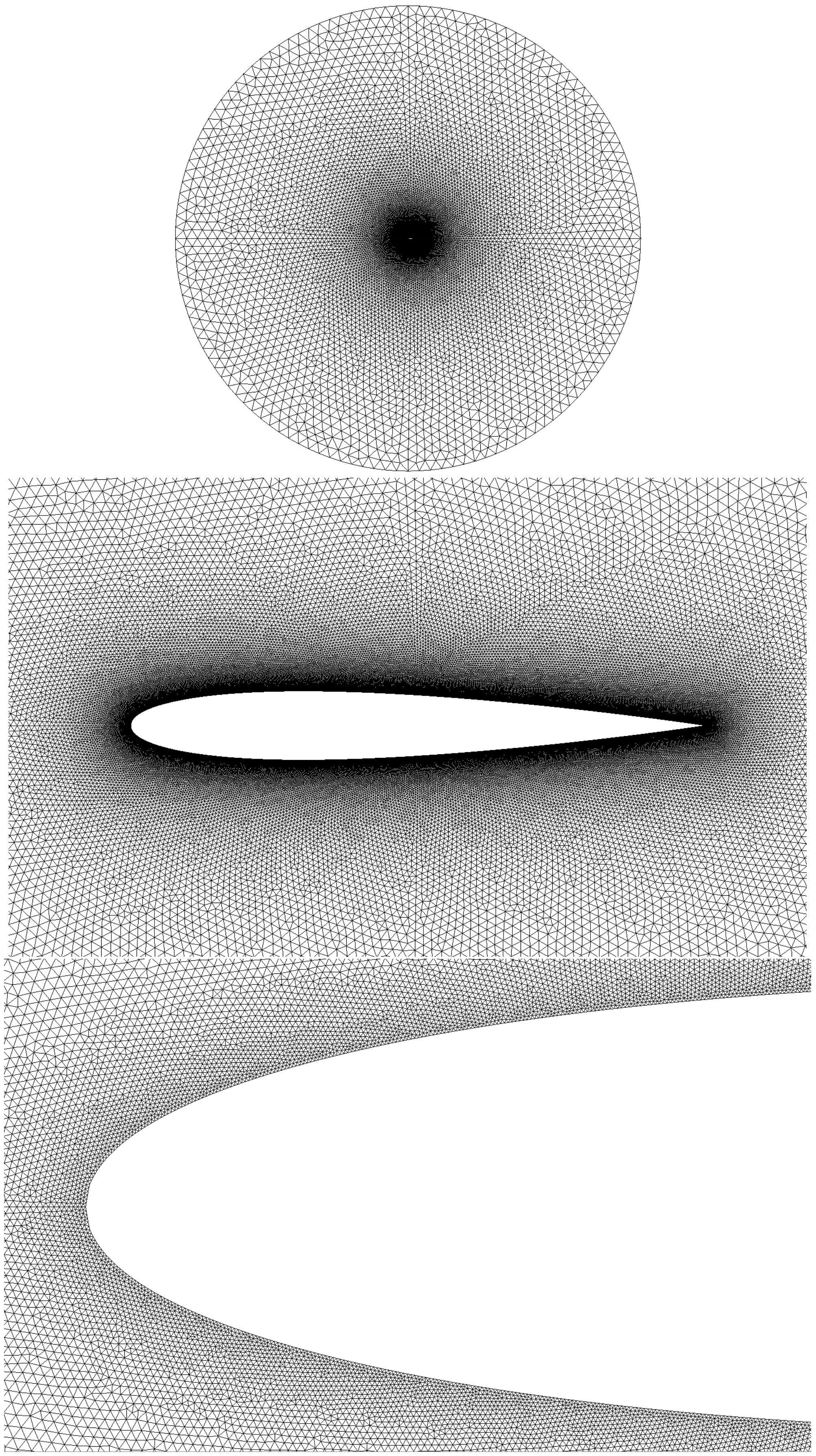


Figure 4.3: The unstructured mesh for steady transonic airfoil test case with different levels of view.

Table 4.2: Free-stream conditions for steady transonic airfoil problem.

Parameter	Value
Mach	0.8
Angle of attack	1.25°
Pressure	1.0 / γ
Density	1.0

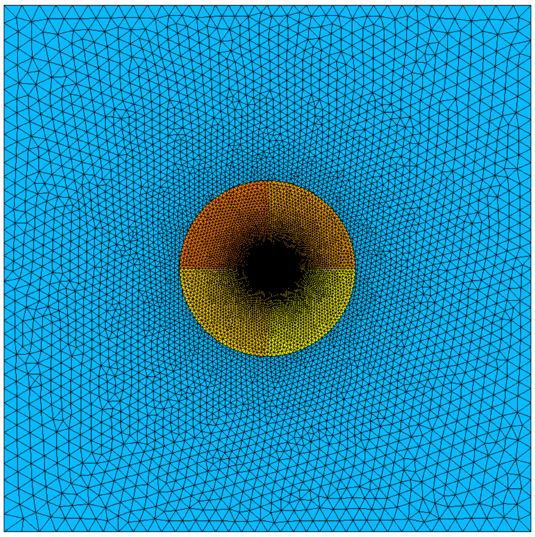


Figure 4.4: The background and airfoil mesh for steady transonic airfoil test case.

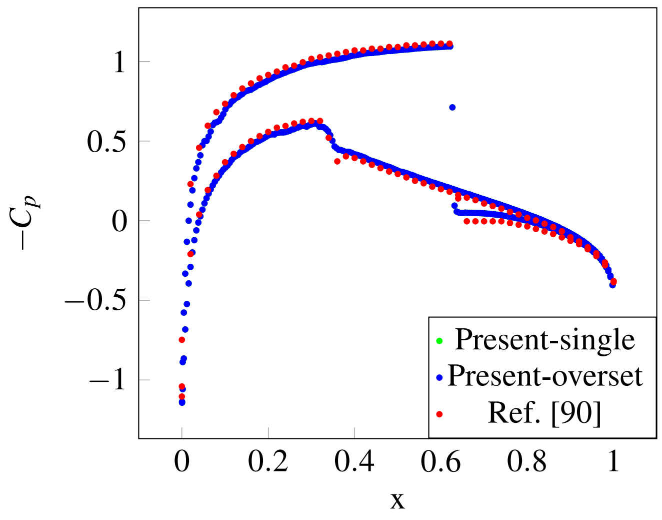


Figure 4.5: Pressure coefficient on the airfoil.

4.1.3 Pitching airfoil

In this test case, the unstructured mesh around NACA 0012 used in steady transonic flow test case undergoes a forced harmonic oscillation around 25% of the airfoil. Initially, the airfoil is rotated 0.016°. Angle of attack is determined with

$$\alpha = \alpha_{\text{mean}} + \alpha_{\text{amp}} \sin(\omega t) \tag{4.1}$$

where, angular frequency ω is found from definition of reduced frequency k such as

$$k = \frac{\omega c}{2u_\infty} \tag{4.2}$$

and c is the chord length and u_∞ is free-stream velocity which is equal to Mach number since free-stream pressure is normalized to make speed of sound unity. Table 4.3 shows parameters used in Equations (4.1) and (4.2).

In order to validate moving mesh formulation, the airfoil mesh is rotated rigidly in each time step with the amount of

$$\Delta\alpha = \dot{\alpha}\Delta t \tag{4.3}$$

where, $\dot{\alpha}$ is the angular velocity

Table 4.3: Harmonic oscillation parameters

Parameter	Symbol	Value
Mean pitch	α_{mean}	0.016°
Amplitude	α_{amp}	2.51°
Reduced frequency	k	0.0814
Chord	c	1
Mach	M	0.755

$$\dot{\alpha} = \alpha_{\text{amp}} \omega \cos(\omega t). \tag{4.4}$$

Tangential face velocity which is to be substracted from flow velocity at a position **r** is computed with

$$\mathbf{b} = \dot{\alpha} \times \mathbf{r}. \tag{4.5}$$

Similar to steady transonic flow case, slip-wall boundary condition is applied on the airfoil surface, however, face velocities are substracted from flow velocity before reversing the direction of interior normal velocity. In addition, pressure gradient is non-zero due to non-zero acceleration and is determined from normal momentum equation

$$\frac{dp}{dn} = -\rho \mathbf{n} \ddot{\mathbf{b}}. \tag{4.6}$$

where, **n** is the face normal vector and $\ddot{\mathbf{b}} = \ddot{\alpha} \times \mathbf{r} = -\alpha_{\text{amp}} \omega^2 \sin(\omega t) \times \mathbf{r}$ is the tangential acceleration.

In addition to the airfoil mesh shown in Figure 4.3, a background mesh with similar cell sizes is used. The background mesh is of rectangular shape (disregarding the depth) with 40-chord length sides in order to assure that pitching airfoil does not get out of bounds of the computational domain. Farfield boundary condition is applied on the surfaces of the background mesh. At the outer surface of the airfoil mesh, overset boundary condition is used to so that the airfoil mesh interpolate flow variables from the background mesh at mandatory receptor cell and also after mesh motion.

Unsteady flow is solved in each time step with implicit second order three-time level

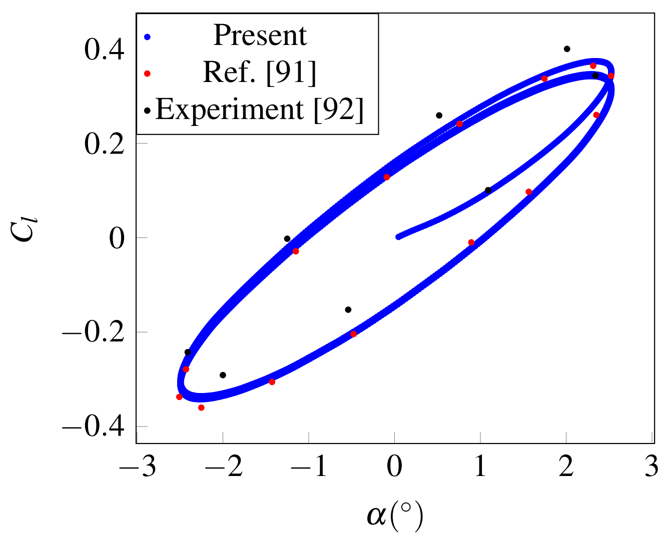


Figure 4.6: Lift coefficient hysteresis.

Euler discretization. In order to allow a relatively large time step $\Delta t = 0.1$, dual-time step formulation is used. Similar to steady transonic flow case, CFL number is taken as 10.

Lift coefficient hysteresis with changing airfoil angle of attack is shown in Figure 4.6. Present results are close to numerical results obtained by Reference [91]. The reason of discrepancy from experimental result [92] is due to inviscid numerical solution.

Figures 4.7, 4.8, 4.9 and 4.10 show pressure contours at different angle of attacks and airfoil pitching directions. At the slip wall, there is no pressure gradient and mass flux causing the flow to speed up at the cost of reduced pressure at the top plane.

Mach contours at the maximum angle of attack are shown in Figure 4.11. At the farfield, pressure is constant due to Riemann boundary condition. It is observed that Mach number is around $M = 0.755$ at the farfield which proves that the farfield boundary

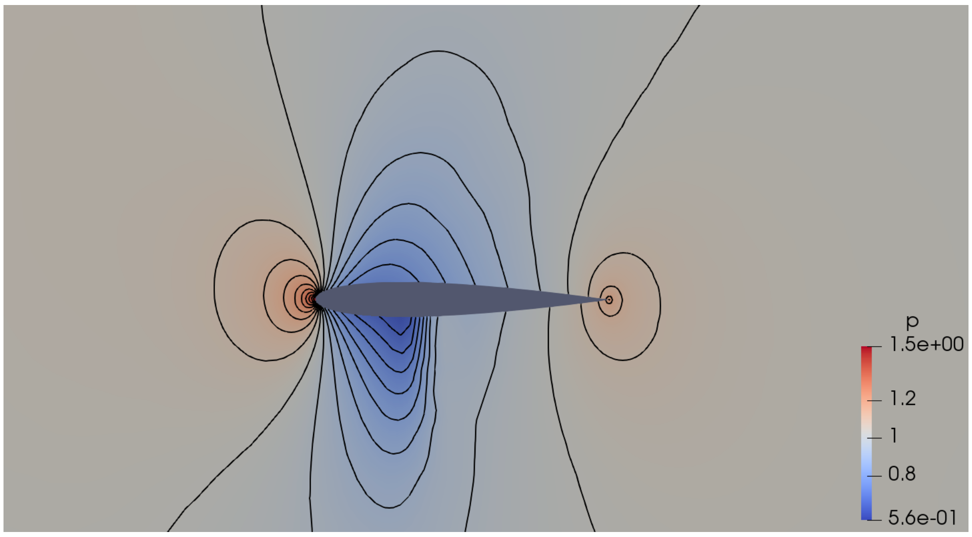


Figure 4.7: Pressure conntours around the NACA0012 airfoil at the mean angle of attack ($\alpha = 0.016°$) during pitching up.

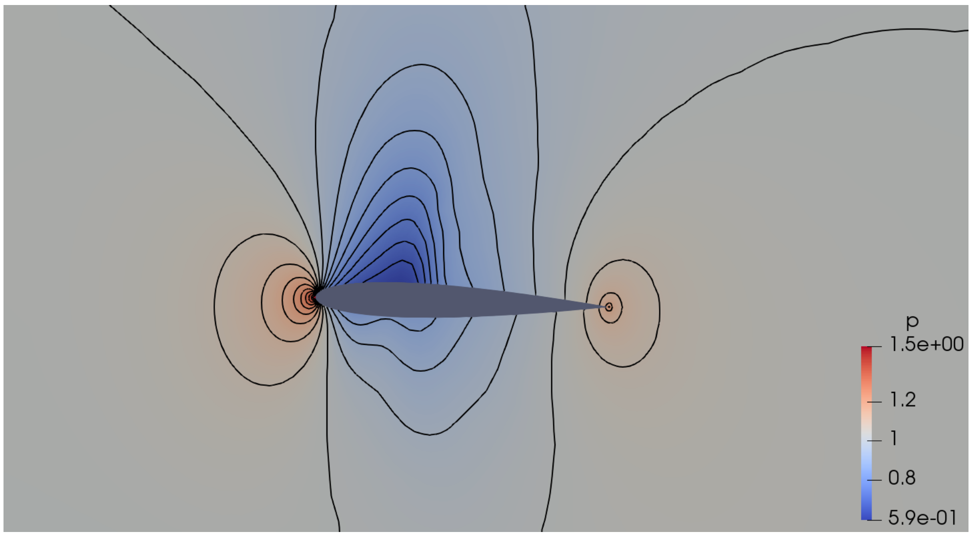


Figure 4.8: Pressure contours around the NACA0012 airfoil at the maximum angle of attack ($\alpha = 2.51°$).

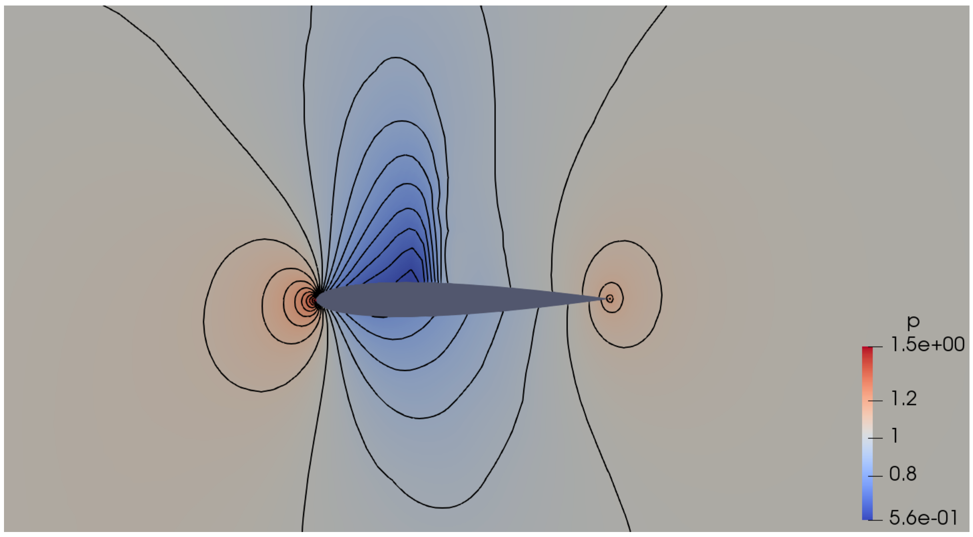


Figure 4.9: Pressure contours around the NACA0012 airfoil at the mean angle of attack ($\alpha = 0.016^{\circ}$) during pitching down.

