

2008

## Parallel Unstructured Grid Generation for Complex Real-World Aerodynamic Simulations

George Zagaris  
*College of William & Mary - Arts & Sciences*

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Aerospace Engineering Commons](#), and the [Computer Sciences Commons](#)

---

### Recommended Citation

Zagaris, George, "Parallel Unstructured Grid Generation for Complex Real-World Aerodynamic Simulations" (2008). *Dissertations, Theses, and Masters Projects*. Paper 1539626877.  
<https://dx.doi.org/doi:10.21220/s2-my1j-p815>

This Thesis is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact [scholarworks@wm.edu](mailto:scholarworks@wm.edu).

PARALLEL UNSTRUCTURED GRID GENERATION FOR  
COMPLEX REAL-WORLD AERODYNAMIC SIMULATIONS

George Zagaris  
Williamsburg, Virginia

Bachelor of Computer Science, College of William and Mary, 2005

A Thesis presented to the Graduate Faculty  
of the College of William and Mary in Candidacy for the Degree of  
Master of Science

Department of Computer Science

The College of William and Mary  
August 2008

# APPROVAL PAGE

This Thesis is submitted in partial fulfillment of  
the requirements for the degree of

Master of Science


  
George Zagaris

Approved by the Committee, June, 2008

  
Committee Chair

Associate Professor Nikos Chrisochoides, Computer Science  
The College of William & Mary

  
Senior Research Engineer Shahyar Pirzadeh, Configuration Aerodynamics Branch  
NASA Langley Research Center

  
Associate Professor Weizhen Mao, Computer Science  
The College of William & Mary

## ABSTRACT PAGE

The focus of this thesis is the design and implementation of a framework for Parallel Unstructured Grid Generation based on the Advancing Front (AF) technique. Parallel Unstructured Grid Generation is a challenging problem for two reasons. First, Domain Decomposition (DD) is a necessary pre-processing step by which the computational domain is decomposed in several smaller sub-domains that can be meshed in parallel. The DD problem is difficult because the size and shape of the separators must be compatible with the local spacing and shape required by the application. Second, Load Balancing (LB) is the assignment of the sub-domains to a set of processors such that the load is evenly distributed. The LB problem is difficult because accurately estimating the load of each sub-domain is impossible due to the heuristic nature of the AF method. The solution to both problems is crucial for Parallel unstructured grid generation which is an enabling technology for large-scale and high performance simulations. Three DD approaches for parallel unstructured grid generation by the AF technique are presented and evaluated in the context of aerodynamic applications. First, a Discrete Domain Decomposition approach is presented by which the domain, defined by a coarse mesh, is decomposed in several sub-domains using robust graph partitioning algorithms. This is a well known approach and is used as the base case. The second approach utilizes the oct-tree as an auxiliary data structure to partition an "empty", i.e., unmeshed, domain by Recursive Coordinate Bisection (RCB). Lastly, an approach which utilizes the Advancing-Partition (AP) strategy that is native to the underlying grid generator is described. In contrast to earlier domain decomposition approaches, the Advancing-Partition strategy benefits from inherent properties of the advancing front algorithm and is particularly suited for parallel unstructured grid generation targeting complex, real-world aerodynamic configurations.

# Table of Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgments</b>	<b>vii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>x</b>
<b>1 Introduction</b>	<b>2</b>
1.1 Motivation . . . . .	5
1.2 Overview of VGRID . . . . .	7
<b>2 Related Work</b>	<b>9</b>
<b>3 Parallel Grid Generation by Discrete Domain Decomposition</b>	<b>12</b>
3.1 Fixing Odd-shape Partitions by Smoothing . . . . .	15
3.2 Closing the Grid by Constrained Delaunay Tetrahedralization . . . . .	16
3.3 Results . . . . .	18
3.3.1 A Wing/Fuselage DLR-F6 Transport Configuration . . . . .	21
3.3.2 A Generic Business Jet . . . . .	22
3.4 Discussion . . . . .	23
<b>4 Parallel Grid Generation by Octree-guided Domain Decomposition</b>	<b>25</b>
4.1 The Oct-tree Guided Domain Decomposition . . . . .	27
4.1.1 Octree Subdivision . . . . .	27
4.1.2 Patch Sub-Domain Extraction . . . . .	27
4.2 Mesh Generation . . . . .	28
4.3 Discussion . . . . .	29
<b>5 Parallel Grid Generation by the Advancing Partition Technique</b>	<b>32</b>
5.1 Parallel Domain Decomposition by the Advancing Partition Technique . .	34
5.2 Parallel Mesh Generation . . . . .	36
5.3 Results . . . . .	37
5.3.1 A Uniform Box Configuration . . . . .	37
5.3.2 A Sphere-in-box Configuration . . . . .	39
5.3.3 A Wing/Fuselage DLR-F6 Transport Configuration . . . . .	41

5.3.4	A Generic Supersonic Jet Configuration . . . . .	44
5.3.5	Energy Efficient Transport Configuration . . . . .	48
5.4	Discussion . . . . .	49
<b>6</b>	<b>Conclusion</b>	<b>55</b>
6.1	Future Work . . . . .	57
<b>A</b>	<b>Pseudocode</b>	<b>58</b>
A.1	Parallel Domain Decomposition . . . . .	58
A.1.1	Master Domain Decomposition Process . . . . .	58
A.1.2	Worker Domain Decomposition Process . . . . .	60
A.2	Parallel Mesh Generation . . . . .	61
A.2.1	Master Parallel Mesh Generation Process . . . . .	61
A.2.2	Worker Parallel Mesh Generation Process . . . . .	62
<b>B</b>	<b>User Guide</b>	<b>63</b>
B.1	Installation . . . . .	63
B.2	Running PVGRID . . . . .	64
	<b>Bibliography</b>	<b>66</b>
	<b>Vita</b>	<b>70</b>

To Nikoleta...

## ACKNOWLEDGMENTS

The author wishes to thank Dr. Nikos Chrisochoides at the College of William and Mary and Dr. Shahyar Pirzadeh from NASA Langley Research Center for their continuous support, for sharing their knowledge and experience in grid generation and for the numerous of insightful, motivating and inspiring discussions. Dr. Andrey Chernikov from the College of William and Mary for contributing the base-code for the METIS partitioner and for helpful discussions on mesh generation algorithms. Dr. Leonidas Linardakis from the College of William and Mary and the Virginia Institute of Marine Science (VIMS) for the helpful conversations on domain decomposition. Andriy Fedorov and Andriy Kot from the College of William and Mary for their insight on software design and parallel computing. Mr Edward B. Parlete from VIGYAN Inc. and Dr. Craig Hunter from NASA Langley Research Center for all their help with VGRID's binary file formats and sharing their experience in mixed FORTRAN/C programming environment. Dr. Mohagna Pandya, from NASA Langley Research Center for his help in running the USM3D solver and interpretation of the CFD solutions with TECPLOT. Dr. Hang Si from Weistrass Institute for Applied Analysis and Stochastics for making TETGEN, a 3D Delaunay Triangulation code, publicly available and for answering numerous questions about the TETGEN API. Kitware, Sandia National Labs, CSimSoft, Los Alamos National Lab and the Army Research Lab for contributing in the development of PARAVIEW, an open-source and freely available Parallel Visualization Application. The results and figures presented in this thesis would not have been possible otherwise. This research is conducted by the Computer Science Department at the College of William and Mary and NASA Langley Research Center (LaRC) and is partially funded by the NASA GSRP (Graduate Student Research Program) grants NL06AA02H (2006) and NNX07AM10H (2007), the National Science Foundation (NSF) grant CCS-0750901, the John Simon Guggenheim Fellowship and by the National Science Foundation (NSF) through TeraGrid resources provided by the National Center for Supercomputing Application (NCSA) at the university of Illinois at Urbana-Champaign.



# List of Tables

3.1	Breakdown of the Parallel Execution for the Generic Transport DLR-F6 configuration . . . . .	21
3.2	Comparative data of the Sequential and the Parallel Execution Times for the Generic Transport DLR-F6 configuration using 100 sub-domains and 32 CPUs . . . . .	22
3.3	Breakdown of the Parallel Execution Time for the Generic Business Jet configuration . . . . .	23
3.4	Comparative data of the Sequential and the Parallel Execution Times for the Generic Business Jet configuration using 100 sub-domains and 32 CPUs	23
5.1	Timings for the Parallel Domain Decomposition, Parallel Mesh Generation phase and the associated speedup for each experiment on the Uniform Box configuration . . . . .	39
5.2	Timings for the Parallel Domain Decomposition, Parallel Mesh Generation phase and the associated speedup for each experiment on the Sphere-in-box configuration . . . . .	41
5.3	Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phase and the associated speedup for each experiment on the DLR-F6 Wing/Fuselage configuration . . . . .	42
5.4	Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment on the Generic Supersonic Jet configuration . . . . .	45
5.5	Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment of the Energy Efficient Transport configuration . . . . .	49
5.6	Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment of the Energy Efficient Transport configuration . . . . .	49

# List of Figures

1.1	Typical, complex aerodynamic configurations (courtesy of Shahyar Pirzadeh, NASA Langley Research Center) (a) Generic High-Lift Transport (b) C-130 sideways view (c) C-130 behind view with parachute deployment . . . . .	6
1.2	Process diagram of the Sequential Framework for VGRID, (courtesy of Shahyar Pirzadeh, NASA Langley Research Center) . . . . .	8
3.1	Parallel Framework by Discrete Domain Decomposition process diagram	14
3.2	(a) Unsmoothed partition interfaces of the sphere configuration (b) Smoothed partition interfaces after 3iterations of Laplacian Smoothing . .	16
3.3	(a) Sample open pocket (b) Closed pocket after applying CDT . . . . .	17
3.4	(a) Sample partition of the DLR-F6 configuration (b) Volume cut of the final merged mesh (c) Pressure distribution (d) Convergence History of the solution error residual . . . . .	18
3.5	(a) Sample partition of the Generic Business Jet configuration (b) Volume cut of the final merged mesh (c) Pressure distribution (d) Convergence of the residual . . . . .	19
3.6	(a) Comparative timing data of the parallel versus the sequential execution on equivalent size meshes (b) Breakdown of the parallel execution .	21
3.7	(a) Comparative timing data of the parallel versus the sequential execution on equivalent size meshes (b) Breakdown of the parallel execution .	22
4.1	The Octree Decomposition Algorithm . . . . .	26
4.2	(a) Interior Bounding Box around the DLR-F6 wing/fuselage configuration (b) Filtered Oct-tree Region around the DLR-F6 (c) Extracted Interface of the quadrilateral faces (shown in red) between the geometry and the Oct-Tree region (d) Sub-domain patches of the partition consisting of the geometry . . . . .	30
4.3	(a) Sample Patched sub-domain partition (b) Sample Patched sub-domain with discretized curves (c) Sample Triangulated sub-domain (after patch triangulation) (d) Sample meshed sub-domains . . . . .	31
5.1	Parallel Framework Process Diagram . . . . .	33
5.2	(a) Sphere-In-Box decomposition at level 1 (b) Sphere-In-Box decomposition at level 2 . . . . .	35

5.3	(a) Sample partition of the Uniform Box configuration (b) Zoom in at the sub-domain boundaries . . . . .	38
5.4	(a) Plot of the speed-up obtained for 2-64 processors (b) Plot of the size distribution across the 64 Partitions of the Uniform Box configuration . .	39
5.5	(a) Sample partitioning of the Sphere-In-Box configuration (b) Zoomed view (c) Plot of the speed-up obtained for 2-64 processors (d) Plot of the grid size distribution across the 64 partitions . . . . .	40
5.6	Sample partitioning of the DLR-F6 . . . . .	41
5.7	(a) Plot of the speed-up obtained for 2-64 processors (b) Plot of the grid size distribution across the 128 partitions. . . . .	42
5.8	(a) Plot of the speed-up obtained using 256 Partitions and 2-64 processors (b) Plot of the speed-up obtained using 512 Partitions and 2-64 processors	43
5.9	Distribution of the entire process illustrating the computation time and idle time on each CPU at every level of Parallel Domain Decomposition and in the Parallel Mesh Generation phase using 32 sub-domains and 8 CPUs. . . . .	44
5.10	Sample partitioning of the Generic Supersonic Jet . . . . .	45
5.11	(a) Plot of the speed-up obtained for 2-16 processors (b) Plot of the grid size distribution across the 16 partitions of the Generic Supersonic Jet Configuration . . . . .	46
5.12	(a) Load distribution of the Generic Supersonic Jet configuration on 4 CPUs (b) Load distribution of the Generic Supersonic Jet configuration on 16 CPUs . . . . .	47
5.13	(a) Partition of the head of the fuselage and wing (b) Partition of the engine attached to the wing (c) Partition of wing tip (d) Partition of the fuselage tail and the wing . . . . .	53
5.14	Grid size distribution of the EET configuration . . . . .	54
5.15	(a) Plot of the speedup for 2 – 32 processors (b) Plot of the speedup for 2 – 32 processors using the Largest-Partition-First (LPF) scheduling policy	54

PARALLEL UNSTRUCTURED GRID GENERATION FOR  
COMPLEX REAL-WORLD AERODYNAMIC SIMULATIONS

# Chapter 1

## Introduction

The focus of this work is geared towards large-scale Computational Fluid Dynamic (CFD) simulations of complex, real-world aerodynamic problems. There are two primary reasons that justify the need for parallel grid generation. First, the need to generate large grids. For CFD simulations, large, high-resolution grids have become quite common and are required. Yet, generating such grids is challenging due to the memory limitations of a single workstation. Second, the desire to improve the performance of the sequential grid generator for such large problems necessitates the use of parallel grid generation technology. For complex aerodynamic configurations, it can take many hours to generate a desired grid sequentially even on modern computers. In a typical aircraft design/modeling setting, it is reasonable for an engineer to have the need to generate several such grids within a single day. Obviously, with sequential grid generation this is not possible. Parallel grid generation is an enabling technology that can offer engineers the opportunity to perform their job in a more efficient manner and this is the main objective of this research. In particular, the focus of this work is in parallelizing NASA's

Unstructured Grid Generator VGRID[26, 27] which is an integral part of TetrUSS[10] : NASA's Tetrahedral Unstructured Software System.

The primary goals and challenges in parallel unstructured grid generation are: to maintain stability, i.e., retain the quality of the mesh generated by state-of-the-art sequential codes without substantial decrease of the scalability of the parallel code [8]. Generally, the parallel unstructured grid generation problem is broken down into two sub-problems: *Domain Decomposition* (DD) and *Load Balancing* (LB). DD and LB are two related problems whose solution is crucial in parallel unstructured grid generation. DD is the process by which the domain to be meshed  $\Omega$  is partitioned into several smaller sub-domains  $\Omega_i$ . DD is difficult because, in order to satisfy the stability criterion, the size and shape of the separators or interfaces (in between the sub-domains) must be compatible with the local spacing and shape determined by the application. Moreover, the partitioning provided by the DD problem must be such that the load is equally distributed on each sub-domain. In practice, exact estimation of the load on each sub-domain is not feasible. Consequently, load imbalances, between the sub-domains, exist which affect the scalability of the parallel code. Hence, LB, the strategy by which the sub-domains  $\Omega_i$  are assigned to a set of processors  $\mathcal{P}$  such that the load is evenly distributed across all of the processing units, plays an important role in the performance of the parallel unstructured grid generator. Although, the solution for the DD and LB problems of an already meshed domain has been addressed and is routinely applied on parallel finite element/finite volume flow solvers, e.g. using graph partitioner such as METIS [2], partitioning of an 'empty', i.e., unmeshed, domain and proper evaluation of the load in a given sub-domain is non-trivial.

This work presents three different methods for parallel unstructured grid generation. First, a method by Discrete Domain Decomposition (i.e. starting from a coarse grid) is presented [37]. The benefit of this approach is that it allows to solve the hard DD problem by graph partitioning, i.e., using METIS [2], which is typically what finite element/finite volume flow solvers do. However, this approach introduces artifacts at the partition interfaces which are not desirable for solving Partial Differential Equations (PDEs) and is subject to over-refinement. Further details and results produced by this approach are presented in Section 3.

Second, a method using Oct-tree guided DD is presented. For an in depth explanation of the oct-tree data-structure and associated algorithms the interested reader is referred to [33, 34]. Experimental results demonstrated that grids produced by this approach are not suitable in the CFD analysis of aerodynamics configurations. First, the Oct-tree vertices affect the stability of the parallel unstructured grid generator. The Oct-tree interfaces constrain the point distribution of the final mesh as well as the size and shape of the mesh elements. Second, the computational cost and high memory requirement associated with generating and maintaining the Oct-tree affects the scalability of the parallel unstructured grid generator. The details of this approach are discussed in more detail in Section 4.

Lastly, an approach based on the Advancing Partition (AP) DD technique [29, 30] is presented. This approach exhibits many properties that are particularly suited for parallel unstructured grid generation for complex, real-world aerodynamic simulations. Among the benefits are: (1) the method is 'native' to the underlying mesh generator, VGRID, (2) no artifacts or artificial partition interfaces are generated by the AP process

as it functions as part of the volume mesh generation procedure and thus, interfaces are generated in a 'natural' way and (3) the computation cost for domain decomposition is minimal in contrast to other approaches which rely on auxiliary data structures and/or operations. The details of this approach as well as experimental results are presented in Section 5.

