Parallel Adaptive Mesh Refinement for High-Order Finite-Volume Schemes in Computational Fluid Dynamics

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Alan Michael Schwing

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Graham V. Candler

August, 2015

Acknowledgements

This document presents the culmination of several years of work as a PhD candidate. The work spanned time in Minneapolis, Minnesota and Houston, Texas and it would not have been possible without the contributions, direct and indirect, of many individuals. I would like to take a moment to recognize those who impacted this work and influenced my concurrent development as a researcher and an engineer.

It would be impossible to not first thank the entirety of Johnson Space Center's Applied Aerospace and Computational Fluid Dynamics branch, EG3. Not only did the branch first inspire me to pursue a career in computational fluids as an undergraduate, but they have also supported my goal of earning a doctoral degree. The members of EG3 continue to provide strong examples of technical excellence and I am proud to include myself in their number.

In specific, I would like to acknowledge Brian Anderson for giving me my first lesson in hypersonics on a white erase board at CUBRC in 2006, Dr. Randy Lillard for motivating me to pursue a doctorate, Darby Vicker for being a fantastic mentor and teacher throughout my tenure in EG3, and James Greathouse for sharing his considerable experience and helping behind the scenes. Jay LeBeau supported my decision to study at the University of Minnesota, lobbied on my behalf, and saved me a place in his branch to return to. Since much of my research was performed in Houston, Dr. Benjamin Kirk served as a vital touchstone and his enthusiasm for this work was invaluable.

Thank you, all.

I am grateful to my advisor, Dr. Graham Candler, for welcoming me into his group at the University of Minnesota. His forward-thinking approach and strong desire for application is as motivational to his students as it is enabling to the field at large. The nature and scope of this work was molded by his expectations - I sincerely hope that I have met them. It is an honor to be associated to a man of his accomplishment.

Also at Minnesota, Dr. Ioannis Nompelis has been a sounding-board for this work and a co-author of these strategies. If his influence can be seen in this final product, I consider that a high compliment.

Dr. Pramod Subbareddy is one of the most deliberate men I have ever met and a meticulous researcher. This work would not have been possible without his insight and experienced advice.

My years in Minnesota were profoundly formative. The research group that Dr. Candler has assembled includes some of the most motivated, intelligent, and interesting people I have ever met. Even with the demands of graduate school, unending research, and looming submission deadlines, the mixture of personalities has an infectious enthusiasm for all aspects of fluid dynamics, numerical methods, and life in a great city. For helping me grow in ways academic and otherwise, I must thank Jason Bender, Eric Stern, Aaron Neville, Joseph Brock, Pietro Ferrero, Derek Dinzl, Ross Chaudhry, Lindsay Kirk, Erik Tylczak, Anand Kartha, Vladimyr Gidzak, Stephanie Jenson, Sidharth GS, Loretta Treviño, Sriram Doraiswamy, Ross Wagnild, Matthew Bartkowicz, Heath Johnson, Travis Dravna, David Peterson, and Jeffrey Komives.

Finally, thank you to my family for their understanding and encouragement over the years. Lue, Michael, Theda, and Terry for making me the man I am today, Barb, Jim, Karlene, and Mike for giving me a second home in Illinois, and Katelyn for her unending patience, her unwavering support, and her profound strength of will - all necessary to maintain a marriage stretched an additional 2,500 mile-years.

For Katelyn

Abstract

For computational fluid dynamics, the governing equations are solved on a discretized domain of nodes, faces, and cells. The quality of the grid or mesh can be a driving source for error in the results. While refinement studies can help guide the creation of a mesh, grid quality is largely determined by user expertise and understanding of the flow physics. Adaptive mesh refinement is a technique for enriching the mesh during a simulation based on metrics for error, impact on important parameters, or location of important flow features. This can offload from the user some of the difficult and ambiguous decisions necessary when discretizing the domain.

This work explores the implementation of adaptive mesh refinement in an implicit, unstructured, finite-volume solver. Consideration is made for applying modern computational techniques in the presence of hanging nodes and refined cells. The approach is developed to be independent of the flow solver in order to provide a path for augmenting existing codes. It is designed to be applicable for unsteady simulations and refinement and coarsening of the grid does not impact the conservatism of the underlying numerics.

The effect on high-order numerical fluxes of fourth- and sixth-order are explored. Provided the criteria for refinement is appropriately selected, solutions obtained using adapted meshes have no additional error when compared to results obtained on traditional, unadapted meshes. In order to leverage large-scale computational resources common today, the methods are parallelized using MPI. Parallel performance is considered for several test problems in order to assess scalability of both adapted and unadapted grids. Dynamic repartitioning of the mesh during refinement is crucial for load balancing an evolving grid. Development of the methods outlined here depend on a dual-memory approach that is described in detail.

Validation of the solver developed here against a number of motivating problems shows favorable comparisons across a range of regimes. Unsteady and steady applications are considered in both subsonic and supersonic flows. Inviscid and viscid simulations achieve similar results at a much reduced cost when employing dynamic mesh adaptation. Several techniques for guiding adaptation are compared.

Detailed analysis of statistics from the instrumented solver enable understanding of the costs associated with adaptation. Adaptive mesh refinement shows promise for the test cases presented here. It can be considerably faster than using conventional grids and provides accurate results. The procedures for adapting the grid are light-weight enough to not require significant computational time and yield significant reductions in grid size.

Contents

Acknowledgements **i**

Abstract **iv**

List of Tables **ix**

List of Figures **xi**

1 Introduction **1**

1.1 Motivation 1

1.2 Chapter Summaries 4

2 Numerical Methods **6**

2.1 Governing Equations 6

2.2 Finite-Volume Formulation 7

2.3 Inviscid Fluxes 10

2.4 Viscous Fluxes and Cell Gradients 14

2.5 Time Integration 15

2.5.1 Explicit 15

2.5.2 Implicit 16

Point Implicit 18

Line Implicit 20

3 Adaptive Mesh Refinement **23**

3.1 Non-Conformal Meshes 25

3.2 Mesh Adaptation . . . 27
3.3 Modifications to Numerical Methods . . . 28
3.3.1 Inviscid Fluxes . . . 28
3.3.2 Viscous Fluxes and Cell Gradients . . . 32
3.3.3 Point Implicit . . . 32
3.3.4 Line Implicit . . . 32
3.4 Memory Management . . . 34
3.4.1 Linked Lists . . . 35
3.4.2 Dynamic Grid Events . . . 38
3.5 Parallelization . . . 40
3.6 Refinement Criteria . . . 45
3.6.1 Geometric . . . 47
3.6.2 Feature-based . . . 47
3.7 Projection and Node Redistribution . . . 49
3.8 Refinement Propagation . . . 51

4 Verification **54**
4.1 Convecting Density Pulse . . . 55
4.1.1 1-D Density Pulse . . . 56
4.1.2 2-D Density Pulse . . . 58
4.2 Parallel Performance . . . 63
4.2.1 Parallel Performance of Unadapted Grids . . . 63
4.2.2 Parallel Performance of Adapted Grids . . . 67

5 Validation and Application **79**
5.1 Sod Shock Tube . . . 79
5.2 Hypersonic Double Wedge . . . 87
5.2.1 Grid Generation . . . 88
5.2.2 Adaptation Strategy . . . 89
5.2.3 Results . . . 91
5.2.4 Computational Efficiency . . . 93
5.3 Hypersonic Double Cone . . . 102
5.3.1 Grid Generation . . . 105

5.3.2 Adaptation Strategy . . . 105
5.3.3 Results . . . 107
5.3.4 Computational Efficiency . . . 109
5.3.5 Parallel Performance . . . 114
5.4 Type IV Shock-Shock Interaction . . . 117
5.4.1 Grid Generation . . . 118
5.4.2 Adaptation Strategy . . . 121
5.4.3 Results . . . 124
5.4.4 Computational Efficiency . . . 132
5.5 Low-Reynolds-Number Cylinders . . . 134
5.5.1 Grid Generation . . . 136
5.5.2 Adaptation Strategy . . . 139
5.5.3 Results . . . 142
$\mathrm{Re_D} = 80$ . . . 142
$\mathrm{Re_D} = 160$ . . . 148
5.6 Capsule Aerodynamics . . . 152
5.6.1 Wind Tunnel Test . . . 153
5.6.2 Grid Generation . . . 153
5.6.3 Adaptation Strategy . . . 155
5.6.4 Results . . . 163
5.6.5 Computational Efficiency . . . 170

6 Summary and Conclusions **175**

Appendix A. Derived Datatypes **186**

List of Tables

5.1 Time required to solve 20 seconds of flow time for the Sod shock tube problem. 84
5.2 Conditions for regions in double wedge show in Fig. 5.6; Mach 9, $\gamma = 1.4$. 88
5.3 Timing and cell count comparisons for uniform 256×256 mesh and equivalent grids with three levels of AMR. 98
5.4 Timing and cell count comparisons for uniform 512×512 mesh and equivalent grids with four levels of AMR. 98
5.5 Timing and cell count comparisons for uniform 1024×1024 mesh and equivalent grids with five levels of AMR. 101
5.6 Freestream conditions for run 35. 103
5.7 Freestream conditions for ONERA experiment. 119
5.8 Grid metrics of computational domains for ONERA simulations. 122
5.9 Percentage of run time in significant subroutines for simulations on ONERA test. 133
5.10 Freestream conditions for cylindrical test cases. 136
5.11 Grid spacings for two-dimensional cylinder grids, with and without AMR. 136
5.12 Drag and Strouhal number results for $\mathrm{Re_D} = 80$. 144
5.13 Percentage of run time in significant subroutines for cylinder solutions ($\mathrm{Re_D} = 80$). 147
5.14 Drag and Strouhal number results for $\mathrm{Re_D} = 160$. 148
5.15 Percentage of run time in significant subroutines for cylinder solutions ($\mathrm{Re_D} = 160$). 151
5.16 Wind tunnel conditions examined in this analysis. 153

5.17 Run time and percentage spent in subroutines for simulations using one level of AMR - Sandy Bridge system. 171
5.18 Run time and percentage spent in subroutines for simulations using two levels of AMR - Sandy Bridge system. 172
5.19 Run time and percentage spent in subroutines for simulations using two levels of AMR - Westmere system. 172
5.20 Run time and percentage spent in subroutines for simulations using three levels of AMR - Sandy Bridge system. 173
A.1 Node Derived Datatype 187
A.2 Face Derived Datatype 188
A.3 Cell Derived Datatype 189
A.4 Node Portal Derived Datatype 190
A.5 Face Portal Derived Datatype 191
A.6 Cell Portal Derived Datatype 191

List of Figures

2.1 Illustration of notional mesh with cells ih, i, ii, and iih identified. . . . 10
2.2 Illustration of lines for relaxation (gray) progressing from viscous boundaries and connecting cells. 21
3.1 Illustration of traditional, conformal mesh and non-conformal mesh with hanging-node. 25
3.2 Illustration of AMR subdivision from an original hexahedral cell. 27
3.3 Options for high-order partner selection for refined face with hanging nodes. 30
3.4 Options for high-order partner selection for coarse face with hanging nodes. 31
3.5 Illustration of DPLR lines (gray) away from viscous boundary for an initial grid and after multiple levels of refinement. 33
3.6 Example of combined array and linked list behavior. 37
3.7 Notional example of CFL ramping with AMR and NaN recovery. 41
3.8 Example of DPLR line identification and repartitioning with AMR on non-conformal mesh. Graph cell weights shown (left); DPLR lines identified in gray (right). 44
3.9 Example of refinement results on region of high curvature and small wall spacings. 52
3.10 Computational meshes with unacceptable and acceptable refinement of adjacent cells. 53
4.1 RMS error for 1-D density pulse after one cycle using uniform grids. . . 56
4.2 Computational mesh, ρ contour, and cell count for 1-D Gaussian pulse at three simulation times using AMR with a refinement tolerance of ρ_{tol}=1E-3. 57
4.3 RMS error for 1-D ρ pulse using AMR and three different values of ρ_{tol}. 59
4.4 RMS error for 2-D density pulse after one cycle using uniform grids. . . 60

4.5 Computational mesh, ρ contour, and cell count for 2-D Gaussian pulse at three simulation times using AMR with a refinement tolerance of ρ_{tol}=1E-3. Images cropped in vertical direction. 61
4.6 RMS error for 2-D ρ pulse using AMR and three different values of ρ_{tol}. 62
4.7 Parallel performance for uniform, unadapted grids. Work is consistent across all grids at a given size. 66
4.8 Parallel performance for uniform, unadapted grids as a function of cells per rank. 67
4.9 Percentage of total runtime in major sections of flow solver for unadapted grids as a function of the number of cores used. Several grid sizes shown. 68
4.10 Cut through the 3-D solution volume showing the initial density pulse and adapted grids. 69
4.11 Parallel performance for adapted grids that are as accurate as an unadapted mesh with 8 million cells. 70
4.12 Minimun, average, and maximum number of cells across all ranks using adapted grids with varying numbers of cores. Parallel efficiency as a function of cells per rank. 72
4.13 Parallel performance of iterating, gradient calculation, and communication on adapted grids and unadapted grids. 73
4.14 Runtime in major sections of flow solver for adapted grids relative to the cost of time stepping as a function of the number of cores. 75
4.15 Comparison of runtime for adapted grids and unadapted grid. 77
4.16 Runtime in major sections of flow solver for adapted grids relative to the cost of time stepping for 400 iterations. 78
5.1 Computational mesh, ρ contour, and cell count for 1-D Sod shock tube at six simulation times. 81
5.2 Comparison of instantaneous ρ on uniform and adapted grid for Sod shock tube at several times. 82
5.3 Isolines and contour of ρ in time and space for Sod shock tube; uniform grid. 84
5.4 Isolines of ρ in time and space for Sod shock tube; uniform and adapted grids. 85

5.5 Number of cells for adaptively refined solutions as a function of simulation time. 86
5.6 Schematic of important flow features in 15°-35° type VI shock-shock interaction. 88
5.7 Images of the mesh for varying levels of refinement and their effective resolutions. 90
5.8 View of cells for 512×512 uniform grid and coarse grid refined to 512×512 with AMR. 91
5.9 Surface pressure on the wedge for a series of uniform grids with varying grid densities. 92
5.10 Surface pressure on the wedge for a grids with varying levels of AMR. . 93
5.11 Cell count versus solution time for cases with three levels of refinement compared to a 256×256 unadapted reference. 95
5.12 Cell count versus solution time for cases with four levels of refinement compared to a 512×512 unadapted reference. 95
5.13 Cell count versus solution time for cases with five levels of refinement compared to a 1024×1024 unadapted reference. 96
5.14 Percent difference in integrated x-direction force for 256×256 uniform grid and those with three levels of AMR. 97
5.15 Percent difference in integrated x-direction force for 512×512 uniform grid and those with four levels of AMR. 99
5.16 Percent difference in integrated x-direction force for 1024×1024 uniform grid and those with five levels of AMR. 100
5.17 Illustration of flow features and magnitude of density gradient in numerical result. 104
5.18 Illustration of grid convergence study on uniformly refined meshes.[50] . 106
5.19 Image of adapted grids superimposed over magnitude of density gradient. 108
5.20 Comparison of wall pressure and heat flux to experimental results. . . . 110
5.21 Comparison of absolute CPU cost for uniform and adapted grids. Both linear and logarithmic y-axis shown. 111
5.22 Runtime in major routines for adapted grids relative to the cost of an average time step as a function of flow time. 112

5.23 Runtime in major routines for adapted grids relative to the cost of an average time step as a function of flow time. 113
5.24 Computational mesh colored by partition rank; contour of $|\nabla\rho|$ in black and white. 114
5.25 Parallel performance for double cone simulations. Adapted and unadapted grids shown. 116
5.26 Test configuration for ONERA shock-shock interaction test; all measurements in millimeters. 118
5.27 Mach number and temperature contours for converged simulation of the ONERA experiment. 120
5.28 Base mesh with boundary conditions for simulations of ONERA test. . . 121
5.29 Final mesh for adapted grid using 4 levels of AMR (867,961 cells). . . . 123
5.30 Comparison of x-direction force in time for uniformly refined grids. . . . 124
5.31 Results from uniform grid refinement study. 126
5.32 DL-CARS comparisons between experiment (symbols) and solutions obtained on uniformly refined grids (lines). 127
5.33 Comparison of x-direction force in time for adapted grids. 128
5.34 Results from uniform grid refinement study. 130
5.35 DL-CARS comparisons between experiment (symbols) and solutions using uniform and adapted grids (lines). 131
5.36 Comparison of absolute CPU cost for uniform and adapted grids. Both linear and logarithmic y-axis shown. 132
5.37 Cylinder base pressure as a function of Reynolds number.[72] 135
5.38 Entire coarse mesh for cylinder simulations. 137
5.39 Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $\mathrm{Re_D} = 160$ cases at various grid resolutions. 138
5.40 Illustration of extents of 'Geometric' AMR, two levels of refinement shown. 139
5.41 Illustration of extents of ω-based AMR, two levels of refinement shown. 140
5.42 Close-up of near-body mesh of cylinder at several grid resolutions. . . . 141
5.43 Comparison of integrated lift coefficient for cylinders at $\mathrm{Re_D} = 80$ with different refinement criteria. 143