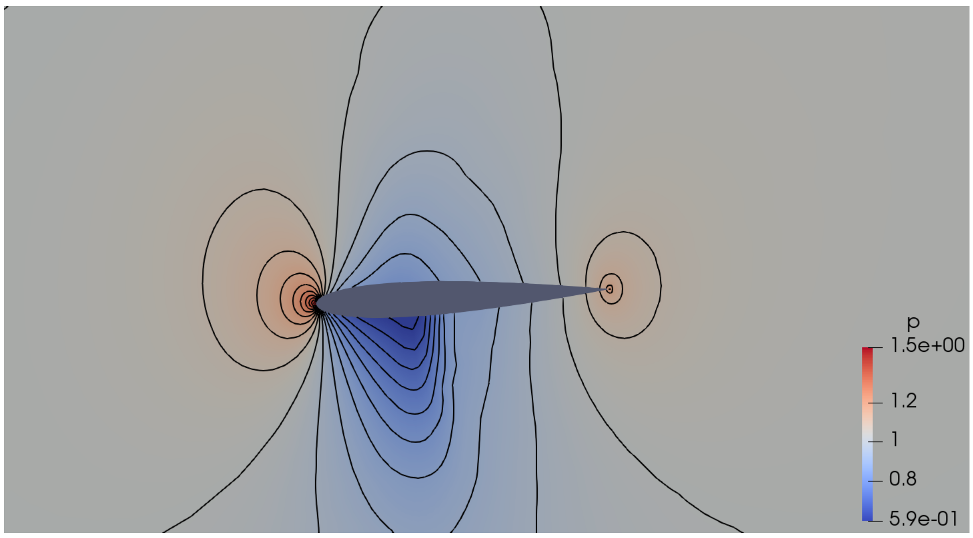


Figure 4.10: Pressure contours around the NACA0012 airfoil at the minimum angle of attack ($\alpha = -2.51^{\circ}$).

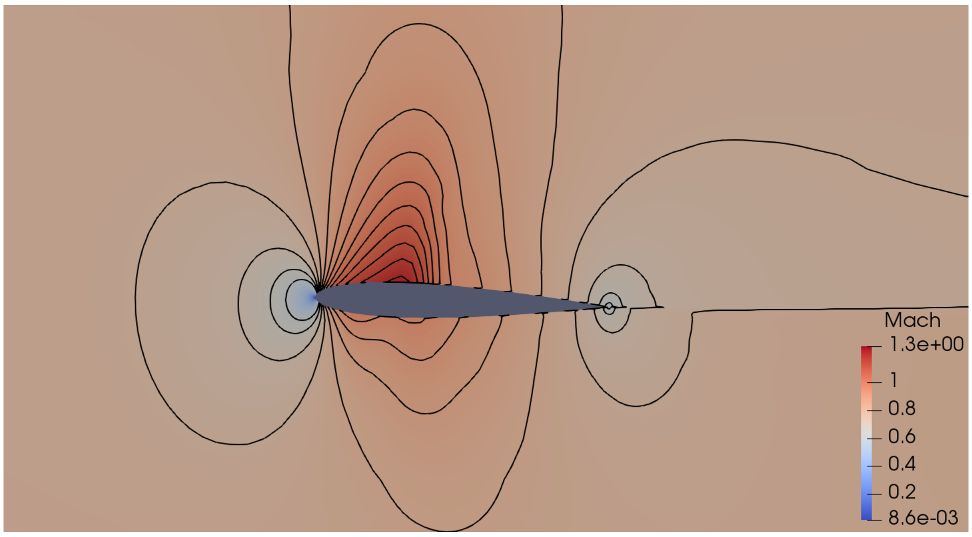


Figure 4.11: Mach contours around the NACA0012 airfoil at the maximum angle of attack ($\alpha = 2.51°$).

condition is appropriate. As mentioned, slip wall boundary condition prevents pressure and mass flux through the wing wall, resulting in higher Mach number at the top plane.

4.1.4 Steady transonic flow over ONERA M6 wing

This test case features three-dimensional inviscid transonic flow with Mach number $M = 0.8395$ over an ONERA M6 wing with angle of attack $\alpha = 3.06°$. Geometric details are explained in Reference [93]. Isometric and top views of the wing half span are shown in Figure 4.12. The reason of showing only half span is because the wing is symmetric at the root.

Figure 4.13 shows the mesh boundaries used in the simulation. A box with 20 span-length in stream-wise and wall-normal dimensions and 10 span-length in span-wise dimension is used to model outer boundaries. In order to reduce the number of cells, symmetry boundary condition is applied on the boundary of the box which coincides with the root of the wing. On the remainder of the box boundaries far-field boundary

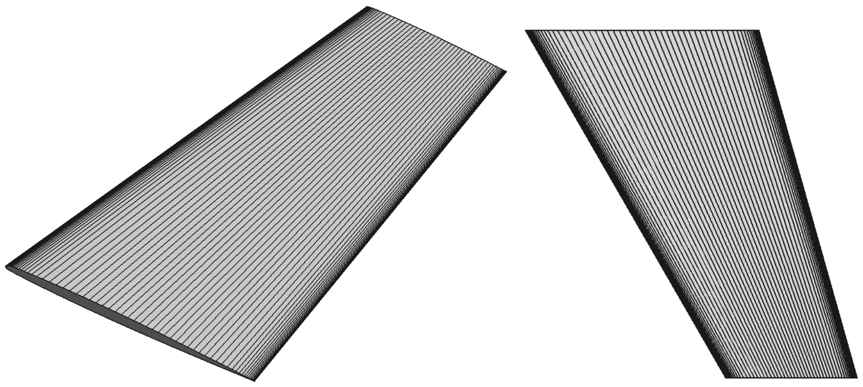


Figure 4.12: Isometric and top views of CAD model of the half-span ONERA M6 wing.

condition is applied. On the wing surface slip-wall boundary condition is applied. The problem is solved using both a single grid and two different overset grid systems. Farfield boundary condition on the outer boundary of the airfoil mesh is replaced with overset mesh boundary condition for overset mesh approach. On outer boundary of the background mesh farfield boundary condition is applied.

In the overset grid case two different grid configurations are used, differing in the size of the airfoil mesh while the background mesh is kept the same in both configurations. Figure 4.14 shows the two overset mesh configurations. Background mesh is of 20×20 span-length dimensions in both overset mesh configurations. However, in the first configuration shown in Figure 4.14a, the airfoil mesh dimensions are 10×10, while in the second configuration, which is shown on the right (Figure 4.14b), the airfoil dimensions are 5×1 span lengths.

Figure 4.15 shows the different views of the volumetric and surface mesh. The airfoil mesh contains approximately 1.74, 1.51 and 1.20 million tetrahedral elements in the single and overset mesh approaches, respectively. The unstructured background mesh

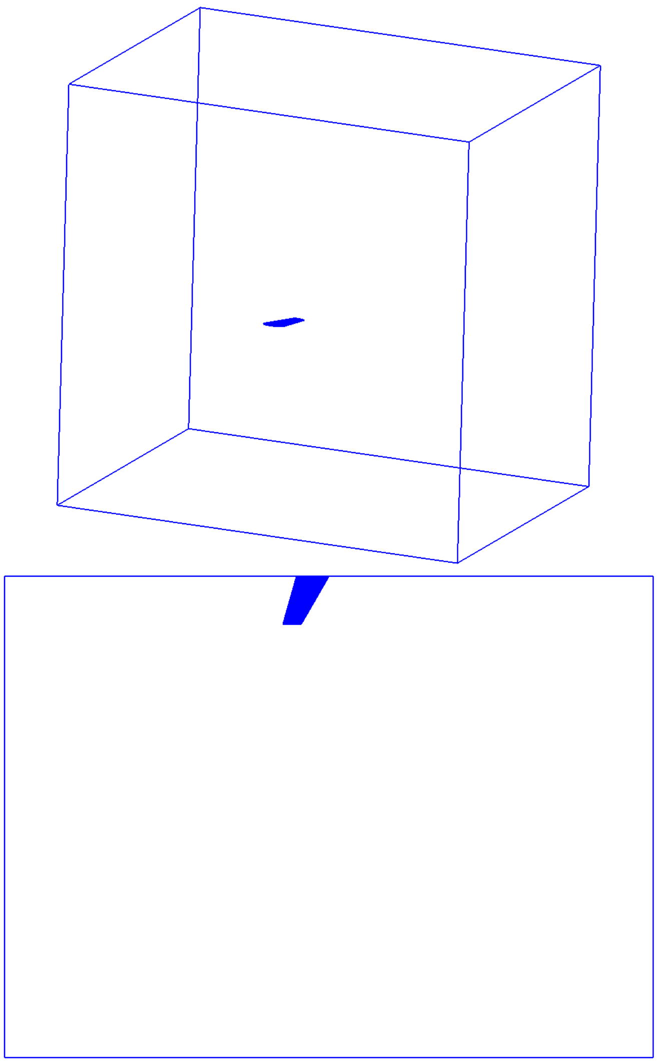


Figure 4.13: Isometric and top views of the mesh boundaries of the ONERA M6 wing.

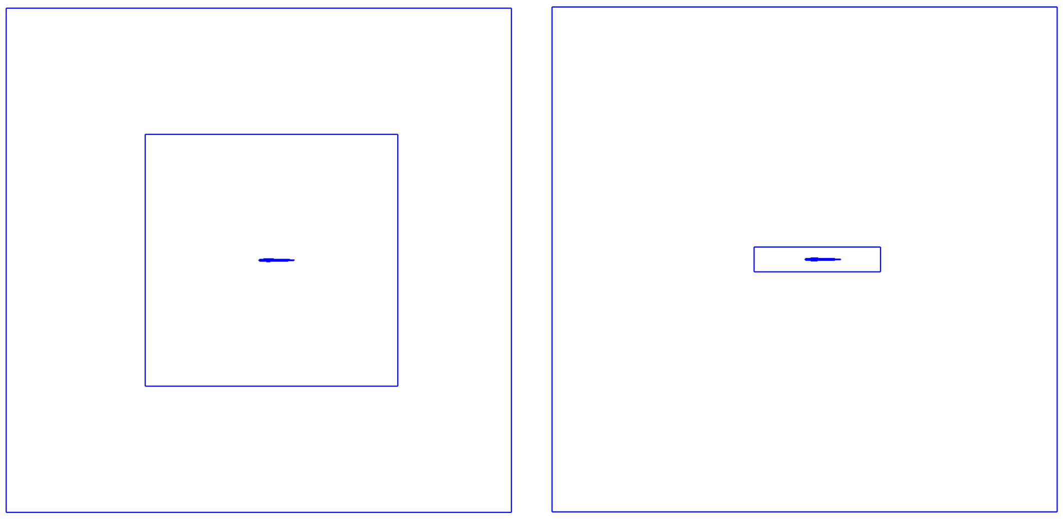

(a) (b)

Figure 4.14: Mesh-system with two different airfoil mesh sizes for ONERA M6 test case. The outer-most boundary belongs to the background mesh. (a) overset-far configuration where the airfoil mesh extends far away the solid surface, (b) overset-close configuration where the airfoil mesh boundary is close to the solid surface.

resolution is similar to that of the airfoil mesh and contains approximately 1.70 million tetrahedral elements.

The Euler equations are solved implicitly with backward Euler method. MUSCL scheme with Venkatakrishnan limiter is used to obtain second order spatial accuracy. CFL number of set to 10.

Table 4.4 shows the computed lift and drag coefficients on the ONERA M6 wing together with the coefficients from a viscous turbulent simulation [94]. In the table, Overset-far results correspond to the configuration where the dimensions of the airfoil outer boundary is half of that of the background mesh and is sufficiently away from the airfoil surface as shown in Figure 4.14a. Similarly, Overset-close results correspond to the configuration where the dimensions of the airfoil outer boundary is much smaller than

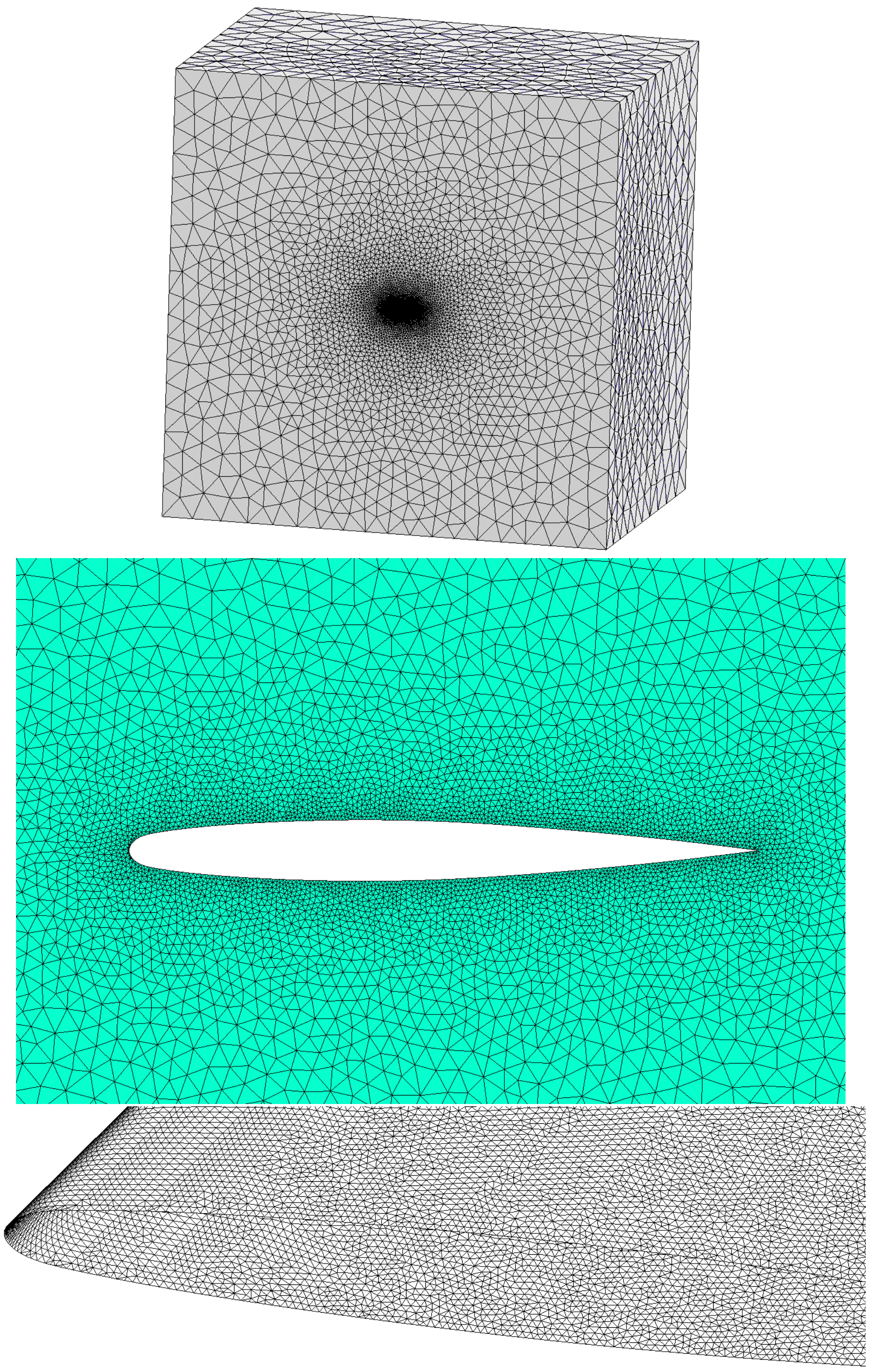


Figure 4.15: Different views of the ONERA M6 mesh.