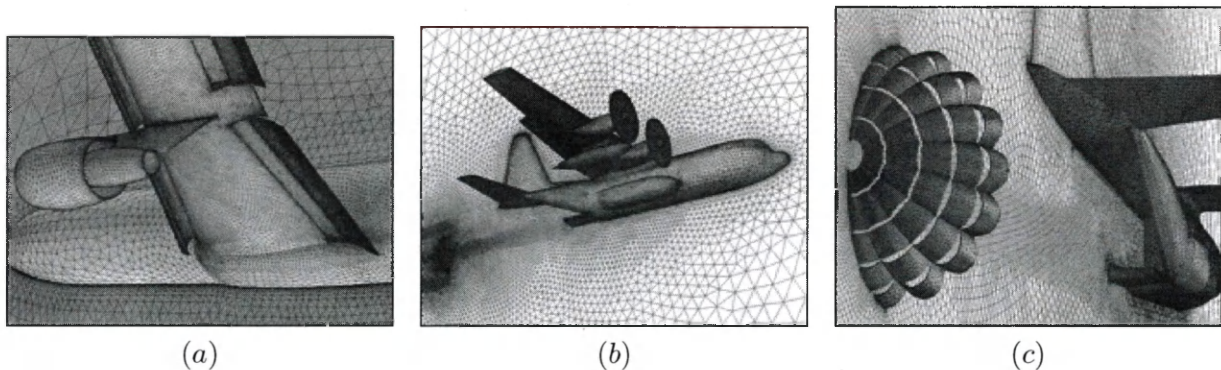
## 1.1 Motivation

CFD is in its nature a multidisciplinary area which encompasses Engineering, Physics, Mathematics, Computer Science and Visualization techniques [23]. The desire to model in more detail complex configurations has led to many recent technological advances in the area of CFD analysis. Notably, the widespread availability of parallel machine architectures, the development of parallel programming libraries such as MPI [12], numerical schemes that can harness the power of parallel architectures and advanced grid generation algorithms for discretizing complex domains are among the most important contributing factors for enabling researchers and engineers to solve complex problems which were intractable a few years ago. In recent years however, the magnitude of the problems increased proportionally with the complexity. While modern flow solvers have been ported to parallel architectures, grid generation remains the serial bottleneck for high performance, large-scale, complex, CFD simulations.

Figure 1.1 illustrates the magnitude and complexity of the problems that arise in the aerodynamics discipline. For such large and challenging problems it takes many hours to generate the desired final grid. The problem is challenging not only because of the magnitude and complexity of the models, but also due to the memory limitations on a



single workstation. On a typical 32-bit workstation the theoretical limit of addressable memory is  $2^{32}$  which translates to 4 GB of memory. Moreover, most modern operating systems reserve 2 GB of address space which further limits the addressable memory to 2 GB. State-of-the-art grid generators, such as VGRID, overcome that physical limit by implementing a grid restart capability, e.g. [26]. The restart capability allows to save the grid when memory is insufficient and restart to mesh the remaining unmeshed region. Grid generators based on the advancing front are particularly suited for such an operation since once an element is created it is not further modified. Also, it should be noted that grid-restarting, although feasible [17], is far more difficult for Delaunay-based schemes.



**Figure 1.1:** Typical, complex aerodynamic configurations (courtesy of Shahyar Pirzadeh, NASA Langley Research Center) (a) Generic High-Lift Transport (b) C-130 sideways view (c) C-130 behind view with parachute deployment

The memory limitation has also been alleviated to a great extent by the deployment of 64-bit architectures and operating systems which are nowadays common even in the non-scientific community. However, problem sizes are expected to increase in both complexity and magnitude and thus serial grid generation still remains a bottleneck.

Furthermore, in the context of adaptive simulations, or time-dependent simulations

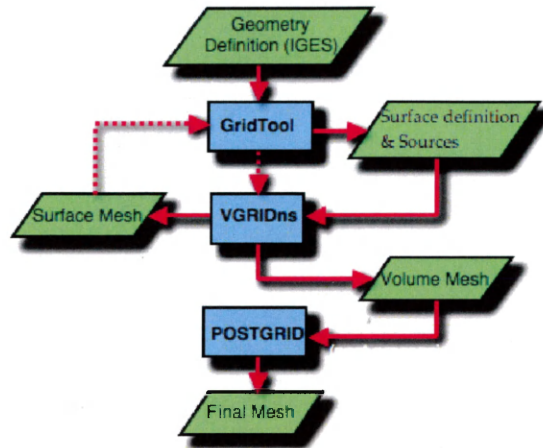
with evolving geometry, re-meshing is an integral part of the simulation. Under this scenario, both the mesher and solver co-exist and are competing for resources (e.g. memory). In this context, a parallel mesher, with distributed memory would yield a more efficient and practical solution.

The main objective of this research is the design and implementation of a framework for parallel unstructured grid generation for routine application in large-scale, complex, aerodynamic CFD simulations.

## 1.2 Overview of VGRID

This section presents an overview of NASA's unstructured grid generation software VGRID developed at NASA Langley Research Center (LaRC). VGRID is based on two marching techniques: (1) The Advancing Front (AF) method [25, 26] for generating Euler (inviscid) grids and (2) The Advancing Layers (AL) method [27] for generating Navier-Stokes (viscous) grids. Figure 1.2 shows a process diagram of the sequential framework for grid generation using VGRID.

The user starts from the geometry specification given in the Initial Graphics Exchange Specification (IGES) format which is widely used within the CFD community. GridTool [32] is used to bridge the gap between CAD and the input to the underlying grid generation system VGRID. This is a user-interactive step by which the user defines the surfaces of the geometry to be meshed and the *source elements* which determine the grid distribution in the field. After the surfaces and source elements are defined, a surface mesh is generated using VGRID. This process may be repeated till the desired surface mesh is obtained. Next, VGRID is invoked to generate the volume mesh. During the



**Figure 1.2:** Process diagram of the Sequential Framework for VGRID, (courtesy of Shahyar Pirzadeh, NASA Langley Research Center)

volume mesh generation procedure, tetrahedral cells are introduced (one-by-one) over the triangular faces on the current front, starting from the surface mesh (initial front), and 'marching' into the interior of the computational domain. This marching process is repeated until no tetrahedron can be formed. After this step most of the computational domain is filled with tetrahedra, however, there still may be some regions left open, called 'pockets', that have not been meshed. Lastly, POSTGRID is invoked to close, or mesh, the remaining pockets by local re-meshing to produce the final grid.

## Chapter 2

# Related Work

Parallel Mesh Generation is not a new problem in the field of applied Computational Fluid Dynamics (CFD). In this section we provide a review of domain decomposition and parallel mesh generation methods that are more closely related to the work presented in this paper. For a more in depth review, the interested reader is referred to a recent survey[8] on parallel mesh generation methods.

Lohner et al. proposed a scheme for Parallel Advancing front[22] that was based on the subdivision of the background grid. This scheme was determined by the authors to be inadequate for a production environment[24] and a new scheme was proposed. The new scheme employs the oct-tree[33, 34] as an auxiliary data structure to parallelize grid generation at each front. A more recent publication[35] indicates the extension of this approach to Reynolds-Average Navier-Stokes (RANS) parallel unstructured grid generation and applicability of this approach in aerodynamic simulations. However, further improvements are necessary for this approach to exhibit better scalability. As the authors note, small imbalances between workloads incur heavy CPU penalty and

decrease in performance[24] .

Cougny and Shephard propose an oct-tree based technique[9] for parallel mesh generation. They build an oct-tree and decompose the entire domain. Octants that are interior to the domain are meshed using template subdivisions and the octants that intersect the domain boundaries are meshed using the advancing front. However, a shortcoming of this approach is that meshing the interface can create difficulties and a re-partitioning strategy is required[9] . Furthermore, the point distribution and thus the size and shape of the mesh elements created by this approach is constrained by the oct-tree vertices and the template subdivisions. Consequently, this approach is not practical for generating anisotropically stretched grids which are desired for aerodynamic simulations.

Several methods based on partitioning a coarse mesh[31, 7, 13, 37] have also been presented. In these methods, a coarse mesh is decomposed into several sub-domains. Next, the boundaries of the sub-domains are refined and then each sub-domain is meshed in parallel using a conventional off-the-shelf mesh generation method. The benefit of this approach is that it circumvents the hard problem of domain decomposition of an empty domain. Instead, the coarse mesh is partitioned using METIS[2] which is a robust and widely used graph partitioning library. However, a short-coming of this approach is that: (1) the method introduces artifacts at the partition interfaces which are not desirable for solving PDEs and (2) this approach is subject to over-refinement. Furthermore, this approach is not suited for RANS grid generation schemes that are essential in aerodynamic simulations nowadays.

A domain decomposition method based on the medial axis[20, 21, 19] was presented

for 2D isotropic grid generation. However, the extension of this approach for 3D geometries is non-trivial primarily due to the computational complexity of the medial axis. Furthermore, in the context of complex, real-world aerodynamic simulations substantial work is required in order to extend this work for (1) anisotropically "stretched" grids and (2) RANS grid generation methods.

Galtier and George[11], present a method of pre-partitioning a surface mesh by triangulating calculated surfaces that intersect the boundaries of the domain. A similar method has also been presented by Ivanov et. al [14] and Larwood et. al [18]. However, the applicability of these methods to anisotropic and RANS grids has not been studied.

## Chapter 3

# Parallel Grid Generation by Discrete Domain Decomposition

This chapter describes the design and implementation of a parallel framework for grid generation that is based on discrete domain decomposition, i.e. starting from a coarse grid. The two fundamental issues by this approach are: (1) grid-termination: the difficulty, in certain cases, of the advancing front (AF) process to complete the grid and (2) grid-quality at the partition interfaces. This work extends and leverages the work described in [7, 13] as an attempt to address the two aforementioned issues. First, smoothing is applied at the partition interfaces in order to improve the quality and second, a constrained Delaunay tetrahedralization is utilized on the remaining regions that the AF process is not able to mesh in order to guarantee termination.

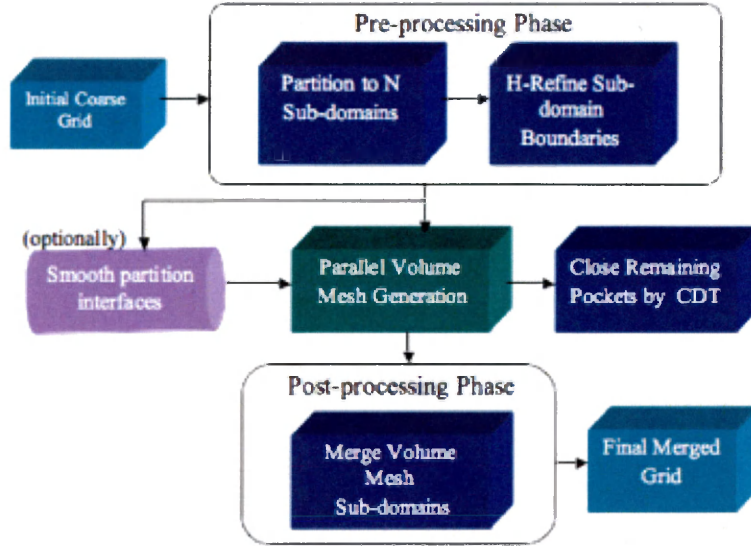
The parallel framework can be viewed as an integrated, pipelined system of modules. Modules were written using the C++ and Fortran90 programming languages and MPI is used for parallel programming. Figure 3.1 shows a process diagram of the framework.

The basic pipeline of execution consists of the following steps:

1. Generate an initial coarse grid.
2. Partition the grid into  $N$  sub-domains.
3. H-refine the sub-domain boundaries at a user-supplied number of iterations (sequentially).
4. Optionally, smooth the partition interfaces in order to fix any odd-shape partitions, remove any artifacts and improve the quality of the faces at the interface.
5. Generate a volume mesh on each sub-domain (in parallel)
6. If there are any regions that the AF process was not able to mesh, close those regions by Constrained Delaunay Tetrahedralization (CDT)
7. Merge the volume mesh sub-domains and produce the final merged grid

As it is shown in Figure 3.1 the framework consists of three phases: (1) The pre-processing phase, (2) the volume meshing phase, and (3) the post-processing phase. The pre-processing phase consists of two main tasks. First, the initial coarse mesh is partitioned into  $N$  sub-domains using METIS [2]. METIS may yield a non-contiguous set of partitions. In order to ensure a contiguous set of partitions, the partitioner assigns new partition IDs to any non-contiguous partition and increments the partition counter. Furthermore, the partitioner extracts only the boundary faces of each sub-domain and outputs a set of sub-domains with void interior. Second, the boundary faces of the sub-domains are refined by h-refinement to the desired level of resolution. In the meshing phase, the partition interfaces may be optionally smoothed in order





**Figure 3.1:** Parallel Framework by Discrete Domain Decomposition process diagram

to fix any odd-shape partitions. Then a volume mesh is produced in parallel on each sub-domain. After the volume mesh generation phase there may be some regions that the AF process was not able to mesh. These regions are identified and are closed by a Constrained Delaunay Tetrahedralization (CDT) method to ensure termination. Smoothing of partition interfaces and applying CDT are discussed later in further detail. Lastly, in the post-processing phase, all the volume sub-domains are merged and a final merged mesh is produced.

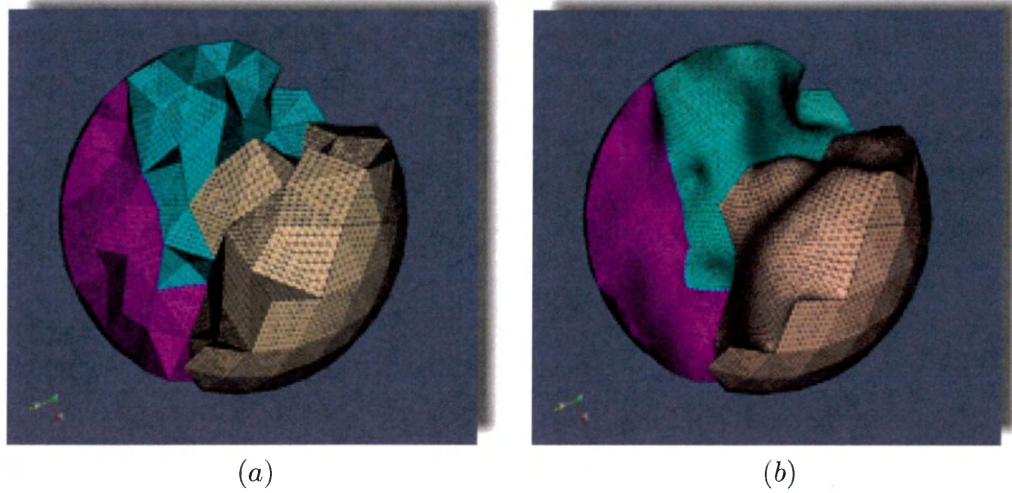
Before proceeding into the further details of the framework it is worth mentioning a couple of technical remarks of the discrete domain decomposition framework. First, post-processing, i.e., merging is performed sequentially and, second, all data communication between modules is done by I/O, i.e., writing and reading to the file-system. As discussed in the Results and Discussion section, merging of the sub-domains and the I/O overheads are the two most dominant tasks of the computation and the primary reasons for

not achieving good parallel performance.

### 3.1 Fixing Odd-shape Partitions by Smoothing

The main problem after decomposing the coarse mesh is that the resulting partitioning may yield odd-shape partitions. Figure 3.2(a) shows a typical sample partitioning of the sphere. As it is illustrated in the figure, such odd-shape partitions, i.e., with sharp corners, low quality faces at the interface and other artifacts may yield a very difficult problem for the AF process and may cause problems in meshing. Partially, the difficulty in meshing is introduced due to the artificial way of refining the boundaries of the sub-domains by h-refinement. H-refinement, deteriorates the quality of the surface mesh and may yield a triangulation that is not compatible with the local spacing and shape defined by the background source elements. In an attempt to fix any odd-shape partitions prior to meshing, Laplacian Smoothing is applied at the partition interfaces to improve the quality of the faces, remove sharp corners and artifacts. This in turn facilitates the AF process by providing a more well-suited input to the mesher. For the particular sphere model shown in Figure 3.2(b) 3 iterations of Laplacian Smoothing have worked really well and the AF process was able to mesh each partition without any problems.

Although Laplacian Smoothing works for simple configurations, such as the sphere, it is not robust for more complex geometries. In the literature it is well known that surface mesh optimization or surface mesh smoothing is a non-trivial problem in 3D due to the geometric constraints that need to be enforced. For example, checks are required to determine if faces are inverted or if vertices are moved outside the domain boundaries etc. Hence, a smoother that implements such constraints, e.g. [15, 16], would yield a



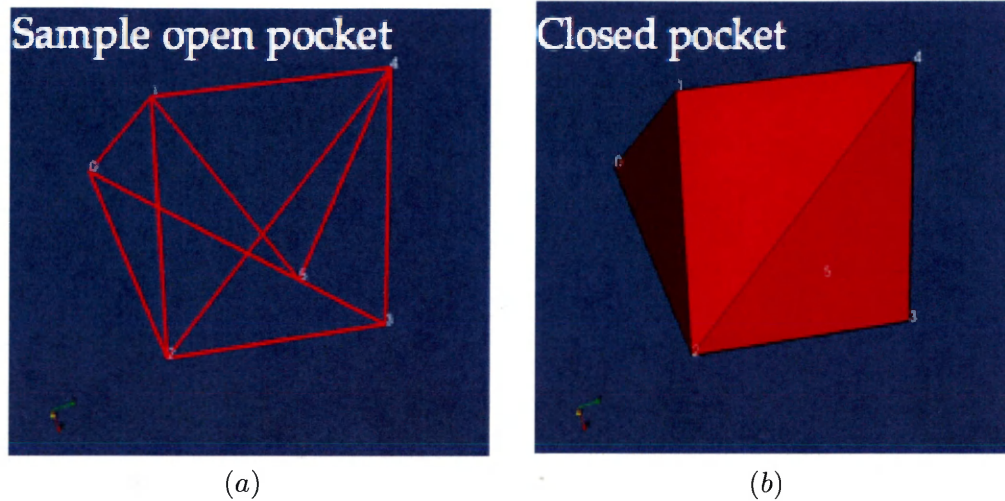
**Figure 3.2:** (a) Unsmoothed partition interfaces of the sphere configuration (b) Smoothed partition interfaces after 3 iterations of Laplacian Smoothing

more robust solution.