5.44 Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $Re_D = 80$ cases using several adaptation criteria. 145
5.45 Comparison of cell counts for adapted grids ($Re_D = 80$). 146
5.46 Comparison of absolute CPU cost for adapted grids. Both linear and logarithmic y-axis shown ($Re_D = 80$). 147
5.47 Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $Re_D = 160$ cases using several adaptation criteria. 149
5.48 Comparison of cell counts for adapted grids ($Re_D = 160$). 150
5.49 Comparison of absolute CPU cost for adapted grids. Both linear and logarithmic y-axis shown ($Re_D = 160$). 150
5.50 Image of pitch plane in baseline mesh with no refinement. 154
5.51 Image of cylindrical volume refinement in the pitch plane; three levels of refinement shown. 156
5.52 Image of feature-based refinement in the pitch plane; three levels of refinement shown. 157
5.53 Near-capsule grid and capsule surface for initial grid (no AMR). 158
5.54 Near-capsule grid and capsule surface for coarse grid (1 level AMR). . . 159
5.55 Near-capsule grid and capsule surface for medium grid (2 levels AMR). . 160
5.56 Near-capsule grid and capsule surface for fine grid (3 levels AMR). . . . 161
5.57 Evidence of interpolation between baseline grid nodes in fine grid. . . . 162
5.58 Instantaneous and average drag coefficient for capsule simulations using several grid resolutions. 164
5.59 Instantaneous and average lift coefficient for capsule simulations using several grid resolutions. 165
5.60 Convergence of drag and lift coefficients with additional refinement. . . 166
5.61 Visualization of coarse grid solution. 167
5.62 Progression of solutions on finer grids with one, two, and three levels of geometric refinement. 168
5.63 Progression of solutions on finer grids with one, two, and three levels of feature-based refinement. 169
5.64 Cell count for capsule simulations using both techniques. 170

Chapter 1

Introduction

1.1 Motivation

Modern aerospace designs are complex. They can include large flight envelopes for which environments are required and the resulting flowfields contain a range of length scales due to high-Reynolds number conditions and ever-more faithful representation of vehicle geometries. Computational Fluid Dynamics (CFD) has become a stable in modern engineering. It is considerable less expensive than ground and flight test and can enable simulation at conditions that are outside the capabilities of other methods. Over the last several decades, CFD has been employed on more and more challenging tasks and dependance on the tools continues to grow.

In order to perform accurate simulations, engineers and researchers must first create a discretized spatial domain. This includes well-resolved surface and volume grids surrounding the interior or exterior mold lines of the geometry under analysis. Errors can manifest in a number of ways and ideal grid topologies for even simple shapes can become quite involved. Grids that are misaligned with flow features or are too coarse are a dominant source of errors in the solution and compromise results. Determining where to best allocate grid points (and as a consequence, simulation time) is an important step when creating a new computational mesh.

Grid generation understandably demands a significant portion of the user's time. Engineers manually create grid topologies, dictate cell spacings, and rely on judgement

and experience to create quality discretizations. Unfortunately, even with state-of-the-art mesh generation tools, there is much in this process that is subjective. Grid resolution studies are necessary to remove subjectivity and ensure that a flowfield is properly discretized. Unfortunately, they are not always performed due to resources constraints.

Changes in the flow environment (Mach number, Reynolds number, or vehicle attitude) modify the resulting solution, moving separation location, affecting shock alignment, and disturbing other important features that were once well-resolved but now lie in a coarser region of the domain. These changes necessitate modifications to the grid and, depending on the topology, several grid systems may be required for a given analysis. One strategy to minimize the number of modifications to the grid is to refine large portions of the grid in order to create larger regions of the grid with adequate refinement to track features that move with changing conditions. This is inefficient, though, and creates an overhead cost for all solutions.

To reduce the demands of grid generation, aspects of it are frequently automated. Several methods see widespread use depending on flow solver and grid technology. Solvers built to use overset grid methods allow for grids to move relative to others in order to follow moving features. Unstructured solvers allow for topological flexibility by arbitrarily meshing volumes between more structure regions. Automated grid translation and smoothing is also used to improve quality in both structured and unstructured grids. While each of these methods can remove some of the subjectivity from the grid generation process, they frequently involve a non-trivial amount of time to properly set-up mechanisms for automation or require more involved numerics and additional time for the flow computation.

A more important problems with conventional grid generation is that it requires a priori knowledge of where to place grid points for accurate computation. Complex and unsteady flows require much more expertise. Inappropriate grid generation and subsequent simulation requires additional work be allocated to both the grid and an additional simulation. Having tools and strategies to correct the grid during the simulation are critical to mitigate these costs and reduce the effort required for grid generation.

This dissertation focuses on leveraging computational resources to automatically refine and adapt grids based on user-provided inputs. This idea is not new and was

pioneered beginning with the work of Berger, Oliger, and Colella.[10;11] Their initial work adapted structured grids for finite-diference flow solvers. Other researchers have shown success with unstructured grids, overset grids, and a number of numerical approaches.

The technique is frequently referred to as Adaptive Mesh Refinement (AMR). At the highest level, the goal of AMR is to select a subset of the computational volume that contains important flow features or has a measurably high error and refine it. This refinement reduces the local error and does not demand a modification of the overall grid topology or attention from the analyst. Most methods allow refinement to move and track unsteady phenomena or a developing solution. If appropriate feature detection or error estimation is used, then strict requirements for a priori knowledge of the flow behavior disappear. This allows the use of an initially coarse and more general grids to be used as a basis for several cases. They are each refined as needed by the flow solver and tailored to the specific flight condition.

In this document, AMR is applied to high-order, implicit flow solver for use with unsteady computations. It extends ideas currently employed in state-of-the-art solvers by including considerations for grids with hanging-nodes or non-conformal boundaries between grid cells of different refinement. Furthermore, it is developed to be efficient for scalable, highly-parallel deployment on modern computer clusters and is focused on providing an example implementation to augment existing flow solvers. Applications to unsteady motivational problems with differing grid topologies are shown.

The purpose of this work is threefold:

1. Show that AMR and high-order numerical stencils are not mutually exclusive and demonstrate higher than second-order results on grids with hanging nodes.

2. Provide a reference implementation for AMR on unstructured grids that is amenable to modern finite-volume numerical methods and is scalable to large computational clusters.

3. Apply the developed procedures and flow solver to a range of motivational problems and compare the results to conventional grids.

These goals are accomplished by creating a new, finite-volume flow solver and augmenting it with an adaptive mesh capability. The resulting solver is heavily instrumented

in order to allow code profiling and facilitate quantitative discussions on algorithm cost. Computational examples are provided for solutions of the Euler and Navier-Stokes equations with explicit and implicit time-stepping across a range of environments, grid topologies, and numerical methods.

Portions of this work have been revised and expanded from two previously presented conference papers by Schwing, Nompelis, and Candler.[61;62] Much of the text from these papers has been modified and reincorporated into a cohesive document reflecting updated understanding, improvements, and new results. In particular, the numerical results from the 2013 paper concerning the double wedge have been recreated due to significant changes in the flow solver between that work and what is described in this document (mostly the use of linked lists and MPI parallelization).

1.2 Chapter Summaries

- Chapter 1 introduces the motivations and objective pursued in this dissertation. It also contains the chapter summaries.

- Chapter 2 outlines the underlying numerical methods and physical models used in this work. It includes initial descriptions of these numerics without augmentation or consideration for grids with hanging nodes.

- Chapter 3 introduces the techniques used in this work for adaptive mesh refinement. Memory management, feature detection, and parallelization are discussed. It also outlines the extensions to the methods described in Chapter 2 necessary for non-conformal meshes.

- Chapter 4 provides verification of the underlying solver and grids obtained with refinement. Results from solutions with and without adapted grids are compared for high-order numerical fluxes. Scalability for the methods outlined previously up to 512 processors is quantified for grids with and without AMR.

- Chapter 5 contains application of the flow solver to several problem. It illustrates advantages over conventional grid for motivational problems. The selected applications cover subsonic, supersonic, and hypersonic flows, simple and complicated

grid topologies, and both steady and unsteady flowfields. Comparisons are made to experimental data were available.

- Chapter 6 presents a final discussion of the work presented in this dissertation.

Chapter 2

Numerical Methods

2.1 Governing Equations

This work focuses on analysis of continuum fluid flow for a thermally and calorically perfect gas. Gas temperatures are below what is required for vibrational or rotational excitation of the gas. The compressible Navier-Stokes equations adequately describe the behavior of flows at these conditions and are written in conservation law form:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0 \tag{2.1}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}\Big(\rho u_i u_j + p\delta_{ij} - \tau_{ij}\Big) = 0 \tag{2.2}$$

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x_j}\Big((E+p)u_j - \tau_{ij}u_i + q_j\Big) = 0 \tag{2.3}$$

In the first equation, the conservation of mass, ρ is the fluid density and u_i is the velocity component in the i-direction, traditionally u, v, and w for $i = 1$, 2, and 3.

The second equation, the conservation of momentum, introduces pressure, p, the Krönecker delta function, δ_{ij}, and the shear stress tensor, τ_{ij}. For a thermally perfect gas, pressure is assumed to be related to the density and temperature by the ideal gas law, $p = \rho RT$ with R begin the specific gas constant (286.9 $[\frac{m^2}{s^2 K}]$ for air).

For a calorically perfect gas, the specific heat at constant pressure (c_p) and constant volume (c_v) are assumed to be invariant with respect to temperature. They are related

to each other by the specific gas constant, R and the ratio of specific heats, γ.

$$R = c_p - c_v \qquad\qquad \gamma = \frac{c_p}{c_v}$$

τ_{ij} is assumed to the proportional to the first derivatives of the velocity,

$$\tau_{ij} = \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) + \lambda\frac{\partial u_k}{\partial x_k}\delta_{ij}$$

where μ is the viscosity and is described the by Sutherland's Law,

$$\mu(T) = \mu_\circ\frac{T^{\frac{3}{2}}}{T + T_\circ}$$

with coefficients $\mu_\circ = 1.458 \times 10^{-6}$ $[\frac{kg}{ms}]$ and $T_\circ = 110.3$ [K]. The second viscosity coefficient is λ and is equal to $-2/3\mu$ for most fluids and conditions.

The third equation is the conservation of energy. E is the total energy of the fluid. E is the sum of the internal energy, $\rho c_v T$, and the kinetic energy, $\frac{1}{2}\rho u_i u_i$. The heat transfer, q_j is assumed to be given by Fourier's law of heat conduction,

$$q_j = \kappa\frac{\partial T}{\partial x_j}$$

where κ is related to the fluid viscosity through the Prandtl number, $\mathrm{Pr} = \frac{c_p\mu}{\kappa}$. In most environments, the Prandlt number for air is approximately 0.72.

2.2 Finite-Volume Formulation

The Navier-Stokes equations can be re-written in a simplified form amenable to more compact manipulation,

$$\frac{\partial U}{\partial t} + \frac{\partial F_j}{\partial x_j} = 0, \tag{2.4}$$

where,

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{pmatrix} \quad \text{and} \quad F_j = \begin{pmatrix} \rho u_j \\ \rho u u_j + p\delta_{1j} - \tau_{1j} \\ \rho v u_j + p\delta_{2j} - \tau_{2j} \\ \rho w u_j + p\delta_{3j} - \tau_{3j} \\ (E+p)u_j - \tau_{ij}u_i + q_i \end{pmatrix}. \tag{2.5}$$

U is the vector of conserved variables and F_i is the vector of fluxes. This work solves the compressible Navier-Stokes equations with a finite-volume scheme. Investigating the equations in the context of a discrete volume, V, and a closed boundary, ∂V, allows for a discrete representation of the governing equations. Integrating over this volume and applying Gauss' theorem, the resulting volume integral is transformed into a surface integral of the fluxes on the boundary. In equation form,

$$\int_V \frac{\partial U}{\partial t} dV = -\int_V \frac{\partial F_j}{\partial x_j} dV$$

is equivalent to

$$\int_V \frac{\partial U}{\partial t} dV = -\oint_{\partial V} \vec{F} \cdot d\vec{S}. \tag{2.6}$$

This work solves a weak form of the conservation equation by replacing the integrated conserved variable in each discrete volume, or cell, with the cell-averaged values, $\bar{U}$. Furthermore, the surface of each cell is discretized into a number of faces. Each face has an outward-facing normal vector, $\vec{n}$ and an area, S. Applying Eq. (2.6) to this discrete model, a form of the equations changes to

$$V\frac{\partial \bar{U}}{\partial t} + \sum_{faces} \left[\vec{F} \cdot \hat{n} S\right] = 0. \tag{2.7}$$

A useful feature of the finite-volume formulation is that it allows for arbitrary polyhedra with a given volume and and arbitrary number of bounding faces. This flexibility

allows for extension of traditional numerical approaches to grids with non-uniform refinement and adaptation. The specifics of the extension are discussed later, but the fundamental flexibility of the finite-volume discretization dovetails nicely into adaptive mesh refinement with non-conformal meshes.

It is convenient to divide the flux vector, $\vec{F}$, into the convective and viscous portions. This is because the nature of these components lend themselves to separate numerical approaches. The convective flux vector, $\vec{F}_c$, contains the advection and pressure terms and is hyperbolic in character. The viscous flux vector, $\vec{F}_v$, is elliptic and includes the viscosity and heat flux terms. When solving the inviscid Euler equations, $\vec{F}_v$ is removed from the governing equations. Equation (2.7) can be rewritten as

$$V\frac{\partial \bar{U}}{\partial t} + \sum_{faces}\left[\vec{F}_c \cdot \hat{n}S\right] + \sum_{faces}\left[\vec{F}_v \cdot \hat{n}S\right] = 0. \tag{2.8}$$

where

$$\bar{U} = \begin{pmatrix} \bar{\rho} \\ \bar{\rho u} \\ \bar{\rho v} \\ \bar{\rho w} \\ \bar{E} \end{pmatrix} \quad \vec{F}_c = \begin{pmatrix} \rho u_j \\ \rho u u_j + p\delta_{1j} \\ \rho v u_j + p\delta_{2j} \\ \rho w u_j + p\delta_{3j} \\ (E+p)u_j \end{pmatrix} \quad \vec{F}_v = \begin{pmatrix} 0 \\ -\tau_{1j} \\ -\tau_{2j} \\ -\tau_{3j} \\ -\tau_{ij}u_i + q_i \end{pmatrix}. \tag{2.9}$$

The cell-averaged conserved variables in Eq. (2.8) are assumed to be at the center of the cell for geometric operations. Similarly, the flux vector is evaluated at the centroid of the bounding face, but is considered constant across the entire face area. There are an infinite number of methods for interpolation or extrapolation to the face centroid from the neighboring cell-averaged solution data.

For the discussions that follow, Fig. 2.1 shows a portion of a notional computational mesh with neighbors to the left and right of a face. The cells are identified by their indices, ih, i, ii, and iih with their cell centers indicated. These designations are relative to the highlighted face where the flux will be evaluated. The face normal for this face is also shown. In the subsequent discussion, all face normals are assumed to be outward of cell i. The numerical methods used in this work are based on those described by Candler *et al.*[18].

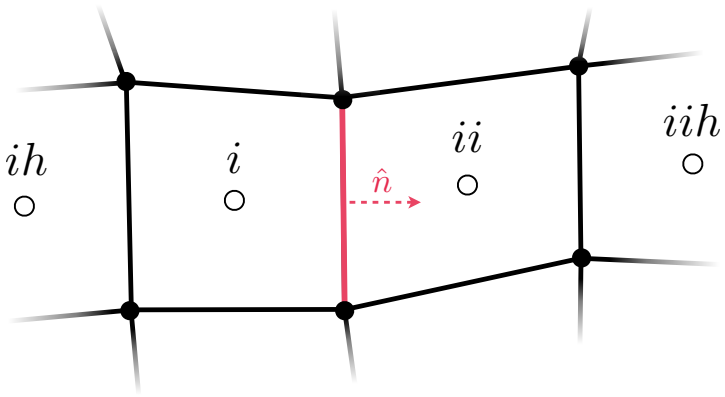


Figure 2.1: Illustration of notional mesh with cells ih, i, ii, and iih identified.