Table 4.4: Aerodynamic force coefficients on the ONERA M6 wing.

Force coef.	Single	Overset-far	Overset-close	Ref. [94]
Lift	0.283	0.283	0.291	0.260
Drag	0.009	0.009	0.011	0.0175

that of the background mesh and is close to the airfoil surface as shown in Figure 4.14b. First, it is clear that the closer the interpolation zone to the solid surface, the higher the discrepancy from the single mesh solution due to interpolation between the airfoil and the backgroung mesh where the gradients are higher than the far away zone from the surface. This is why in overset mesh applications require more attention to mesh generation than single mesh applications. Second, the reason of discrepancy of lift coefficient between that of Reference [94] and the present work is the absence of turbulent boundary layer interacting with the shock to produce a recirculation zone at the upper surface as explained in Reference [94]. Inviscid drag coefficient is expected to be lower than the viscous counterpart since the only source of drag is the pressure drag.

Figure 4.16 shows pressure coefficients along the 44% of the wing span obtained by the present overset mesh configurations, numerical reference [95] which also solves the Euler equations and experimental reference [96]. As shown in the figure, there is a good agreement between the present, reference numerical and reference experimental results. The main difference between two numerical methods is the resolution of shock at $x/c = 0.6$. In Reference, [95] the jump in shock remains in the range of experimental values. On the other hand, with the present method the change in shock profile is more abrupt, therefore, resolves discontinuity better than Reference [95]. Also, pressure difference between the upper and lower surfaces of the wing is higher for the

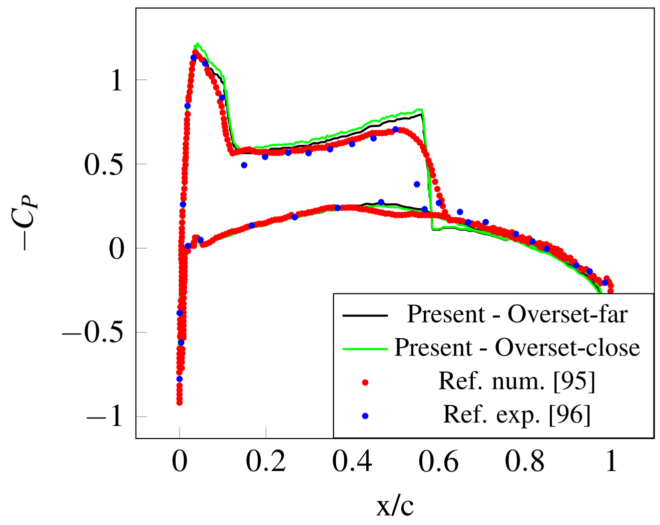


Figure 4.16: Pressure coefficients at 44% profile of ONERA M6 wing.

overset-close configuration than the overset-far configuration resulting in a higher lift.

Figure 4.17 shows pressure contours at 44% of the wing span obtained by the single mesh approach. Similar to pitching airfoil test case, at the slip wall, in the absence pressure gradient and mass flux, the flow at the top plane speeds up at the cost of reduced pressure whereas, at the farfield, pressure is constant due to Riemann boundary condition. Figure 4.18 also shows the pressure contours, however, for overset-close configuration. Compared to Figure 4.17, the transition of pressure contours from the overset to the background mesh is not smooth which is expected due to unconservative interpolation of flow variables across overset meshes.

Figure 4.19 shows Mach contours at 44% of the wing span obtained by the single mesh approach. Mach number is around $M = 0.8$ at the farfield as expected.

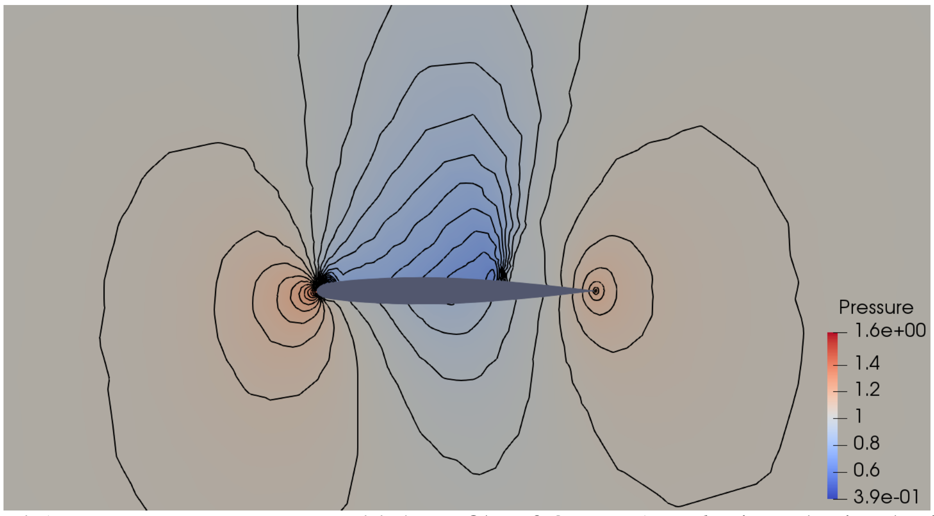


Figure 4.17: Pressure contours at 44% profile of ONERA M6 wing obtained with the single mesh approach.

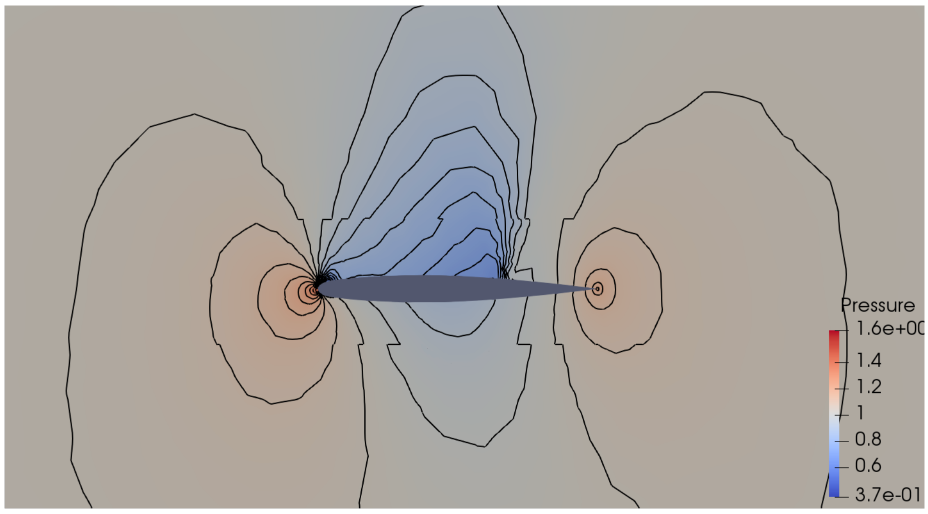


Figure 4.18: Pressure contours at 44% profile of ONERA M6 wing obtained with overset-close configuration.

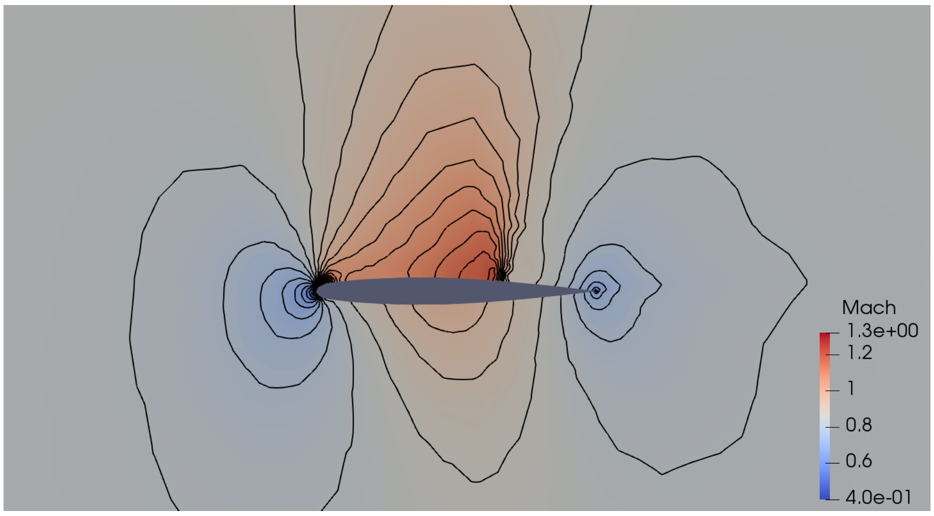


Figure 4.19: Mach contours at 44% profile of ONERA M6 wing obtained with the single mesh approach.

4.1.5 Rotor-fuselage interaction

A generic helicopter configuration based on Reference [97] consisting of a fuselage, pylon, hub and four NACA0012-profiled rotor blades is considered to evaluate a three-dimensional unsteady flow.

4.1.5.1 CAD models and meshes

In Reference [97], several flight modes are tested whereas in this simulation only the near hover mode is considered. Control parameters are shown in Table 4.5. Note that the collective pitch of $\theta_{c,75} = 9.4^\circ$ is measured at 75 percent of the blade radius. Built-in linear twist is $\theta_t = -8^\circ$. The lateral and longitudinal cyclic pitches, shown in Figure 4.20, are $A_1 = -0.1$ and $B_1 = 0.2$. Note that in Reference [97], coning angle is not mentioned, therefore, in this study, coning angle is assumed to be zero. Variation of cyclic pitch with azimuth angle, ϕ is shown in Figure 4.21.

Sketch of a blade and the spanwise collective pitch are shown in Figure 4.22. The total pitch of a blade is calculated with

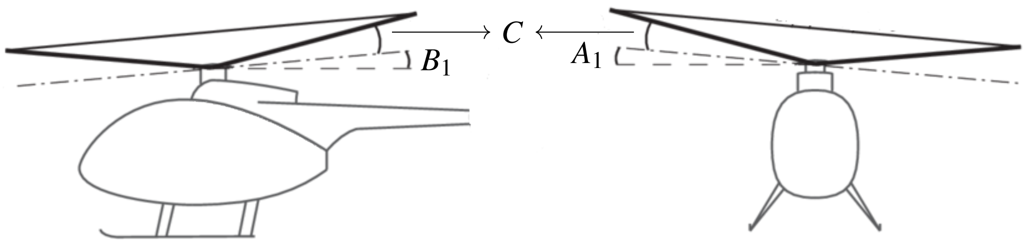


Figure 4.20: Lateral cyclic pitch, A_1, longitudinal cyclic pitch, B_1 and coning angle C which is assumed to be zero. Adapted from Reference [98].

Table 4.5: Control parameters.

Parameter	Value
Collective pitch, $\theta_{c,75}$	9.4°
Linear twist, θ_t	−8°
Lateral cyclic pitch, A_1	−0.1°
Longitudinal cyclic pitch, B_1	0.2°
Shaft angle, α_s	0°
Advance ratio, μ	0.01

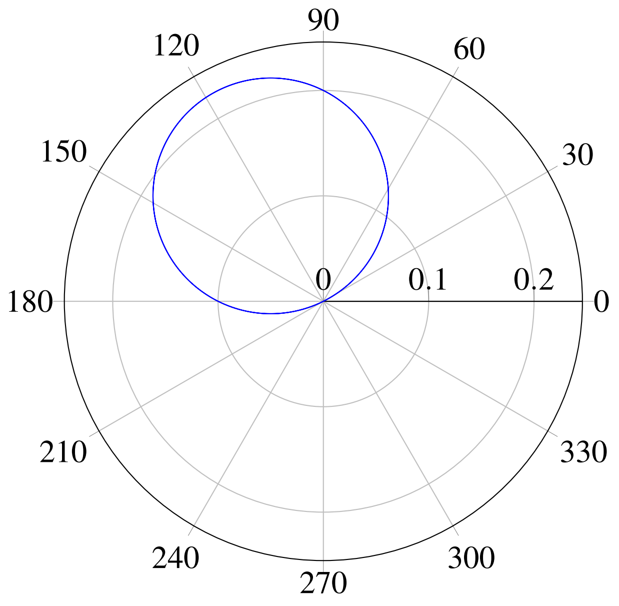


Figure 4.21: Variation of cyclic pitch with azimuth angle.

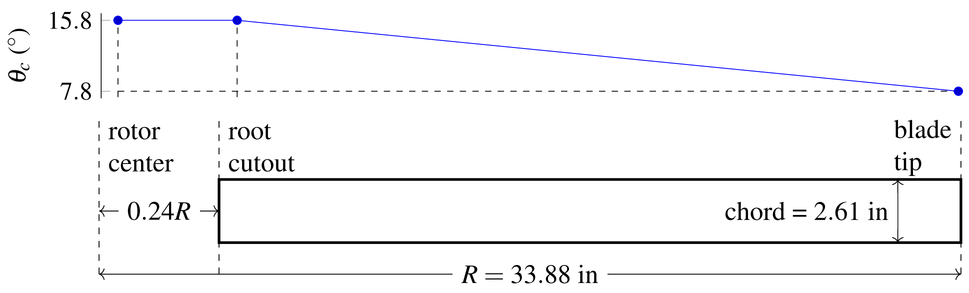


Figure 4.22: Sketch of a rotor blade in generic helicopter configuration and the spanwise collective pitch.

$$\theta = \theta_c - \frac{r}{R}\theta_t + A_1 \sin\phi + B_1 \cos\phi \tag{4.7}$$

where, r is the radial position of a blade and R is the rotor radius as shown in Figure 4.22.

Datum azimuth position, $\phi = 0$, is located at the tail of the fuselage. In Equation (4.7), θ_c is the collective pitch at the blade root and can be deduced by setting $A_1 = B_1 = 0$ or, in other words, keeping the swashplate perpendicular to the shaft axis.

The shapes of the fuselage and pylon are defined with super-ellipse equations:

$$\begin{bmatrix} H(x/l) \\ W(x/l) \\ Z_0(x/l) \\ N(x/l) \end{bmatrix} = C_6 + C_7 \left(C_1 + C_2 \left| \frac{x/l + C_3}{C_4} \right|^{C_5} \right)^{1/C_8}. \tag{4.8}$$

x/l is the non-dimensional longitudinal coordinate with limits [0, 1.997] for fuselage and [0.400, 1.018] for pylon and l is the desired length of fuselage which is 78.57 inches. Dimensions given in imperial units in Reference [97] were not converted to metric units to avoid ambiguity. Coefficients (C_1 to C_8) are shown in Table A.1 in appendices.

Cross-sectional coordinates y/l and z/l are obtained with

$$y/l = r \sin\phi$$

$$z/l = r \cos\phi + Z_0$$

where, ϕ ranges from 0 to 2π and r is defined as

$$r = \left[\frac{\left(\frac{HW}{4}\right)^N}{\left|\frac{H}{2}\sin\phi\right|^N + \left|\frac{W}{2}\cos\phi\right|^N} \right]^{1/N} . \tag{4.9}$$

Equation (4.8) and the coefficients provided in Reference [97] are modified since their original counterparts caused problems such as division by zero. Equations (4.8) and (4.9) are different from the counterparts by modulus operator around $\frac{x/l+C_3}{C_4}$, and $\frac{H}{2}\sin\phi$ and $\frac{W}{2}\cos\phi$. The modified coefficients are encircled in Table A.1.

Figure 4.23 shows CAD models of fuselage, pylon, four rotor blades, and hub modelled with FreeCAD [99]. Leading and trailing faces of fuselage are blunted slightly in order to have satisfactory mesh quality. Hub is modelled as cylinder with diameter 0.1 m and length 0.05 m and extensions which connect the hub to the blades have the same profile and angle of attack with the roots of respective blades. The extensions span from the hub such that they overlap 5% of respective blades. FreeCAD Python scripts to generate the fuselage and the rotor blades are provided in Figures B.1 and B.2 in Appendix B.