### 3.2 Closing the Grid by Constrained Delaunay Tetrahedralization

A subtle issue with parallel unstructured grid generation by is that some partitions may be too difficult to mesh by the AF process. After the volume mesh generation phase the regions that the AF process was not able to mesh or close, called pockets, are identified. Figure 3.3(a) shows a sample persistent pocket that the AF process was not able to mesh. In this work, a Constrained Delaunay Tetrahedralization (CDT) method [36] is applied to mesh these pockets. The Constrained Delaunay Tetrahedralization algorithm used in this approach was implemented in TetGen [6], a robust, open-source 3D Delaunay Triangulator. The set of vertices and face connectivity of the pocket makes up the input to the CDT method. Given that information the CDT method creates tetrahedral

elements in the empty region and closes the pockets as illustrated in Figure 3.3(b).

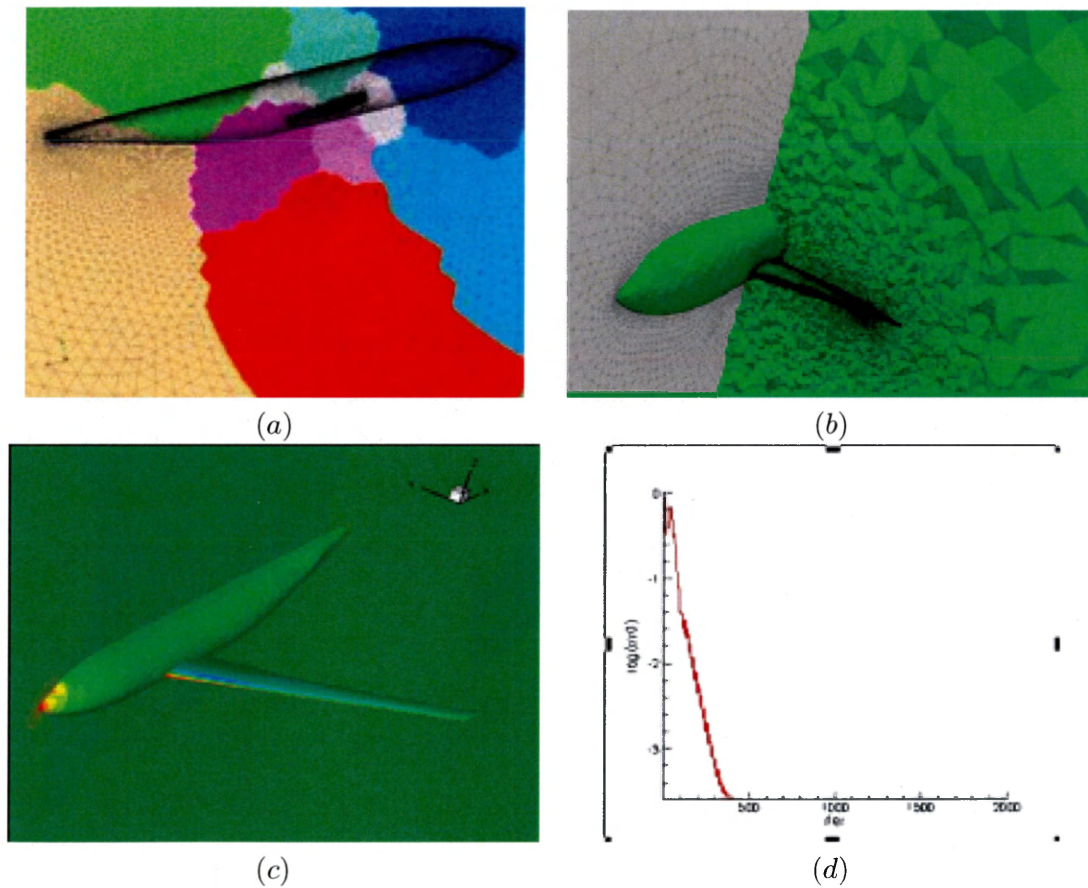


**Figure 3.3:** (a) Sample open pocket (b) Closed pocket after applying CDT

The CDT method may insert vertices in the interior of the pocket but it preserves the pocket boundaries. Thus, a conforming mesh is guaranteed after the pocket is closed and no re-meshing is necessary. Although, the CDT approach is simple, robust and practical, there are no theoretical guarantees about grid quality. If the pocket is very tight CDT may create slivers and thus the final grid may have to be post-processed to improve the quality. Another alternative is to allow the insertion of points on the pocket boundaries. This would facilitate the Delaunay algorithm to provide an optimal quality tetrahedralization. However, insertion of points on the boundary of the pocket would yield a non-conforming mesh and would require re-meshing which would propagate into the interior of the domain. Furthermore, re-meshing especially in the context of parallel grid generation, can become a very complicated procedure.

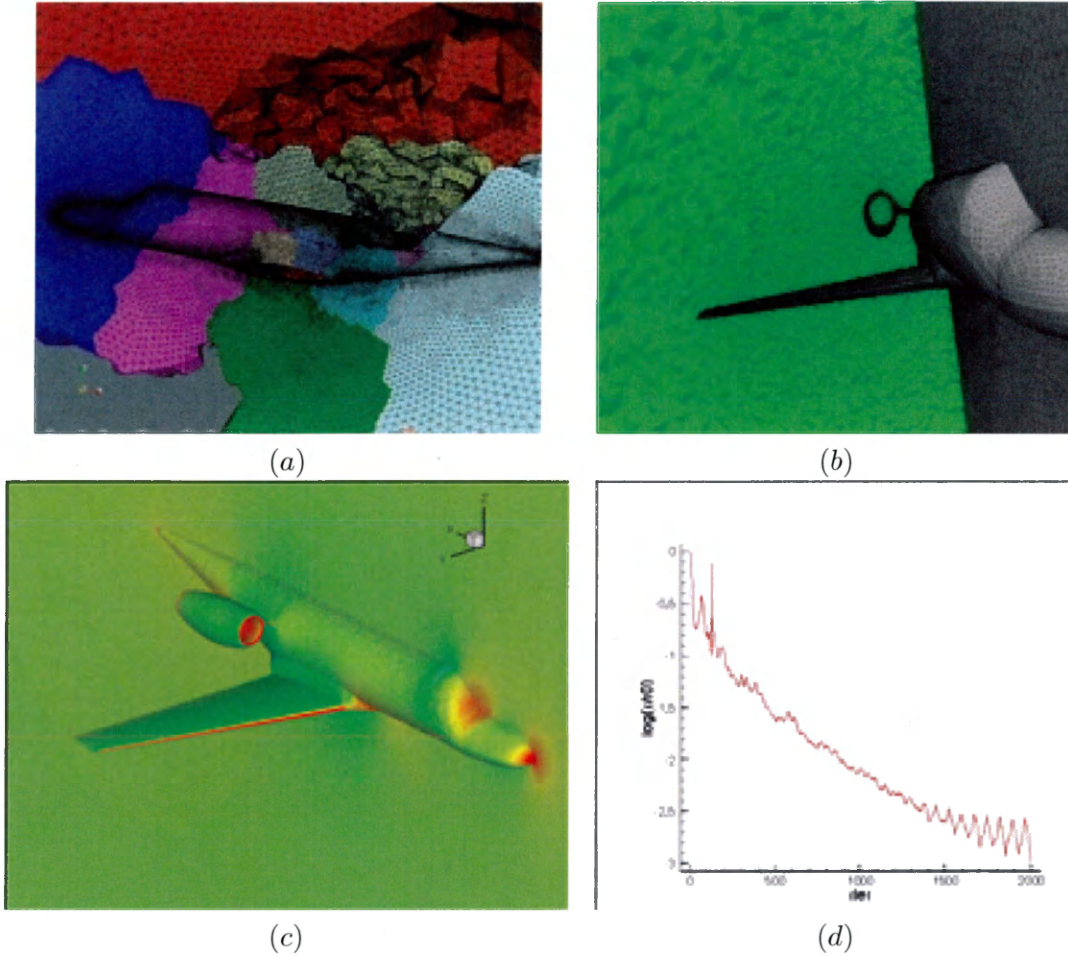
### 3.3 Results

This section presents numerical performance results of the current implementation of the framework starting from a coarse mesh. The experiments were conducted using a Generic Business Jet configuration and a Generic Transport DLR-F6 wing/fuselage configuration shown in Figures 3.5 and 3.4 respectively.



**Figure 3.4:** (a) Sample partition of the DLR-F6 configuration (b) Volume cut of the final merged mesh (c) Pressure distribution (d) Convergence History of the solution error residual

For each configuration a set of meshes, varying in size, are generated. Additionally, equivalent size meshes are generated sequentially. This preliminary performance eval-



**Figure 3.5:** (a) Sample partition of the Generic Business Jet configuration (b) Volume cut of the final merged mesh (c) Pressure distribution (d) Convergence of the residual

uation provides two sets of data: (1) comparative timings of the online performance of the parallel framework data (excluding the I/O overheads and the time to merge the sub-domains) with the sequential grid generator on equivalent size grids and (2) a breakdown of the execution times of each stage of the parallel execution. The speedup of the parallel code is given by  $\frac{T_s}{T_p}$ , where  $T_s$  is the time for sequential mesh generation and,  $T_p$  is the total time of the parallel code, including domain decomposition, parallel mesh generation and merging.

The underlying architecture that used in these experiments consisted of a 6-node Linux cluster. The exact hardware specifications of each of the nodes are the following:

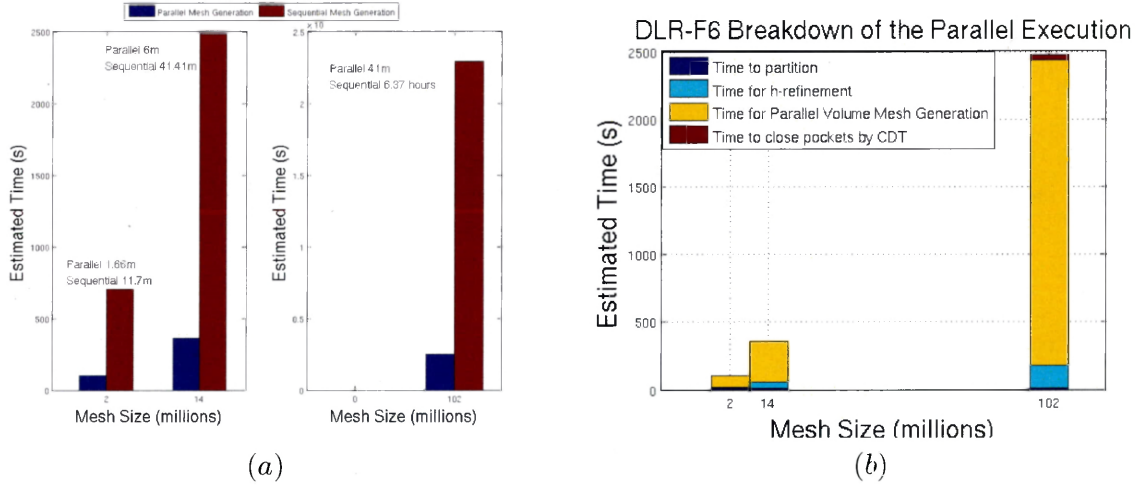
- 3 (nodes)  $\times$  8 CPUs total (in each core): Intel Xeon @3.73GHz, 4 hyper-threaded cores, 8GB of physical RAM in each core.
- 1 (node), 4 CPUs: Intel Xeon @3.6GHz, 2 hyper-threaded cores, 4GB of physical RAM.
- 2 (nodes)  $\times$  2 CPUs total (in each core): Intel Xeon @3.6GHz, 1 hyper-threaded core, 1GB of physical RAM in each core.

Under these specifications, the resulting configuration consisted of a total of 32 CPUS and a total of 13 GB of physical RAM.

For the parallel execution a static, block-cyclic distribution of sub-domains to CPUS was applied. First, the estimated work-load of sub-domain  $i$ ,  $\mathcal{L}_i$ , is computed by the following formula:  $\mathcal{L}_i = \frac{\mathcal{V}_i \mathcal{N}_{f_i}}{\mathcal{S}_i}$  where  $\mathcal{V}_i$  is the volume,  $\mathcal{N}_{f_i}$  is the number of faces, and  $\mathcal{S}_i$  is the surface area of the sub-domain. Next, the sub-domains are sorted based on the value of the estimated work-load  $\mathcal{L}_i$  and statically assigned to the CPUs based on array-block-cyclic assignment. In general, under the aforementioned guidelines, the set of sub-domains,  $\mathcal{M}_i$ , assigned to CPU  $P_i$  is rigorously defined as follows:  $M_i = \{\bigcup(P_i + jN) \ni: (P_i + jN) \leq N\}$  where  $N$  is the number of sub-domains and  $j \geq 0$ , a positive integer.

### 3.3.1 A Wing/Fuselage DLR-F6 Transport Configuration

The experiments using the DLR-F6 generic transport wing/fuselage configuration, shown in Figure 3.4, start from partitioning an initial coarse grid of approximately 290 thousand tetrahedral elements into a 100 sub-domains. Grids of 3 different sizes are generated by the parallel code. In addition, equivalent size grids are generated sequentially. The corresponding execution times of the sequential and parallel runs are compared. The results are summarized in Figure 3.6 and Tables 3.1, 3.2.



**Figure 3.6:** (a) Comparative timing data of the parallel versus the sequential execution on equivalent size meshes (b) Breakdown of the parallel execution

Mesh Size	Partition	Refinement	Volume Mesh	CDT
2M	9.12s	13.2997s	77.7356s	0s
14M	9.12s	44.2073s	305.5s	1.63s
102M	9.12s	166.36s	2260.6s	41.1716s

**Table 3.1:** Breakdown of the Parallel Execution for the Generic Transport DLR-F6 configuration

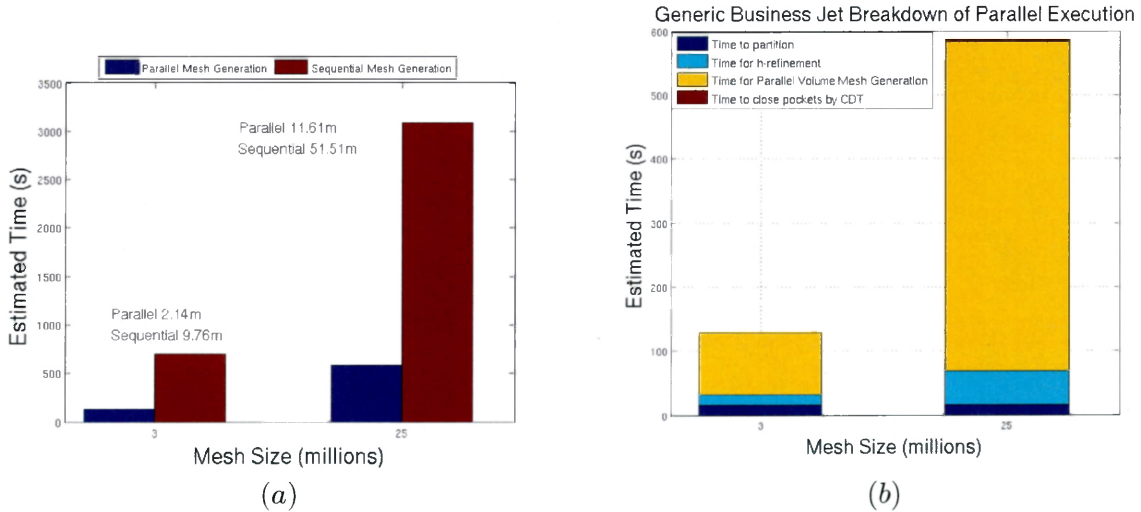


Mesh Size	Parallel Execution Time	Sequential Execution Time	Speedup
2M	100.1553s	704.404s	7.03
14M	360.4573s	2484.719s	6.89
102M	2477.2516s	22923.637s	9.25

**Table 3.2:** Comparative data of the Sequential and the Parallel Execution Times for the Generic Transport DLR-F6 configuration using 100 sub-domains and 32 CPUs

### 3.3.2 A Generic Business Jet

Similarly, the experiments using the Generic Business Jet configuration, shown in figure 3.5, start from partitioning an initial coarse grid of approximately 430 thousand tetrahedral elements into a 100 sub-domains. Grids of 2 different sizes are generated by the parallel code and equivalent size grids are generated also sequentially. The corresponding execution times of the sequential and parallel runs are compared. The results are summarized in Figure 3.7 and Tables 3.3, 3.4.