2.3 Inviscid Fluxes

For supersonic flow, the inviscid, convective fluxes are hyperbolic in character with information traveling along characteristics. Furthermore, the fluxes are linear and homogenous with respect to the vector of conserved variables, $U = (\rho, \rho u, \rho v, \rho w, E)$. Due to this, it is equivalent to relate the inviscid flux, $\vec{F_c}$, to an arbitrarily scaled vector of conserved variables, U, like so,

$$\vec{F_c}(\lambda U) = \lambda \vec{F_c}(U).$$

Furthermore, the fluxes' homogeneity allows the fluxes to be linearized with respect to the conserved variables by introducing the jacobian, A,

$$\vec{F_c}(U) = \frac{\partial \vec{F_c}}{\partial U} U = AU.$$

To analyze the characteristics of the inviscid fluxes, it is convenient to examine them with respect to the vector of primitive variables, $V = (\rho, u, v, w, p)$. The specifics of the derivations are only briefly covered here. In developing numerical methods for the inviscid fluxes, the flux jacobian is modified by use of a transformation matrix, $S = \frac{\partial V}{\partial U}$ and its inverse. The flux jacobian A is decomposed into,

$$A = \frac{\partial U}{\partial V}\frac{\partial V}{\partial U}\frac{\partial \vec{F_c}}{\partial V}\frac{\partial V}{\partial U} = S^{-1}MS,$$

where the matrix, M, is diagonalized to find the matrix of eigenvalues, Λ,

$$\frac{\partial V}{\partial U}\frac{\partial \vec{F_c}}{\partial V} = M = C^{-1}\Lambda C.$$

The matrix C is comprised of the eigenvectors. For the Euler equations, the eigenvalues are a, $u' + a$, and $u' - a$, where a is the speed of sound and u' is the component of the velocity normal to the face where the flux is being calculated. Once the fluxes have been decomposed into this form, they are split into positive and negative components,

$$\Lambda^{\pm} = \frac{\Lambda \pm |\Lambda|}{2}.$$

The positive components, Λ^+, act in the direction of the face normal and the negative components, Λ^-, are anti-parallel to the face normal. They are also referred to as left- and right-running characteristics, respectively. By using these decomposition, the original inviscid flux at the face can be represented in terms of two fluxes,

$$\begin{array}{cc} \vec{F_c} = F_c^+ + F_c^- & \text{where} \\ \vec{F_c^+} = A^+U = S^{-1}C^{-1}\Lambda^+CSU & \text{and} \\ \vec{F_c^-} = A^-U = S^{-1}C^{-1}\Lambda^-CSU. & \end{array}$$

the subscript c and vector notation have been dropped from the two fluxes, $\vec{F_c^+}$ and $\vec{F_c^-}$, for convenience in the following notation.

In summary, F_c^+ and F_c^- are the upwinded components of the inviscid flux and are in the direction of the positive and negative running eigenvalues with respect to the face. To numerically solve for these fluxes, a first-order approximation can be made; this method and result are referred to as Steger-Warming fluxes.[66] Referring back to Fig. 2.1, the upwinded Steger-Warming fluxes for the face would be,

$$F^+_{i+1/2} = A^+_i U_i \quad \text{and} \quad F^-_{i+1/2} = A^-_{ii} U_{ii}. \tag{2.10}$$

In (2.10), the values at the jacobians are evaluated from values in the neighboring cells and the solution quantities in those cells are used for the left and right components of the solution. This method had been found to be more dissipative that necessary for certain conditions. The Modified Steger-Warming fluxes are very similar in form

but provide less dissipation and evaluate the jacobian by first averaging the primitive variables on either side of the face,[17]

$$F^+_{i+1/2} = A^+_{i+1/2} U_i \quad \text{and} \quad F^-_{i+1/2} = A^-_{i+1/2} U_{ii}. \tag{2.11}$$

Higher-order approximations of the inviscid fluxes are possible by finding a more accurate prediction for the solution at the cell face. In the previous equations, the solution at the face is assumed to be equal to the solutions in the neighboring cells, U_i and U_{ii}. This is equivalent to first-order extrapolation to the face. Rewriting (2.11) with U_L and U_R as the solution variables from the left- and right-side of the face provides

$$F^+_{i+1/2} = A^+_{i+1/2} U_L \quad \text{and} \quad F^-_{i+1/2} = A^-_{i+1/2} U_R. \tag{2.12}$$

This is at best second-order. On an uniform mesh, a second-order accurate prediction for U_L and U_R, with cells identified as in Fig. 2.1, would be

$$U_L = \frac{3U_i - U_{ih}}{2} \quad \text{and} \quad U_R = \frac{3U_{ii} - U_{iih}}{2}. \tag{2.13}$$

Conventionally, a higher-order MUSCL reconstruction is used to calculate U_L and U_R. A Method of gradient reconstruction is also useful and has particular benefit for unstructured grids with non-hexahedral elements.[16].

Additional manipulations of the upwinded fluxes present opportunities for flux calculations that are higher than second-order accurate. First, it is necessary to rewrite the Modified Steger-Warming fluxes as a central and upwinded component.[67] Starting with (2.12),

$$\begin{aligned}
F_{i+1/2} &= F^+ + F^- = A^+ U_L + A^- U_R \\
&= \left(S^{-1}C^{-1}\Lambda^+ CS\right)\cdot U_L + \left(S^{-1}C^{-1}\Lambda^- CS\right)\cdot U_R \\
&= \left(S^{-1}C^{-1}\tfrac{\Lambda+|\Lambda|}{2}CS\right)\cdot U_L + \left(S^{-1}C^{-1}\tfrac{\Lambda-|\Lambda|}{2}CS\right)\cdot U_R \\
&= \left(S^{-1}C^{-1}\Lambda CS\right)\cdot\tfrac{(U_L+U_R)}{2} + \left(S^{-1}C^{-1}|\Lambda|CS\right)\cdot\tfrac{(U_L-U_R)}{2} \\
F_{i+1/2} &= \underbrace{A\cdot\frac{(U_L+U_R)}{2}}_{central} + \underbrace{|A|\cdot\frac{(U_L-U_R)}{2}}_{upwinded}
\end{aligned}$$

This new form conveniently presents the inviscid flux as a combination of a central

difference across the face and an upwinded component. The upwinded component is dissipative and for ideal computations, could be removed in order to limit numerical dissipation and error in the solution. However, the numerical dissipation is stabilizing and it is necessary for flows with discontinuities (such as a shock wave). A technique that allows for inclusion of the dissipative portion of the flux includes a new parameter, Φ, that modulates the inclusion of numerical dissipation,

$$F = F_{\text{central}} + \Phi \cdot F_{\text{upwinded}}.$$

The term Φ is a sensor that goes to zero in regions of the flow where no dissipation is required. For hypervelocity flows, the Ducros shock-sensing switch is a practical choice for Φ,

$$\Phi = \frac{(\nabla \cdot \vec{u})^2}{(\nabla \cdot \vec{u})^2 + \omega^2 + \epsilon}.$$

It depends on the ratio of dilation to dilation and vorticity in a cell.[26] A small number, ϵ, is added to the demonenator to prevent division by zero. Dissipation is added predominantly in regions of high compression (shock waves), where Φ approaches unity. This sensor is designed to limit numerical dissipation in regions with high vorticity, such as a turbulent wake or shear flow. Other sensors for dissipation could depend on large gradients in flow variables or other problem-specific criteria.

Research has shown that a high-order accurate central flux can be derived for the central portion of the flux. This formulation removes the dependance on the jacobian and instead replaces it with a central stencil that requires information from the neighbors, i and ii, as well as their second neighbors, ih and iih. There are second-, fourth-, and sixth-order accurate central fluxes and all are used in this work. The fluxes obey a secondary conservation of kinetic energy and are referred to in this document as Kinetic Engergy Consistant (KEC) fluxes. The KEC fluxes are less dissipative than the MSW fluxes and allow for more accurate resolution of small-scale structures and a broader energy spectrum.[6]

Depending on the order, solution quantities and their spatial derivatives are required when extrapolating to obtain face-located quantities. The general form of the stencil to obtain a solution variable, ϕ, at the face, ϕ_f, is

$$\phi_f = \alpha\left(\phi_i + \phi_{ii}\right) + \beta\left(\nabla\phi_i \cdot \vec{x_i} + \nabla\phi_{ii} \cdot \vec{x_{ii}}\right) + \gamma\left(\nabla\phi_{ih} \cdot \vec{x_{ih}} + \nabla\phi_{iih} \cdot \vec{x_{iih}}\right).$$

Where α, β, and γ are weights unique to the order of the method, and have been discussed at length elsewhere.[6] The primitive values at the face, ϕ_f, are used to calculate the convective flux directly using (2.9). Dissipation is added where identified by the switch in (2.3).

Finally, another modification to the fluxes is the inclusion of a sonic glitch correction. The sonic glitch correction is a modification made to the magnitude of the eigenvalues when evaluating the jacobian. For the inviscid fluxes, issues arise when the magnitude of an eigenvalue approaches zero - sonic or stagnation regions. At the stagnation region, error can accumulate and lead to a 'carbuncle'. To alleviate these issue, the eigenvectors ($\Lambda^{\pm}$) are replaced with

$$\Lambda^{\pm} = \frac{\Lambda^{\pm} \pm \sqrt{\left(\Lambda^{\pm}\right)^2 + \left(a\epsilon\right)^2}}{2},$$

where ϵ is a small, user-defined value - typically with a magnitude near 1E-1 and a is the local speed of sound.

2.4 Viscous Fluxes and Cell Gradients

The viscous fluxes are computed at each face and as implemented here do not contain as much ambiguity as the inviscid fluxes. Scalar values and gradient of the solution variables are averaged to the face centroid and used in a direct calculation of the viscous flux, F_v. Viscous jacobians are calculated as required for implicit time integration.

Viscous terms depend on the spatial derivatives of the flow quantities at the face. In order to approximate these derivatives, the cell-centered gradients are extrapolated to the location of the face centroids. Identical to the work described by Nompelis *et al.*, these cell-centered gradients are averaged at the faces and improved with application of deferred correction.[43;51].

Both the viscous fluxes and the inviscid, KEC fluxes require the cell gradients throughout the flowfield. For an unstructured mesh, calculating the spatial gradients is

not always straightforward. This approach used here is a weighted least squares method that includes all cells neighboring a cell whose gradient is required. In brief, to find the gradient in three-dimensions for cell $c_\circ$ with m neighboring cells $(c_1 ... c_m)$ the following matrix solution is required to determine the values of ϕ_x, ϕ_y, and ϕ_z - the spatial derivatives for variable ϕ.

$$\begin{bmatrix} \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}^2}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}\Delta y_{\circ,j}}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}\Delta z_{\circ,j}}{w_{\circ,j}} \\ \sum_{j=1}^{m} \frac{\Delta y_{\circ,j}\Delta x_{\circ,j}}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta y_{\circ,j}^2}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta y_{\circ,j}\Delta z_{\circ,j}}{w_{\circ,j}} \\ \sum_{j=1}^{m} \frac{\Delta z_{\circ,j}\Delta x_{\circ,j}}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta z_{\circ,j}\Delta y_{\circ,j}}{w_{\circ,j}} & \sum_{j=1}^{m} \frac{\Delta z_{\circ,j}^2}{w_{\circ,j}} \end{bmatrix} \begin{bmatrix} \phi_x \\ \phi_y \\ \phi_z \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}\Delta \phi_{\circ,j}}{w_{\circ,j}} \\ \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}\Delta \phi_{\circ,j}}{w_{\circ,j}} \\ \sum_{j=1}^{m} \frac{\Delta x_{\circ,j}\Delta \phi_{\circ,j}}{w_{\circ,j}} \end{bmatrix},$$

where $\Delta x_{i,j}$ is the distance in the x-direction between the cell centers of c_i and c_j, $(x_j - x_i)$. $\Delta y_{i,j}$ and $\Delta z_{i,j}$ are similar for the y- and z-directions. The difference between the scalar values, $(\phi_j - \phi_i)$, is $\Delta\phi_{i,j}$. A weighting parameter, $w_{i,j}$ is included as well. For this work, it is the square of the distance between cell centers i and j.

The solution to the least squares calculation involves a matrix solve with the right-hand side updated every timestep to reflect the changing flow quantities, ϕ. Fortunately, the left-hand matrix is only a function of the geometry and can be inverted once and stored. This matrix does need to be re-inverted if the mesh changes due to a dynamic grid event - refinement or coarsening of cells, movement of grid points, or repartitioning of the domain.

2.5 Time Integration

2.5.1 Explicit

Recall the finite-volume formulation presented in Eq. (2.8),

$$V\frac{\partial \bar{U}}{\partial t} + \sum_{faces}\left[\vec{F}_c \cdot \hat{n}S\right] + \sum_{faces}\left[\vec{F}_v \cdot \hat{n}S\right] = 0.$$

Rearranging the terms and solving for the rate of change of the cell-averaged quantities $\bar{U}$ gives

$$\frac{\partial \bar{U}}{\partial t} = -\frac{1}{V}\sum_{faces}\left[\vec{F}_c \cdot \hat{n}S\right] + \frac{1}{V}\sum_{faces}\left[\vec{F}_v \cdot \hat{n}S\right].$$

Using a first-order, explicit time integration scheme between time level n and $n+1$, $\frac{\partial \bar{U}}{\partial t}$ is approximated as $\frac{\bar{U}^{n+1}-\bar{U}^n}{\Delta t}$. For such a scheme, the fluxes are calculated at the current time level, n. Separating the convective flux into the positive- and negative-running upwinded fluxes, this method can be written as

$$\bar{U}^{n+1} = \bar{U}^n - \frac{\Delta t}{V}\sum_{faces}\left[\left(\vec{F}_- + \vec{F}_+\right)^n \cdot \hat{n}S\right] + \frac{\Delta t}{V}\sum_{faces}\left[\left(\vec{F}_v\right)^n \cdot \hat{n}S\right] = \bar{U}^n + \Delta\bar{U} \quad (2.14)$$

with $\Delta\bar{U}$ being shorthand for the explicit update to the cell-averaged value.

It is simple to extend (2.14) for multi-step methods, such as the Runge-Kutta family of methods, to achieve higher than first-order. A third-order accurate Runge-Kutta scheme by Gottlieb and Shu is used in this work.[32] It is also possible to use information from previous time steps ($^{n-1}$,$^{n-2}$, ...) to improve the order of the error, but schemes of this type are not used here. For explicit methods, the main theme is that the flux values are calculated at the current iteration or stored from prior iterations.

2.5.2 Implicit

Explicit integration using (2.14) is stable for time step sizes limited to a maximum allowable time step given by the Courant-Friedrichs-Lewy (CFL) condition. In general, the requirement states that $\Delta t \leq \frac{l}{\lambda}$ with l being a cell's smallest linear dimension and λ the magnitude of its largest eigenvalue. Grids with small cells or very large velocities (for the Euler and Navier-Stokes equations) require very small time steps in order to remain stable.

For viscous problems, well-resolved grids have very fine near-wall cell spacing, the stable time step is very small and can make many problems intractable with explicit time stepping. Most motivational problems involve hypersonic or viscous flows with similar demands on stability. For steady-state computations where temporal error does not influence the result, it is efficient to iterate using a time step much larger than the maximum stable explicit value. To do this, it is necessary to introduce implicit methods.

This work considers two, Full-Matrix Point-Relaxation (FMPR) or 'point implicit' and line implicit.[74] Line implicit implemented for large parallel computation is also referred to as Data-Parallel Line Relaxation (DPLR).

The choice of implicit method strongly influences the convergence properties and robustness of the resulting Navier-Stokes solver.[48] Studies of numerical results have included FMPR (point implicit) and DPLR (line implicit) as well as Krylov methods and GMRES solvers. Other researchers working on AMR have had success with a variety of implicit approaches. Popular for both finite-volume and finite-element solvers employing AMR are Newton-Krylov implicit methods.[12;35;44]. These methods are parallel and provide a robust capability for implicit time advancement. Using the PETSc library, researchers can leverage existing modular subroutines for robust application of these methods in existing codes.[4]

Instead of using the PETSc libraries, the point and line implicit methods used here tightly couple the linear solve with the data structures already in the code and takes advantage of the flow physics inherent to the governing equations. They has also been demonstrated to require a smaller memory overheard as compared to the PETSc libraries by using data structures already in the flow solver.[52] In regions of the flow where there is a strong coupling to adjacent cells, viscous boundary layers for instance, line implicit is favorable. Far from viscous walls, an unbiased point implicit operator is favorable. The two method are compatable and in a given solution it is likely that both are used.