CAD models are transferred to the mesh generator, GMSH [100] to generate tetrahedral mesh for each component. Rotor blades have cylindrical meshes with diameter $10\times$ chord and length $20\times$ chord. Fuselage has a spherical mesh with diameter of 20 inches enclosing all blade and hub meshes. The hub also has a spherical mesh with diameter

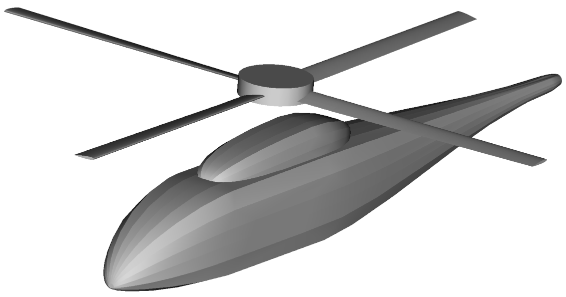
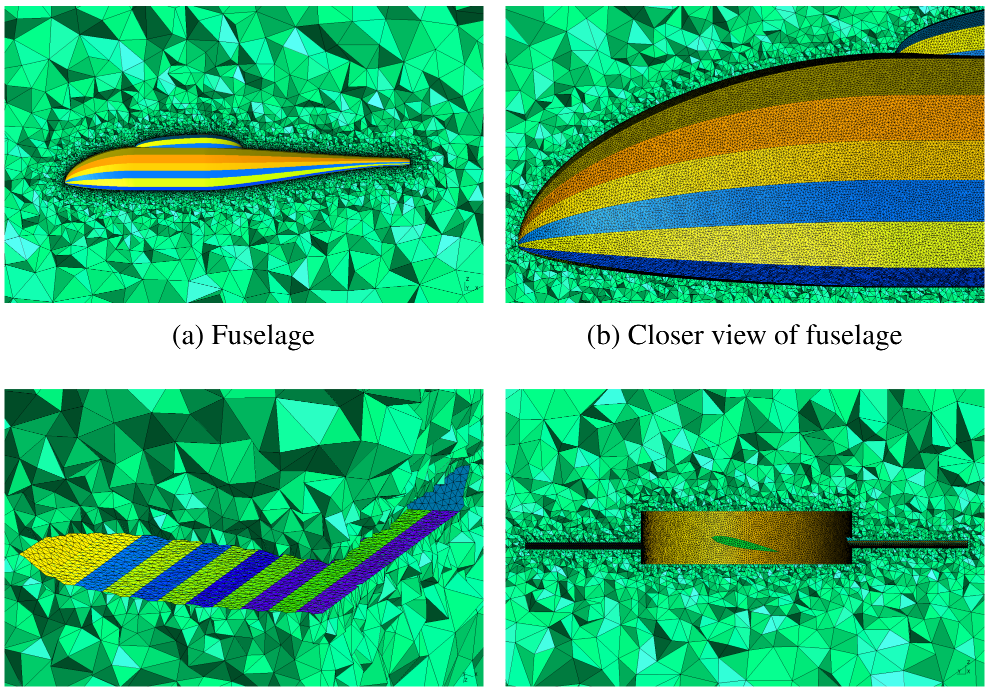


Figure 4.23: CAD model of generic helicopter configuration.


(a) Fuselage
(b) Closer view of fuselage
(c) Rotor blade
(d) Hub and extensions
Figure 4.24: Cross-sections of component meshes.

scaling with root cutout length such that $1.5 \times 0.24R$. Figure 4.24 shows cross-sections of the fuselage, one of the blades, and the hub. Element size s on fuselage, blade, and hub surfaces are 3 mm, 1 mm, and 2 mm. Volumetric element size is calculated with $s + d/5$ for all components where, $s = 0.01$ is the element size on any component wall surface and d is the shortest distance from respective wall surface. Table 4.6 shows the number of cells in each mesh.

Table 4.6: Number of cells in each mesh.

Fuselage & pylon	1,645,011
Blade 1	1,066,344
Blade 2	1,067,491
Blade 3	1,067,878
Blade 4	1,066,421
Hub	510,924
Total	5,356,578

Figure 4.25 shows a slice of field cells after donor search. Each color corresponds to a different mesh. In the figure, there are cells that are undesirably larger than the surrounding cells such the one enclosed in yellow circle. These cells were receptors having candidate donors which themselves were mandatory receptors or by definition, they were orphans. Since they have to be the donors of mandatory receptors, they are converted to field cells.

Flow variables interpolated via receptor cells are used by the neighbor field cells. A receptor which has only receptor neighbors can be excluded from the mesh-system as the flow variables interpolated via the receptor would not be used by the neighbor receptors. The procedure of excluding redundant receptors is called overlap minimization. Figure 4.26a shows receptors of a slice of fuselage mesh and Figure 4.26b shows the same mesh after exclusion of redundant receptors. Overlap minimization reduces the number of interpolations, hence, total simulation time.

The computational results are compared against experimental results of Reference [97] in terms of aerodynamic coefficients and unsteady and averaged-pressure coefficients.

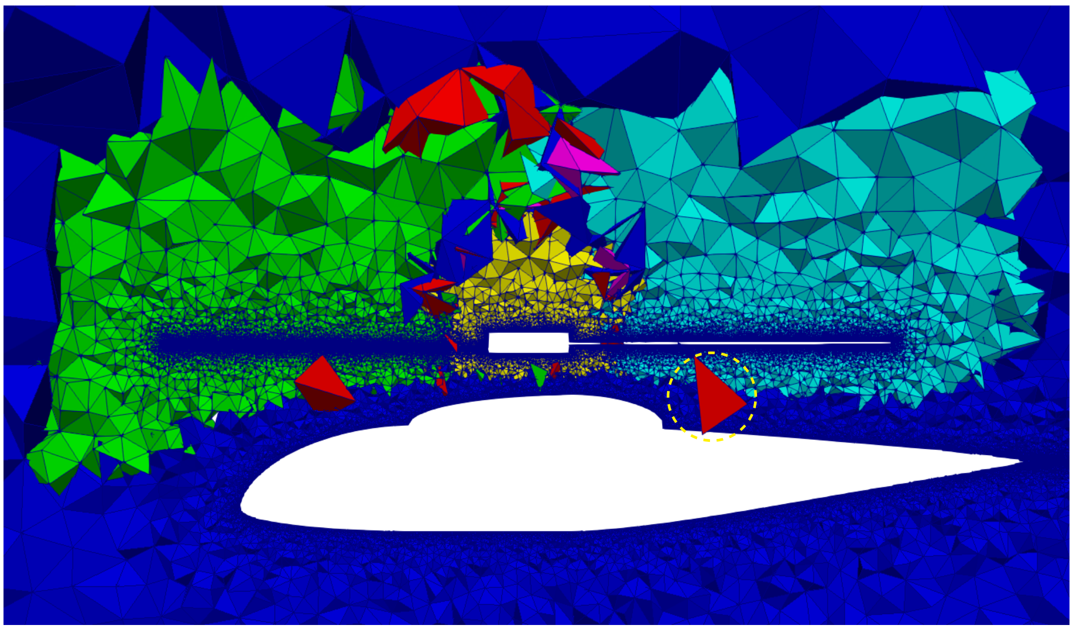


Figure 4.25: A slice of field cells after donor search. Each mesh is shown with a different color. The cell enclosed in yellow circle was an orphan converted to a field cell.

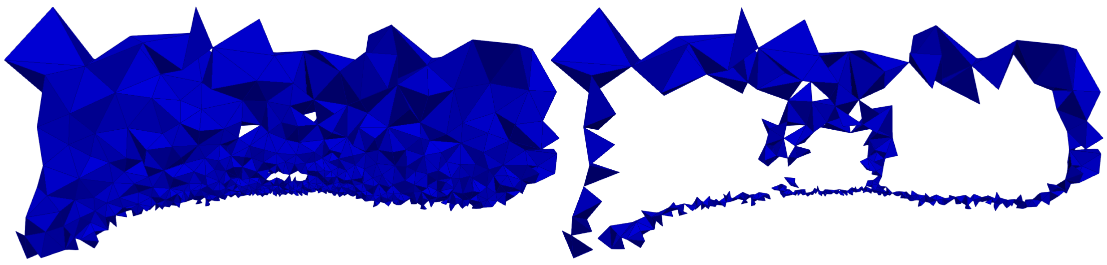

(a) Before (b) After

Figure 4.26: Slice of fuselage mesh before and after overlap minimization.

4.1.5.2 Solver parameters

Helicopter configuration is tested in near hover conditions with advance ratio, $\mu = 0.01$. Advance ratio is defined as

$$\mu = \frac{V_\infty}{V_t} \tag{4.10}$$

where, V_∞ is the freestream velocity and V_t is the blade tip velocity calculated with $V_t = \omega \times r$. The angular speed is calculated as $\omega = 2\pi(2000\,\mathrm{rev/min})/(60\,\mathrm{s/min}) \approx 209\,\mathrm{m/s}$. Blade span length is $r = 33.88\,\mathrm{inch} \times 0.0254\,\mathrm{m/inch} \approx 0.86\,\mathrm{m}$. As a result, the blade tip velocity is $V_t \approx 180\,\mathrm{m/s}$. From Equation (4.10), freestream velocity is $V_\infty = 0.01 \times 180\,\mathrm{m/s} = 1.8\,\mathrm{m/s}$.

For an explicit scheme, physical time step has to be

$$\Delta t = \mathrm{CFL}\frac{L}{V_t} \tag{4.11}$$

where, L is the characteristic length and is calculated as cubic root of the volume of cell at the tip of a blade such as $L = \sqrt[3]{10^{-12}} = 10^{-4}$. Characterictic length is particularly considered at the tip of a blade in order to compute the minimum physical time step. CFL for the explicit formulation has to be $\mathrm{CFL} < 1$ in real time, hence, it is taken as $\mathrm{CFL} = 0.9$. Therefore, from Equation (4.11)

$$\Delta t = \mathrm{CFL}\frac{L}{V_t} = 0.9\frac{10^{-4}\,\mathrm{m}}{180\,\mathrm{m/s}} \approx 10^{-7}\,\mathrm{s} \tag{4.12}$$

Pseudo steady state is reached in 10 pseudo time steps by iteratively computing flux $\mathbf{F}$ and updating U^{s+1}. Once pseudo steady is reached, conservative variables at new time level is updated as $U^{n+1} = U^{s+1} = U^s$.

In hover conditions, aerodynamic coefficients would yield high values since free-stream variables such as velocity, and in turn dynamic pressure, is much lower than forward flight. Therefore, pressure coefficient is calculated with modified dynamic pressure using blade tip speed instead of free-stream velocity. Modified dynamic pressure is

$$p_d = \rho_\infty V_t^2 \tag{4.13}$$

and modified pressure coefficient is

$$C_p = \frac{p - p_\infty}{p_d} \tag{4.14}$$

In order to avoid confusion, sign conventions for forces and moments are the same as in Reference [97] as shown in Figure 4.27. In the figure, C_Y, C_A and C_N are lateral, axial and normal force coefficients acting on the fuselage. C_n, C_m and C_l are yaw, pitch and roll coefficients acting on the fuselage.

Rotor thrust, T, is calculated by summing up the axial (x), lateral (y) and normal (z) force components acting on surfaces of the rotor blades such that

$$\mathbf{T} = \sum_{\text{face}} \mathbf{F}_x + \mathbf{F}_y + \mathbf{F}_z \tag{4.15}$$

Similar to pressure coefficient, rotor thrust coefficient is also calculated with modified thrust coefficient:

$$C_T = \frac{|\mathbf{T}|}{p_d A} \tag{4.16}$$

where, $A = \pi R^2$ is the rotor disc area. Other force coefficients on the fuselage are calculated with the same dynamic pressure and rotor disc area as shown in

Equation (4.17).

$$C_N = \frac{F_z}{p_d A}, \quad C_A = \frac{F_x}{p_d A}, \quad C_Y = \frac{F_y}{p_d A} \tag{4.17}$$

Moment coefficients are additionaly divided by rotor radius, R as shown in Equation (4.18).

$$C_m = \frac{M_y}{p_d A R}, \quad C_n = \frac{M_z}{p_d A R}, \quad C_l = \frac{M_x}{p_d A R} \tag{4.18}$$

In Equation (4.18), M_x, M_y and M_z are moments around x-, y- and z- axis, respectively, acting on the fuselage about the center point $(0.696L_f, 0, 0.322L_f)$, where, $L_f = 78.57$ inch is the longitudinal length of the fuselage.

As in Reference [101], aerodynamic coefficients are corrected with the ratio of computational to experimental thrust coefficient such as

$$C_i = \frac{C_T}{C_{T,exp}} C_i \tag{4.19}$$

where, C_i is a force or moment coefficient.

Of various components in the experimental configuration, it is common to model only the fuselage and the blades and exclude the hub and the fuselage support [101–104]. However, exclusion of the hub and the support is known to reduce solution accuracy at the upper and the lower regions of the fuselage [104]. In the present simulation, the hub is modelled along with the fuselage and the blades, however, the support is omitted.

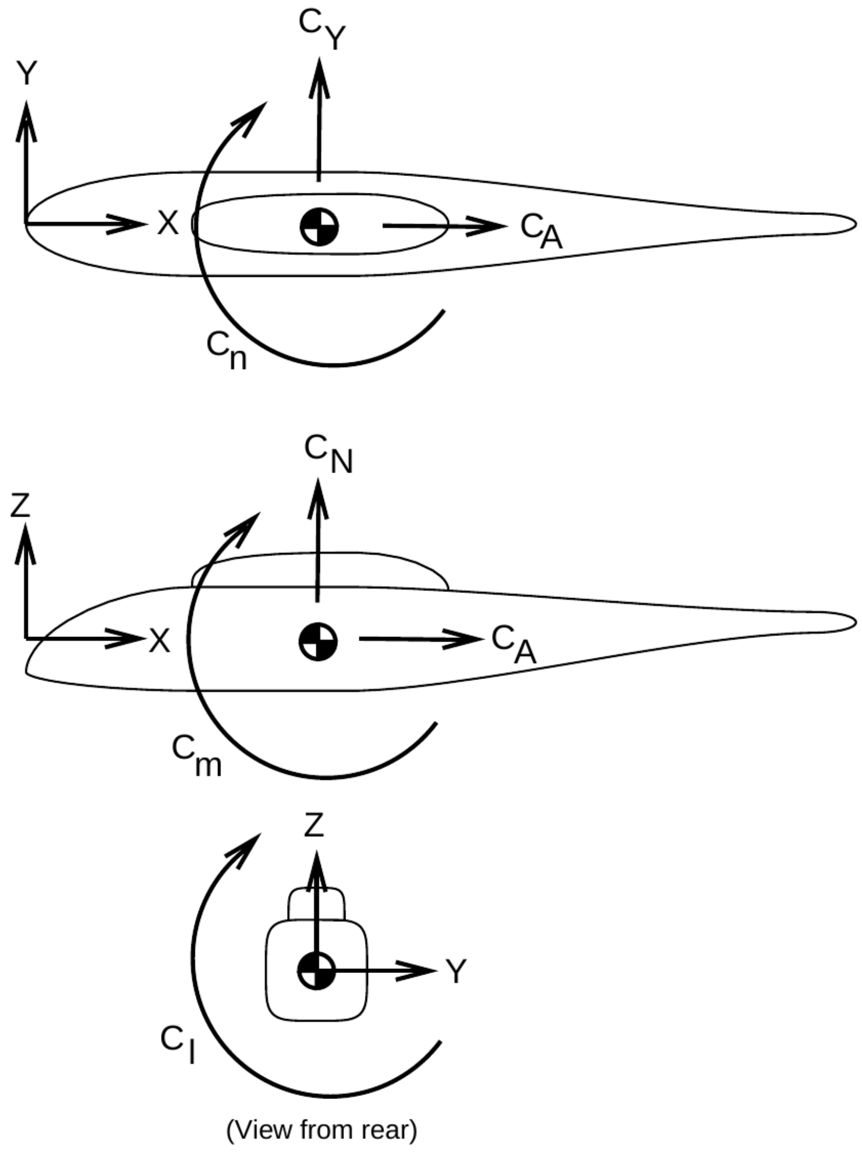


Figure 4.27: Sign conventions for forces and moments. The figure is taken from Reference [97].