**Figure 3.7:** (a) Comparative timing data of the parallel versus the sequential execution on equivalent size meshes (b) Breakdown of the parallel execution

Mesh Size	Partition	Refinement	Volume Mesh	CDT
3M	16.1266s	16.0598s	96.4348s	0s
25M	16.1266s	52.3522s	515.978s	1.71967s

**Table 3.3:** Breakdown of the Parallel Execution Time for the Generic Business Jet configuration

Mesh Size	Parallel Execution Time	Sequential Execution Time	Speedup
3M	128.6212s	696.977s	5.42
25M	586.17647s	3090.931s	5.27

**Table 3.4:** Comparative data of the Sequential and the Parallel Execution Times for the Generic Business Jet configuration using 100 sub-domains and 32 CPUs

### 3.4 Discussion

The benefit of this approach is that it solves the difficult DD problem using robust graph partitioning libraries such as METIS. Consequently, the problem of Parallel Unstructured Grid Generation is substantially simplified which enables a straightforward design and implementation. However, there are many inherent properties of this approach that complicate its routine application in a real-world, production environment. Among the top reasons for this are:

1. The coarse grid requirement. The quality of the final grid is related to coarseness of the initial grid. In particular, the initial grid needs to be coarse enough such that the smallest features of the domain are sufficiently resolved. Knowledge of the coarseness level of the initial grid assumes the experience of the user with the underlying grid generator VGRID.
2. This approach is not applicable for the parallelization of RANS grid generation methods which are, nowadays, essential in computational fluid aerodynamics. Im-

portantly, for RANS grid generation methods the direction of the interfaces must be compatible. Generating compatible interfaces by partitioning a coarse mesh is difficult to achieve.

3. In the present implementation, the artificial way of refining the sub-domain boundaries, by h-refinement, yields an over-refined final grid. Consequently, a larger problem is given to the flow-solver.

In practice, this approach appears to be more suitable in the context of Adaptive Refinement where a coarse mesh is already defined, but, it is not suitable for the Parallel Mesh Generation of an 'empty' domain.

## Chapter 4

# Parallel Grid Generation by Octree-guided Domain Decomposition

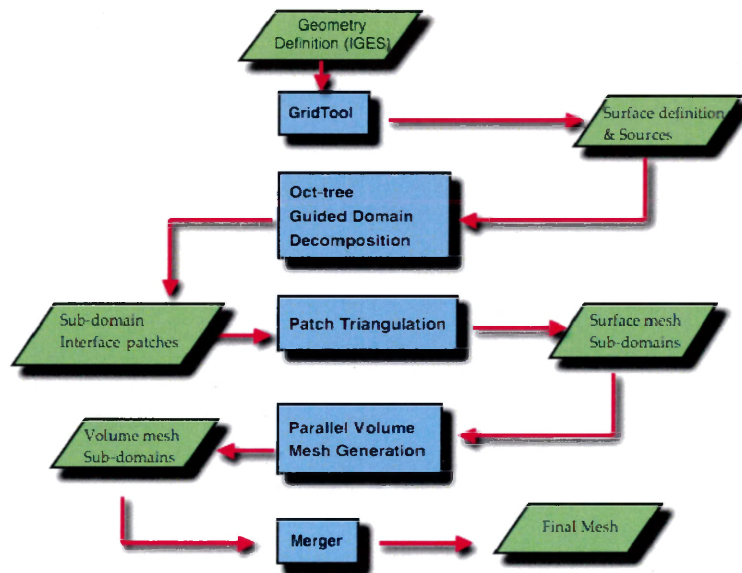
The primary motivation for the Octree approach is to devise a mechanism for creating the interfaces to partition an empty domain  $\Omega$ . Moreover, since in external flow CFD simulations, the domain typically consists of an outer boundary often in the form of a bounding box, the oct-tree data-structure appears to be a natural selection for domain decomposition. For a thorough review on the octree data-structure, associated algorithms and its applications the interested reader is referred to [33, 34].

Figure 4.1 illustrates the basic steps of this approach which are outlined below:

1. As in the sequential framework, GridTool[32] is used to define the surface patches and sources of the domain.
2. An Oct-tree is generated which partitions the domain. The quadrilateral faces of

the oct-tree shared between partitions define the sub-domain interface patches.

3. The patches are triangulated defining the set of sub-domains. This is achieved in two steps. First, the patch curves are discretized and then the patches are triangulated.
4. The sub-domains are meshed in parallel
5. The meshed sub-domains are merged to produce the final mesh.



**Figure 4.1:** The Octree Decomposition Algorithm

The following sections present in more detail the octree decomposition algorithm and preliminary results obtained by this approach.

## 4.1 The Oct-tree Guided Domain Decomposition

This section discusses in more detail the basic steps in the Oct-tree guided domain decomposition that are enumerated below:

1. Octree Subdivision
2. Sub-domain Patch Extraction

### 4.1.1 Octree Subdivision

The oct-tree guided decomposition algorithm starts from the bounding box of the domain as the root octant and performs a balanced 2:1 refinement. A balanced 2:1 refinement requires that each octant can have at most four face-adjacent neighbors and such a balancing strategy has proven useful for providing a smooth gradation in the field of the computational domain. The gradation of the octree is controlled by the background source elements that are specified by the user using GridTool[32] and the source interpolation module of VGRID. Notably, for domain decomposition only a coarse oct-tree is required since not all the features need to be resolved. The user inputs an additional spacing parameter, called  $\Delta\mathcal{X}$  to control the coarsening factor for the octree, i.e. the number of times the final octree will be coarser relative to the final grid spacing determined by the background source elements.

### 4.1.2 Patch Sub-Domain Extraction

An interior bounding box surrounding the surface patches of the geometry inside the outer boundary of the computational domain is computed, e.g. see Figure 4.2(a). The

octants within the interior bounding box are filtered out and a gap separating the geometry and the region of the computational domain covered by the Oct-tree is created, e.g., see Figure 4.2(b). The quadrilateral faces between the gap and the Oct-tree are extracted, e.g., see Figure 4.2(c). The extracted Quadrilateral faces and the surface patches of the geometry compose one of the sub-domains, e.g., see 4.2(d). The interior octants are partitioned using either METIS or Recursive Coordinate Bisection (RCB) and the boundary quadrilateral faces of each partition are extracted. Additionally, at this stage of the algorithm the following data-structures are maintained:

- a global-to-local mapping of the partition boundaries for each sub-domain is generated in order to maintain a coherent global representation of the sub-domains.
- Each face is oriented (according to the right-hand-rule) with respect to each of the sub-domains. VGRID requires that face (patch) normals are pointing inwards to the computational domain.

After this step, each sub-domain is defined by a set of boundary quadrilateral faces.

## 4.2 Mesh Generation

Mesh generation proceeds in two steps. First, the quadrilateral partition boundary faces are converted into a set of VGRID planar patches, e.g. see Figure 4.3(a), and triangulated using VGRID. VGRID triangulates patches by first discretizing the patch boundary curves, e.g. see Figure 4.3(b), and then triangulating the interior of each patch by the conventional 2D Advancing Front method, e.g. see Figure 4.3(c). Second, each triangulated patch is oriented and stored separately according to the given sub-domain.

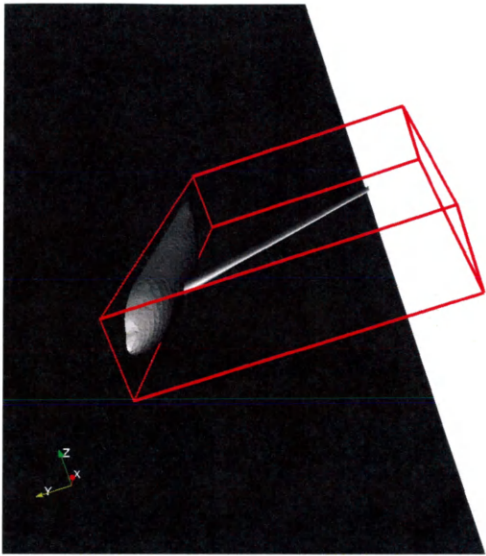
Lastly, the sub-domains are meshed in parallel using VGRID, e.g. see Figure 4.3(d). Additionally, the global-to-local mapping is maintained in order to maintain a coherent representation of the final distributed mesh.

### 4.3 Discussion

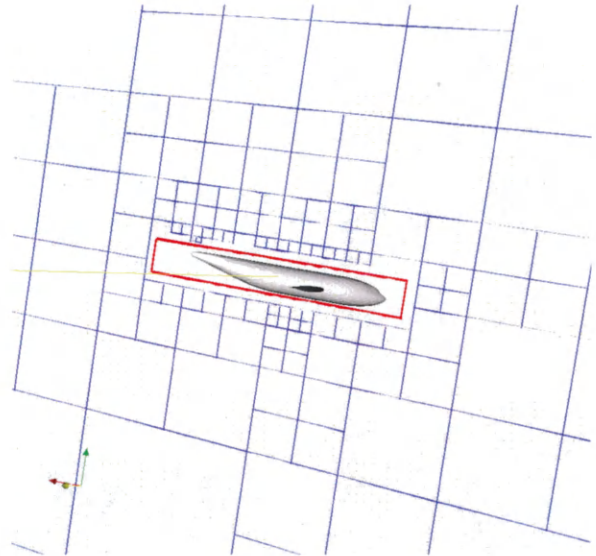
The primary motivation and intention for the oct-tree guided approach is the implementation of a generalized domain decomposer which may be used with different mesh generators, e.g. VGRID [27], TETGEN [6]. A byproduct of this approach is the implementation of a general Octree library which can be used and extended in the development of other algorithms. The Octree Library is an integral part of the Parallel Unstructured Grid Generation Framework [1].

Preliminary results demonstrated that grids produced by this approach are not suitable in the CFD analysis of aerodynamics configurations. First, the Oct-tree vertices constrain the point distribution of the final grid as well as the size and shape of the mesh elements. This is particularly undesirable in CFD analysis and especially for the generation of anisotropic grids. Second, the computational cost and high memory requirement associated with generating and maintaining the Oct-tree defeats the purpose of parallel unstructured grid generation. Additionally, other technical difficulties, such as the automatic patching of octree interfaces, need for a more accurate calculation of the gap between the surface geometry and the octree partitions were among the top reasons for abandoning this approach.

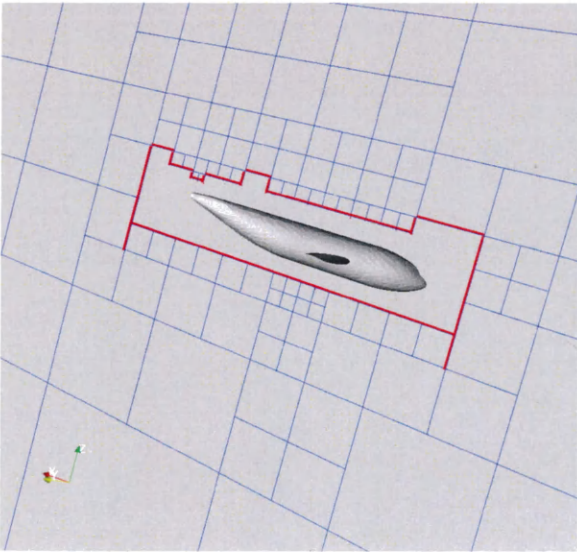




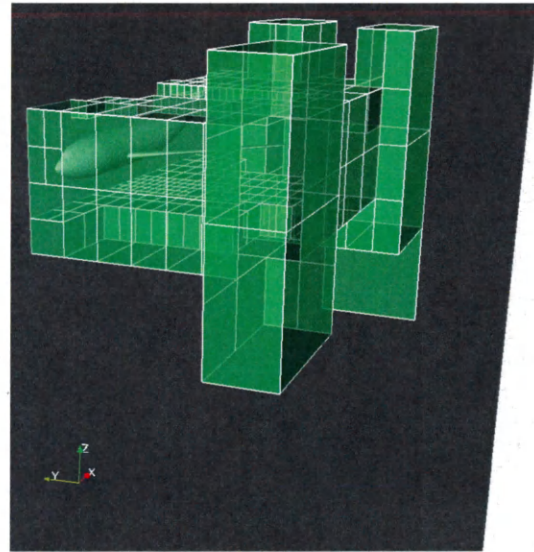
(a)



(b)

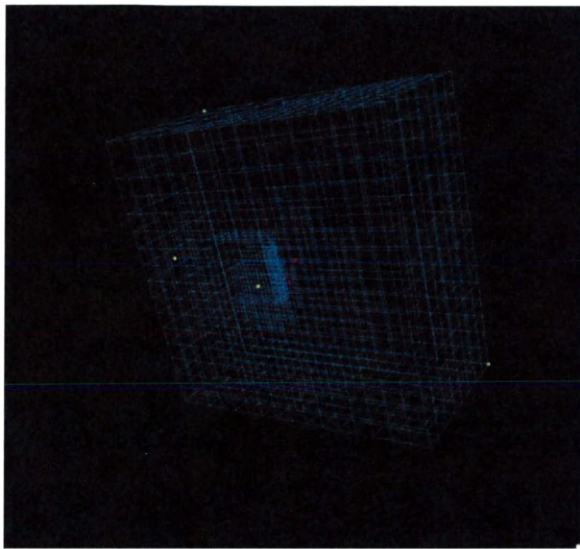


(c)

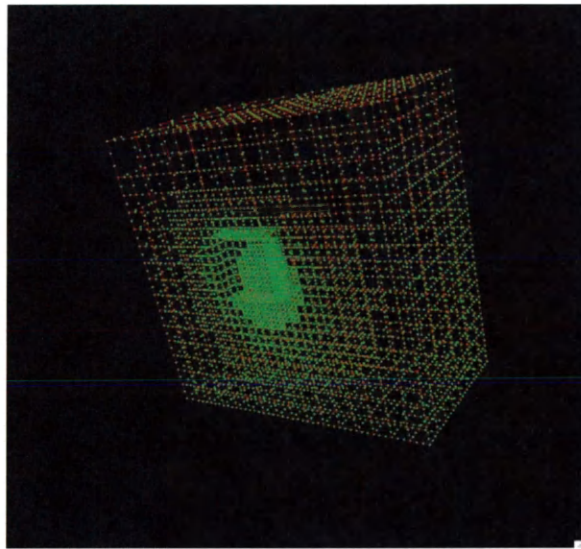


(d)

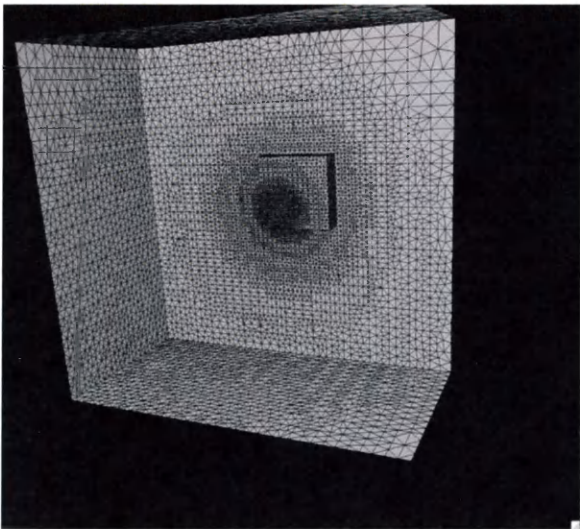
**Figure 4.2:** (a) Interior Bounding Box around the DLR-F6 wing/fuselage configuration (b) Filtered Oct-tree Region around the DLR-F6 (c) Extracted Interface of the quadrilateral faces (shown in red) between the geometry and the Oct-Tree region (d) Sub-domain patches of the partition consisting of the geometry



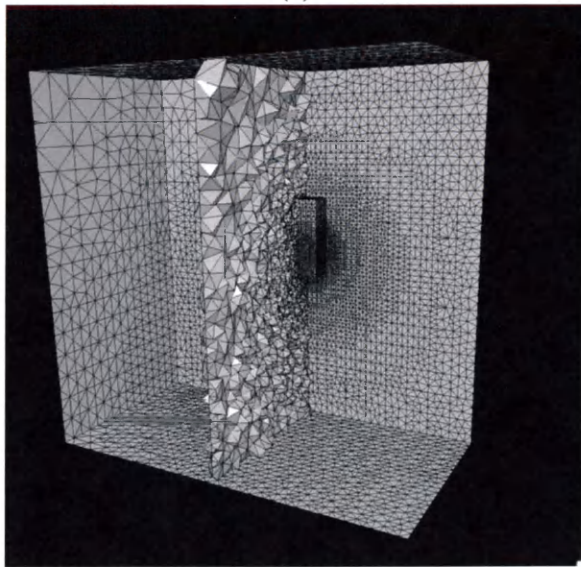
(a)



(b)



(c)



(d)

**Figure 4.3:** (a) Sample Patched sub-domain partition (b) Sample Patched sub-domain with discretized curves (c) Sample Triangulated sub-domain (after patch triangulation) (d) Sample meshed sub-domains

## Chapter 5

# Parallel Grid Generation by the Advancing Partition Technique

This chapter presents a framework for Parallel Unstructured Grid Generation [38] by an Advancing-Partition (AP) [29, 30] technique. The parallel framework consists of two phases: (a) the parallel domain decomposition phase and (b) the parallel mesh generation phase. The AP technique, developed at NASA Langley Research Center, is employed for the parallel domain decomposition phase. In both phases, the Master-Worker model is employed to mitigate any load imbalances. The framework can be viewed as an integrated, pipelined, system of modules. These modules were written using the C++ and Fortran90 programming languages and MPI [12] was used for parallel programming. Figure 5.1 shows a process diagram of the framework.