The explicit time integration presented earlier can be made implicit by evaluating all of the numerical fluxes at the future time level, $n+1$, instead of the current time level, n. For the inviscid convective fluxes, F_c, a first-order linearization is performed,

$$\begin{aligned}
F_c^{n+1} &= F_c^n + \frac{\partial F}{\partial \bar{U}}^n \left(\bar{U}^{n+1} - \bar{U}^n\right) + \mathcal{O}(\Delta t^2) \\
&= F_c^n + A_+^n \underbrace{\left(\bar{U}_i^{n+1} - \bar{U}_i^n\right)}_{\delta \bar{U}_i} + A_-^n \underbrace{\left(\bar{U}_{ii}^{n+1} - \bar{U}_{ii}^n\right)}_{\delta \bar{U}_{ii}} \\
F_c^{n+1} &= F_c^n + A_+^n \delta \bar{U}_i + A_-^n \delta \bar{U}_{ii}
\end{aligned}$$

where A_+^n and A_-^n are the right- and left-running flux jacobians at the face. There is a slight nomenclature change from that used in previous sections where left- and right-running jacobians now use a subscript $\pm$ so as to not conflict with the superscripts ^{n}

and $^{n+1}$. $\delta\bar{U}_i$ and $\delta\bar{U}_{ii}$ are the implicit updates to the conserved variables with i being the index of the current cell and ii the index of its face neighbor.

The viscous fluxes are also approximated at the future time level $n+1$. To do this, we introduce an approximate viscous jacobian, M, and a rotation matrix, R. It is also necessary to define a matrix, N, that transforms from the conserved variables to the primitive variables. This results in:

$$\begin{aligned} F_v^{n+1} &= F_v^n + \delta F_v^n \\ &= F_v^n + \left(R^{-1}MR\right)\frac{\partial}{\partial\eta}\left(N^n\delta U^n\right) \end{aligned}$$

Where the derivative, $\frac{\partial}{\partial\eta}$, is in the face-normal direction. The linear system that results from combining the convective and viscous fluxes with the governing finite volume formulation is:

$$\delta\bar{U}_i + \frac{\Delta t}{V_i}\sum_{faces}\left[\left(A_+^n\delta\bar{U}_i + A_-^n\delta\bar{U}_{ii} + \left(R^{-1}MR\right)N\left(\delta\bar{U}_i - \delta\bar{U}_{ii}\right)\right)\cdot\hat{n}S\right] = \Delta\bar{U}_i \quad (2.15)$$

By storing the Jacobian matrices and performing an iterative solution procedure, the values for $\delta\bar{U}_i$ are obtained based on the explicit flux, $\Delta\bar{U}_i$.

The result is a very large matrix of equations that must be solved simultaneously. Fortunately, the nature of the fluxes is local and each cell only depends on the immediately adjacent cells. The local stencil results in a matrix that is sparse and contains predominantly zeros in the off-diagonal terms. Point and line implicit methods as explored in this work instead use relaxation to iteratively solve the matrix of equations.

Point Implicit

The point implicit method is an unbiased, iterative implicit solve. It uses a block-diagonal matrix to solve Eq. (2.15) while maintaining tight coupling for all degrees of freedom within a computational cell. Furthermore, since the method has no bias in how the implicit linear system of equations is relaxed to convergence, it is suitable for unsteady, time-accurate simulations.

Equation (2.15) shows the coupling of each cell to its neighboring cells. It can be manipulated to highlight this by identifying the relationships between a cell i and the

j faces that separate it from its m adjacent neighbors. Combining like terms,

$$\left[I + \frac{\Delta t}{V_i}\sum_{j=1}^{m}\left[A_+^n + \left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]\delta\bar{U}_i + \frac{\Delta t}{V_i}\sum_{j=1}^{m}\left[\left[A_-^n - \left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]\delta\bar{U}_j = \Delta\bar{U}_i$$

The solution for $\delta\bar{U}_i$ is initialized to zero and iteratively solved with k sub-iterations at each time step. Non-$\delta\bar{U}_i$ terms are relaxed and evaluated using the previous sub-iteration's solution variables. Applying this relaxation and moving all relaxed terms to the right-hand side and simplifying,

$$\underbrace{\left[\frac{V_i}{\Delta t} + \sum_{j=1}^{m}\left[A_+^n + \left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]}_{\hat{A}_i}\delta\bar{U}_i^{k+1} = \underbrace{\frac{V_i}{\Delta t}\Delta\bar{U}_i}_{\Delta\bar{U}_i^*} - \sum_{j=1}^{m}\underbrace{\left[\left[A_-^n - \left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]}_{\hat{B}_{i,j}}\delta\bar{U}_j^k$$

In matrix form, the relaxed equation is shown below. Typically, no more than five k iterations are required for good sub-iteration convergence. The left-hand side is inverted once using a block diagonal algorithm at the start of the sub-iterations and stored. One stored, it can be used to update $\delta\bar{U}_i$ based on the right-hand side. Each k iteration, the right-hand side is updated in order to propagate the influence of the change in adjacent

cells (via $\delta\bar{U}_j^k$).

$$\begin{bmatrix} \ddots & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \hat{A}_i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \ddots \end{bmatrix} \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ \partial\bar{U}_i \\ \vdots \\ \vdots \\ \vdots \end{pmatrix}^{k+1} = \begin{pmatrix} \vdots \\ \vdots \\ \vdots \\ \Delta\bar{U}_i^* - \sum_{j=1}^{m} \hat{B}_{i,j}\delta\bar{U}_j^k \\ \vdots \\ \vdots \\ \vdots \end{pmatrix}^{k} \tag{2.16}$$

Line Implicit

This work also employs a second implicit method, the line implicit operator also referred to as line relaxation. When investigating the viscous Navier-Stokes equations, there are even more strenuous demands placed on the implicit solver due to the extremely small cell spacings near viscous walls.

The line relaxation operator tightly couples the linear solve with the data structures already in the code and takes advantage of the flow physics inherent to the Navier-Stokes equations. It has also been demonstrated to require a smaller memory overheard as compared to the PETSc libraries by using data structures already in the flow solver.[52]

We employ Data-Parallel Line Relaxation (DPLR) when contiguous lines of cells can be grown from viscous walls. In regions of the domain where such lines do not exist, the Full-Matrix Point-Implicit method is used. For more details, refer to prior publications on the specifics.[51;74]. Both implicit methods are amenable to distributed computation.

Figure 2.2 shows a sample mesh that is bounded on four sides by viscous walls. To take advantage of the strong coupling near the walls, line relaxation is used in the wall-normal directions to couple contiguous lines of cells. Where multiple lines intersect at a cell, the cell is arbitrarily ascribed to only one line; the other line terminates. In this example, lines extend throughout the domain until truncated by other lines. More generally, the user can designate a maximum line length. Cells further than that length away from the walls use point relaxation on their faces.

For a fixed, unstructured grids, lines are identified once - before time marching

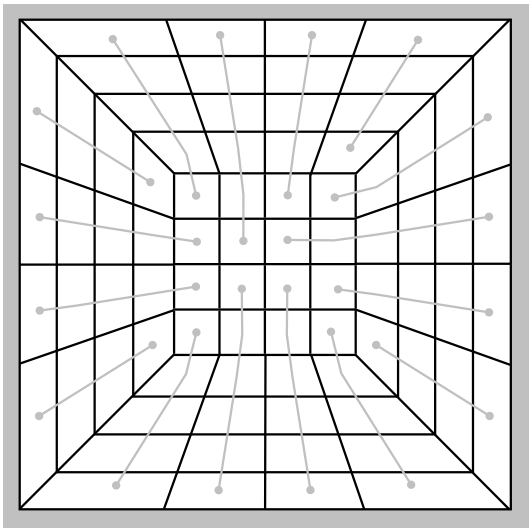


Figure 2.2: Illustration of lines for relaxation (gray) progressing from viscous boundaries and connecting cells.

begins. If the solution is run in parallel, lines are redistributed so that all cells and faces in a line are located on the same computational rank. This makes it extremely efficient when performing the linear solve since the highly-coupled lines require no interprocessor communication. Operations that do not change cell-to-face connectivity, such as moving meshes or grid smoothing, do not change the line groupings.

Following the simplifications for the point implicit method, line relaxation moves many of the terms from Eq. (2.15) to the right-hand side. In the following numerical description, cells l and u are the adjacent neighbors (in the line) to cell i.

$$\underbrace{\left[\frac{V_i}{\Delta t}+\sum_{j=1}^{m}\left[A_+^n+\left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]}_{\hat{A}_i}\delta\bar{U}_i^{k+1}+$$

$$\underbrace{\left[\left[A_-^n-\left(R^{-1}MR\right)N\cdot\hat{n}S\right]_l\right]}_{\hat{C}_i}\delta\bar{U}_l^{k+1}+\underbrace{\left[\left[A_-^n-\left(R^{-1}MR\right)N\cdot\hat{n}S\right]_m\right]}_{\hat{D}_i}\delta\bar{U}_m^{k+1}=$$

$$\underbrace{\frac{V_i}{\Delta t}\Delta\bar{U}_i}_{\Delta\bar{U}_i^*}-\sum_{j=1}^{m}\underbrace{\left[\left[A_-^n-\left(R^{-1}MR\right)N\cdot\hat{n}S\right]_j\right]}_{\hat{B}_{i,j}}\delta\bar{U}_j^k$$

The following matrix assume that the lines are arranged such that they are sequential in the solution vector.

$$\begin{bmatrix}\ddots&\ddots&0&0&0&0&0\\ \ddots&\ddots&\ddots&0&0&0&0\\ 0&\ddots&\ddots&\ddots&0&0&0\\ 0&0&\hat{C}_i&\hat{A}_i&\hat{D}_i&0&0\\ 0&0&0&\ddots&\ddots&\ddots&0\\ 0&0&0&0&\ddots&\ddots&\ddots\\ 0&0&0&0&0&\ddots&\ddots\end{bmatrix}\begin{pmatrix}\vdots\\ \vdots\\ \partial\bar{U}_l\\ \partial\bar{U}_i\\ \partial\bar{U}_u\\ \vdots\\ \vdots\end{pmatrix}^{k+1}=\begin{pmatrix}\vdots\\ \vdots\\ \vdots\\ \Delta\bar{U}_i^*-\sum_{j=1}^{m}\hat{B}_{i,j}\delta\bar{U}_j^k\\ \vdots\\ \vdots\\ \vdots\end{pmatrix}^{k} \tag{2.17}$$

The left-hand side is inverted once using a block LU-decomposition at the start of the sub-iterations and stored. One stored, it can be used to update $\delta\bar{U}_i$ based on the right-hand side. Each k iteration, the left-hand side is updated in order to propagate the influence of the change in adjacent cells ($\delta\bar{U}_j^k$).

The hybrid point-line implicit method is relatively straight forward. Cells that are not part of a DPLR line and are relaxed using point implicit contain zeros in the rows on the RHS of Eq. (2.17) and their LHS are augmented with the influence of all of their neighbors. This hybrid approach is used for the work contained in this document that depends on implicit time advancement.

Chapter 3

Adaptive Mesh Refinement

Adaptive Mesh Refinement is not a new technique and was pioneered by a number of authors over many years. Some of the earliest work was performed by Berger, Oliger, and Colella.[10;11] The initial work used dynamic adaptation of structured grids with finite-difference solvers. Success with unstructured grids, overset grids, and a number of numerical approaches have been widely documented since.

Both structured and unstructured grids are popular in modern flow solvers. Structured grids provide a lightweight, implicit connectivity that enables minimal memory overhead and can help create more efficient computation. Adaptive solvers using structured grids and cartesian grids have shown success for practical problems at both subsonic and supersonic conditions.[2;15;73] Unstructured approaches are typically more expensive (in memory and computational cost) but can simplify grid generation and provide greater flexibility in data structures. Grids with tetrahedral elements, prisms, and pyramids augment the more traditional hexahedral cells found in structured grids. For finite-volume and finite-element codes, unstructured grids dovetail nicely into the underlying numerical theory and are well represented in the literature.[12;35;44]

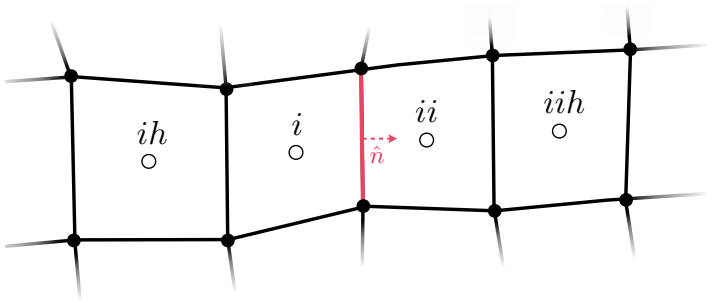
For basic subdivision of all element types, bisection methods that results in isotropic refinement are common. Bisection involves subdivision at the midpoint of all edges of an element. In three dimensions, isotropic bisection of a single hexahedral cell results in eight children each of roughly one-eighth the size of the parent cell. Many physical flows have strong gradients in only one prevailing direction (e.g., shock waves, boundary layers, shear layers, pressure waves) and lend themselves to an anisotropic strategy.

Isotropic AMR can inflate the number of cells created by refinement and includes additional resolution in directions where it was already sufficient. Anisotropic subdivision allows for more flexibility and is used by a number of researchers in the field. It does, however, increase the complexity of the AMR procedure and can lead to large changes in cell aspect ratio and size in localized regions.[36;75]. Anisotropic subdivision only yields substantial benefits, however, provided that the grid is aligned with the features of interest.[71]

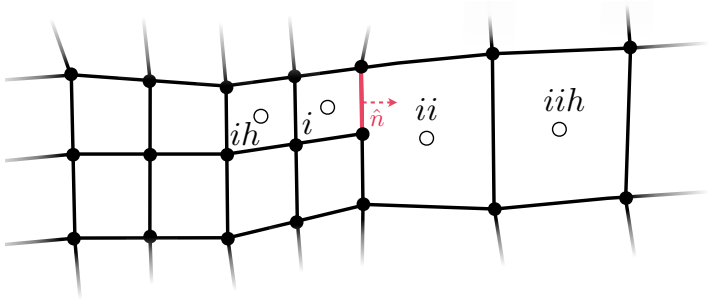
Automated mesh refinement and adaptation replaces expertise in grid generation with expertise in the selection of this criteria. If the criteria is too broad, then the solution becomes expensive and it eliminates the potential for a more efficient mesh. Too specific of a criteria and the resulting, adapted grid may be too coarse in regions that are critical for accurate results. For this reason, one of the most important features for a solver using adaptive mesh refinement is the selection of the refinement criteria.

A simple algorithm can employ feature detection and refine based on a flow variable, derivative, or a derived quantity. Such sensors are simple to implement and use, but require case-specific criteria. Refining the solution based on a metric for the error in each cell is also popular and continues to receive attention in recent literature.[58;59] Targeting residual or error can help remove user-dependance on the effects of refinement. Multiresolution methods provide a reasonable metric for refinement and have shown promising results on a number of applications.[13;14;37] Another popular criteria is an adjoint-method which depends on a global functional or engineering metric of importance to the researcher. This approach is an active topic of research for steady and unsteady problems.[23;24;47;68]

For codes that employ AMR, parallelization of the adaptation procedure is necessary for efficient scalability. Estimates using Amdahl's law state that a serial AMR process in an otherwise parallel code can limit the potential speedup to a mere order of magnitude.[70] For modern compute clusters that include thousands of cores, to take full advantage of resources it is imperative that the adaptation process itself is parallelized and can function in a scalable, distributed fashion. Furthermore, load balancing adaptive grids requires additional assumptions and considerations when compared to those required for static grids.



(a) Conformal mesh.



(b) Non-conformal mesh.

Figure 3.1: Illustration of traditional, conformal mesh and non-conformal mesh with hanging-node.

3.1 Non-Conformal Meshes

This work centers on the use of adaptive mesh refinement to locally enrich portions of the domain with additional computational cells. It uses h-adaptation to subdivide the existing cells into smaller ones. When AMR operation are performed, there is no attempt made to recreate a conformal mesh, where each node seamlessly connects adjacent edges or faces (2-D or 3-D) and no edge or face truncates in the interior of the computational domain. Figure 3.1(a) illustrates a region of the computational domain that is conformal. Figure Fig. 3.1(b) also illustrates a non-conformal mesh with all of the cells to the left of the highlighted face being refined. Since cell ii was not refined, there is an edge bisecting the the face between cell ii and what was the coarse cell i in Fig. 3.1(a). This creates a hanging-node on the boundary of cell ii.