4.1.5.3 Boundary conditions

At the outer surface of the fuselage mesh farfield boundary condition with a free-stream velocity of $V_\infty = 1.8m/s$ is applied. At the surfaces of all the components, slip-wall boundary condition is used. Face velocities are added to the flow velocities in order assure no mass flows through wall surfaces. Finally, at the outer surfaces of the blades and the hub, overset mesh boundary condition is used.

4.1.5.4 Validation of aerodynamic coefficients

Periodic steady state is achieved in nearly 22 rotor revolutions as shown in Figure 4.28. Simulation is run until 40 rotor revolutions in order to avoid any transitional effects. Average rotor thrust coefficient is approximately 0.00602 which is higher than the experimental thrust coefficient 0.004018 (multiplied by the rotor solidity, $\sigma = 0.098$). The difference in thrust coefficients is 33% which is not unordinary for this problem. Reference [105] which solves the same problem with overset mesh technique and Euler equations but with advance ratio $\mu = 0.15$, reports 41% difference in thrust coefficients. In Reference [97], convergence history of thrust coefficient is not provided, therefore, the experimental thrust coefficient is shown as a straight line. It is possible to adjust blade pitch angles until the experimental and the computational thrust coefficients match however, in this case, the pitch angles are kept the same as in Reference [97].

Table 4.7 lists time-averaged computational aerodynamic coefficients obtained from the present work, the numerical Reference [105] and the experimental Reference [97]. The computed download, that is negative normal force coefficient, C_N and longitudinal or axial force coefficient, C_A are lower than their experimental counterpart. Since viscosity is ignored in the Euler equations, lower computational download and axial forces are

Table 4.7: Time-averaged computational and experimental aerodynamic coefficients.

Coefficient	Computational	Experimental [97]
Vertical force, C_N	-1.3×10^{-5}	-1.5084×10^{-4}
Longitudinal force, C_A	-3.5×10^{-8}	-1.2363×10^{-5}
Lateral force, C_Y	1.2×10^{-5}	3.9706×10^{-4}
Pitch moment, C_m	1.1×10^{-7}	1.2228×10^{-5}
Yaw moment, C_n	-3.1×10^{-6}	-4.7272×10^{-5}
Roll moment, C_l	0.2×10^{-7}	4.4642×10^{-7}

expected. As shown in Figure 4.21, due to the cyclic pitch, the rotor is inclined towards the advancing side, therefore, causing fuselage to drift towards 120° direction. As a result, axial and lateral (C_Y) forces are in negative and positive directions, respectively. Since the flight mode is near hover, the flow is dominated by rotor speed and freestream wind is insufficient to result in net positive axial force.

Similar to force coefficients, moment coefficients are lower than experimental counterparts, especially the pitch moment, C_m which is due to the higher pressure on the nose and the lower pressure on the tail boom, computational pitching moment is much lower than the experimental pitching moment. Yaw moment, C_n, as expected, is in the direction of rotor rotation. In real case, when the rotor and the fuselage are physically attached, fuselage would rotate in the reverse direction of the rotor. In that case, tail rotor would be added to balance the yaw moment of the fuselage. There is a slight positive roll due to drifting in 120° direction.

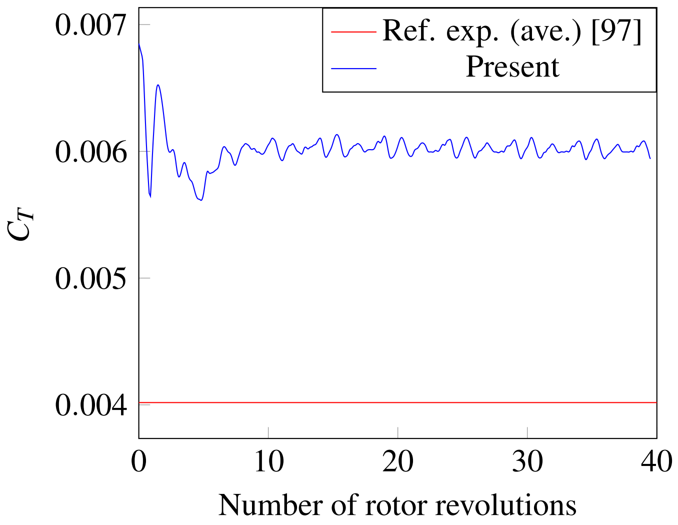


Figure 4.28: Convergence of thrust coefficient with the number of rotor revolutions.

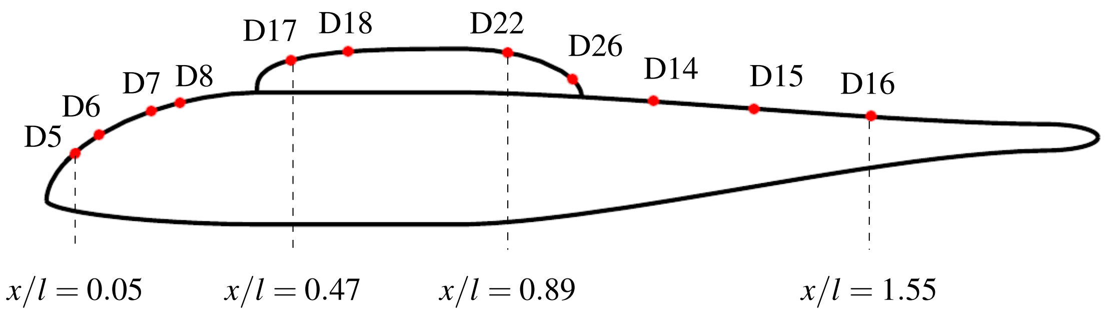


Figure 4.29: The control points at top centerline of the fuselage.

4.1.5.5 Unsteady pressure coefficients

Once periodic steady state is reached, unsteady modified pressure coefficients are evaluated in a time step and on different control points on upper centerline of the fuselage (Figure 4.29) and around the fuselage (Figure 4.30). Exact locations of the control points are shown in Table 4.8.

Figure 4.31 shows modified unsteady pressure coefficients at some of the control points locations. Similar to Reference [104], the experimental data is shifted in phase by 252° in order to accomodate for measurement lag of experimental equipment. Four pulses

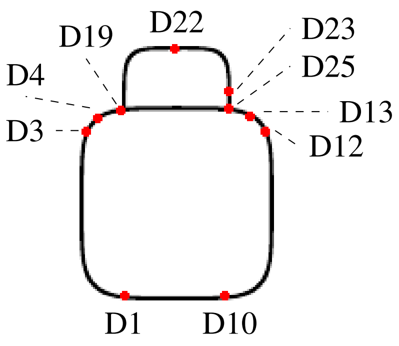


Figure 4.30: The control points around the fuselage at $x/l = 0.89$.

Table 4.8: Coordinates of control points on the fuselage.

Top centerline				Circumferential			
Point	x/l	y/l	z/l	Point	x/l	y/l	z/l
D5	0.052	0.007	0.004	D1	0.897	-0.091	-0.117
D6	0.096	0.006	0.037	D3	0.895	-0.117	0.080
D8	0.201	0.007	0.090	D4	0.895	-0.096	0.106
D9	0.256	0.007	0.110	D19	0.895	-0.067	0.125
D17	0.467	0.007	0.185	D22	0.895	0.007	0.200
D18	0.600	0.007	0.202	D23	0.895	0.067	0.150
D22	0.896	0.007	0.200	D25	0.895	0.067	0.125
D26	1.001	0.007	0.150	D13	0.895	0.094	0.109
D14	1.180	0.007	0.100	D12	0.897	0.116	0.086
D15	1.368	0.007	0.087	D10	0.897	0.094	-0.115
D16	1.556	0.007	0.073				

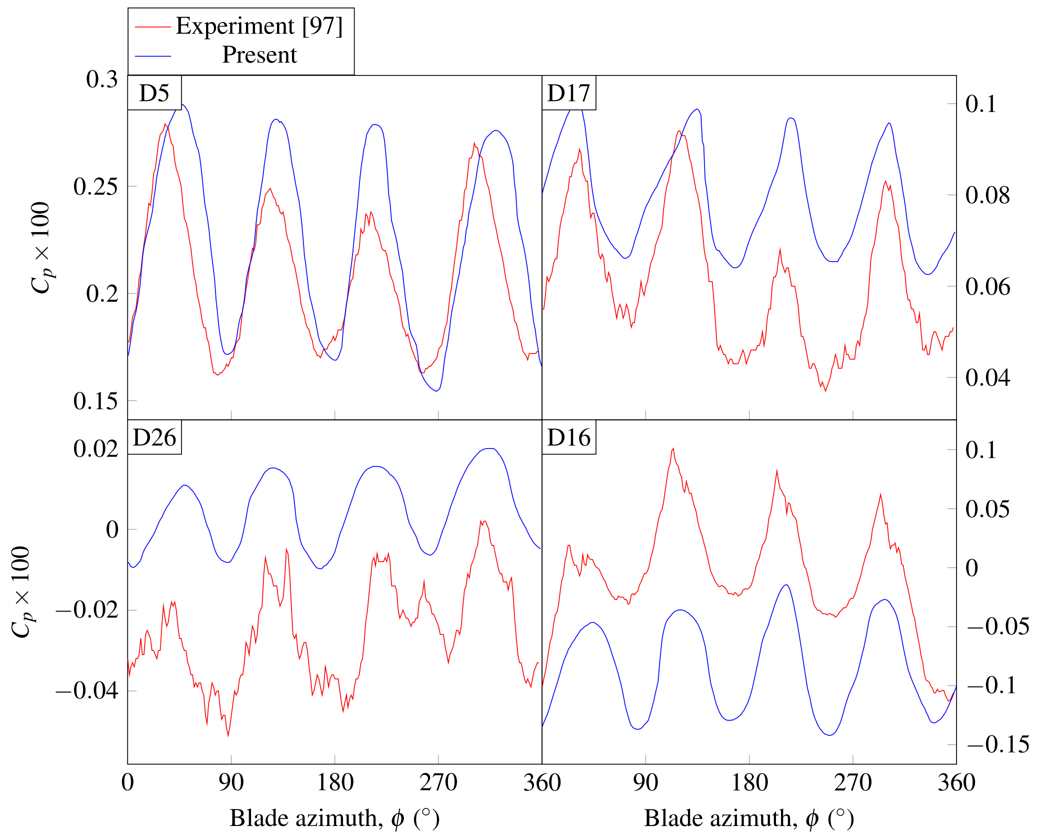


Figure 4.31: Unsteady modified pressure coefficients at different control points on top centerline of fuselage.

in the figures correspond to four blades each passing over the control points. In most locations, the experimental pulses are clearly distinguisable however, the experimental pulses are more oscillatory at D17 and D26 as they are located underneath the hub where blades exert approximately equal amount of pressure on the fuselage.

Pressures in all but tail control points are over-predicted by the simulation. The results at positions close to the nose of the fuselage such as D5 match better compared to other control points.

The present and experimental results over the pylon such as D17 and D26 are not in good agreement due to complex intergrid region at the intersection of meshes of the fuselage, four blades and the hub. The number of interpolations in this overlapping region is much higher than the rest of the field, causing the solution accuracy to lower significantly. Another source of discrepancy is due to the fact that interpolation between meshes does not conserve intercell fluxes.

It is known from Reference [106] which simulates a viscous flow around the generic helicopter configuration, that the flow separates at the end of the pylon or from D26 to D14 points. Since the flow seperation is not captured in the simulation, the pressure at D26 is higher than the experimental value.

The D16 point is located at the tail boom and is in the downwash of rotor where tip vortices, in the experimental case, pass over. As a result, pressure is under-predicted at tail points in the absence of tip vortices from the blades.

Figure 4.32 shows unsteady modified pressure coefficients on some of the control points around the fuselage at $x/l = 0.89$. Locations of control points are shown in Figure 4.30. Asymmetry in experimental results on advancing (e.g. D1 and D3) and retreating sides of (e.g. D10 and D12) is obvious from Figure 4.32. It is expected that the asymmetry is because of the blade root attachments that are located at advancing side of the rotor. At lower points of the fuselage (D1 and D10) due to absence of fuselage support, computational results show some discrepancy relative to experimental ones.

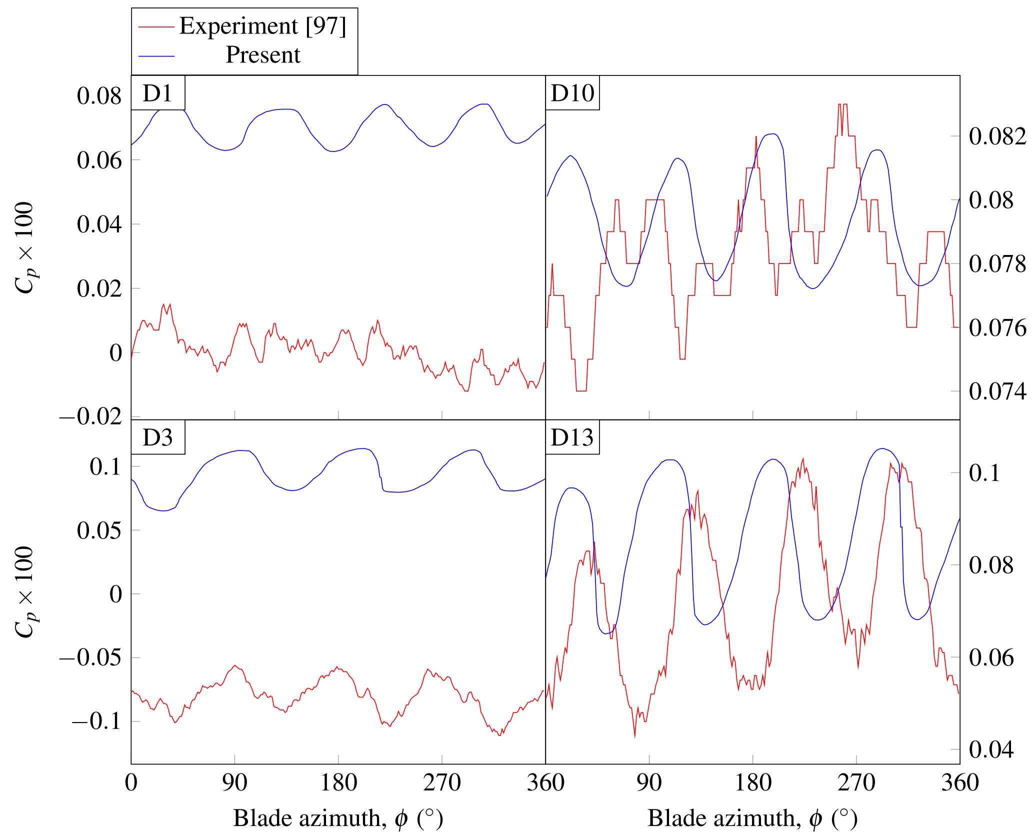


Figure 4.32: Unsteady modified pressure coefficients at different stations around the fuselage at $x/l = 0.89$.

4.1.5.6 Time-averaged pressure coefficients

In Reference [97], coefficients are averaged for 30 revolutions. In this study, since data files consume considerable memory, some of the intermediate data files are deleted during simulation to save memory. As a result, pressure coefficients are averaged for the last revolution.

Figure 4.33 shows averaged pressure coefficients calculated in the last revolution. From the figure, it is shown that pressure is over-estimated from the nose to the end of the pylon and under-predicted there on over the tail boom. In addition, Table 4.9 shows relative errors of the time-averaged pressure coefficients. It is clear that over the pylon (D17, D18, D22 and D26) the discrepancy is increasing compared to the nose due to interpolations between overset meshes. At D26, particularly, due to inability to capture of flow seperation, the discrepancy is even higher than the rest of the pylon control points. Towards the tail boom the discrepancy increases due to insufficient solution of vortex detachments from the blades that hit the tail boom and increase pressure coefficient. At critical regions such as over the pylon and tail boom, mesh resolution should be higher and overset mesh region should be as far as possible from the component surfaces in order to capture physical features.