The user starts by generating the surface mesh of the domain, as in the sequential framework, using VGRID. Next, in the parallel domain decomposition phase the computational domain described by a surface mesh is decomposed in  $2^l$  sub-domains, where

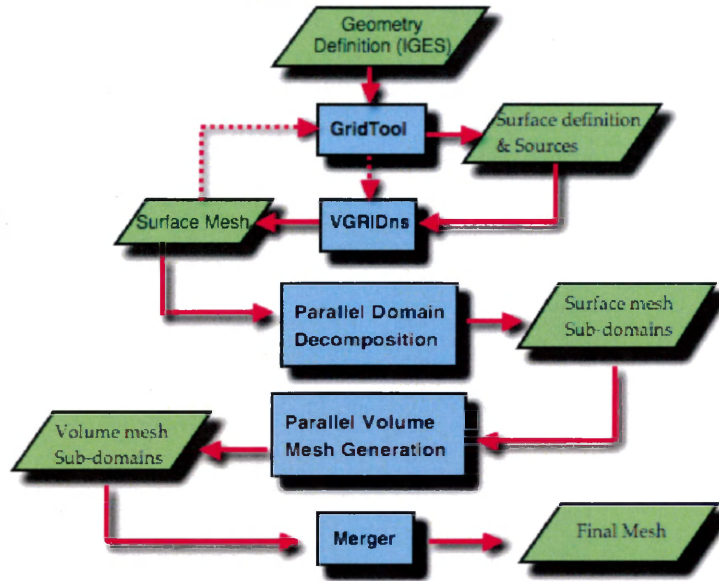


Figure 5.1: Parallel Framework Process Diagram

$l$  is the desired level of subdivisions prescribed by the user. After the sub-domains are generated, they are meshed in the parallel mesh generation phase. In both phases, the Master-Worker model is employed for parallelization. The Master process starts by setting up a work-queue of tasks. There are two task types: (a) Domain Decomposition task and (b) Mesh Generation task. First, the tasks are scattered to the Workers. As soon as a Worker completes a task, it sends an acknowledgment to the Master and is assigned a new task. This process is repeated until all tasks are completed and the work-queue is empty. Finally, all the sub-domains are re-numbered and merged to produce the final mesh. The Parallel Domain Decomposition algorithm (PDD) and the Parallel Mesh Generation algorithm (PMG) are discussed in more detail in the following sections.

## 5.1 Parallel Domain Decomposition by the Advancing Partition Technique

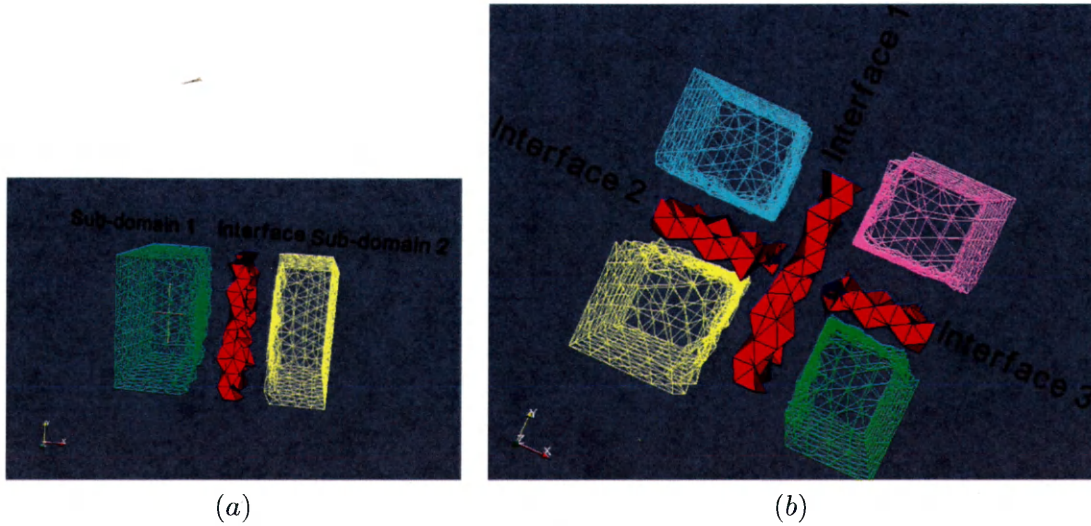
The algorithm utilizes the Advancing Partition (AP) technique implemented within VGRID. The AP method partitions a domain (parent), given in the form of a boundary triangulation, into two sub-domains (children). First, a Cartesian plane is computed at the Center-of-Mass density to logically define the location of the interface in space. Then, a modified advancing front technique is used to march a zone of tetrahedral elements along the Cartesian plane. The zone of tetrahedral elements constitutes the interface or separator between the two logical sub-domains. Lastly, the sub-domains are created by extracting and locally renumbering the triangular faces from the interface and the parent. The PDD algorithm applies the AP technique in a recursive manner and maintains a coherent global mesh representation across all the sub-domains. The recursive nature of the algorithm is particularly suited for the parallel implementation.

The PDD algorithm accepts as input the surface mesh of the domain to be decomposed and the desired level of subdivisions  $L_{max}$  prescribed by the user. At each level  $l$ , there are  $2^l$  sub-domains, defined by a boundary triangulation, and  $2^l - 1$  interfaces defined by a zone of tetrahedral cells. Figures 5.2(a) and 5.2(b) demonstrate the basic idea of the parallel domain decomposition algorithm using a simple box configuration.

The PDD algorithm consists of three primary functions:

1. The DECOMPOSE function applies the AP technique to a domain or sub-domain.

The AP technique generates an interface at the Center-of-Mass density, defined logically by an "imaginary" Cartesian plane, and generates the two sub-domains.



**Figure 5.2:** (a) Sphere-In-Box decomposition at level 1 (b) Sphere-In-Box decomposition at level 2

2. The MASTER DOMAIN DECOMPOSITION process is responsible for the following three tasks:
  - (a) Directing the initial domain decomposition stage.
  - (b) Maintaining a consistent global numbering of the sub-domains with respect to the global mesh.
  - (c) Communicating with the workers, i.e. sending work and receiving acknowledgments.
  
3. The WORKER DOMAIN DECOMPOSITION process performs the following three tasks:
  - (a) Receive a domain decomposition job from the master.
  - (b) Perform the domain decomposition job.
  - (c) Send acknowledgment and request another job.

## 5.2 Parallel Mesh Generation

The algorithm utilizes the Advancing Front (AF) technique for volume mesh generation which is implemented within VGRID[27, 26]. The AF technique starts from a surface mesh definition, which describes the initial front for the algorithm. Then, the algorithm inserts points and creates tetrahedra growing inwards to the domain. The spacing and stretching of the tetrahedra is controlled by the sizing function defined by the background source elements prescribed by the user. The process is repeated till the entire domain is filled or meshed.

Given the set of sub-domains generated by the PDD algorithm described earlier, the PMG algorithm meshes the sub-domains in parallel by applying the AF technique. The AF technique is particularly suited for parallelization since the initial front is not changed and new points are introduced only in the interior of the domain. In the context of the present parallel framework, this allows to mesh the sub-domains without the need for any communication since, for each sub-domain, the sub-domain boundary faces are not changed.

The PMG algorithm is implemented by the following three functions:

- The MESHSUBDOMAIN function calls VGRID to mesh the given sub-domain
- The MASTER MESH GENERATION function is responsible for coordinating submission of mesh generation tasks to the workers.
- The WORKER MESH GENERATION function is responsible for performing the mesh generation task and sending an acknowledgment to the Master upon completion.

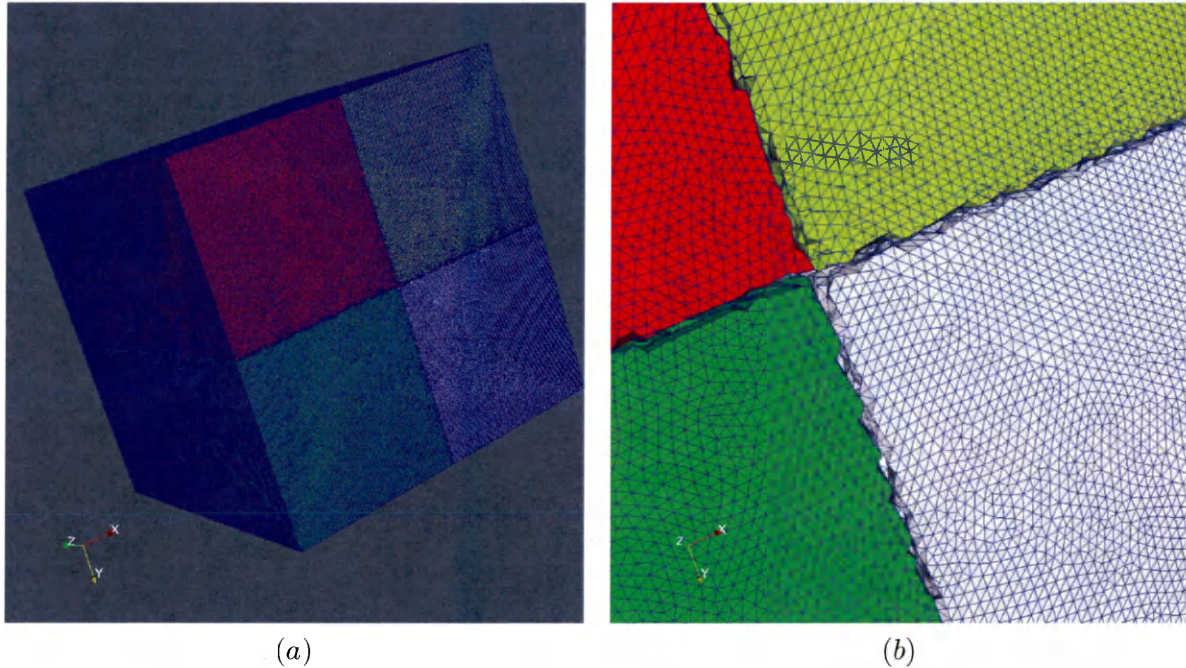
## 5.3 Results

This section presents a performance evaluation of the present parallel framework. The Mercury cluster on TeraGrid [5], supported by the NCSA[3] at the University of Illinois at Urbana-Champaign, was chosen as the target platform for this evaluation. Mercury is an IBM IA-64 Linux cluster equipped with 887 IBM dual-core Itanium 2 (1.3/1.5GHz) processors with 4GB or 12GB of memory per node. Five sample configurations were used for the performance evaluation: (1) a Uniform box, (2) a Sphere-in-box configuration, (3) a transport wing/fuselage (DLR-F6) configuration, (4) a generic Supersonic jet (SLE) configuration, and (5) an Energy Efficient Transport (EET) configuration. Each of the abovementioned configurations adds an additional layer of complexity offering further insight in the evaluation of this approach which assisted in further improving the implementation and the sophistication of the experiments. The speedup, denoted by  $\mathcal{S}$ , for the parallel domain decomposition phase, parallel mesh generation phase and the overall process is computed by the formula  $\mathcal{S} = \frac{T_s}{T_p}$  where  $T_s$  is the sequential execution time and  $T_p$  is the parallel execution time of the domain decomposition phase, mesh generation phase and overall process respectively.

### 5.3.1 A Uniform Box Configuration

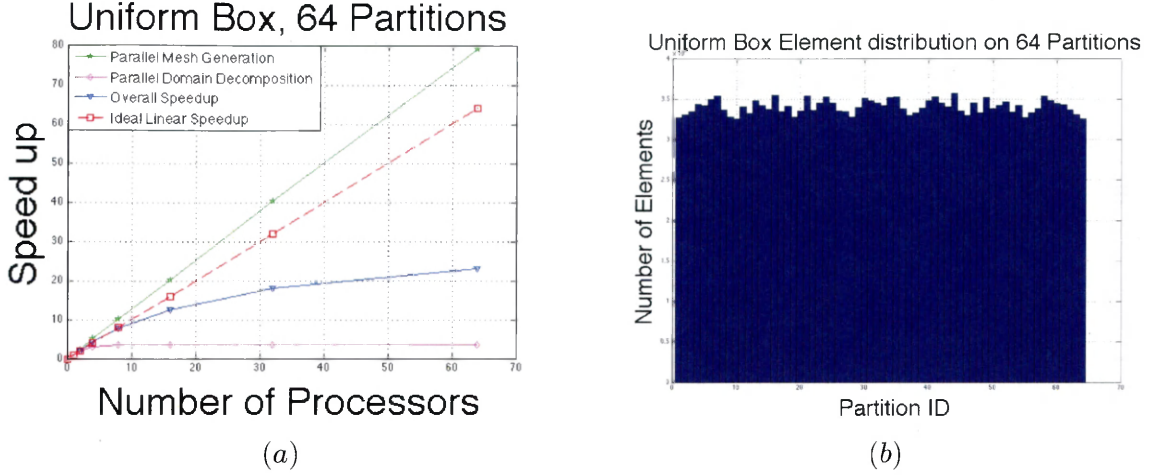
The uniform box configuration is a simple and useful example for understanding how the method works in an idealistic scenario where the geometry is simple and the mesh is uniform, so there are no work-load imbalances. Figure 5.3 illustrates the sample partitioning for this configuration and Figure 5.4 and Table 5.1 summarize the performance results of the method.





**Figure 5.3:** (a) Sample partition of the Uniform Box configuration (b) Zoom in at the sub-domain boundaries

A 23 million cell mesh was generated for this configuration. The domain defined by the surface mesh was partitioned in 64 sub-domains and the experiments were conducted by varying the number of processors from 2 to 64. Figure 5.4 (a) shows the speed up obtained from the parallel mesh generation experiments. As it is illustrated in this special case the method exhibits a super-linear speed up for the parallel mesh generation phase and good overall performance as expected. Notably, domain decomposition yields sub-domains of roughly equal size as it is illustrated in Figure 5.4 (b). Consequently, the load on the CPUs is equally distributed. Load-balancing, as it is discussed later, has a crucial role in the performance of the overall process.



**Figure 5.4:** (a) Plot of the speed-up obtained for 2-64 processors (b) Plot of the size distribution across the 64 Partitions of the Uniform Box configuration

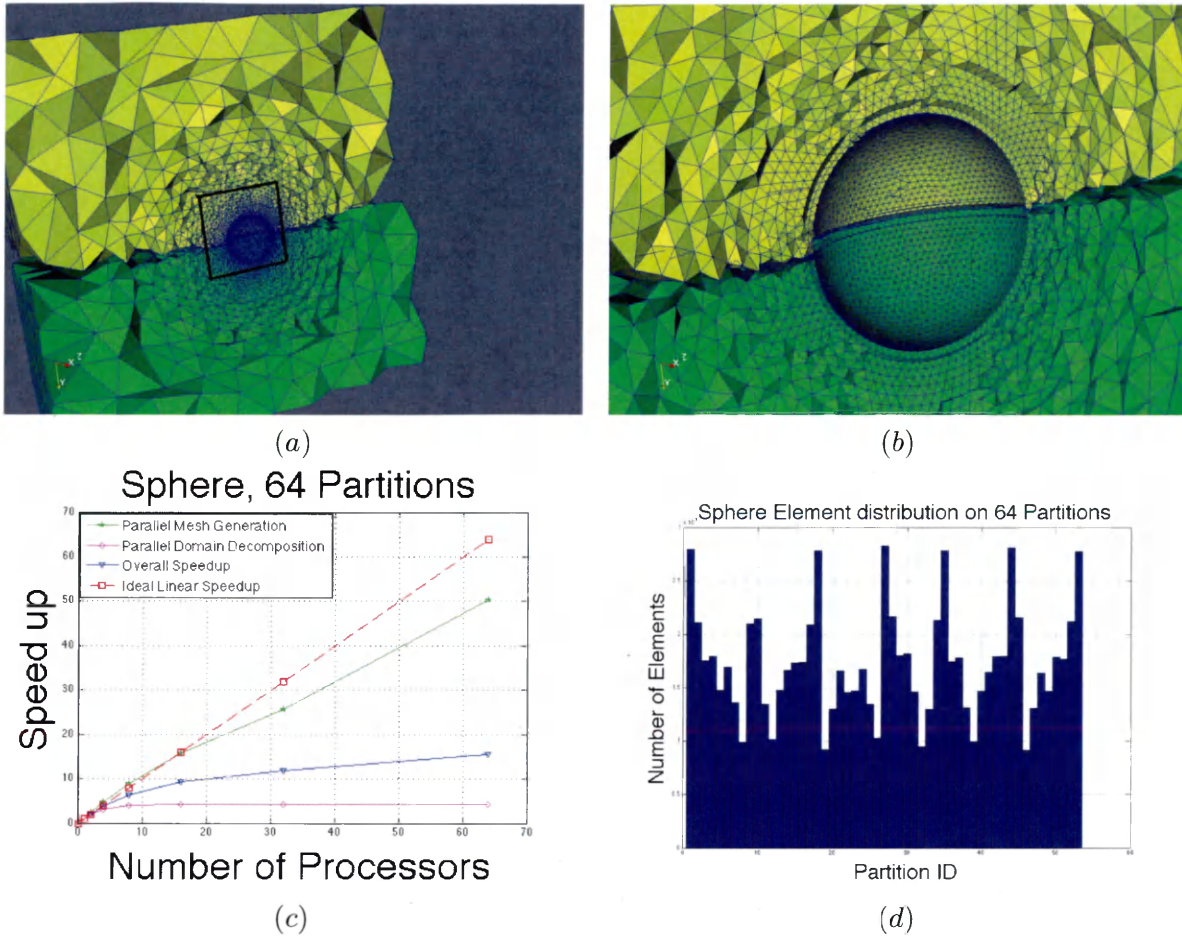
CPU's	Domain Decomposition	Mesh Generation	Total	Speedup
1	N/A	N/A	14325.84s	N/A
2	652.294s	6429.58s	7162.92s	2
4	415.489s	2788.41s	3273.35s	4.37
8	375.186s	1403.47s	1848.57s	7.74
16	373.28s	709.41s	1152.26s	12.43
32	373.33s	354.71s	797.94s	17.95
64	373.97s	180.896s	624.056s	22.95

**Table 5.1:** Timings for the Parallel Domain Decomposition, Parallel Mesh Generation phase and the associated speedup for each experiment on the Uniform Box configuration

### 5.3.2 A Sphere-in-box Configuration

The Sphere-in-box configuration is another good example for demonstrating how the method works on a simple non-uniform mesh where the load per sub-domain is not balanced. Figure 5.5 and Table 5.2 summarize the results.