The fundamental numerics of the finite-volume method lend themselves to grids that contain hanging nodes. Cell identifiers relative to the highlighted face in Fig. 3.1 are

updated to illustrate this point. In principle, subdivision of faces merely creates more faces, each of whom have a unique pair of neighboring cells. This augments the sum in Eq. (2.8), but otherwise does not change the application of the governing equations. However, as will be discussed in this section, there are modifications to the numerical methods as previously described that must be made to accommodate grids with hanging nodes.

Initially, all of the cells in the domain are described as being 'level zero' cells or faces. This indicates that they have not been refined and are the coarsest level that will ever be found in the domain. When a cell or face is refined, the results of this subdivision are referred to as its 'children'. The cell or face that was refined is similarly their 'parent'. The children of the parent are considered to be one level finer than the parent and the grid level for these elements is increased by one. For instance, if a level zero cell was refined isotropically for a three-dimensional problem, it would be the parent of eight 'level one' cells. If any of those new children were later refined, they would be replaced by eight 'level two' children, and so on.

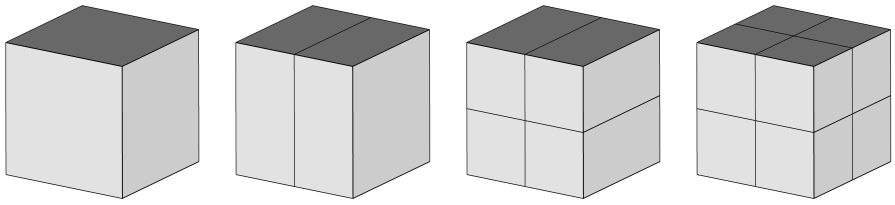
When a parent cell is refined, its volume and its faces are subdivided to create the child cells and faces. These faces replace the parent in the numerical solve and inclusion of both the parent and the child is redundant. For this reason, the coarse cell and faces are 'blanked' and hidden from all subsequent flow solve operations. When a face or cell is blanked, it is retained in memory, but not included when computational arrays are allocated (more on this later). Nodes are never blanked; cells and faces always require the nodes of their parents. Cells, faces, and nodes that are not blanked are referred to as 'active'.

In contrast, when refined cells are later coarsened, the child cells and faces are dropped and removed from memory entirely. Each process has a finite amount of memory and it is computationally inexpensive to create new geometry and perform the required prolongation operations. The memory overhead required to maintain all of the discrete geometry ever used during the course of an unsteady simulation could potentially become quite large. Since the current approach already requires more memory than a conventional fixed grid due to the dual memory approach, it was not considered advantageous to retain blanked children.

3.2 Mesh Adaptation

In a general sense, there is no limit to the number of cells that a parent cell can be divided into. An obvious choice for subdivision is to refine a cell by reducing its size by a factor of two in each solution direction. This is also referred to as isotropic subdivision. In a three-dimensional cell, this would equate to dividing a cell into eight smaller cells. Depending on the skewness of a hexahedral element, these new cells need not have equal size.

Figure 3.2 shows an example of hexahedral cell subdivisions for problems of varying dimensionality. At initialization, the boundary conditions on each of a cell's faces inform the solver of a cell's dimensionality. If it has boundary conditions on opposite faces, then refinement in that direction is not considered. This logic enables one- and two-dimensional problems using hexahedral grids with no change in the solver or refinement strategy.



(a) Original Cell (b) 1-D Subdivision (c) 2-D Subdivision (d) 3-D Subdivision

Figure 3.2: Illustration of AMR subdivision from an original hexahedral cell.

Working with an unstructured grid, each cell can be refined independent of the others. This provides a great deal of flexibility and reduces the creation of superfluous elements. Solution quantities are conserved during the refinement and coarsening operations. For refinement, each of the child cells are initialized to the solution variables in the coarse parent cell. When replacing a set of children with their parent cell during coarsening, the parent cell's solution variables are overwritten with the volume-weighted sum of the conserved variables from the child cells.

3.3 Modifications to Numerical Methods

The grids in this work are constrained to be six-sided hexahedral cells. For non-trivial geometries, these hexahedral cells are rarely cubic. While the finite-volume method can work on arbitrary polyhedra, using hexahedral cells enables more accurate resolution of discontinuities - such at shocks, which frequently accompany hypersonic flows - and have been shown to more precisely predict vehicle surface quantities.[16;31] Additionally, hexahedral shapes provide face connectivity structures that enable larger computational stencils. These stencils are required for the calculation of high-order numerical fluxes.

While not implemented in this work, subdivision of non-hexahedral elements should not effect the considerations in this section. Similar to the assumptions in the finite-volume method, all strategies and algorithms are face-based and agnostic to the polyhedra upon which they are applied. The one exception is in the line implicit operator - identification of the lines requires elements to have 'opposite' faces. The following subsections outline necessary modifications to the methods presented in Section 2.

3.3.1 Inviscid Fluxes

In general, flux methods higher than first-order require specifications of the high-order partners to each face. The high-order partners are the second-neighbors to the face in question. As shown previously when discussion the numerics, conformal meshes of hexahedral cells have a unique set of neighbors and high-order partners for each face. Figure 3.1(a) shows the face neighbors, i and ii, and the high-order partners, ih and iih, to the highlighted face.

Grids with hanging nodes can present obvious selections for the high-order partners, too. Suppose cells on the left-hand side of the previous case are subdivided to create a new set of faces and cells as shown in Fig. 3.1(b). The new, highlighted face still has a unique and easily determined set of neighbors - this is always the case. For this situation, the high-order partners are also easy to identify and very little changes from the uniform grid case. Unfortunately, not all faces in adapted meshes provide ideal circumstances for high-order partner selection.

Some ambiguity presents itself with more exotic arrangements of non-conformal cells. Figure 3.3 shows a situation where the cells on the far-right have also been subdivided

and second-neighbors for the highlighted, refined face are required. The selection of neighbors remains unchanged and so does the left-hand high-order partner, ih. Four possible choices for the right-hand high-order partner, iih, are shown.

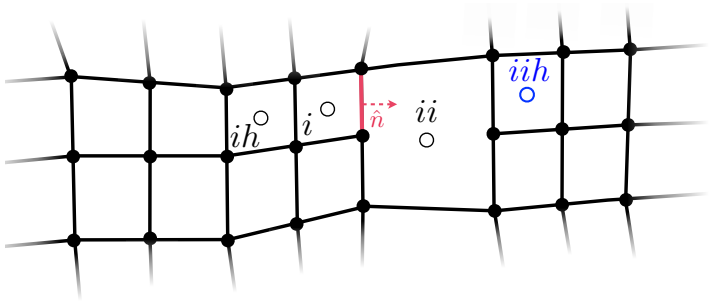
Figure 3.3(a) One option for selection is to use the closets fine cell that is in the normal direction of the fine face. This is in the spirit of the upwinding discussed previously where extrapolation to the face depended on the cells normal to it. It does result in an irregular spacing of the cell centers on the left-hand side of the face.

Figure 3.3(b) This option suggests using a combination of the two new cells that share a face with the coarse neighbor, cell ii. It is similar to the previous option, but includes as the second-neighbors all of the neighbors to the nearest-neighbor.

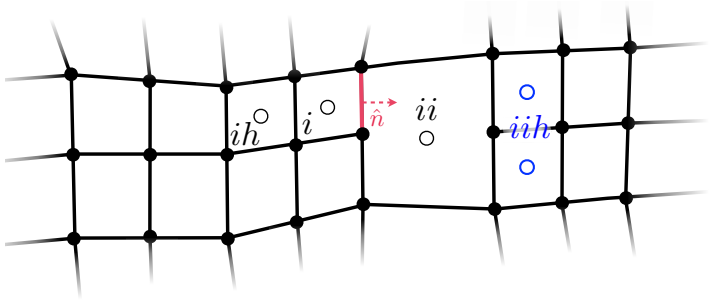
Figure 3.3(c) Using the gradient information in cell ii, it's possible to interpolate a fictitious solution value for the location identified by the dashed 'cell center' iih. This is not an actual cell but a temporary values used for the flux evaluation at this face. Depending on geometry used when performing the interpolation, the value for ii could be adjusted as well. This arrangement most closely resembles an ideal condition where all cells in the mesh were refined.

Figure 3.3(d) The last option discussed here considers the original neighbors and high-order partners to the coarse parent to the red face. It uses a restriction operator on the children of the coarse cell iih to update the solution values for the blanked coarse cell. A restriction of the gradients in the child cells replaces the gradient in the coarse cell. A more exact gradient might be determined by using a least-squares solve based on the values in the child cells. It create a stencil identical to the one that was used prior to grid refinement and the introduction of hanging nodes. This does create a lopsided stencil for either side the face, however.

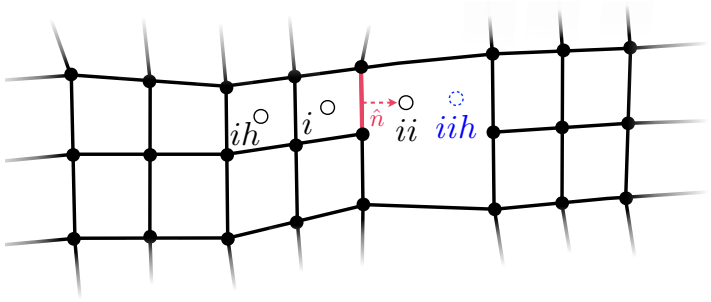
One final situation is when examining a coarse face in a grid with refined cells. Figure 3.4 presents two options for the high-order partner on the left-hand side of the face, ih. Shown are two of the options explored earlier for the fine face. The other two options identified previously, do not have a strong relevance for this topology. The first uses a combination of the two nearest fine cells to the neighbor (Fig. 3.4(a)) and



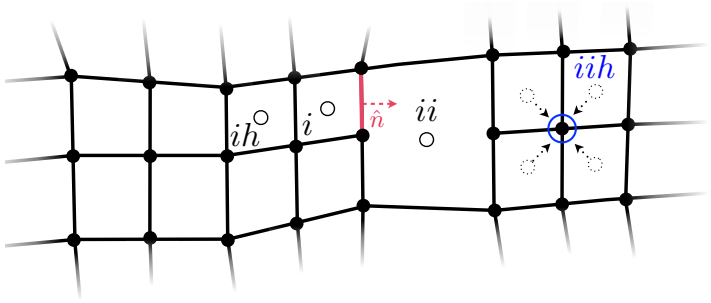
(a) Nearest refined cell selection.



(b) Neighbor's neighbors selection.

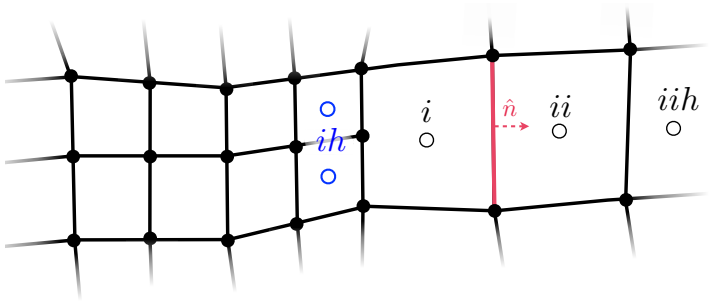


(c) Gradient-based reconstruction selection.

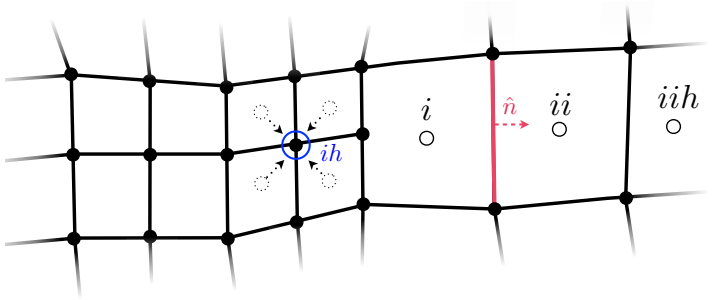


(d) Restricted coarse cell selection.

Figure 3.3: Options for high-order partner selection for refined face with hanging nodes.



(a) Neighbor's neighbors selection.



(b) Restricted coarse cell selection.

Figure 3.4: Options for high-order partner selection for coarse face with hanging nodes.

the second uses the restriction of the fine cells that made up the now-blanked coarse high-order partner (Fig. 3.4(b)).

It is the last approach, Fig. 3.4(d) and Fig. 3.3(b), that is used in this work when ambiguity arises in high-order partner identification for non-conformal meshes. The connectivity and search algorithm is simpler and the mechanisms for restriction are already available in the solver. Consistency between the two scenarios is also convenient, but it is possible that opportunities exists for better selection. There is potential for a more rigorous investigation into best practices for the construction of numerical stencils on grids with hanging nodes. This work investigates only the option selected here. The following results do not appear to show a deleterious effect on solution quality provided an appropriate refinement tolerance is selected.

3.3.2 Viscous Fluxes and Cell Gradients

No substantial change is required to the viscous flux calculation when considering cells with hanging nodes. The viscous fluxes are already localized and depend only on the quantities in the cells neighboring each face. No ambiguity in neighbor selection arises for subdivided cells and faces. As the grid is refined, the number of faces for which viscous fluxes are required increases, but the numerics do not change.

The gradient calculation is similarly unaltered when including hanging nodes. Use of weighted least-squares allows for an arbitrary number of neighboring cells in the matrix solve. When neighboring faces are subdivided, their children are now included in a cell's least-square matrices. Similarly, fine cells use neighboring coarse cells when at the edge of a region of refinement.

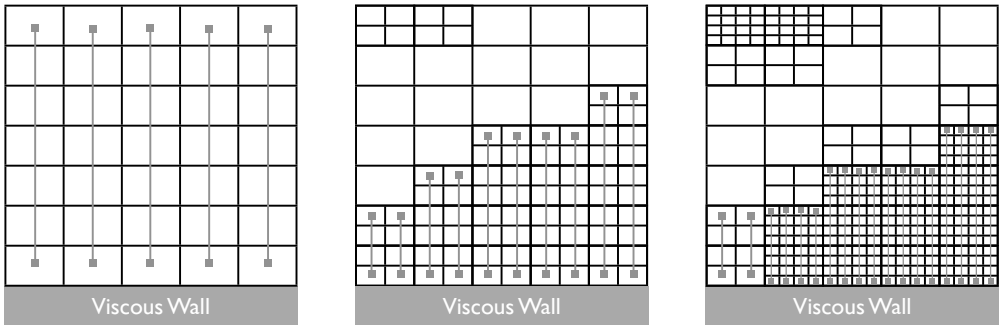
3.3.3 Point Implicit

Implementation of this implicit method within an AMR framework is natural and straightforward due to the unstructured nature of the numerical approach. Connectivity between cells in grids generated using AMR is maintained by the faces and the finite-volume solver is constructed to perform all operations over the list of active faces. In the matrix used for point relaxation in Eq. (2.16), the appropriate row for each cell contains summations over the faces bounding each computational cell. The $\hat{A}_i$ matrix includes all of a cell's right-running upwinded fluxes and the $\hat{B}_{i,j}$ matrices contain the left-running upwinded fluxes from neighboring cells.

Augmenting the point implicit operator involves increasing the number of terms in the $\hat{A}$ and $\hat{B}$ matrices. Since the number of faces has increased to account for the effects of adaptation and the code to create these matrixes loops over all active faces, it is handled automatically without additional consideration. It is, of course, important that parent faces are blanked and are not included in the summation for cell i.

3.3.4 Line Implicit

The line implicit operator is more sophisticated than the point implicit operator and requires attentional consideration. Based on a review of available literature, application of line relaxation for adapted grids with hanging nodes has not been performed. In



(a) Initial Grid (b) 1 Level of Refinement (c) 2 Levels of Refinement

Figure 3.5: Illustration of DPLR lines (gray) away from viscous boundary for an initial grid and after multiple levels of refinement.

addition to augmenting the implicit solve, there needs to be accommodation in how lines are created due to the presence of hanging nodes. Additionally, because the implicit operator depends on the grid topology and connectivity the form of the left-hand side in the iterative matrix solve needs to change as AMR changes cell connectivity. Much of this is accomplished by additional searches to recreate appropriate adjacency structures following adaptation.