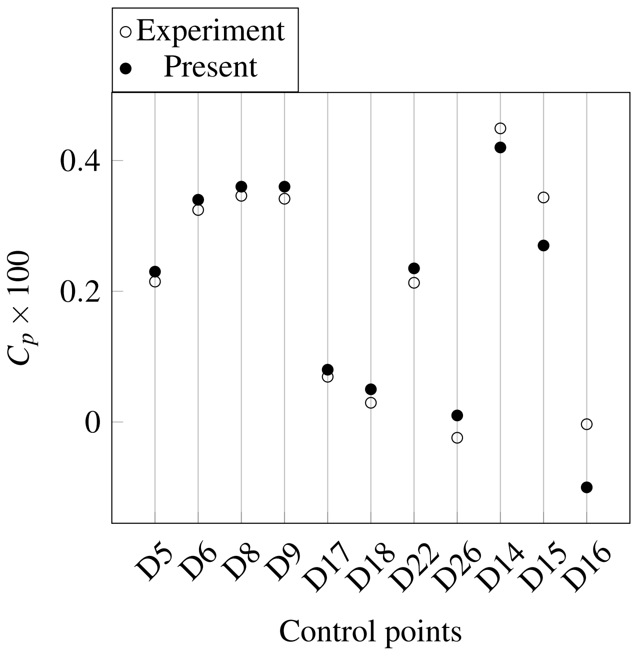


Figure 4.33: Averaged modified pressure coefficients at different control points on top centerline of the fuselage.

Table 4.9: Relative error of time-averaged pressure coefficients.

Control point	Relative Error
D5	-0.00015
D6	-0.00016
D8	-0.00014
D9	-0.00018
D17	-0.00011
D18	-0.00021
D22	-0.00022
D26	-0.00034
D14	0.00029
D15	0.00073
D16	0.00096

4.2 Parallel performance results

Test cases are run on TRUBA's [107] clusters. In order to reduce queue time for initialization of the simulation, computations are run on different set/family of nodes having different specifications, as shown in Table 4.10. When a certain number of cores is requested, the resource manager provides a mixture of cores that belong to nodes of Barbun, Sardalya and Levrek groups. However, in this heteregenous computing environment where nodes can have different specifications, load balance results are expected to vary depending on the variety of node specifications. Note that, in High Performace Computing jargon, a node may refer to a server to execute a specific task such as computing, user login and data transfer. In this context, a node specifically refers to a computing server which contains a motherboard with multiple cores, several layers of memory storage and network sockets. For example, a node in Barbun group, in Table 4.10, contains 20 cores and 384 GB of RAM. In this work, hyperthreading is not used and virtual processors are mapped to cores, therefore, processor is an alias for core for the rest of the paper. Since a node can have a maximum 20 cores, for a job requesting more than 20 processors, cores cannot fit into a single node, but, they will belong to different nodes. Therefore, data transfer takes place between cores having both shared memory and distributed memories. Open MPI's Byte Transfer Layer (BTL) framework uses the best component such as shared memory, TCP, Infiniband and so on for data transfer between cores. Again, in order to reduce queue time, the number of nodes is left to be determined by resource manager. The results in this section are obtained by using maximum allowable number of processors per user (128) usually distributed over 7-10 nodes belonging to different set of nodes.

Table 4.10: Hardware specifications.

Group	Processor Model	Core per node	RAM (GB)
Barbun	Intel Xeon Scalable 6148	20	384
Sardalya	Intel Xeon E5-2690 V4	14	256
Levrek	Intel Xeon i5-2690	16	256

The most intensive memory usage is observed during data exchange between processors. Exchange of spatial partitions during load (re)balance and exchange of mesh cells after mesh motion are two major tasks involving intensive MPI communication. Although it is verified by Valgrind [108] that no memory leak occurs, memory usage increases after the mentioned tasks. It is expected that the reason for increasing memory usage is memory fragmentation caused by frequent allocation/deallocation of buffers leaving fragmented chunks of memory space. An obvious solution is using fixed buffers for data exchange. However, after load rebalance partitioning layout changes, therefore, the buffers used for data exchange have to be deallocated. Once memory usage hits the memory limit (8 GB per core), the simulation is halted by resource manager. In order not to lose progress, crucial data structures of assembler and solver are serialized with Boost's serialization library (which is also used by Boost MPI for data transfer), and written to binary archives for every quarter (90°) rotation. Processors read deserialized data from respective archives and restart simulation with reset memory space.

The most time consuming tasks per time step are shown in Figure 4.34. These tasks remain to be the most time consuming in all time steps. However, note that `Rebalance` may not be needed for all time steps. The execution time of `Solve` is based on the maximum number of pseudo time steps (10 in this study) to solve Euler equations. Since the assembler constitutes less time compared to the solver, it is unnecessary to

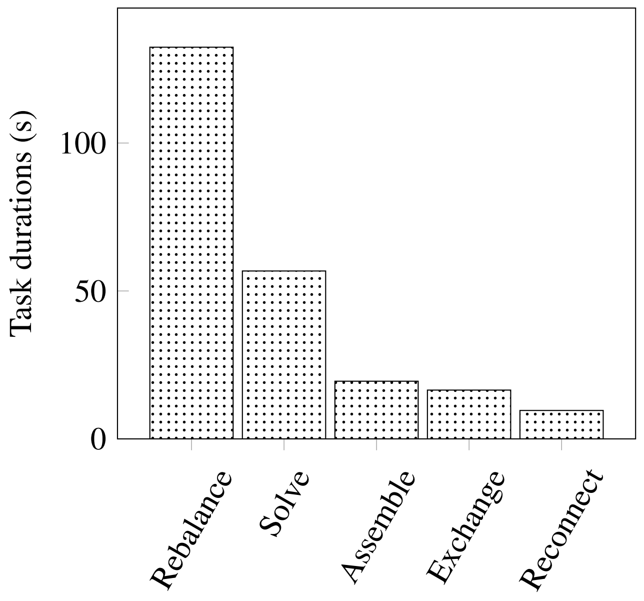


Figure 4.34: The most time consuming tasks in a time step. The task durations belong to the slowest processor.

optimize load balance for the assembler. As discussed in Section 3.2, it is possible to have separate partitions for the assembler and the flow solver in order to optimize load balance differently. However, having two separate partitions requires mapping of data across different partitioning layouts. The case with two different partitions is also tested and it is found that mapping of data increases total run time of simulation.

Frequency of load rebalance depends on relative time-cost of load rebalancing and the solver. In the case of 10 pseudo time steps for the solver, rebalancing is more costly than the solver, therefore, `Rebalance` is called when the ratio of the maximum to the minimum number of cells across processors exceeds 1000. The rebalance threshold is not the optimum and can be improved by an algorithm involving relative costs of tasks. Figure 4.35 shows the run time per time step with and without rebalance. Areas under each curve represents total run time over quarter rotation of rotor. The time saved with load rebalancing is 13% which adds up periodically for every quarter rotation due to

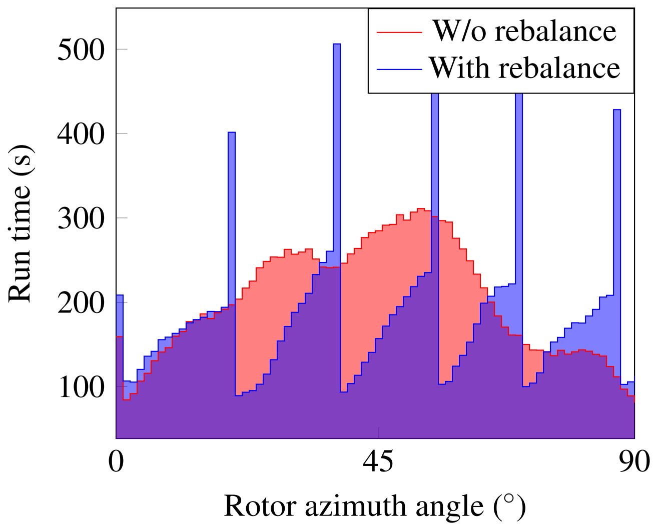


Figure 4.35: Run time of simulation in each time step with and without load rebalance. Area under each curve represents total run time over 90° rotation of rotor.

the fact that the initial spatial partitioning is restored for every quarter rotation.

Downside of the proposed load balancing method is that the whole load balancing procedure (octree construction and adaptive refinement) should be repeated from scratch. One of the main reasons of high cost of load rebalancing is high number of bins resulting from refinement of octree. As explained in Section 3.2, bins are merged after refinement of octree in order to reduce communication among bins. Time spent on merging of octree bins is proportional to the number of octree bins. In order to reduce load imbalance across processors below 10%, 344 refinement steps are executed, as shown in Figure 4.36 resulting in 1040 bins. Merging all 1040 bins into 128 (the number of processors) bins causes the load balancing to be the most time consuming task. Frequency of load rebalancing is also related to the number of merged octree bins. The higher the number of merged octree bins, the more is the number of mesh cells that move across octree bins after mesh motion in every time step.

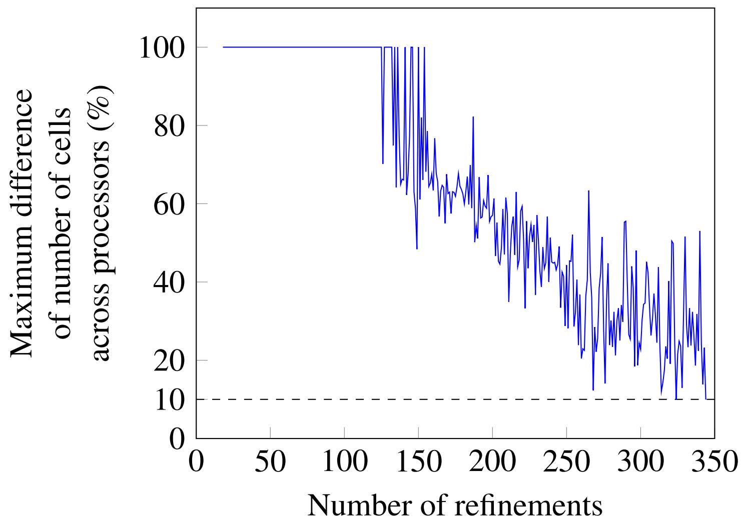


Figure 4.36: Reduction of load imbalance with increasing number of octree refinements. Dashed line indicates the threshold under which refinement stops.

The tasks `Exchange` and `Reconnect` are for exchange of mesh cells after mesh motion and reconnection of arrival mesh cells to the local mesh-blocks. `Exchange` has to be called in every time step in order to keep mesh cells in correct partitions/processors even though load rebalancing is to be executed.

Figure 4.37 shows total run time and speed-up for different number of processors. Speed-up is calculated by dividing run time by the run time for 64 processors. One of the reasons of speed-up, as expected, is distribution of computational load among more computational resources. Another reason of speed-up, in the particular case of partially shared memory computing, may be the increase in shared memory. Resource manager allocates 8 GB memory per core. Although memory is provided per core basis, cores are free to read/write any portion of cumulative shared memory space assigned to a user.

The number of 'walks' in stencil walk algorithm ranged from 3 to more than 1000.

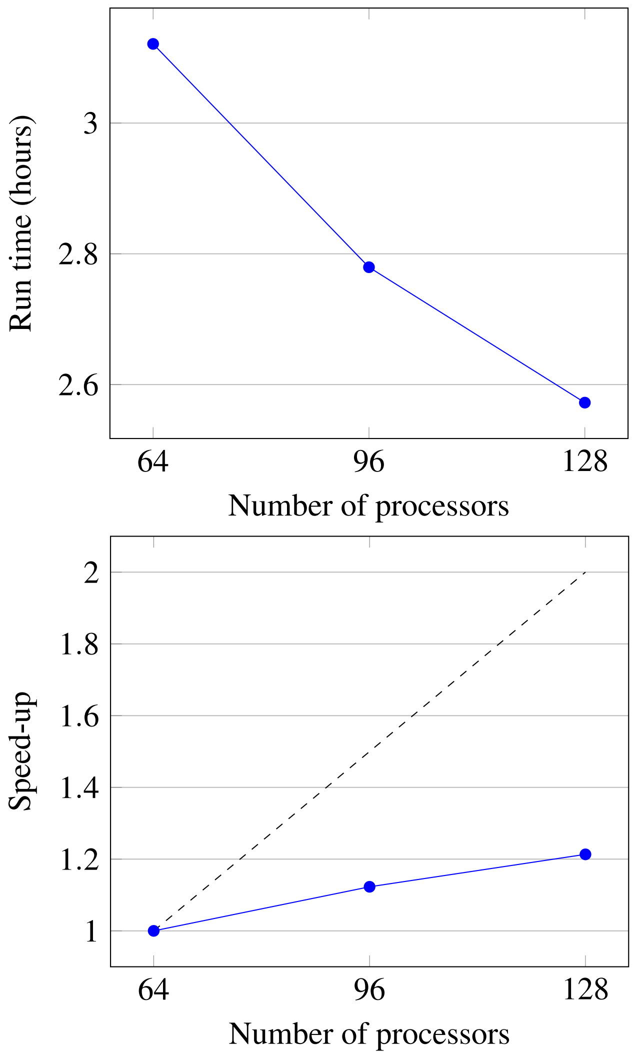


Figure 4.37: Run time and speed-up over 90° of rotor rotation.

Whenever the number of walks exceeded 1000, donor search algorithm switched from stencil walking to ADT. Usage of stencil walk algorithm saves 5% percent of time of donor search compared with only using ADT in every time step. However, in problems where time step is sufficiently large and current donor cells are sufficiently far away from the seeds, the high number of cell-to-cell walks will cause stencil walk algorithm to perform slower than ADT.

Although some of the tasks (e.g. donor search) are common in research papers pertaining overset mesh methodology [1, 8, 109], due to lack of exact definitions and implementation details of tasks, comparisons of parallel performance parameters can be misleading even if better results are obtained. In terms of partitioning method, this paper is an improvement over Suggar++ [1] by usage of adaptive refinement instead of predetermined volumes for spatial partitioning, therefore, it is useful to compare parallel performance results of Suggar++ with the results obtained in this paper. In Reference [1], wall clock time of combination of tasks that are hole cut, donor search and overlap minimization and respective speed-up are presented for the case of helicopter fuselage and blades for up to 8 processors. Since problem sizes (the number of cells) are different in Reference [1] and in this study, it is appropriate to compare speed-up results only. Figure 4.38 shows speed-up for the mentioned tasks. The main inhibitor of speed-up is hole cutting task as it requires inter-processor communication for AABB of mesh walls. However, for up to 8 processors communication cost across processors is not substantial. Donor search and overlap minimization are completely local tasks meaning that there is no inter-processor communication required. Therefore, speed-up for these three tasks remains nearly linear for up to 8 processors. Note that, in

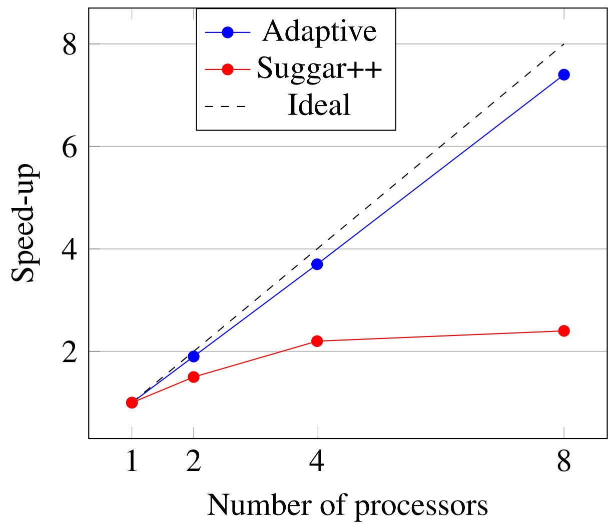


Figure 4.38: Speed-up for combination of tasks hole cutting, donor search, and overlap minimization.

this work, hole cutting is an integral part of donor search as opposed to Suggar++'s direct hole cutting which is a standalone task. As implementation details of Suggar++ are unknown, it is unclear why Suggar++ shows such non-linear behavior. It should also be noted that spatial partitioning is a memory intensive operation. Since memory cannot explicitly be requested and is provided per core basis in this work, some of the modules such as the flow solver have to be deactivated in order to obtain performance results for 8 and less number processors.