A 12 million cell mesh was generated for the Sphere-in-box configuration. The domain defined by the surface mesh was partitioned in 64 partitions. A sample partitioning is shown in Figures 5.5(a) and 5.5(b). The experiments were conducted by varying the



**Figure 5.5:** (a) Sample partitioning of the Sphere-In-Box configuration (b) Zoomed view (c) Plot of the speed-up obtained for 2-64 processors (d) Plot of the grid size distribution across the 64 partitions

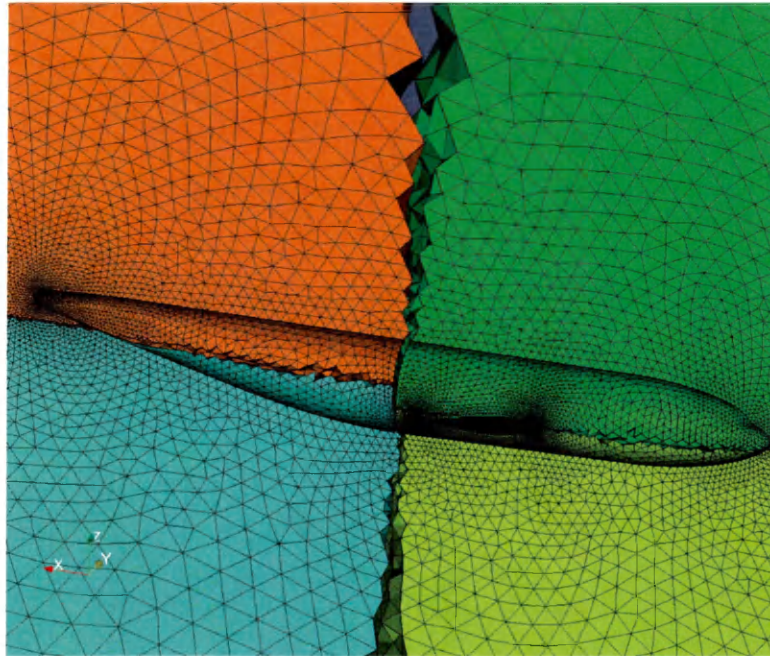
number of processors from 2 to 64. Figures 5.5 (c) and 5.5 (d) show the speed-up graph and size distribution across the partitions respectively. For this case there are load-imbalances since the mesh is not uniform. Despite the imbalances however, the method exhibits great scalability up to 8 processors. Additionally, the size of the problem is a lot smaller in comparison to the Uniform Box configuration. Sub-domain imbalances and the size of the problem are the primary reasons for the poor scalability of the method beyond 8 processors.

CPU	Domain Decomposition	Mesh Generation	Total	Speedup
1	N/A	N/A	6860.76s	N/A
2	587.48s	2806.73s	3430.38s	2
4	380.144s	1432.28s	1849.04s	3.71
8	288.99s	772.315s	1097.76s	6.25
16	270.194s	437.664s	744.223s	9.22
32	270.026s	266.512s	573.508s	11.96
64	270.388s	136.053s	443.447s	15.47

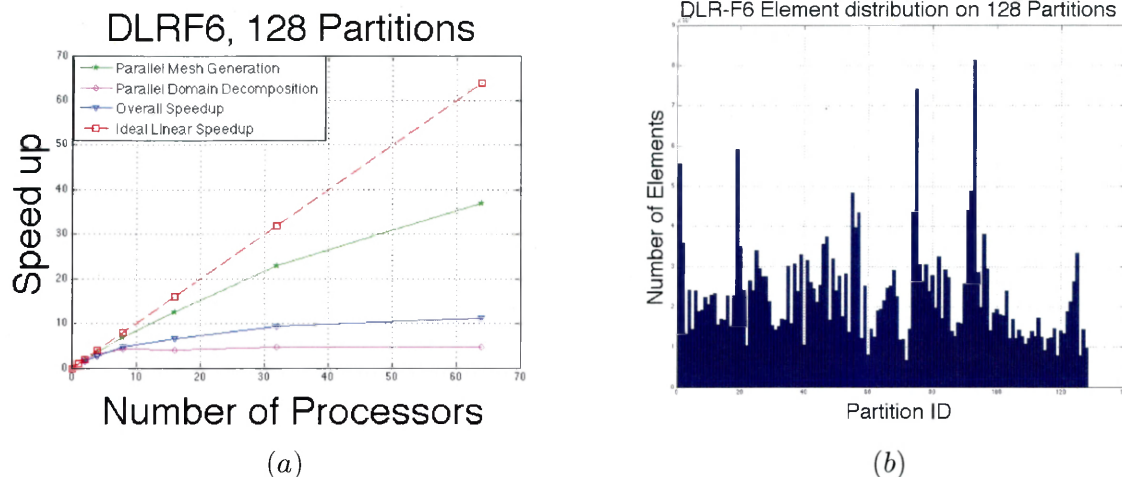
**Table 5.2:** Timings for the Parallel Domain Decomposition, Parallel Mesh Generation phase and the associated speedup for each experiment on the Sphere-in-box configuration

### 5.3.3 A Wing/Fuselage DLR-F6 Transport Configuration

The DLR-F6 Transport is a typical aircraft model consisting of a wing and fuselage. Figure 5.7 and Table 5.3 summarize the results obtained for a mesh of 32 million tetrahedral cells for this configuration.



**Figure 5.6:** Sample partitioning of the DLR-F6



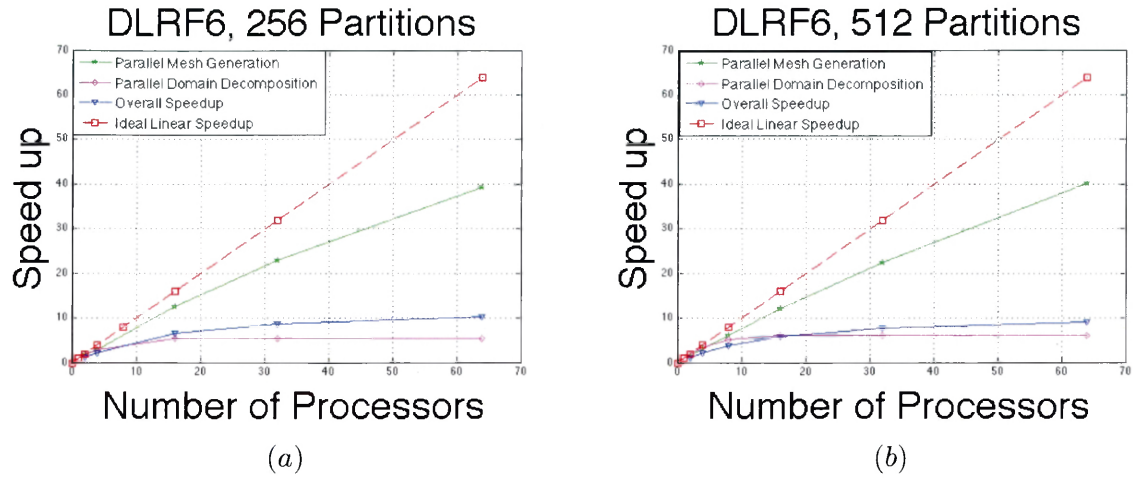
**Figure 5.7:** (a) Plot of the speed-up obtained for 2-64 processors (b) Plot of the grid size distribution across the 128 partitions.

CPUs	Domain Decomposition	Mesh Generation	Total	Speedup
1	N/A	N/A	15075.39s	N/A
2	1954.11s	8237.57s	10290.60s	1.46
4	1236.40s	4205.31s	5540.86s	2.72
8	933.607s	2191.10s	3222.90s	4.68
16	993.751s	1198.62s	2292.89s	6.57
32	846.308s	661.11s	1605.59s	9.38
64	847.754s	408.136s	1354.85s	11.13

**Table 5.3:** Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phase and the associated speedup for each experiment on the DLR-F6 Wing/Fuselage configuration

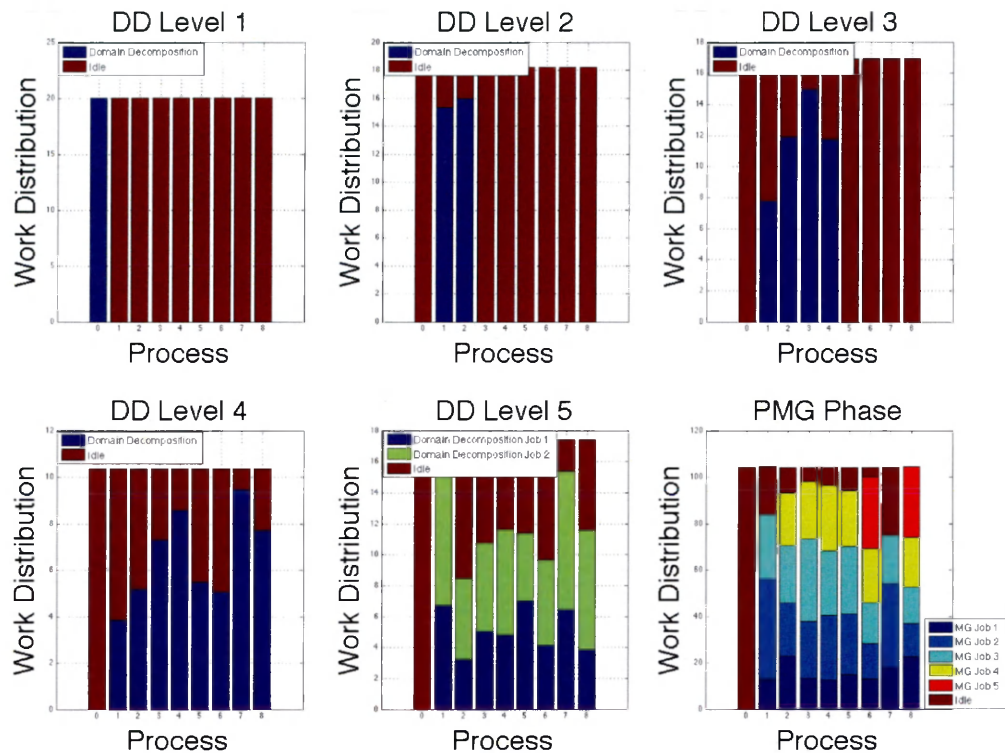
The domain defined by the surface mesh of the DLR-F6 was decomposed into 128 sub-domains. A sample partitioning of this configuration is shown in Figure 5.6. The experiments were conducted by varying the number of processors from 2 to 64. In practice, ensuring that the number of partitions is at least twice the number of processors facilitates the Master-Worker model in balancing the load on the CPUs. Further experiments on this particular configuration revealed that applying more levels of decomposition, i.e., generating 256 and 512 sub-domains, slightly reduced the overall performance. Figure

5.8 summarizes the effects of over-decomposition.



**Figure 5.8:** (a) Plot of the speed-up obtained using 256 Partitions and 2-64 processors (b) Plot of the speed-up obtained using 512 Partitions and 2-64 processors

Over-decomposition improves the performance of the Parallel Mesh Generation Phase. More sub-domains facilitate the Master-Worker model in maximizing the utilization of the CPUs. This benefit is illustrated in Figure 5.9, bottom right, in the load distribution plot for the PMG phase using 8 CPUs and 32 sub-domains. However, more time is spent in the decomposition phase and consequently the overall performance is decreased. As discussed later and inferred from Figure 5.9, the primary reason for the overall performance decrease, when over-decomposition is applied, is that the current decomposition phase is not scalable. This is partially attributed to the synchronization required in the current implementation before proceeding to the next level of decomposition.



**Figure 5.9:** Distribution of the entire process illustrating the computation time and idle time on each CPU at every level of Parallel Domain Decomposition and in the Parallel Mesh Generation phase using 32 sub-domains and 8 CPUs.

### 5.3.4 A Generic Supersonic Jet Configuration

The Generic Supersonic Jet configuration is a good example to demonstrate the applicability of the method for grids containing high-aspect ratio cells. Domain Decomposition by the AP technique and Parallel Grid generation by the AF technique for this particular case is difficult due to the highly stretched grid elements. For more information about issues concerning anisotropic grid generation and meshing requirements for computing the sonic-boom problem on this configuration, the interested reader is referred to [28]. In contrast to the previous examples where block partitioning is applied, for this con-

figuration a strip partitioning strategy is employed. Figure 5.11 summarizes the results obtained for a mesh of 16 million tetrahedral elements.



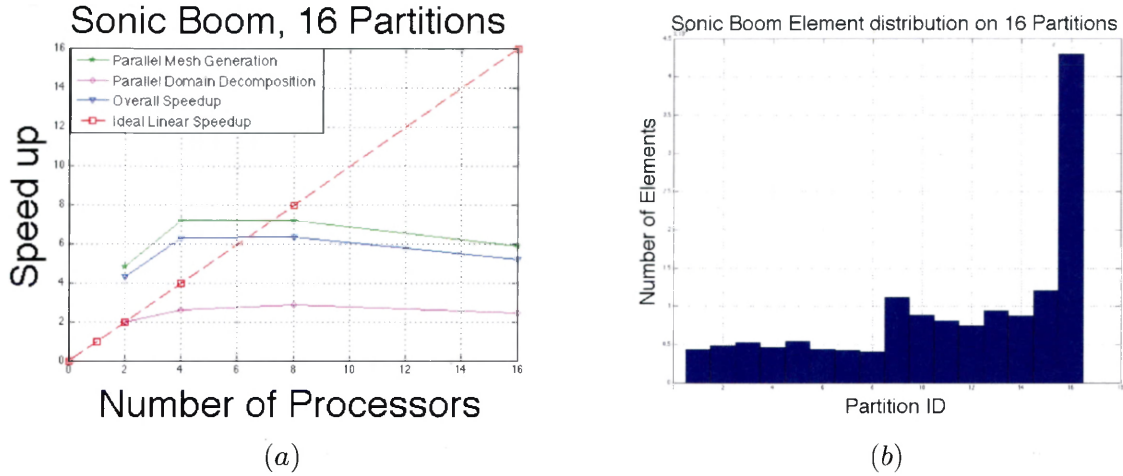
**Figure 5.10:** Sample partitioning of the Generic Supersonic Jet

CPUs	Domain Decomposition	Mesh Generation	Total	Speedup
1	N/A	N/A	37686.27s	N/A
2	903.62s	7866.10s	8819.87s	4.27
4	692.40s	5255.10s	5997.92s	6.28
8	631.423s	5243.23s	5925.73s	6.35
16	735.88s	6447.86s	7242.05s	5.20

**Table 5.4:** Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment on the Generic Supersonic Jet configuration

The domain defined by the surface mesh is decomposed into 16 strip partitions and the experiments were conducted by varying the number of processors from 2 to 16. Due to the high aspect ratio cells, further decomposition of this configuration is not feasible.

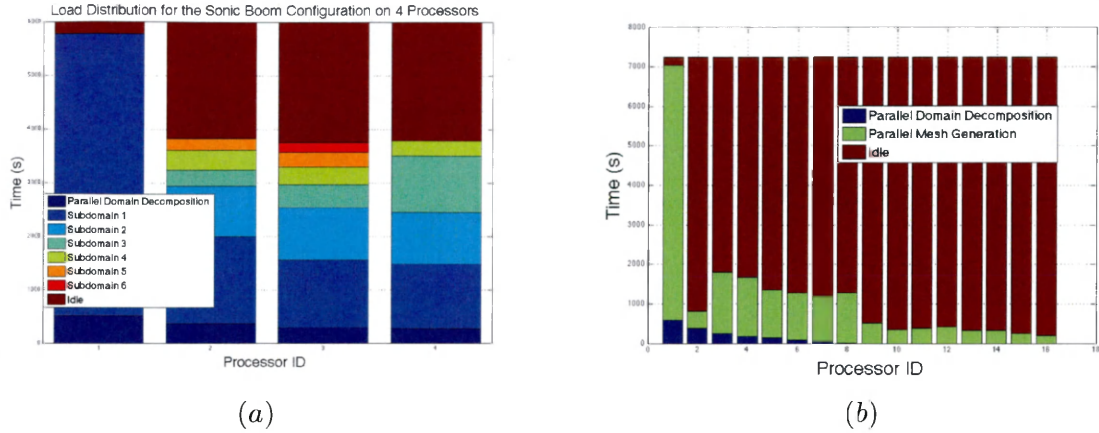




**Figure 5.11:** (a) Plot of the speed-up obtained for 2-16 processors (b) Plot of the grid size distribution across the 16 partitions of the Generic Supersonic Jet Configuration

Hence, the performance of the method is evaluated up to 8 CPUs and the results for the 16-CPU case are shown to demonstrate the importance of over-decomposition when the sub-domains exhibit imbalanced loads. As it is shown in Figure 5.11(a) super-linear speedup is achieved using 2 and 4 processors with best performance in the latter case. Based on observation<sup>1</sup>, the AF method is slower for highly stretched grids. Partitioning the domain minimizes the search space for the AF algorithm. In particular, the number of faces to be checked during the face-removal procedure, a primary function of the AF algorithm, is reduced which enables the method to achieve such a good performance. However, beyond 8 CPUs there is not enough work to facilitate the Master-Worker model in dynamic load balancing. A closer look at the load distribution for the 4 and 16 CPU case demonstrates the benefits of the Master-Worker model as well as justify the need for adaptive, i.e. graded, decomposition. The load distribution for the pre-mentioned cases is shown in Figures 5.12(a) and 5.12(b) respectively.