For application on adapted grids with hanging nodes, the basic strategy for line relaxation is altered simply by only considering cells that are of the same level of refinement as the faces at viscous surfaces. Figure 3.5 has a simple illustration showing refinement of near-body cells and its effect on the implicit lines. An initial grid without any refinement, Fig. 3.5(a), grows relaxed lines of cells away from the surface to the opposite boundary of the domain. Notice that the lines of cells (identified in gray) run from one side of the figure to the other.

If the near-wall grid has been adapted, the relaxed lines only extend so far as there is an unbroken string of cells at the same grid level as the viscous boundary faces. Cells that are not part of a line use point relaxation. Figures 3.5(b) and (c) demonstrate a successively refined mesh and the resulting lines identified for relaxation on grids with hanging nodes. The lines do not extend to the refined region in the top-left corner of the mesh because there is not an unbroken group of refined cells from the viscous wall.

This modification and additional constraint on the identification of lines for relaxation is fairly straight-forward. For implementation, the solver is flexible in its definition of the cell lines and modifies them in memory after every dynamic grid event. The method depends on the assumption that cells in which the flow physics are tightly coupled have been co-identified by the refinement sensor. This follows the underlying premise for line relaxation which couples the numerics with the physics.

3.4 Memory Management

The code developed for this work is written in FORTRAN. Much of the computational strategy is based on US3D, a computational flow solver that uses predominately one-dimensional arrays to perform calculations. 1-D arrays are very efficient in FORTRAN and can take full advantage of vectorized operations and compiler optimizations for fast and efficient computation. For a fixed grid, one-dimensional arrays are generated at solution initialization or code start-up and remain in place for the entire simulation. The number of nodes, cells, and faces is invariant. Even with dynamic simulations with moving bodies, grid smoothing, or shock-fitting the computational geometry can change or deform and merely update the node locations and grid metrics without modifying memory structures.

For adapted grids, the concept of a Dynamic Grid Event (DGE) is introduced. A dynamic grid event is an event that triggers a change to the structures in memory by the addition or removal of nodes, faces, and cells. Adaptive mesh refinement is the most obvious dynamic grid event. Parallelization will be discussed in detail in a subsequent section, but one critical component of this approach involves repartitioning the mesh across multiple compute processes (ranks). This repartitioning shuttles cells, faces, and nodes between the ranks and requires modification to the memory structures since elements have been added or removed locally (on that rank). Repartitioning is another example of a dynamic grid event.

The AMR strategy used here employs a dual datatype approach. Linked lists of derived datatypes are ideally suited for adaptation and for repartitioning where it is important to be able to rapidly add, remove, or reposition elements. Unfortunately, linked lists are not ideal for efficient computation. Memory adjacency and compiler

optimization are much more amenable to arrays.

To support dynamic grid events, the flow solver was designed to be flexible in the way that it prepared memory prior to beginning time integration. Initially, all geometry and solution data is stored in linked lists. Once computation begins, much of this data from the linked lists are stored in 1-D arrays that are allocated for rapid iteration and time advancement. The portions of the code responsible for time stepping, gradient calculation, and output depend on these arrays. Before a DGE, the code stores the most recent flow variables in the linked lists and deallocates the 1-D arrays. All operations during a dynamic grid event depend solely on the linked lists and update them as required. Following the DGE, new 1-D arrays are allocated and updated with the most current information and memory structure informed by the linked lists.

The most immediate issue with this strategy is that it requires more memory than an approach that depends on only linked lists or 1-D arrays alone. It also requires that 1-D arrays be deallocated and reallocated on either side of a dynamic grid event. It should be noted that an alternative that depends only on 1-D arrays would require a similar amount of deallocation and reallocation, but might incur a smaller memory footprint with appropriate memory management. Using linked lists alone would reduce computational efficiency afforded by one-dimensional arrays, but could be designed to eliminate the majority of reallocation.

Another motivation for this system of dual memory is to provide an avenue for augmentation of existing solvers. Many flow solvers are designed around arrays of data in order to take advantage of their computational efficiency. By using linked lists for only the dynamic portion of mesh adaptation and load balancing, the AMR and repartitioning subroutines become a module that can interface with existing codes. It works behind the scenes to replace the flow solver's existing arrays with new ones without requiring a complete rewrite of current software. Understanding the performance of this approach can validate its approach (or similar) for those who might abstract it in this fashion.

3.4.1 Linked Lists

There are three major derived datatypes used in the solver; one for each of the cells, faces, and nodes. Each of them is described in detail in Appendix A. The derived data

types are allocated into arrays that are oversized relative to the demands of the current grid and are very seldom expanded. As new elements are created, they are placed into the most recently emptied location in that array and connected via pointers to the set of active elements to create a linked list of active cells, faces, and nodes. When elements are blanked, they remain in the array but are removed from the linked list. If an element is removed (from a coarsening operation), its entry in the array is re-initialized and it is added to the list of empty locations in the array.

As an example, consider a mesh with only four cells. Figure 3.6 provides an simple example of how the two main pointers (active and empty) and the datatype arrays work together. The array for the cell data types might initially be 10 elements long with only the first four items populated and connected via linked lists, see Fig. 3.6(a). There are two linked lists: one that connects all of the active cells and another that is referenced from a pointer and connects all of the empty array references ('Active' and 'Empty' in this example).

Initially, only four cells are populated (Fig. 3.6(a)) in this notional one-dimensional problem. Figure 3.6(b) assumes that two of the cells were refined in order to create four refined children. Those children are added to the linked list and array in the order that they are generated and the first four elements of the 'Empty' list are requested and filled. As a cell is refined, it is replaced in the 'Active' linked list by its children (which are added to the end of the list). Since the coarse cell is merely blanked, its information is still maintained in the array.

At a later time, the refinement criteria determines that refined children of one of the initial coarse cell are no longer needed. At this point, the refined children are removed from memory and their locations in the array are cleared and added to the front of the 'Empty' list. By their reference to their parent, it is added back to the 'Active' list - see Fig. 3.6(c). In this simple example, all lists move left to right down the memory array, in actuality, the connectivity of the linked lists is arbitrary and based on the evolution of the adapted solution and repartition events.

Additional pointers keep track of significant locations in the linked lists. For example, the face and cell linked lists are referenced by a pointer that tracks the most recent element for which up-to-date face and cell metrics exist. By interrogating to see if this pointer is the last element in the list, it is known if metric calculation is required.

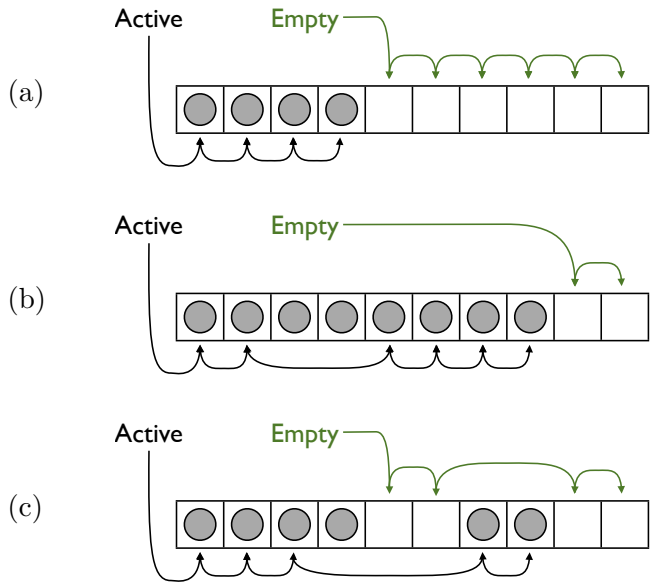


Figure 3.6: Example of combined array and linked list behavior.

Traversing the linked list from the 'Metric' pointer until the end calculates metrics for the faces and cells that are new to the rank (either by adaptation or repartitioning) at which point the pointer then points to the end of the list.

These operations are made easier by a module of simple commands that quickly return the first item in the empty linked list when a new node, face, or cell is added to the domain. Most of the subroutines outside of the flow calculations depend on traversing the linked lists and a break in the chain causes everything to fall apart. Robust subroutines ensure that connectivity between the linked lists never becomes broken. Similarly, when elements are dropped from memory, routines pull them out of the linked list, connect their previous and next elements to each other, and store that array location at the front of the empty linked list.

Based on the specifics of the implementation, additional linked lists might be required or desired. For instance, a linked list that connected all of the level-zero elements or one that held the blanked elements could provide avenues for streamlining certain algorithms. None of these were included in this work aside from the 'Active', 'Empty', and 'Metric' linked lists explicitly mentioned here.

An important point should be made before leaving this section. This work uses an oversized array for the derived datatypes as a mater of convenience. For proof-of-concept work, algorithm development, and debugging, it is straight-forward to reference a linked list element by its location in the large array when the linked lists break down. Computational forensics are much more tractable if the most recent address of the offender or location of the failure is know.

Subsequent work using the code and strategy described here do not depend on allocation of a large, oversized array. Implementation can be restricted to only require dynamic allocation/deallocation of linked list elements (more on this in Appendix A). The module for interacting with the linked lists is designed with generic subroutines that were made specific to this implementation. Using only linked lists would update these subroutines, but not dramatically change their behavior. With such an approach, the 'Empty' linked list would be unnecessary as new grid elements would be conjured on-the-fly.

3.4.2 Dynamic Grid Events

Proper handling of the dynamic grid events are a critical component of this work. A Dynamic Grid Event (DGE) is any modification to the grid that requires the 1D arrays to be updated. In the current implementation, that includes AMR or repartitioning of the mesh when running a distributed computation. Test problems use adaptation and repartitioning often and robust strategies for ensure that the fundamental connectivity in the mesh is maintained as the grid is refined and chunks of memory are shuttled between processors. Due to the nature of the memory management, not only is grid connectivity important, but connectivity inside of the linked lists is also crucial.

Development of routines to prepare and recover from DGEs was a substantial portion of this effort. When dealing with multiple grid levels, linked list connectivity, and repartitioning events there are numerous challenges that require robust methods. Approaches described in this section have success when dealing with three-dimensional problems across many ranks in a parallel computation with AMR and repartitioning.

The main time advancement loop attempts to minimize the impact of the dynamic grid events, but takes a conservative approach that rebuilds the connectivity and the FORTRAN one-dimensional arrays after AMR or repartitioning. Opportunities exist

to refine the approach and minimize some of the work associated with DGEs. There is wasted effort in the circumstance where AMR and repartitioning occur on the same time step (often). A high-level description of the main time advancement loop is shown in Algorithm 1. Frequency-based operations (*do_restart*, *do_output*, *do_AMR*, *do_repart*) are performed at user-specified times based on iteration count, physical time, or the magnitude of the residual.

Algorithm 1: Main Time Advancement Loop

```
do while time < time_final
    Calculate Δt from CFL;
    Advance solution Δt;
    Check for NaN;
    Evaluate do_restart, do_output, do_AMR, do_repart;
    if do_restart then Write restart file;
    if do_output then Write visualization files;
    if do_AMR then
        Update linked lists and drop 1-D arrays;
        Evaluate refinement criteria;
        Perform AMR;
        Project surface nodes and redistribute off-body grid;
        CALL MPI_Prep
    endif
    if do_repart then
        Update linked lists and drop 1-D arrays;
        Build current partitioning graph and call ParMETIS;
        Perform repartition across ranks;
        CALL MPI_Prep
    endif
enddo
subroutine MPI_Prep :
    Renumber node, face, and cell counts and sync global IDs;
    if line_relaxation then Find DPLR lines;
    Rebuild parallel connectivity to minimize communication;
    Create 1-D arrays from updated linked lists;
end
```

Adaptive mesh refinement can cause significant disruption to the solution and local refinement can introduce numerical instability. Also, transient events in solution

convergence or too quick of a CFL ramping schedule cause code failure. Both of these phenomena motivated development of a robust CFL scheduler that handles DGEs as well as more traditional flow solver failures and generally creates bullet-proof time advancement. Two main features merit descriptor here.

The first is a NaN-catcher that monitors the residual to determine when a NaN has taken place. Instead of exiting the code with an error, the solution is reverted a certain number of iterations and the CFL is dropped to half of its prior value. Over a specified number of iterations, the CFL ramps back to the prior value at a more gradual pace. Implementation only requires a memory overhead of the solution values in the one-dimensional arrays.

The second is an adjustment to the CFL after an adaptation that increases the maximum grid level in the domain. After cell subdivision, even a previously steady solution has unsteady character with the additional degrees of freedom. In high-gradient regions and near boundaries, there can be considerable change in the solution before and after adaptation. Dropping the CFL to a lower value and ramping it back to the previous one greatly improves robustness of the adaptation and does not incur significant cost and a short ramping is sufficient in practice.

Figure 3.7 shows an example of these mechanisms for a problem that includes mesh adaptation and high-gradient regions. It results in a steady flowfield solution and time-accuracy is not important. After an initial CFL ramping to 1.0, the code was instructed to double the CFL every 500 iterations. After a specified increment of simluated flow time, the grid is refined. After AMR, the CFL drops and quickly ramps back to the previous value and continues to double until the maximum CFL of 500 is reached. Every invocation of the AMR routine creates a similar dip in CFL. At iteration 2,000 a NaN is encountered. The solver reverts to a prior solution, drops the CFL, and ramps back to the previous value. Afterwards, the solution continues.

3.5 Parallelization

The goal of distributed computation is to spread the problem across a large number of computation cores, or ranks, to more quickly iterate. This incurs fixed costs for communication between the ranks, but subdivides the work to leverage large compute

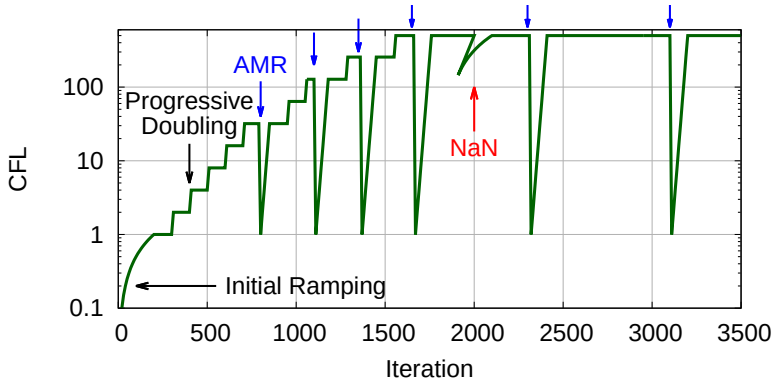


Figure 3.7: Notional example of CFL ramping with AMR and NaN recovery.

clusters with hundreds of cores. It is imperative that the code is structured to minimize the cost of communication. Futhermore, differences in the number of cells or faces between cores can cause load balance issues where one rank is waiting on one or more of its peers to complete before it can continue. This inefficiency is common for parallel computation and must be limited. The flow solver developed for this work uses the standard OpenMPI library in order to achieve distributed computation. [29]

Fringe faces, those at the periphery of a rank's partition, require the solution values at the neighboring process' cells in order to compute the numerical flux and vice versa. This necessitates a ghost cell for the fringe face's neighbors and frequent communication between processes. MPI communication in the code predominantly uses non-blocking send and receive calls that allow numerical work to be performed while variable exchange is taking place. This is enabled by the data dependencies implied in the DPLR algorithm and by deliberate ordering of solution arrays. In most cases, this serves to hide the communication and allows the simulation to avoid message passing bottlenecks. As the number of fringe faces grows large with respect to the number of internal faces, the effect of communication bottlenecks become evident, however.

When constructing the list of fringe faces, the code builds lists of the required cells from the computational ranks for which is has shared information. It then collapses these lists in order to ensure that information for each computational cell is only shared once to each rank that requires it. This minimizes the communication that must take place during iterations.

For an unbalanced partitioning graph, where the computational ranks have a widely different number of elements or where the partitions have not been optimized with respect to their number of fringe elements, even the best attempts at hiding communication costs become insufficient. If a partition of the grid is thought of as a physical volume, then there is a strong desire to minimize the surface area to volume ratio. This code uses ParMETIS to derive an appropriate loading for each of the ranks in the computation and rebalance the grid as indicated.[41] When using unadapted grids, this rebalancing is done once at the beginning of the computation before iteration begins. Grids generated using AMR require additional attention and provide unique challenges in this regard.

AMR can quickly cause previously load-balanced simulations to quickly fall out of balance. This is mainly due to unsteady or transient phenomena or to a naive initial partitioning that did not consider future growth of the grid. As cells are refined disproportionately inside the domain of one rank and not others, inefficiencies manifest. To counteract these imbalances, the evolving grid can be repartitioned based on the current grid densities. ParMETIS allows for adaptation of the current mesh by means of the *AdaptiveRepart* routine. Initial partitions and adaptive partitions are refined with successive calls to *RefineKway*.