Chapter 5

CONCLUSIONS

In the present study, an unsteady compressible flow solver with Overset Mesh capability was developed for running in distributed computing environment. Domain connectivity was achieved with a developed overset grid assembler. Overlapping cells were identified with Alternating Digital Tree (ADT) of cell AABBs in the first time step. In subsequent time steps, stencil walk algorithm was implemented by starting from a seed cell and 'walking' to the containing cell using cell-to-cell connectivity. The overlapping cells that were identified in the first time step with ADT were used as seeds in the stencil-walk algorithm. It was found that usage of stencil walk algorithm saved 5% percent of time spent on domain connectivity compared to ADT in every time step.

In Reference [1], pre-defined volumes were used for spatial partitions and no load balancing was performed. This study improved the spatial partitioning approach in Reference [1] by avoiding problem dependent usage of pre-defined volumes and also by implementing adaptive load balancing after spatial partitioning. Load balancing has usually been ignored in Overset Mesh simulations and, to the knowledge of the present author, has never been applied to spatially partitioned mesh-systems.

It was observed that time-consuming tasks remain to be the same in all time steps. Even though load (re-)balancing was found to be the most time consuming task, it was shown that frequent load balancing reduced total simulation time considerably. The time saved with load rebalancing was 13% which added up periodically for every quarter rotation. The main reason for the high cost of load balancing was the high number of octree bins resulting from refinement of octree in order to reduce imbalance under a predefined threshold.

Downside of the proposed load balancing method was that the whole load balancing procedure (octree construction and adaptive refinement) had to be repeated from scratch in subsequent time steps. One of the main reasons of high cost of load rebalancing was high number of bins resulting from refinement of octree. Merging all bins resulted in the load balancing to be the most time consuming task.

Frequency of load rebalance depended on relative time-cost of load rebalancing and the solver. The re-balancing algorithm was executed when the ratio of the maximum to the minimum number of cells across processors exceeded a pre-defined threshold. The rebalance threshold was not the optimum and can be improved by an algorithm involving relative costs of tasks.

The assembler constituted less time compared to the solver, therefore, it was unnecessary to have separate partitions for the assembler and the flow solver in order to optimize load balance differently for the assembler and the solver. The case with two different partitions was also tested and it was found that having two separate partitions increased total run time of simulation because of mapping of data across different partitioning layouts.

Speed-up results for combination of tasks (hole cut, donor search and overlap minimization) in the present work were compared with Suggar++ [1] which provided speed-up results for up to 8 processors. It was observed that present speed-up results showed linear behaviour compared to non-linear speed-up in Suggar++. Additionally, higher speed-up was obtained compared with Suggar++. In this work, the main inhibitor of speed-up was hole cutting task as it required inter-processor communication for AABB of wall boundaries. However, for up to 8 processors communication cost across processors was not substantial.

Memory usage after major tasks of the assembler and the solver were recorded. It was found that data exchange across processors was the most memory intensive operation. Memory usage kept increasing throughout the simulation. The reason of constant increase in memory usage was the allocation/deallocation of instances after data exchange. OpenMPI which is an implementation of Message Passing Interface (MPI) tends to reuse send/receive buffers and does not return buffer memory even if requested. Improvement of the assembler and the solver by reusage of send/receive buffers is one of the future works of this study. In order to restart the simulation after running out of memory, data files were serialized/deserialized at certain time step intervals.

In order to validate the assembler and the solver, several test cases were simulated including the rotor-fuselage interaction on a generic helicopter configuration which consisted of six components, namely, a fuselage, four NACA0012 blades and a hub. The present results were compared with experimental results in Reference [97]. The experimental configuration in Reference [97] additionally included two attachments:

One for connecting the hub to the ceiling and another one for attaching the fuselage to the floor. In flow simulations around generic helicopter configurations, usually, only the fuselage and the blades are included. In this study, the hub was also included in the CAD model to further increase the accuracy of flow solution.

A near hover flight condition with advance ratio of 0.01 was evaluated. Thrust coefficient was over-predicted by the simulation. Over-prediction of the thrust coefficient is usual in flow simulations. However, computational coefficients were found to be lower than the experimental counterparts by about one order of magnitude. The discrepancy in coefficients was partly due to the Euler equations for which viscous effects were ignored.

Unsteady pressure coefficients were compared with the experimental measurements at control points located on top centerline of the fuselage and around the fuselage at the end of the pylon. On all control points, phases of four pulses corresponding to the four blades were accurately captured. However, magnitudes of pressure coefficients showed some discrepancy at some control points. Computational pressure coefficients were in good agreement at the regions closer the nose of the fuselage. Towards the pylon, where, all component meshes overlap, there was a discrepancy between the computational and the experimental pressure coefficients due to extensive amount of interpolations across the component meshes. Although the Euler equations provided useful insight to the flow simulation around the generic helicopter configuration, assessing from the discrepancy in pressure coefficients between computational and experimental results due to flow separation and tip vortex shedding, viscous effects need to be accounted for better results.

5.1 Future work

Some improvements can be implemented to the overset mesh solver in order to speed up the execution time.

First, whenever load imbalance exceeds a threshold, the whole load balancing procedure (octree construction and adaptive refinement) had to be repeated from scratch in subsequent time steps. As load-balancing is the bottleneck, high speed-up is expected if load recomputation involves previous iteration without constructing the octree.

Second, frequency of load rebalance depends on relative time-cost of load rebalancing and the solver. The re-balancing algorithm is executed when the ratio of the maximum to the minimum number of cells across processors exceeded a pre-defined threshold. This approach is definitely not the optimum and can be improved by an algorithm involving relative costs of tasks.

Third, data exchange across processors is the most memory intensive operation. Memory usage keep increasing throughout the time-consuming simulations due to the allocation/deallocation of instances after data exchange. Usage of global buffers for send and receive buffers would solve the constly increasing memory usage.

REFERENCES

[1] R. W. Noack, D. A. Boger, R. F. Kunz, and P. M. Carrica, “SUGGAR++: An improved general overset grid assembly capability,” *Proceedings of the 47th AIAA Aerospace Science and Exhibit*, pp. 22–25, 2009.

[2] H. Hiester, M. Piggott, P. Farrell, and P. Allison, “Assessment of spurious mixing in adaptive mesh simulations of the two-dimensional lock-exchange,” *Ocean Modelling*, vol. 73, pp. 30–44, 2014.

[3] “Comsol,” https://www.comsol.com/blogs/your-guide-to-meshing-techniques-for-efficient-cfd-modeling/.

[4] “Air craft systems,” https://www.aircraftsystemstech.com/p/helicopter-structures.html.

[5] B. G. van der Wall, C. L. Burley, Y. Yu, H. Richard, K. Pengel, and P. Beaumier, “The HART II test – measurement of helicopter rotor wakes,” *Aerospace Science and Technology*, vol. 8, no. 4, pp. 273–284, Jun. 2004.

[6] J. Benek, J. Steger, and F. C. Dougherty, “A flexible grid embedding technique with application to the Euler equations,” in *6th Computational Fluid Dynamics Conference*, 1983, p. 1944.

[7] G. Karypis and V. Kumar, "METIS: Unstructured graph partitioning and sparse matrix ordering system, version 5.0," Department of Computer Science, University of Minnesota, Tech. Rep., 1995.

[8] B. Roget and J. Sitaraman, "Robust and efficient overset grid assembly for partitioned unstructured meshes," *Journal of Computational Physics*, vol. 260, pp. 1–24, 2014.

[9] G. Chesshire and W. D. Henshaw, "Composite overlapping meshes for the solution of partial differential equations," *Journal of Computational Physics*, vol. 90, no. 1, pp. 1–64, 1990.

[10] N. Suhs and R. Tramel, "PEGSUS 4.0 user's manual," Arnold Engineering Development Center, Tech. Rep., 1991.

[11] W. M. Chan and P. G. Buning, "User's manual for FOMOCO utilities-force and moment computation tools for overset grids," NASA Technical Memorandum, Tech. Rep., 1996.

[12] D. Brown, W. Henshaw, and D. Quinlan, "Overture-object-oriented tools for overset grid applications," in *17th Applied Aerodynamics Conference*, 1999, p. 3130.

[13] N. A. Petersson, "An algorithm for assembling overlapping grid systems," *SIAM*

Journal on Scientific Computing, vol. 20, no. 6, pp. 1995–2022, 1999.

[14] Z. Wang, V. Parthasarathy, and N. Hariharan, "A fully automated Chimera methodology for multiple moving body problems," *International Journal for Numerical Methods in Fluids*, vol. 33, no. 7, pp. 919–938, 2000.

[15] D. D. Chandar, J. Sitaraman, and D. J. Mavriplis, "A GPU-based incompressible Navier–Stokes solver on moving overset grids," *International Journal of Computational Fluid Dynamics*, vol. 27, no. 6-7, pp. 268–282, 2013.

[16] K. Soni, D. D. Chandar, and J. Sitaraman, "Development of an overset grid computational fluid dynamics solver on graphical processing units," *Computers & Fluids*, vol. 58, pp. 1–14, 2012.

[17] A. Mishra, D. Jude, and J. D. Baeder, "A GPU accelerated adjoint solver for shape optimization," in *2018 Fluid Dynamics Conference*, 2018, p. 3557.

[18] A. Marongiu and L. Benini, "Efficient OpenMP support and extensions for MPSoCs with explicitly managed memory hierarchy," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 809–814.

[19] M. P. Forum, "MPI: A message-passing interface standard," University of Tennessee, USA, Tech. Rep., 1994.

[20] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[21] W. Gropp, "MPICH2: A new start for MPI implementations," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2002, pp. 7–7.

[22] "Intel MPI library," https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/mpi-library.html.

[23] B. L. Chamberlain, *Chapel (Cray Inc. HPCS Language)*. Boston, MA: Springer US, 2011, pp. 249–256. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_54

[24] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671

[25] C. L. Weeks, "Concurrent extensions to the FORTRAN language for parallel programming of computational fluid dynamics algorithms," NASA Technical Memorandum, Tech. Rep., 1986.

[26] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[27] "Elasticluster," https://github.com/elasticluster/elasticluster.

[28] N. C. Prewitt, D. M. Belk, and W. Shyy, "Parallel computing of overset grids for aerodynamic problems with moving objects," *Progress in Aerospace Sciences*, vol. 36, no. 2, pp. 117–172, 2000.

[29] J. Cai, H. M. Tsai, and F. Liu, "An overset grid solver for viscous computations with multigrid and parallel computing," in *16th AIAA Computational Fluid Dynamics Conference*, 2003, p. 4232.

[30] M. J. Djomehri and R. Biswas, "Performance enhancement strategies for multi-block overset grid CFD applications," *Parallel Computing*, vol. 29, no. 11-12, pp. 1791–1810, 2003.

[31] N. Suhs, S. Rogers, and W. Dietz, "Pegasus 5: An automated pre-processor for overset-grid CFD," in *32nd AIAA Fluid Dynamics Conference and Exhibit*, 2002, p. 3186.

[32] M. J. Djomehri, R. Biswas, and N. Lopez-Benitez, "Load balancing strategies for multi-block overset grid applications." in *Computers and Their Applications*,

2003, pp. 373–378.

[33] E. Kim, J.-H. Kwon, and S. H. Park, "Parallel performance assessment of moving body overset grid application on PC cluster," in *Parallel Computational Fluid Dynamics*, vol. 18. Amsterdam; Oxford; Elsevier, 2006, pp. 59–66.

[34] J. Cai, H. M. Tsai, and F. Liu, "A parallel viscous flow solver on multi-block overset grids," *Computers & Fluids*, vol. 35, no. 10, pp. 1290–1301, 2006.

[35] J. J. Alonso, S. Hahn, F. Ham, M. Herrmann, G. Iaccarino, G. Kalitzin, P. LeGresley, K. Mattsson, G. Medic, P. Moin *et al.*, "Chimps: A high-performance scalable module for multi-physics simulations," in *Proceedings of the 42nd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Sacramento, CA, July*, 2006, pp. 9–12.

[36] J. Sitaraman, M. Floros, A. Wissink, M. Potsdam, and V. Sankaran, "Parallel unsteady overset mesh methodology for a multi-solver paradigm with adaptive Cartesian grids," in *26th AIAA Applied Aerodynamics Conference*, 2008, p. 7177.

[37] G. Zagaris, M. T. Campbell, D. J. Bodony, E. Shaffer, and M. D. Brandyberry, "A toolkit for parallel overset grid assembly targeting large-scale moving body aerodynamic simulations." in *IMR*. Springer, 2010, pp. 385–401.

[38] J. Sitaraman, M. Floros, A. Wissink, and M. Potsdam, "Parallel domain

connectivity algorithm for unsteady flow computations using overlapping and adaptive grids," *Journal of Computational Physics*, vol. 229, no. 12, pp. 4703–4723, 2010.

[39] P. G. Buning and T. H. Pulliam, "Cartesian off-body grid adaption for viscous time-accurate flow simulations," in *20th AIAA Computational Fluid Dynamics Conference*, 2011, pp. 27–30.

[40] B. Landmann and M. Montagnac, "A highly automated parallel Chimera method for overset grids based on the implicit hole cutting technique," *International Journal for Numerical Methods in Fluids*, vol. 66, no. 6, pp. 778–804, 2011.

[41] E. J. Nielsen and B. Diskin, "Discrete adjoint-based design for unsteady turbulent flows on dynamic overset unstructured grids," *AIAA Journal*, vol. 51, no. 6, pp. 1355–1373, 2013.

[42] M. J. Brazell, J. Sitaraman, and D. J. Mavriplis, "An overset mesh approach for 3D mixed element high-order discretizations," *Journal of Computational Physics*, vol. 322, pp. 33–51, 2016.

[43] C. Liu, "A stabilized finite element dynamic overset method for the Navier–Stokes equations," Ph.D. dissertation, University of Tennessee at Chattanooga, 2016.

[44] G. K. Kenway, A. Mishra, N. R. Secco, K. Duraisamy, and J. Martins, “An efficient parallel overset method for aerodynamic shape optimization,” in *58th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2017, p. 0357.

[45] X. Hu, Z. Lu, J. Zhang, W. Yuan, X. Liu, and N. Nie, “An efficient parallel Chimera grid flow solver based on implicit hole cutting method,” in *AIP Conference Proceedings*, vol. 1978, no. 1. AIP Publishing LLC, 2018, p. 230002.

[46] J. Crabill, F. D. Witherden, and A. Jameson, “A parallel direct cut algorithm for high-order overset methods with application to a spinning golf ball,” *Journal of Computational Physics*, vol. 374, pp. 692–723, 2018.

[47] X. Chang, R. Ma, N. Wang, and L. Zhang, “Parallel implicit hole-cutting method for unstructured Chimera Grid,” *Acta Aeronautica et Astronautica Sinica*, vol. 39, no. 6, pp. 48–58, 2018.

[48] W. Wang, C. Yan, S. Wang, Y. Huang, and W. Yuan, “An efficient, robust and automatic overlapping grid assembly approach for partitioned multi-block structured grids,” *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 233, no. 4, pp. 1217–1236, 2019.

[49] R. Meakin and A. Wissink, “Unsteady aerodynamic simulation of static and

moving bodies using scalable computers," in *14th Computational Fluid Dynamics Conference*, 1999, p. 3302.

[50] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[51] N. Prewitt, D. Belk, and W. Shyy, "Distribution of work and data for parallel grid assembly," in *37th Aerospace Sciences Meeting and Exhibit*, 1999, p. 913.