<sup>1</sup>Private communication with Dr. Shahyar Pirzadeh



**Figure 5.12:** (a) Load distribution of the Generic Supersonic Jet configuration on 4 CPUs (b) Load distribution of the Generic Supersonic Jet configuration on 16 CPUs

As one can observe in the 4 CPU case, Processor 1 is assigned the sub-domain that exhibits the largest computational load while Processors 2 – 4 execute the remainder. For the 16 CPU case however, all 16 sub-domains are scattered on to the 16 CPUs. Processors 2 – 16 are assigned the sub-domains with minimal computational load when compared to the computational load of the sub-domain assigned to Processor 1. Consequently, processors 2 – 16 are mostly idle, during the Parallel Mesh Generation phase, waiting to synchronize with Processor 1. Intuitively, a solution to this is to decompose the sub-domain that exhibits the largest computational load, i.e. the sub-domain assigned on Processor 1, further. This would in turn generate more sub-domains to facilitate the Master-Worker model in better dynamic load balancing and provide an equal element distribution among the sub-domains as opposed to the current element distribution shown in Figure 5.11(c). Such a decomposition strategy is feasible by Adaptive Domain Decomposition, i.e. decompose the sub-domains that have more work. However, the challenge in implementing an Adaptive Domain Decomposition strategy is in

describing a metric that estimates the load of a sub-domain. An obvious criterion for estimating the load of a sub-domain is the number of boundary faces of a sub-domain. Although this criterion can be representative of the load for certain cases it is not sufficient for the cases where the sub-domains are further refined in the interior. Better load-estimation is critical for Adaptive Domain Decomposition, which will be addressed in future work.

### 5.3.5 Energy Efficient Transport Configuration

The Energy Efficient Transport (EET) configuration is a good example to demonstrate the applicability of the method on complex configurations. The final mesh consisted of 23 million tetrahedral cells. The domain defined by the surface was decomposed in 64 sub-domains and the experiments were conducted by varying the number of processors from 2 to 32. Sample partitions of this configuration are illustrated in Figure 5.13.

For this configuration, two sets of experiments were conducted. The first experiment was conducted in a similar manner as in the previous cases. The results are summarized in Figures 5.15(a) and Table 5.5. For the second experiment, the grid size distribution, obtained from the first experiment, and shown in Figure 5.14, was used to assign a priority to each sub-domain. The largest sub-domain (largest in terms of grid size) has highest priority and is executed first. Next, in the parallel mesh generation phase the sub-domains are executed in Largest-Partition-First (LPF) order. The results of this experiment are summarized in Figure 5.15(b) and Table 5.6.

The data shown in Table 5.5 and Table 5.6 demonstrate the effects of the LPF policy. For two CPUs LPF does not admit any performance benefits. As the number of

CPU	Domain Decomposition	Mesh Generation	Total	Speedup
1	N/A	N/A	19367.34s	N/A
2	9673.74s	11689.5s	21447.4s	0.9
4	5818.15s	5191.71s	11083.1s	1.75
8	5678.99s	3064.98s	8827.55s	2.19
16	4571.43s	1411.954s	6055.68s	3.19
32	4459.61s	898.112s	5430.59s	3.57

**Table 5.5:** Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment of the Energy Efficient Transport configuration

CPU	Domain Decomposition	Mesh Generation (LPF)	Total	Speedup
1	N/A	N/A	19367.34s	N/A
2	9673.74s	11704.4s	21462.5s	0.9
4	5811.21s	5111.63s	10995.8s	1.76
8	4923.33s	2729.23s	7725.39s	2.50
16	4583.12s	1398.56s	6054.82s	3.24
32	4452.75s	834.138s	5359.7s	3.6

**Table 5.6:** Timings for the Parallel Domain Decomposition and Parallel Mesh Generation phases and the associated speedup for each experiment of the Energy Efficient Transport configuration

processors increases however, LPF admits a slight increase in the performance. Another important aspect that the data reveal is that when the number of sub-domains is much larger than the number of CPUs the scheduling policy does not have a crucial role in the performance, the Master-Worker model takes care of any load imbalances. In contrast, the scheduling policy admits some benefit as the number of sub-domains in excess the number of processors decrease.

## 5.4 Discussion

In summary, the results obtained by the Parallel Unstructured Grid Generation Framework using the AP technique demonstrated the applicability of this approach for com-

plex, real-world aerodynamic applications. Among the benefits of this approach are: (1) the AP technique is 'native' to the mesh generator, (2) the method is suitable for anisotropically stretched grids and (3) the method does not require the computation and complexity of auxiliary data structures in contrast to other approaches.

A notable characteristic of the performance results is that the parallel domain decomposition phase does not scale as well as the parallel mesh generation phase. Consequently, the overall performance is decreased when over-decomposition is applied. Among the top reasons for poor scalability are: (1) At each level of decomposition  $i$ , there are  $2^i$  sub-domains that can be decomposed in parallel utilizing at most  $2^i$  processors and (2) as the number of sub-domains increases the amount of work per sub-domain (problem size) decreases.

As mentioned earlier, in the current implementation synchronization is required for re-numbering the interfaces and sub-domains accordingly in order to ensure a coherent global mesh representation at each level of decomposition. Re-numbering is performed only by the Master process and hence the Worker processes are idle until re-numbering completes to proceed to the next level. Consequently, Worker processes are not well utilized for the parallel domain decomposition phase which is another contributing factor for the poor scalability of the parallel domain decomposition phase. A possible solution is to remove the synchronization phase and perform the re-numbering at the end. This would enable a better utilization of the resources and increase in the performance.

The performance results have also demonstrated the importance and effects of load-balancing (LB). Load-balancing is crucial in the overall performance. In the current implementation, LB is addressed in two ways:

1. Static Load Balancing (SLB): In the domain decomposition phase the AP method computes the cut plane based on mesh density of the sub-domain boundary triangulation such that the number of elements generated in each sub-domain (and thus the work associated with each sub-domain) is approximately equal.
2. Dynamic Load Balancing (DLB): the Master-Worker model assigns the next task in the queue to a worker as soon as a worker completes a given task. Consequently, provided that there is enough work (over-decomposition), the load on each processing unit is equally distributed.

Both SLB and DLB approaches can benefit from a better load-estimator. It is difficult to predict the amount of work required per sub-domain. Although the mesh density of the sub-domain boundary triangulation can give a good approximation of the load for some cases, it is not sufficient for the cases that exhibit more refinement in the interior of a sub-domain. Moreover, the number of elements on a sub-domain does not yield a one-to-one correspondence with the amount of computation required. Experience has shown that grid generation of highly stretched, anisotropic elements requires more computation<sup>2</sup>. A feasible solution for better load-estimation is the use of a background, coarse, oct-tree. Preliminary experiments using the oct-tree have demonstrated positive results for the SLB approach.

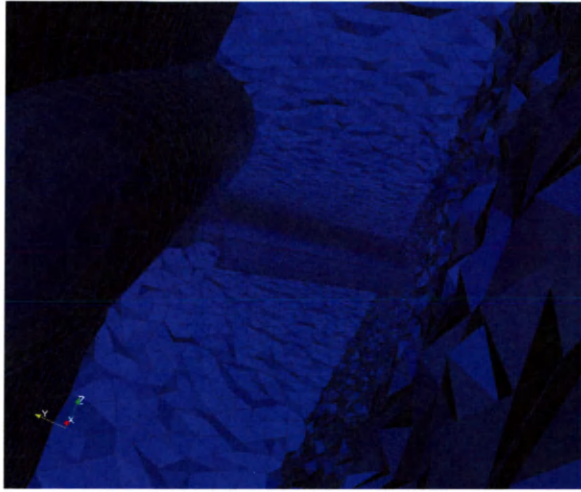
Additionally, a better load-estimator would enable the implementation of an Adaptive Domain Decomposition strategy. The experiments using the Generic Supersonic Jet justify the need for Adaptive Domain Decomposition. Adaptive Domain Decomposition has been successfully employed in [20, 21] in order to maintain balanced memory

---

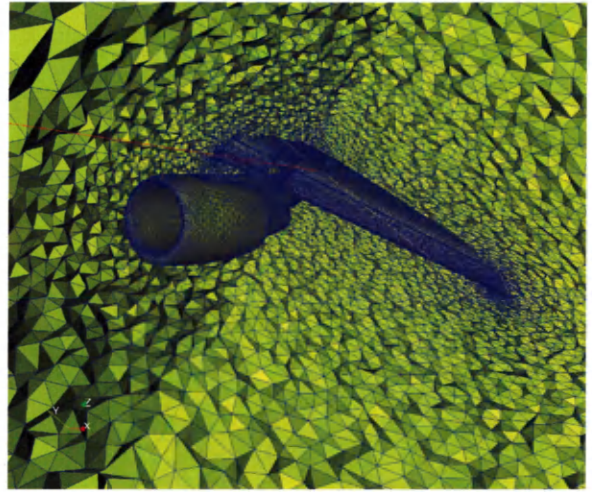
<sup>2</sup>Private Communication: Dr. Shahyar Pirzadeh

requirements and work-load. By Adaptive Domain Decomposition, the sub-domains that exhibit a lot of computation are further decomposed to further balance the load. However, the quality of load-balancing by this approach depends on the accuracy of the load-estimator.

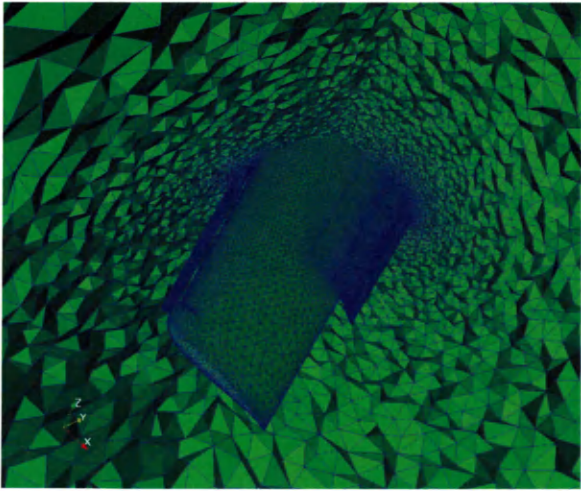
Lastly, experience with the parallel implementation has revealed that the *problem size*, *number of processing units* and the *number of partitions* have a crucial role in the performance. By observation, there exists an upper-bound to the number of processing units that can be effectively utilized for a given problem size. Moreover, given the problem size and the number of processing units available there exists an optimal number of partitions that achieves 'best performance'. Experience revealed a lower-bound, namely, the number of partitions must be at least twice the number of processors. Future efforts are focusing on a mathematical model of these parameters to enable their a priori estimation.



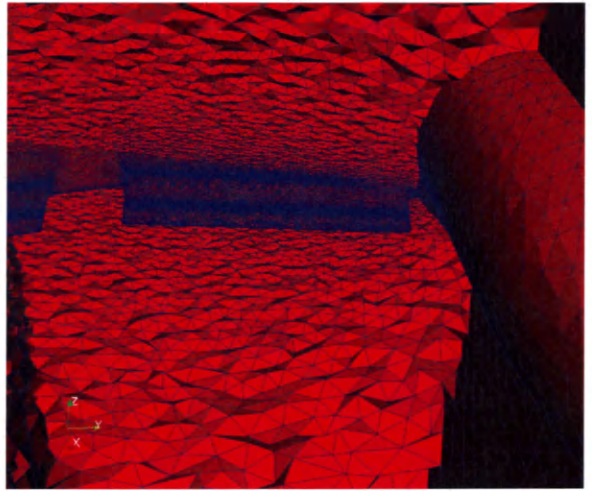
(a)



(b)



(c)



(d)

**Figure 5.13:** (a) Partition of the head of the fuselage and wing (b) Partition of the engine attached to the wing (c) Partition of wing tip (d) Partition of the fuselage tail and the wing



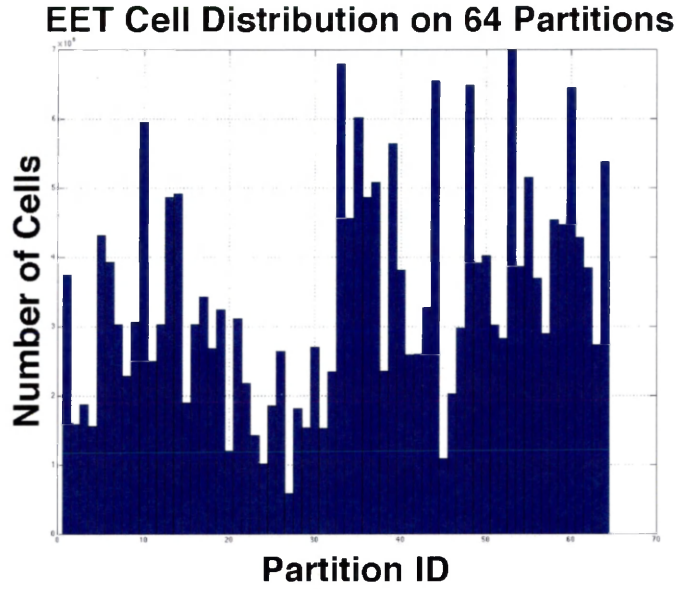


Figure 5.14: Grid size distribution of the EET configuration

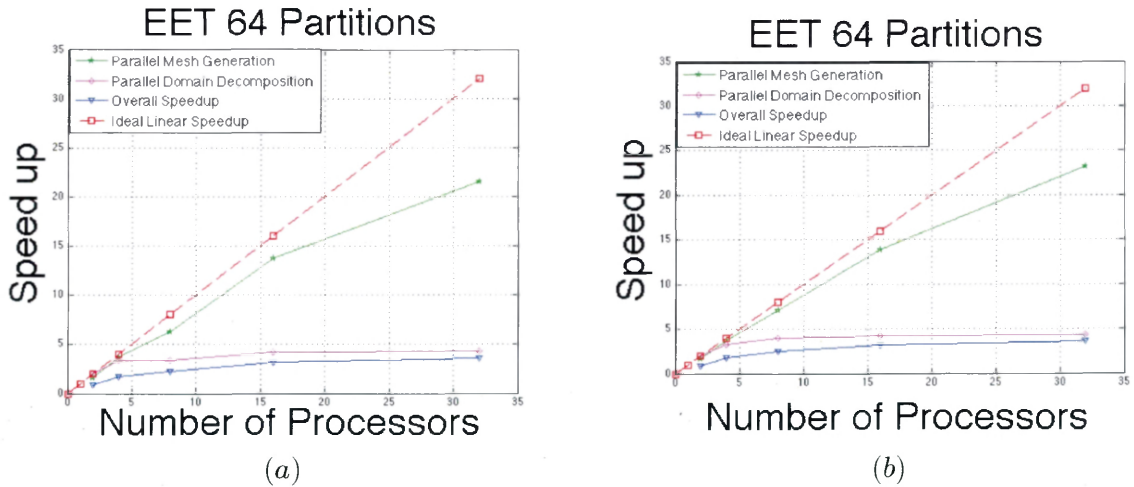


Figure 5.15: (a) Plot of the speedup for 2 – 32 processors (b) Plot of the speedup for 2 – 32 processors using the Largest-Partition-First (LPF) scheduling policy

## Chapter 6

# Conclusion

The infinitely increasing magnitude and complexity of problem sizes demands the use of parallel unstructured grid generation. This thesis presents a framework for parallel unstructured grid generation, using VGRID, for the routine application in aerodynamic simulations.

Three domain decomposition approaches for parallel unstructured grid generation by an Advancing Front technique are presented. First, a method by Discrete Domain Decomposition (i.e. starting from a coarse grid) is presented[37]. The benefit of this approach is that it treats the DD problem using robust graph partitioning libraries, such as METIS [2], which is typically the methodology employed in modern state-of-the-art finite element/finite volume flow solvers. However, the short-coming by this approach is that the method creates artifacts at the partition interfaces. The application of H-refinement at the interfaces deteriorates the quality of the triangulation. Moreover, the spacing at the interfaces is not compatible with the local spacing determined by the sources. Consequently, due to the incompatible spacing, the meshing process may

create tetrahedral cells of lower quality at the interfaces of the sub-domains and the final grid will be over-refined. Such artifacts are undesirable for solving Partial Differential Equations (PDEs).

Second, a method using Oct-tree guided DD is presented. Experimental results demonstrated that grids produced by this approach are not suitable in the CFD analysis of aerodynamics configurations. The two most important shortcomings of this approach are: (1) the Oct-tree vertices constrain the point distribution, size and shape of the mesh elements which is undesirable in CFD analysis, (2) the computational cost and high memory requirement associated for generating and maintaining the Oct-tree defeats the purpose of parallel unstructured grid generation.