This strategy for load balancing and redistribution operates on the level zero cells in the mesh. By operating on the coarsest cells in the mesh, it ensures that a cell and all of its descendants are collocated on a single processor. This helps minimize some of the costs associated with bookkeeping and also simplifies logic internal to the code. It does, however, reduce the effective number of vertices in the graph used for partitioning. This can become an issue when using the line implicit operator across a significant number of ranks because it further constrains the number of vertices in the graph.

The procedure builds the partition graph of level zero cells and weights each cell by the number of active children that it has. It also provides edge weightings for each level zero face with the total number of children for each face on the partition fringe. Since ParMETIS can provide a partition that takes both edge and cell weightings into account, this additional information allows for minimal communication between ranks. In order to take advantage of the inherent parallelism in the DPLR lines it is necessary that cells in a single DPLR line are partitioned together. This is easily accomplished

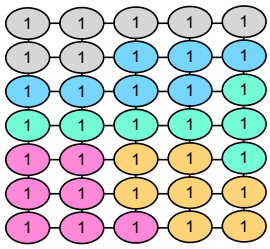
by collapsing each line into a single element and appropriately weighting the cells and faces by their sum.

Figure 3.8 is a simple example shown a series of graph partitions for the series of adapted grids used in Fig. 3.5. This sample assumes that there are five ranks running a simulation using implicit line relaxation with two levels of AMR. An initial mesh of 35 cells are partitioned and repartitioned over the coarse of the simulation. DPLR lines are co-located in order to take advantage of parallel implicit solve and cells are partitioned at level zero.

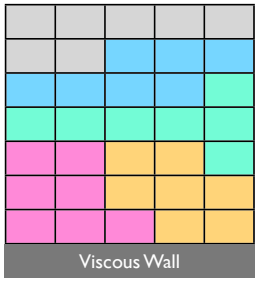
In the figure, the first column of each line represents the vertex graph that is delivered to ParMETIS. The vertices represent cells or lines of cells and are colored to denote the rank that they are on. The number inside the vertex it its weight (the number of cells in the line). Not shown here are the weights associated with the faces or the lines on the partition graph. The second column represents the computational mesh corresponding to the vertex graph. Again, cells are colored by the rank that owns them. Finally, the third column shows the computational mesh with updated DPLR line identification. The second and third rows included examples of adaptation between the second and third column.

Moving from the top-left to the bottom-right, therefore, should give the impression of watching both refinement and repartitioning occur during the course of the simulation. The final state of the grid is not ideally load balanced. On the last row, the five ranks have 44, 45, 59, 60, and 64 cells. A more equitable arrangement is certainly possible, but the heavily-weighted vertex in the bottom-right of the mesh makes it impossible to perfectly balance the arrangement. Collapsing the DPLR lines results in limited degrees of freedom in the final partition. Relaxing the constraint on the level zero mesh would provide opportunities for better load balancing, but incur added complexity during memory management.

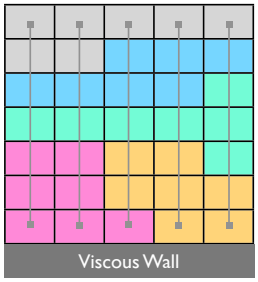
Grid redistribution can be triggered by a number of criteria: it can be called after each grid adaptation, based on a metric for grid balance, or be dependent on measured runtimes orrelevant time scale. In practice, repartitioning can be an expensive operation and the selection of an ideal frequency for redistribution may be a problem-specific choice. The following results use either a fixed repartition frequency to provide a controlled environment for study of the algorithm or tie repartitioning to AMR events in



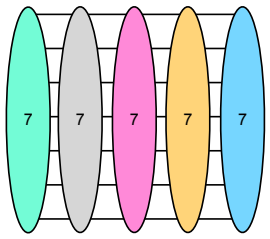
Initial partition graph.



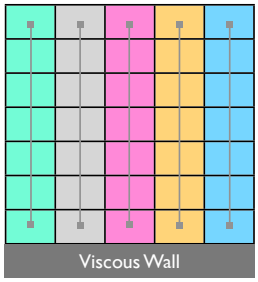
Partitioned grid.



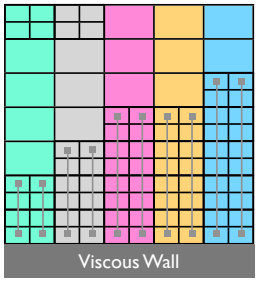
DPLR line identification on initial partition.



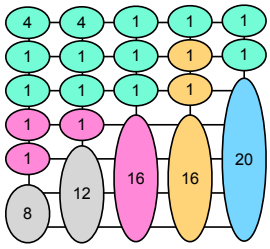
Updated partition graph; co-located DPLR lines.



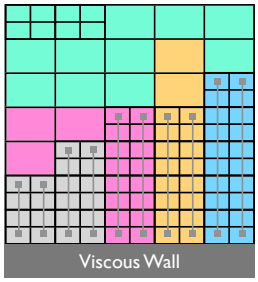
Repartitioned grid.



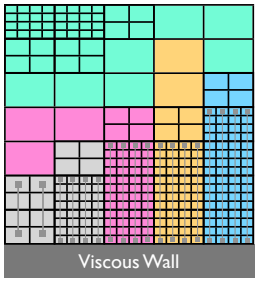
AMR refinement and line identification.



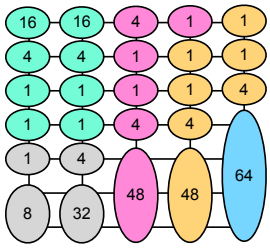
Updated partition graph.



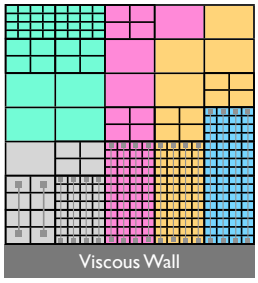
Repartitioned grid.



AMR refinement and line identification.



Updated partition graph.



Repartitioned grid.

Figure 3.8: Example of DPLR line identification and repartitioning with AMR on non-conformal mesh. Graph cell weights shown (left); DPLR lines identified in gray (right).

order to minimize imbalanced during time stepping.

Algorithm 2 shows pseudo-code for a reparation event. It illustrates the high-level considerations, but does not show the specifics. The term *MPI_portal* is used to describe one of the three derived datatypes used for migrating data between ranks. Specifics for each of the MPI portals are discussed in detail in Appendix A. The portals are specified such that they contain the minimum information required to recreate grid metrics, connectivity, and the solution state vector on the receiving end of a reparation.

Cells, faces, and nodes are numbered in a global sense from zero to the total number across all ranks in the simulation. These numbers are unique to the grid element and are synced for all shared faces and nodes after a repartition event. Each grid element is also given a local number from zero to the total number on that rank (this corresponds to its place in the oversized derived datatype array). When transferring elements between nodes, all references that use local identifiers are converted to global identifiers.

3.6 Refinement Criteria

Previous research into mesh adaptation by many authors has identified a large number of possible refinement criteria. For the purposes of this work, a complete assessment of refinement criteria is not important. Selection of an appropriate refinement criteria and tolerance is problem-dependent and availability of many criteria allows flexibility and empower users to tailor the refinement for a selected problem.

This document focus on understanding implementation, scalability, and performance of AMR. The choice of refinement metric is secondary to that and future development of this approach can employ existing and new refinement criteria. Important to this work are criteria that allow for comparison between adapted and unadapted grids and illustrate the advantages and issues of the method. The refinement criteria used in the work that follows are introduced in this section.

The flow solver applies the criteria at specified times throughout the simulation - specified frequencies in iteration number or simulated time. Each criteria used in a simulation has a maximum level of refinement, in grid levels, that it can refine cells to. When applying multiple sensors, cells are flagged for the highest level of refinement dictated by all active sensors.

Algorithm 2: Repartioning mesh - Rank *rank_ID*

```
Build partition graph for ParMETIS;
Call ParMETIS to color partition graph;
Count number of cells, faces, and nodes coming to and going from rank_ID;
foreach linked_list
    foreach item in linked_list
        if ParMETIS(item) ≠ rank_ID then
            children ← all_children(item);      ! Returns item and children
            foreach item in children
                Expand local ID references to global ID references for
                children/parent;
                Add item to MPI_portal;
                Drop cell item from cell linked list;          ! Cell only
            end
        endif
    end
    Call MPI_Send/Recv(MPI_portal);
    foreach item in MPI_portal
        Expand global ID references to local ID references to children/parent;
        Add item to linked list;
    end
end
foreach face in face linked_list
    if ParMETIS(face) ≠ rank_ID and face not in local cells then
        Drop face from face linked list
    endif
end
foreach node in node linked_list
    if ParMETIS(node) ≠ rank_ID and node not in local faces then
        Drop node from node linked list
    endif
end
```

3.6.1 Geometric

The most basic criteria for adaptive refinement are those that are tied to a problem's geometry. For a non-moving mesh, these criteria are unchanging and refinement location and magnitude is known a priori, before numerical simulation begins. Refinement of this type is an extension of the grid generation process and provides an additional level of control for researchers when setting-up a simulations.

In this work, several simulations use geometric refinement applied over a predetermined region of the grid. A simple criteria identifies a box bounded by (x_{min}, y_{min}, z_{min}) and (x_{max}, y_{max}, z_{max}) and aligned with the principle axes. Cells whose centers are contained inside the box can be identified and refined to a predetermined level prior to time stepping. Geometric criteria can also be used to identify cells contained by spheres in space, rotated box grids, or more exotic geometric definitions.

Conversely, cells inside a region of space can be limited to a maximum level of refinement. This is designed to overrule other sensors that might have a higher maximum grid level in a region of the grid. One possible use for this is to create sponge layers or force adaptation to ignore a region of the domain (by setting the maximum level to zero inside of a region of space).

Another refinement criteria is useful when considering wall-bounded or viscous flows. This work uses a criteria that refines all cells without a specified distance to a grid boundary. To do this, it uses a KD-Tree lookup to calculate the wall distance to the specific boundary in parallel.[9;42] Cells within a specified distance of the boundary faces are refined to the grid level specified by the user.

3.6.2 Feature-based

The current work uses several feature-based refinement criteria. For a given simulation, any combination may be used and they can augment the geometric criteria mentioned previously. Feature-based refinement criteria have limitations, but are simple to implement and provide reasonable results for many problems.

This work uses feature-based criteria for flow variables ϕ (density, velocity, pressure, temperature, and/or vorticity) with several options for normalization. The first feature-based criteria is an undivided difference. Undivided differences inform the solver

that there is a sufficiently large change in relevant quantities between adjacent cells to require additional spatial resolution. As implemented here, this criteria is face-based and depends on values in neighboring cells i and ii.

Equation (3.1) shows the undivided difference used for refinement of a flow variable, ϕ, based on the tolerance, ϕ_{tol}. The code traverses all active faces and evaluates the difference between its neighbors. Both neighbors are flagged for refinement when the right-hand side of the equation exceeds the specified tolerance. Notice that Eq. (3.1) normalizes the difference to the local minimum in the flow variable ϕ. Normalization is not required and the option also exists to normalize to the global maximum if desired.

$$\phi_{tol} < \frac{|\phi_i - \phi_{ii}|}{min(\phi_i, \phi_{ii})} \tag{3.1}$$

Another method used in the following results is a sensor based on the gradient of a flow variable. This is a cell-based sensor and operates on all cells i. Shown in Eq. (3.2), the sensor depends on a pre-determined tolerance for the gradient and involves no normalization. It is evaluated in each cell and does not required any information from the neighbors (once the gradients are known). By inspection of a well-resolved simulation, an appropriate value for ϕ_{tol} is determined.

$$\phi_{tol} < |\nabla \phi_i| \tag{3.2}$$

Flows with separated wakes involve shear layers and turbulent wake that create regions of high vorticity, ω. Refinement on the scalar value of vorticity is reasonable and the code allows for selection of vorticity as a refinement criteria. Kamkar *et al* have show that an alternative criteria is to use a non-dimensional Q-criterion instead. [40]

The advantage of this sensor is two-fold. First, its simplicity can remove the requirement for a priori judgment on what magnitude of vorticity should trigger mesh refinement. Second, it has been shown to accurately track propagation of vortices in the solution. Equation (3.3) shows the equation for non-dimeisonal Q-criterion.

$$\bar{Q} = \frac{1}{2}\left(\frac{||\Omega||^2}{||S||^2} - 1\right) \tag{3.3}$$

Where Ω is the rotation rate tensor and S is the strain rate tensor. A threshold for

refinement is placed on the non-dimensional value of $\bar{Q}$. A refinement tolerance of unity is suggested for this sensor. Previous work has found that the inclusion of a filter is necessary to remove improper selection of regions with low vorticity but with a significantly large non-dimensional value. Based on the work Kamkar, cells whose dimensional Q-criterion magnitude is less than 0.25 percent of the global maximum are removed from consideration when using this refinement criteria.

3.7 Projection and Node Redistribution

Grid smoothing and surface projection are also necessary to move beyond simple shapes and flat boundaries. Most aerodynamic shapes include non-linear surfaces and simple subdivision of boundary faces is insufficient to properly resolve the flow features. For viscous cases with very small body-normal spacings near the wall, projection and subsequent node redistribution is required in order to ensure that no negative volumes develop and ensure that grid lines remain aligned with the surface.

A simple method to achieve both goals of projection and alignment of unstructured grids with hanging nodes is presented here. It finds lines of nodes from the surface by 'climbing' the connectivity from the lines constructed for line relaxation. Each of the resulting node lines has a normal direction associated with it. The direction is the average of the face normals from all faces that have the surface node on one of their edges. To support large simulations, all aspects of this approach are done in a distributed fashion and are fully parallel.

In order to resolve a more detailed surface than exists in an unadapted coarse grid, a finer representation of the source geometry is required. Watertight input from a CAD package could provide a baseline set of surfaces for coarse grid generation as well as subsequent adaptation and is an attractive option for future work. For this work, a simple method accepts very fine triangulations in the triq format and builds a KD-tree of triangle centroids.[1] After AMR refinement is complete, surface nodes are projected to the fine representation by a KD-tree lookup and Algorithm 3.

For viscous problems at high Reynolds numbers, it is important to include sufficient resolution at the wall to resolve the very high gradients that exist. Furthermore, it is desirable that the method provides a smooth distribution of grid cells away from

Algorithm 3: Surface Node Projection

```
foreach node on boundary
    project ← .T.;
    location ← (node_x, node_y, node_z);
    do while project
        Lookup closest triangle to location with KD-Tree → target;
        Project location to plane of target with node line normal → point;
        Check to see if point is inside of area contained by target;
        if inside then
            (node_x, node_y, node_z) ← (point_x, point_y, point_z);
            project ← .F.;
        else
            location ← (point_x, point_y, point_z)
        endif
    enddo
end
```

viscous boundaries and does not cause cell spacing discontinuities in the domain. This work employs two methods for redistributing the cells away from viscous boundaries: geometric and hyperbolic tangent stretching functions.[69]

Most grids have relatively simple near-body grids that are imbedded in a more complex topology. Ideally the grid spacings begin at the small near-wall value and stretch away from the surface and match the existing values away form the wall. The methods used here require two Δx values: one at the viscous wall and another that is equal to the current spacing at the farthest end of the node line. These are referred to as Δx_{wall} and Δx_{far}, respectively.

The value for Δx_{wall} is specified by the user for the particular run and is likely dictated by a target y^+ value. After adaptation, the code traverses each line of nodes emanating from the surface and records a connectivity array. It also records the existing spacing at the end of the line, Δ_{far}. For consistent behavior across a range of adaptation cycles, the user provides the desired wall spacing for the finest grid. Lines that are of a level coarser than the finest level use a wall spacing that is scaled by $2^{L_{fine}-L_{line}}$ where L_{fine} is the finest grid level prescribed by the user and L_{line} is the current level of nodes in the line.