[52] A. M. Wissink and R. L. Meakin, "On parallel implementations of dynamic overset grid methods," in *SC'97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*. IEEE, 1997, pp. 15–15.

[53] R. Meakin, "A new method for establishing intergrid communication among systems of overset grids," in *10th Computational Fluid Dynamics Conference*, 1991, p. 1586.

[54] J. Bonet and J. Peraire, "An Alternating Digital Tree (ADT) algorithm for 3D geometric searching and intersection problems," *International Journal for Numerical Methods in Engineering*, vol. 31, no. 1, pp. 1–17, jan 1991.

[55] L. Hall and V. Parthasarathy, "Validation of an automated Chimera/6-dof methodology for multiple moving body problems," in *36th AIAA Aerospace*

Sciences Meeting and Exhibit, 1998, p. 753.

[56] D. Belk and R. Maple, "Automated assembly of structured grids for moving body problems," in *12th Computational Fluid Dynamics Conference*, 1995, p. 1680.

[57] S. Pissanetzky and F. G. Basombrío, "Efficient calculation of numerical values of a polyhedral function," *International Journal for Numerical Methods in Engineering*, vol. 17, no. 2, pp. 231–237, 1981.

[58] M. Khoshniat, G. R. Stuhne, and D. A. Steinman, "Relative performance of geometric search algorithms for interpolating unstructured mesh data," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2003, pp. 391–398.

[59] B. Roget and J. Sitaraman, "Wall distance search algorithm using voxelized marching spheres," *Journal of Computational Physics*, vol. 241, pp. 76–94, 2013.

[60] I.-T. Chiu and R. Meakin, "On automating domain connectivity for overset grids," in *33rd Aerospace Sciences Meeting and Exhibit*, 1995, p. 854.

[61] G. Mei, "Realmodel: A system for modeling and visualizing sedimentary rocks," Ph.D. dissertation, University of Freiburg, 2014.

[62] "MPI memory," https://stackoverflow.com/questions/13088772/mpi-send-takes-huge-part-of-virtual-memory.

[63] K. Chand, "Component-based hybrid mesh generation," *International Journal for Numerical Methods in Engineering*, vol. 62, no. 6, pp. 747–773, 2005.

[64] M. S. Jung and O. J. Kwon, "A conservative overset mesh scheme via intergrid boundary reconnection on unstructured meshes," in *19th AIAA Computational Fluid Dynamics*, 2009, p. 3536.

[65] E. Rinaldi, P. Colonna, and R. Pecnik, "Flux-conserving treatment of non-conformal interfaces for finite-volume discretization of conservation laws," *Computers & Fluids*, vol. 120, pp. 126–139, 2015.

[66] K. P. Fishkin and B. A. Barsky, "An analysis and algorithm for filling propagation," in *Computer-generated Images*. Springer, 1985, pp. 56–76.

[67] S. Lemaire, G. Vaz, and S. Turnock, "On the need for higher order interpolation with overset grid methods," 2019.

[68] E. F. Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer Science & Business Media, 2013.

[69] B. Diskin and J. L. Thomas, "Comparison of node-centered and cell-centered

unstructured finite-volume discretizations: inviscid fluxes," *AIAA journal*, vol. 49, no. 4, pp. 836–854, 2011.

[70] A. Jeffrey, "Quasilinear hyperbolic systems and waves," *London*, 1976.

[71] P. L. Roe, "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, vol. 43, no. 2, pp. 357–372, 1981.

[72] P. Roe and J. Pike, "Efficient construction and utilisation of approximate Riemann solutions," in *Proc. of the sixth Int'l. Symposium on Computing methods in Applied Sciences and Engineering, VI*, 1985, pp. 499–518.

[73] A. Harten and J. M. Hyman, "Self adjusting grid methods for one-dimensional hyperbolic conservation laws," *Journal of Computational Physics*, vol. 50, no. 2, pp. 235–269, 1983.

[74] P. L. Roe, "Sonic flux formulae," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 2, pp. 611–630, 1992.

[75] F. Dubois and G. Mehlman, "Nonparameterized entropy fix for Roe's method," *AIAA Journal*, vol. 31, no. 1, pp. 199–200, 1993.

[76] B. Einfeldt, C.-D. Munz, P. L. Roe, and B. Sjögreen, "On Godunov-type methods near low densities," *Journal of Computational Physics*, vol. 92, no. 2, pp. 273–

295, 1991.

[77] A. Harten, P. D. Lax, and B. v. Leer, “On upstream differencing and Godunov-type schemes for hyperbolic conservation laws,” *SIAM Review*, vol. 25, no. 1, pp. 35–61, 1983.

[78] S. Davis, “Simplified second-order Godunov-type methods,” *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 3, pp. 445–473, 1988.

[79] B. Einfeldt, “On Godunov-type methods for gas dynamics,” *SIAM Journal on Numerical Analysis*, vol. 25, no. 2, pp. 294–318, 1988.

[80] E. F. Toro, M. Spruce, and W. Speares, “Restoration of the contact surface in the HLL–Riemann solver,” *Shock Waves*, vol. 4, no. 1, pp. 25–34, 1994.

[81] E. Toro, “A linearized Riemann solver for the time-dependent Euler equations of gas dynamics,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 434, no. 1892, pp. 683–693, 1991.

[82] B. Van Leer, “Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov’s method,” *Journal of computational Physics*, vol. 32, no. 1, pp. 101–136, 1979.

[83] J. A. White, H. Nishikawa, and R. A. Baurle, “Weighted least-squares cell-

average gradient construction methods for the VULCAN-CFD second-order accurate unstructured grid cell-centered finite-volume solver," in *AIAA Scitech 2019 Forum*, 2019, p. 0127.

[84] A. Haselbacher and J. Blazek, "Accurate and efficient discretization of navier-stokes equations on mixed grids," *AIAA journal*, vol. 38, no. 11, pp. 2094–2102, 2000.

[85] T. Barth and D. Jespersen, "The design and application of upwind schemes on unstructured meshes," in *27th Aerospace Sciences Meeting*, 1989, p. 366.

[86] V. Venkatakrishnan, "On the accuracy of limiters and convergence to steady state solutions," in *31st Aerospace Sciences Meeting*, 1993, p. 880.

[87] S. Venkateswaran and C. Merkle, "Dual time-stepping and preconditioning for unsteady computations," in *33rd Aerospace Sciences Meeting and Exhibit*, 1995, p. 78.

[88] D. Demidov, "AMGCL – A C++ library for efficient solution of large sparse linear systems," *Software Impacts*, vol. 6, p. 100037, 2020. [Online]. Available: https://doi.org/10.1016/j.simpa.2020.100037

[89] G. Mengaldo, D. De Grazia, F. Witherden, A. Farrington, P. Vincent, S. Sherwin, and J. Peiro, "A guide to the implementation of boundary conditions in compact

high-order methods for compressible aerodynamics," in *7th AIAA Theoretical Fluid Mechanics Conference*, 2014.

[90] L. Manzano, J. Lassaline, and D. Zingg, "A newton-krylov algorithm for the euler equations using unstructured grids," in *41st Aerospace Sciences Meeting and Exhibit*, 2003, p. 274.

[91] V. Venkatakrishnan and D. Mavriplis, "Implicit method for the computation of unsteady flows on unstructured grids," *Journal of Computational Physics*, vol. 127, no. 2, pp. 380–397, 1996.

[92] R. Landon, "Data from AGARD Report 702: NACA 64 A 006 Oscillating Flap; NACA 012 Oscillatory and Transient Pitching; NLR 7301 Supercritical Airfoil Oscillatory Pitching and Oscillating Flap; and ZKP Wing, Oscillating Aileron," *2000.*, 2000.

[93] V. Schmitt, "Pressure distributions on the ONERA M6-wing at transonic mach numbers, experimental data base for computer program assessment," *AGARD AR-138*, 1979.

[94] G. Araya, "Turbulence model assessment in compressible flows around complex geometries with unstructured grids," *Fluids*, vol. 4, no. 2, p. 81, 2019.

[95] M. K. Singh, V. Ramesh, and N. Balakrishnan, "Implicit scheme for meshless

compressible euler solver," *Engineering Applications of Computational Fluid Mechanics*, vol. 9, no. 1, pp. 382–398, 2015.

[96] P. Cook, M. Mcdonald, and M. Firmin, "Experimental data base for computer program assessment," *AGARD AR*, vol. 138, 1979.

[97] R. E. Mineck, *Steady and periodic pressure measurements on a generic helicopter fuselage model in the presence of a rotor*. NASA Langley Research Center, 2000.

[98] M. Insulander, "Development of a helicopter simulation for operator interface research," Master's thesis, Institution for Man-System Interaction, Swedish Defense Research Agency, 2008.

[99] J. Riegel, W. Mayer, and Y. van Havre, "FreeCAD (version 0.18.4)," http://www.freecadweb.org.

[100] C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[101] B. Kubrak and D. Snyder, "CFD code vailidation of rotor/fuselage interaction using the commercial software STAR-CCM+ 8.04," SIEMENS, Tech. Rep., 2013.

[102] Y. Tanabe, I. Otani, and S. Saito, *Validation of computational results of rotor/fuselage interaction analysis using rFlow3D code*. Japan Aerospace Exploration Agency, 2010.

[103] B.-S. Lee, M.-S. Jung, O.-J. Kwon, and H.-J. Kang, "Numerical simulation of rotor-fuselage aerodynamic interaction using an unstructured overset mesh technique," *International Journal of Aeronautical and Space Sciences*, vol. 11, no. 1, pp. 1–9, 2010.

[104] A. R. Kenyon and R. E. Brown, "Wake dynamics and rotor-fuselage aerodynamic interactions," *Journal of the American Helicopter Society*, vol. 54, no. 1, pp. 12 003–12 003, 2009.

[105] H. Xu and S. Zhang, "Aerodynamic investigation of unsteady flow past robin helicopter with four-bladed rotor in forward-flight," in *32nd European Rotorcraft Forum*, 2006.

[106] R. E. Mineck, "Application of an unstructured grid Navier–Stokes solver to a generic helicopter body," NASA Technical Memorandum, Tech. Rep., 1999.

[107] "TRUBA," https://www.truba.gov.tr/index.php/en/main-page/.

[108] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *ACM Sigplan notices*, vol. 42, no. 6, pp. 89–100, 2007.

[109] W. J. Horne and K. Mahesh, “A massively-parallel, unstructured overset method for mesh connectivity,” *Journal of Computational Physics*, vol. 376, pp. 585–596, Jan 2019.

APPENDICES

Appendix A: Coefficients in super-ellipse equations

Table A.1: Coefficients for the shape of fuselage

$0.0 < x/l < 0.4$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	1.0	-1.0	-0.4	0.4	1.8	0.0	0.25	1.8
W	1.0	-1.0	-0.4	0.4	2.0	0.0	0.25	2.0
Z_0	1.0	-1.0	-0.4	0.4	1.8	-0.08	0.08	1.8
N	2.0	3.0	0.0	0.4	1.0	0.0	1.0	1.0
$0.4 \leq x/l < 0.8$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	0.0	0.0	0.0	0.0	0.0	0.25	0.0	0.0
W	0.0	0.0	0.0	0.0	0.0	0.25	0.0	0.0
Z_0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
N	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0
$0.8 \leq x/l < 1.9$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	1.0	-1.0	-0.8	1.1	1.5	0.05	0.2	0.6
W	1.0	-1.0	-0.8	1.1	1.5	0.05	0.2	0.6
Z_0	1.0	-1.0	-0.8	1.1	1.5	0.04	-0.04	0.6
N	5.0	-3.0	-0.8	1.1	1.0	0.0	1.0	1.0
$1.9 \leq x/l < 2.0$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	1.0	-1.0	-1.9	0.1	2.0	0.0	0.05	2.0
W	1.0	-1.0	-1.9	0.1	2.0	0.0	0.05	2.0
Z_0	0.0	0.0	0.0	0.0	0.0	0.04	0.0	0.0
N	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0

Table A.2: Coefficients for the shape of pylon

$0.4 < x/l < 0.8$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	1.0	-1.0	-0.8	0.4	3.0	0.0	0.145	3.0
W	1.0	-1.0	-0.8	0.4	3.0	0.0	0.166	3.0
Z_0	0.0	0.0	0.0	0.0	0.0	0.125	0.0	0.0
N	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0

$0.8 \leq x/l < 1.018$								
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
H	1.0	-1.0	-0.8	0.218	2.0	0.0	0.145	2.0
W	1.0	-1.0	-0.8	0.218	2.0	0.0	0.166	2.0
Z_0	1.0	-1.0	-0.8	1.1	1.5	0.065	0.06	0.6
N	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0

Appendix B: Scripts

```
import sys
import math
import numpy
import Part
import BOPTools.JoinAPI
from mesh import *
from coef_fuselage import *
from coef_pylon import *
from make_component import *
from constant import *
from modify_geo import *

doc = FreeCAD.newDocument('newdoc')
fuselage_start = 0.001
fuselage_end = 1.997
pylon_start = 0.40001
pylon_end = 1.018
radial_start = 0.00001
nx_fuselage = 10
nx_pylon = 10
np = 10
L_inch = 78.57
L = L_inch * INCH_TO_METER / fuselage_end
dia = 50

px_fuselage = numpy.linspace(
fuselage_start, fuselage_end, nx_fuselage)
px_pylon = numpy.linspace(
pylon_start, pylon_end, nx_pylon)
pr = numpy.linspace(radial_start, 2.0*math.pi, np)
fuselage = make_component(Fuselage, px_fuselage, pr, L)
pylon = make_component(Pylon, px_pylon, pr, L) # make pylon.
helicopter = BOPTools.JoinAPI.connect([fuselage, pylon])
solid_helicopter = Part.Solid(helicopter)
sphere = Part.makeSphere(dia,FreeCAD.Vector(1,0,0))
cut = sphere.cut(solid_helicopter)
cut_object = doc.addObject("Part::Feature","Cut")
cut_object.Shape = cut
mesh(cut_object, 'main_body', doc)
file_name = "fuselage_pylon"
modify_geo(file_name, 'Cut')
```

Figure B.1: FreeCAD Python script for the generic fuselage model.

```
import sys
import numpy
import BOPTools.JoinAPI
from mesh import *
from make_component import *
from constant import *
from modify_geo import *
from NACA0012 import *
from blade_helper import *

doc = App.newDocument('newdoc')
fuselage_end = 1.997
L_inch = 78.57
L = L_inch * INCH_TO_METER / fuselage_end
chord_inch = 2.61
chord = chord_inch * INCH_TO_METER
aoa = 4;
twist = 8;
blade_radius_inch = 33.88
blade_radius = blade_radius_inch * INCH_TO_METER
root_cut_out = 0.24 * blade_radius
blade_length = blade_radius - root_cut_out
nx = 10;
hub_center = [0.696 * L, 0, 0.322 * L]
cylinder_radius = chord * 10
cylinder_length = cylinder_radius * 2

x = numpy.linspace(0.00001, chord, nx)
points = make_NACA0012_points(x, chord)
blade = [make_blade(points, aoa, twist, blade_length)

for i in range(4):
    if i == 0 or i == 2:
        ref_blade = blade[0]
    if i == 1 or i == 3:
        ref_blade = blade[1]
    pos = reposition_blade(
    ref_blade, blade[i], i, blade_radius, hub_center, chord)
    blade_object = doc.addObject("Part::Feature","Blade"+str(i))
    blade_object.Shape = blade[i]
    cylinder = make_cylinder(i, cylinder_radius, chord,
    cylinder_length, blade_length, hub_center[2], pos)
    cut = cylinder.cut(blade)
    cut_object = doc.addObject(
    "Part::Feature","Cut" + str(i))
    cut_object.Shape = cut
    mesh(cut_object, 'blade', doc)
    file_name = "blade" + str(i)
    modify_geo(file_name, 'Cut', hub_center)
```

Figure B.2: FreeCAD Python script for the rotor blades.