Lastly, an approach based on the Advancing-Partition DD technique [29, 30] is presented. This approach exhibits many properties that are particularly suited for parallel unstructured grid generation for complex, real-world aerodynamic simulations. Among the benefits are: (1) the method is 'native' to the underlying mesh generator, VGRID, (2) no artifacts, partition interfaces are generated by the advancing front process as part of the volume mesh in a 'natural' way and (3) the computation cost for domain decomposition is minimal in contrast to other approaches which rely on auxiliary data structures and/or operations. However, the performance of the current implementation can be improved. The performance results for a variety of cases, presented earlier, demonstrate that the poor scalability of the parallel domain decomposition phase is the primary contributing factor affecting the overall performance. Future efforts will address the performance bottlenecks in the present implementation.

## 6.1 Future Work

Future work is focusing on the following improvements for the current approach:

1. Extension of the AP method within a user-programmable framework. The current implementation is specific and native to VGRID. Due to the close link with the underlying mesh generator, the current parallel framework exhibits a moderate degree of flexibility. Future work will leverage the functionality provided in the current implementation by providing a more flexible interface. Hence, the AP method can be integrated with different mesh generators and, optimized and tailored for the specific application.
2. Removing synchronization from the parallel domain decomposition phase to improve the scalability of the parallel domain decomposition phase.
3. Use of background, coarse Oct-tree for better load-estimation to:
  - Compute more accurately the location of the cutting plane such that the number of elements generated in each sub-domain is approximately equal.
  - Adaptive Domain Decomposition
4. A Mathematical model for a priori approximation of the optimal values for the number of processing units and partitions to use given a problem size.
5. Gateway portal development for the more effective use of TeraGrid.

# Appendix A

## Pseudocode

### A.1 Parallel Domain Decomposition

#### A.1.1 Master Domain Decomposition Process

**Require:** The input consists of:

1. Surface mesh (front)  $S$  to decompose
2. the initial split direction  $D$  ( $D \in \{x, y, z\}$ ),
3. the maximum level of subdivisions  $L_{max}$
4. alternating, a flag that indicates whether the split direction will be alternating at subsequent levels

**Ensure:** The output consists of:

1. The final merged mesh  $\mathcal{M}$
- 1: Initialize the level counter  $L$  to zero
  - 2:  $(S_i, S_1, S_2) = \text{call DECOMPOSE}(S, D)$

```

3: Push  $S_i$  in the list of interfaces  $\mathcal{I}$ 

4: Push the subdomains  $S_1, S_2$  to the subdomain queue  $\mathcal{S}$ 

5: Increment level counter  $L$ 

6: if (alternating) then

7:   Update  $D$ 

8: end if

9: Synchronize with workers before proceeding to subsequent levels of subdivision

10: Broadcast  $L$  and  $D$  to worker processes

11: while ( $L < L_{max}$ ) do

12:   Scatter the sub-domains in the subdomain queue  $\mathcal{S}$  to the worker processes

13:   while ( $\mathcal{S}$  is not empty ) do

14:     Pop subdomains  $S_n$  from the subdomain queue  $\mathcal{S}$ 

15:     Receive an acknowledgment from  $P_i$  for a subdomain  $S$ 

16:     Send  $S_n$  to  $P_i$  for decomposition

17:     Renumber the global ids in respect to a global mesh of the interface  $S_i$  of  $S$ 

18:     Renumber the global ids in respect to a global mesh of the two sub-domains
         $S_1, S_2$  of  $S$  with interface  $S_i$ 

19:     Push  $S_i$  in the list of interfaces  $\mathcal{I}$ 

20:     Push the subdomains  $S_1, S_2$  in the subdomain queue  $\mathcal{S}$  and remove the parent
         $S$ 

21:   end while

22: repeat

23:   Receive an acknowledgment from  $P_i$  for a subdomain  $S$ 

```

- 24: Renumber the global ids in respect to a global mesh of the interface  $S_i$  of  $S$
- 25: Renumber the global ids in respect to a global mesh of the two sub-domains  $S_1, S_2$  of  $S$  with interface  $S_i$
- 26: Push  $S_i$  in the list of interfaces  $\mathcal{I}$
- 27: Push the subdomains  $S_1, S_2$  in the subdomain queue  $\mathcal{S}$  and remove the parent  $S$
- 28: **until** all acknowledgments are received
- 29: Increment level counter  $L$
- 30: **if** (alternating) **then**
- 31:     Update  $D$
- 32: **end if**
- 33: Signal workers to go the next level
- 34: Broadcast  $L$  and  $D$  to worker processes
- 35: **end while**
- 36: Synchronize with workers for completion of the parallel domain decomposition phase

### A.1.2 Worker Domain Decomposition Process

- 1: Synchronize with the master for subsequent levels of subdivision
- 2: Receive Broadcasted values for  $L$  and  $D$
- 3: **while** ( $L < L_{max}$ ) **do**
- 4:     **repeat**
- 5:         Receive a domain decomposition job for subdomain  $S$  from the master process
- 6:          $(S_i, S_1, S_2) = \text{call DECOMPOSE}(S, D)$

- 7:     Send acknowledgment for  $S$  to the Master
- 8:     **until** Signaled to go to the next level
- 9:     Receive Broadcasted values for  $L$  and  $D$
- 10:  **end while**
- 11: Synchronize with master for completion of the parallel domain decomposition phase

## A.2 Parallel Mesh Generation

### A.2.1 Master Parallel Mesh Generation Process

**Require:** The domain to be decomposed in to a set of sub-domain

**Ensure:** All subdomains are meshed and the sub-domains are merged

- 1: Synchronize with workers for the parallel mesh generation phase
- 2: Set up a work queue of the subdomains to be meshed
- 3: Scatter the sub-domains to the worker processes
- 4: **while** workqueue is no empty **do**
- 5:     pop a sub-domain  $S$  from the workqueue
- 6:     wait for a job request from a worker  $P_i$
- 7:     Send sub-domain  $S$  to  $P_i$  for meshing
- 8: **end while**
- 9: Send signal to workers to exit
- 10: Merge the sub-domains to a single Finite element mesh
- 11: Synchronize with workers for completion of the parallel mesh generation phase



### A.2.2 Worker Parallel Mesh Generation Process

- 1: Synchronize with master for the parallel mesh generation phase
- 2: **repeat**
- 3: Receive sub-domain  $S$  to mesh from the master
- 4: call `MESHSUBDOMAIN( $S$ )` to mesh  $S$
- 5: Send a request for a new sub-domain
- 6: **until** Signaled to exit
- 7: Synchronize with master for completion of the parallel mesh generation phase

# Appendix B

## User Guide

### B.1 Installation

The following instructions assume that the user is familiar with UNIX and MPI is already installed on the target platform. If MPI is not installed, or you do not know consult your systems administrator. Alternatively, you may install a copy of MPI by downloading from <http://www-unix.mcs.anl.gov/mpi/> and follow the instructions therein.

The PVGRID Package consists of the following binaries:

- VGRID: the unstructured grid generator and partitioner
- POSTGRID: the post-processing code for closing the remaining unmeshed regions
- PARTFRONTS: a utility code for separating the fronts after partitioning
- PVGRIDPDD: the parallel code which performs parallel domain decomposition, parallel mesh generation and merging

PVGRID has been tested on the following architectures using Intel compilers:

- Intel/MAC OS X
- Intel/Linux OS
- SGI Altix/Linux OS

1. Install the PVGRID Package in the desired location `$PVGRID_HOME`

2. Ensure that `$PVGRID_HOME` is in your path

(a) If you are using `bash` add the following line to your `.bashrc` file

```
export PATH=$PVGRID_HOME:$PATH;
```

(b) If you are using `cs`h add the following line to your `.cshrc` file

```
setenv PATH $PVGRID_HOME:$PATH;
```

(c) If you are using another shell environment please consult your manual, or ask your systems administrator

## B.2 Running PVGRID

The following arguments are currently implemented in PVGRID:

- `-p {project file name}` Sets the project file name.
- `-L {#}` Sets the desired level of subdivisions. E.g. if  $x$  is the desired level of subdivisions  $2^x$  sub-domains will be generated. Default is 2.
- `-constant-direction` Disables alternating direction of subdivision at each level. Default is alternating direction.

- `-split-direction {#}` Sets the initial split direction. Possible values are 1, 2, 3 representing  $x, y, z$  respectively. If this value is set to 0, VGRID computes all 3 directions and picks the 'best'. Default is 1.
- `-vtk` outputs the sub-domains in `vtk` formatted files for visualization with Paraview [4]. By default sub-domains are not written in `vtk` files.

A typical command line for PVGRID is the following:

```
mpirun -n $N pvgridpdd -p $projectName -L $x
```

where `$N` is the number of MPI processes to use, `$projectName` is the file name of the project (without extension) and `$x` is the number of levels of subdivision.

# Bibliography

- [1] Framework for parallel grid generation. <http://www.cs.wm.edu/gxzaga/PMESHLib>.
- [2] Metis, family of multilevel partitioning algorithms. <http://glaros.dtc.umn.edu/gkhome/views/metis>.
- [3] National center for supercomputing applications. <http://www.ncsa.uiuc.edu/>.
- [4] Paraview, parallel visualization application. <http://www.paraview.org>.
- [5] Teragrid cluster. <http://www.teragrid.org>.
- [6] Tetgen, a quality tetrahedral mesh generator and three-dimensional triangulator. <http://tetgen.berlios.de/index.html>.
- [7] ANDREY CHERNIKOV, NIKOS CHRISOCHOIDES, AND KEVIN BARKER. Parallel programming environment for mesh generation. In *Proceedings of the Eighth International Conference on Numerical Grid Generation in Computational Field Simulations*, pages 805–814, June 2002. Honolulu, HI, USA.
- [8] NIKOS CHRISOCHOIDES. *Numerical Solution of Partial Differential Equations on Parallel Computers*, chapter 7 Parallel Mesh Generation, pages 237–255. Springer-Verlag, 2005.
- [9] H. L. DE COUGNY AND M. S. SHEPHARD. Parallel volume meshing using face removals and hierarchical repartitioning. *Computer Methods in applied mechanics and engineering*, 174:275–298, 1999.
- [10] N. T. FRINK, SHAHYAR Z. PIRZADEH, P. C. PARIKH, MOHAGNA J. PANDYA, AND M. K. BHAT. The NASA tetrahedral unstructured software system. *The Aeronautical Journal*, 104(1040):491–499, October 2000.
- [11] JEROME GALTIER AND PAUL LOUIS GEORGE. Prepartitioning as a way to mesh subdomains in parallel. In *Proceedings of the Fifth International Meshing Roundtable*, pages 107–121, 1996.
- [12] WILLIAM GROPP, EWING LUSK, AND ANTHONY SKJELLUM. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT press, 1999.
- [13] YASUSHI ITO, ALAN M. SHIH, ANIL K. ERUKALA, BHARAT K. SONI, ANDREY CHERNIKOV, NIKOS P. CHRISOCHOIDES, AND KAZUHIRO NAKAHASHI. Parallel

- unstructured mesh generation by an advancing front. *Mathematics and Computers in Simulation*, 75:200–209, January 2007.
- [14] E.G. IVANOV, H. ANDRA, AND A. N. KUDRYAVTSEV. Domain decomposition approach for automatic parallel generation of tetrahedral grids. *Computational Methods in Applied Mathematics*, 6(2):178–193, 2006.
- [15] X. JIAO. Volume and feature preservation in surface mesh optimization. In *Proceedings of the Fifteenth International Meshing Roundtable*, pages 359–374, September 2006. Birmingham, AL, USA.
- [16] X. JIAO, N. R. BAYYANA, AND H. ZHA. Optimizing surface triangulation via near isometry with reference meshes. In *Proceeding of International Conference on Computational Science*, pages 334–341, May 2007. Beijing, China.
- [17] ANDRIY KOT, ANDREY CHERNIKOV, AND NIKOS CHRISOCHOIDES. Effective out-of-core parallel Delaunay mesh refinement using off-the-shelf software. In *20th IEEE International Parallel and Distributed Processing Symposium*, Rhodes Island, Greece, April 2006.
- [18] B. G. LARWOOD, N. P. WEATHERILL, O. HASSAN, AND K. MORGAN. Domain decomposition approach for parallel unstructured mesh generation. *International Journal for Numerical Methods in Engineering*, 58:177–188, 2003.
- [19] LEONIDAS LINARDAKIS. *Decoupling Method for Parallel Delaunay 2D Mesh Generation*. PhD thesis, College of William and Mary, Williamsburg, Virginia, July 2007.
- [20] LEONIDAS LINARDAKIS AND NIKOS CHRISOCHOIDES. Delaunay decoupling method for parallel guaranteed quality planar mesh refinement. *SIAM Sci. Comput.*, 27(4):1394–1423.
- [21] LEONIDAS LINARDAKIS AND NIKOS CHRISOCHOIDES. A static medial axis domain decomposition for 2D geometries. *ACM Transactions on Mathematical Software*, 34(1):1–19, November 2005.
- [22] R. LOHNER, J. CAMBEROS, AND M. MERRIAM. Parallel unstructured grid generation. *Computer Methods in Applied Mechanics and Engineering*, 95:343–357, 1992.
- [23] RAINALD LOHNER. *Applied CFD Techniques: An Introduction based on Finite Elements*. Wiley, 2001.
- [24] RAINALD LOHNER AND JUAN R. CEBRAL. Parallel advancing front grid generation. In *Proceedings of the Eighth International Meshing Roundtable*, pages 67–74, October 1999. South Lake, Tahoe, CA, USA.
- [25] PARESH PARIKH, SHAHYAR PIRZADEH, AND RAINALD LOHNER. A package for 3-d unstructured grid generation, finite-element flow and flow field visualization. NASA Technical Report NASA-CR-182090, NASA (non Center Specific), September 1990.

- [26] SHAHYAR PIRZADEH. Recent progress in unstructured grid generation. *American Institute of Aeronautics and Astronautics (AIAA)*, 1992.
- [27] SHAHYAR PIRZADEH. Three-dimensional unstructured viscous grids by the advancing-layers method. *American Institute of Aeronautics and Astronautics (AIAA)*, 34(1):43–49, January 1996.
- [28] SHAHYAR PIRZADEH. Advanced unstructured grid generation for challenging aerodynamic applications. In *Proceedings of the 26th American Institute of Aeronautics and Astronautics (AIAA) Applied Aerodynamics Meeting*, August 2008. (Abstract Accepted).
- [29] SHAHYAR PIRZADEH. Domain decomposition by the advancing-partition method. NASA New Technology Report NTR-5025177, NASA Langley Research Center, February 2008.
- [30] SHAHYAR PIRZADEH AND GEORGE ZAGARIS. Domain decomposition by the advancing-partition method for parallel unstructured grid generation. In *Proceedings of the 47th American Institute of Aeronautics and Astronautics (AIAA) Aerospace Sciences Meeting*, January 2009. Orlando, Florida (submitted for publication).
- [31] R. SAID, N. P. WEATHERILL, K. MORGAN, AND N. A. VERHOEVEN. Distributed parallel Delaunay mesh generation. *Computer Method in Applied Mechanics and Engineering*, 177:109–125, 1999.
- [32] J. SAMAREH. Gridtool: A surface modelling and grid generation tool. In *Proceedings of the Workshop on Surface Modelling, Grid Generation and Related Issues in CFD Solutions*, 9-11 May 1995.
- [33] HANAN SAMET. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
- [34] HANAN SAMET. *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*. Addison-Wesley, 1995.
- [35] DMTRI SHAROV, HONG LUO, JOSEPH D BAUM, AND RAINALD LOHNER. Unstructured navier-stokes grid generation at corners and ridges. *International Journal for Numerical Methods in Fluids*, 43:717–728, 2003.
- [36] H. SI AND K. GAERTNER. Meshing piecewise linear complexes by constrained Delaunay tetrahedralizations. In *Proceedings of the 14th International Meshing Roundtable*, pages 147–163, September 2005.
- [37] GEORGE ZAGARIS, SHAHYAR PIRZADEH, ANDREY CHERNIKOV, AND NIKOS CHRISOCHOIDES. Parallel advancing front for CFD simulations of real-world aerodynamic problems. In *9th US National Congress on Computational Mechanics*, July 2007. San Francisco, CA, USA.

- [38] GEORGE ZAGARIS, SHAHYAR PIRZADEH, AND NIKOS CHRISOCHOIDES. A framework for parallel unstructured grid generation for complex aerodynamic simulations. In *Proceedings of the 47th American Institute of Aeronautics and Astronautics (AIAA) Aerospace Sciences Meeting*, January 2009. Orlando, Florida (submitted for publication).



## VITA

### George Zagaris

George Zagaris was born in Athens, Greece on February 6, 1982. In May 2005, George completed his B.Sc. degree with honors in Computer Science and a minor in mathematics from the College of William & Mary. During the summer of 2005 to August 2008, George pursued his M.Sc. degree in Computer Science at the College of William & Mary. While a graduate student, George was awarded the Graduate Research Student Program (GSRP) Fellowship at NASA Langley Research Center to work as a Visiting Research Assistant Fellow on Parallel Unstructured Grid Generation for complex real-world aerodynamic simulations under the direction of Dr. Nikos Chrisochoides, Associate Professor from the College of William & Mary and Dr. Shahyar Pirzadeh, Senior Research Engineer from the Configurations and Aerodynamics Branch at NASA Langley Research Center. Starting in the Fall of 2008, George will be pursuing his PhD degree in the Applied Science department at the College of William & Mary.