Figure 3.9 shows an example of both projection and smoothing on a simple grid

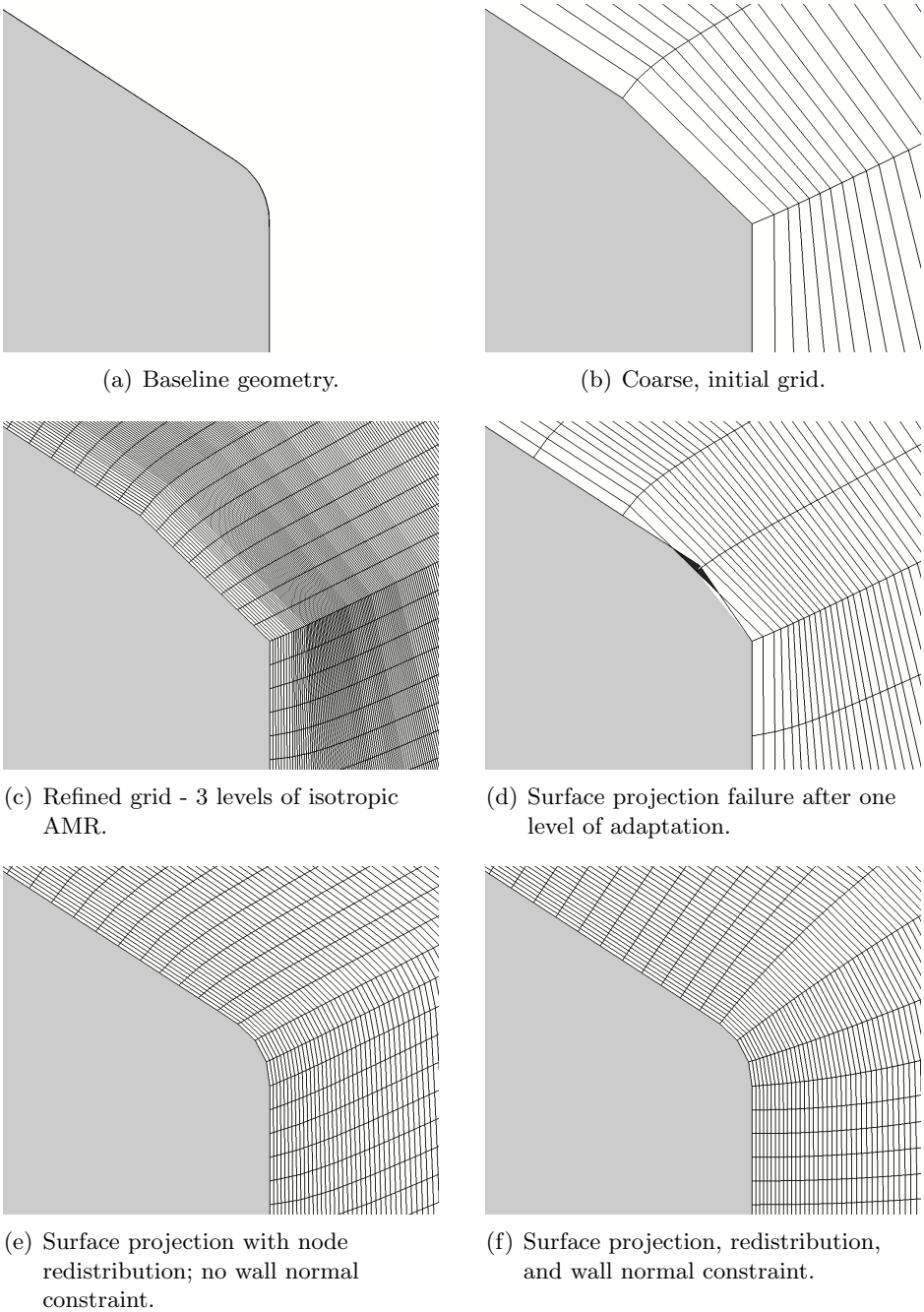
designed to stress grid adaptation. A fine representation of the geometry is shown in Fig. 3.9(a). The initial, coarse mesh rounds off the region of high curvature (Fig. 3.9(b)). Isotropic AMR on this domain yields a non-ideal mesh, Fig. 3.9(c), with no additional resolution of the underlying geometry and clear striations in the grid corresponding to the poor growth away from the surface in the original grid.

Projection of the surface nodes alone is insufficient to improve the results. Figure 3.9(d) shows failure after the first adaptation with the wall-bounded faces being projected past the near-wall faces. This is due to the small wall spacing and poor representation of the initial mesh. Application of projection coupled with node redistribution provides a much more appealing grid. Grid spacings and surface representation are much improved in Fig. 3.9(e). Figure 3.9(f) is a grid after three levels of AMR. It shows a mesh with the surface normals adjusted to ensure normal grid lines away from the body. For curved boundary flows, this work uses all three techniques: projection, redistribution, and enforced normalcy.

3.8 Refinement Propagation

With a face-based unstructured numerical method, neighboring cells may be subdivided an arbitrary number of times relative to the neighbor cell. Figure 3.10(a) shows an example of a cell (far-left) that contains subcells two levels higher than the neighbor cell (the center cell). The center cell (in 2-D), now has seven faces defining its perimeter. Such an arrangement would not impact the finite volume formulation, but the current method seeks to avoid dramatic changes in cell sizes. To accomplish this, the code enforces that adjacent cells not have more than one grid level difference between them. The accompanying figure, Fig. 3.10(b), presents an acceptable configuration.

Our adaptive refinement incorporates 'buffer cells'. Regions that are flagged for refinement based on the criteria described previously are expanded by a number of buffer cells in all directions. These cells create an inflated region of refinement that allows flow features to propagate over several time steps without drifting into coarser regions of the grid. By conservatively choosing the size of this buffer region, the researcher can confidently reduce the frequency for AMR calls. As will be shown later, excessive buffer cells can reduce the overhead associated with AMR, but comes at the cost of increased



(a) Baseline geometry.

(b) Coarse, initial grid.

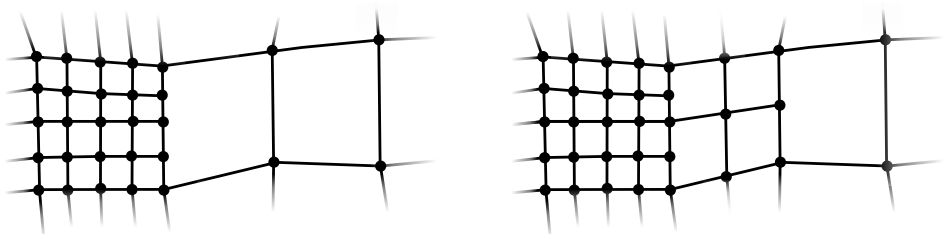
(c) Refined grid - 3 levels of isotropic AMR.

(d) Surface projection failure after one level of adaptation.

(e) Surface projection with node redistribution; no wall normal constraint.

(f) Surface projection, redistribution, and wall normal constraint.

Figure 3.9: Example of refinement results on region of high curvature and small wall spacings.



(a) Unacceptable refinement of face neighbors.

(b) Accepted refinement of face neighbors.

Figure 3.10: Computational meshes with unacceptable and acceptable refinement of adjacent cells.

grid size.

It can be difficult to ensure proper gradation of cell sizes in three dimensions when using multiple refinement and coarsening techniques with a number of grid levels. The approach developed for this work is face-based, robust, and fully parallel. It relies on the existing adjacent structures and propagates refinement through the multilevel mesh with consideration of both the cell and face levels and the local level of refinement.

Chapter 4

Verification

The numerical methods designed to solve the governing equations are complex. Implementation is often non-trivial and there are many opportunities for error. Furthermore, the truncation error from the methods used must be compared to benchmarks in order to show that they achieve the stated accuracy for ideal meshes. Verification is critical when developing or benchmarking a new computational tool. A new solver is employed in this work and its capabilities and behavior must be understood before attempting validation cases and more challenging problems. Verification is, put simply, ensuring that you are correctly solving the numerical equations that you intend to solve.

Specific to this work, two different fluxes are used when calculating the convective portion of the flux vector: Modified Steger-Warming fluxes and the Kinetic Energy Consistent fluxes. These fluxes range in accuracy from first- to sixth-order. They have different dependancies on the solver and are both compared to exact solutions for both the Euler and Navier-Stokes equations. The example problem involves inviscid convection of a pulse in density through a periodic domain. Without dissipation or viscosity, the exact solution known. Application of adaptive mesh refinement for the convecting density pulse is also shown.

Also shown are numerical experiments highlighting the parallellization via MPI. A simple problem is considered using both adapted and unadapted grids. Using a fixed amount of work, the problem is distributed across up to 512 processors. Detailed analysis of subroutine timings is included as function of grid size and partition.

4.1 Convecting Density Pulse

A convecting density pulse provides a trivial, analytical solution to the Euler equations. In a uniform velocity flow, the pulse should convect with the fluid at the bulk flow velocity. Comparisons to the exact solution involves shifting the convected pulse by V_∞/t and comparing to the initial condition - where t is elapsed time and V_∞ is the bulk flow velocity. Low-dissipation methods are ideal for flows without discontinuities and this is an identical test case to the one used in previous investigation into the KEC numerics.[6]

This test case is selected because it contains a localized region of the flow with a strong gradient (the pulse) that moves in time. The density pulse is unsteady and provides an opportunity to highlight both refinement and coarsening of computational cells. With this verification suite, solutions obtained uniform grids will be compared to those found on cells generated using adaptation.

The initial conditions for the density pulse are shown in Eq. (4.1) with $\bar{r}$ being the spatial coordinate of the computational element. Solutions for one-, two-, and three-dimensional pulses are examined. For the one-dimensional pulse, $\bar{r}$ is merely the x-coordinate of the cell centroid, for the two-dimensional pulse $\bar{r} = \sqrt{x^2 + y^2}$, and for the three-dimensional pulse $\bar{r} = \sqrt{x^2 + y^2 + z^2}$. The computational domains for each problem extend from -5 [m] to 5 [m] in all relevant coordinate directions and the boundary conditions are periodic. Finally, for the two-dimensional pulse, $v = 1.0$ $[m/s]$ and for the three-dimensional pulse, both v and $w = 1.0$ $[m/s]$.

$$\begin{array}{lllllllllll} u & = & 1.0 & [m/s], & w & = & 0.0 & [m/s], & p & = & 1.0 \quad [Pa], \\ v & = & 0.0 & [m/s], & \rho & = & 1.0 + \frac{1}{10} e^{-\frac{(\bar{r}-5.0)^2}{2}} & [kg/m^3], & T & = & \frac{p}{\rho R} \quad [K]. \end{array} \tag{4.1}$$

All simulations use a timestep consistent with a CFL of 0.1 and a 3$^{\text{rd}}$-order explicit Runge-Kutta method. One cycle over the periodic domain requires 10 seconds of simulated time. Afterwards, the density pulse returns to its initial location and the error in ρ is calculated. The RMS of the error (weighted by volume) across all computational cells provides a scalar measure of accuracy.

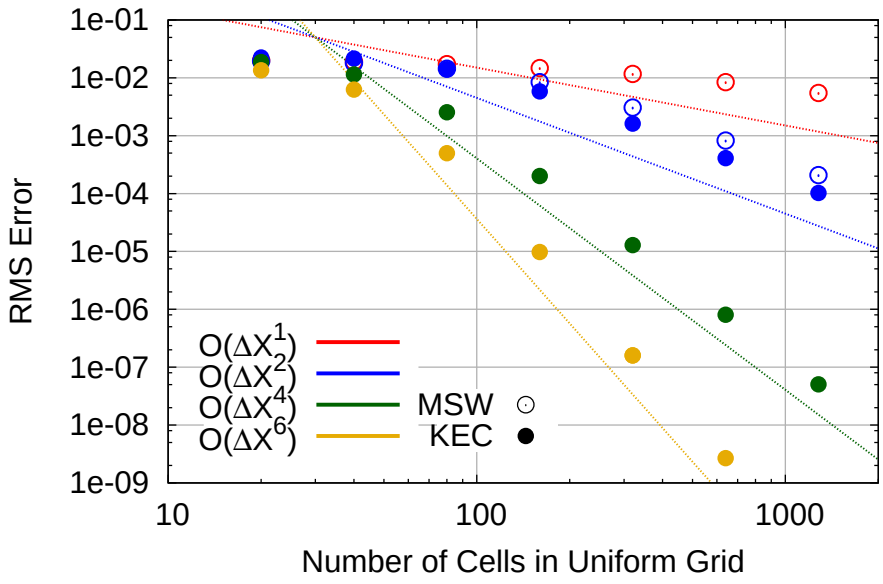
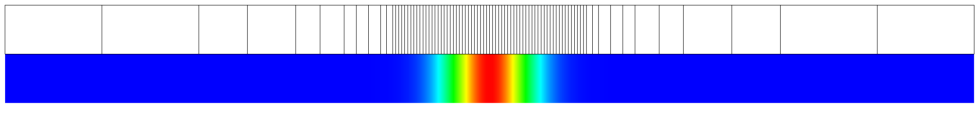


Figure 4.1: RMS error for 1-D density pulse after one cycle using uniform grids.

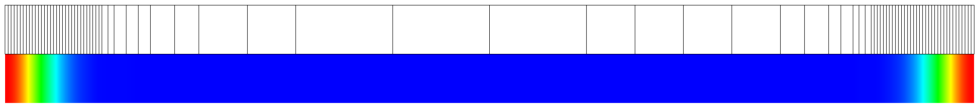
4.1.1 1-D Density Pulse

Figure 4.1 shows the final error in the domain after one cycle for a range of numerical fluxes across several grids. The flow solver was used on a range of uniform grid sizes in order to obtain a measure of the spatial order of error for the flux calculation routines. Two forms of the Modified-Steiger Warming fluxes and three of the KEC fluxes are evaluated. Lines showing theoretical convergence of first-, second-, fourth-, and sixth-order are shown. The symbols represent the error from the simulations. Ideal behavior is characterized by the reduction in error between adjacent symbols to be identical to the slope of corresponding dashed lines. As expected, all of the methods recover the predicted order as the grids are refined on uniform grids.

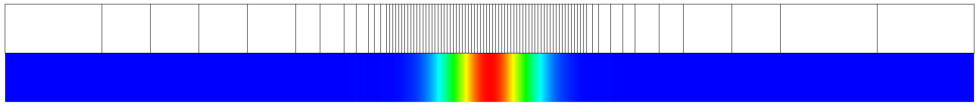
For comparison with AMR, a baseline grid with only ten cells across was constructed. The domain was then adapted to initial density pulse with a specified limit on the maximum number of grid levels. Using isotropic subdivision, the addition of each grid level reduces the smallest grid spacing by a factor of two. For instance, if the grid was uniformly refined five grid levels, then there would be 320 cells in the resulting mesh ($10 \cdot 2^5 = 320$). Each cell in that domain would have a grid spacing that is equivalent to a uniform grid with 320 cells. If adaptation was targeted and only a portion of the



(a) t = 0 s (86 Cells)



(b) t = 5 s (92 Cells)



(c) t = 10 s (89 Cells)

Figure 4.2: Computational mesh, ρ contour, and cell count for 1-D Gaussian pulse at three simulation times using AMR with a refinement tolerance of ρ_{tol}=1E-3.

grid was refined five grid levels, then the finest cells would be the same size as those in the 320-cell uniform mesh, but there would be fewer than 320 cells in the domain.

The solutions are initialized on grids refined to capture the gradients in the prescribed density profile (Eq. (4.1)). As the solution ran, AMR was performed ever 5 iterations in order to ensure that the refinement tracked the convecting pulse. A simple, feature-based refinement criteria is used for this analysis and applied in each cell, Eq. (3.1). The variable ρ_{tol} was changed in the results that follow. Cells were coarsened when the refinement propagation and coarsening metrics indicated that they were no longer required.

Figure 4.2 shows the adapted mesh at the initial time, half-way through the simulation, and at the final time. A ρ_{tol} value of 1E-3 was used for this example. Qualitatively, both refinement and coarsening of the mesh perform well. The adapting grid tracks the pulse in time and fine cells are removed from the domain when no longer needed. This case illustrates five levels of adaptive refinement starting from the coarse baseline grid. Contours in the figures show the value of density obtained using the sixth-order KEC fluxes.

Results of the spatial convergence study using AMR show much similarity to the results obtained on uniform grids. Figures 4.3(a)-4.3(c) show the results when using

refinement criteria of ρ_{tol} = 1.0E-4, 1.0E-6, and 1.0E-8, respectively. The finest cells in a refined grid are equivalent in size to those in a uniform grid with a number of points equal to the 'Number of Cells in Uniform Grid', the x-axis in the plots.

Three values of ρ_{tol} are used in order to illustrate sensitivity to the AMR refinement tolerance. When using the larger tolerances, the higher-order fluxes achieve a minimum error in excess of what was observed on the uniform grid. It is only when the refinement tolerance is reduced that the error is able to match the results show previously for the uniform grids.

4.1.2 2-D Density Pulse

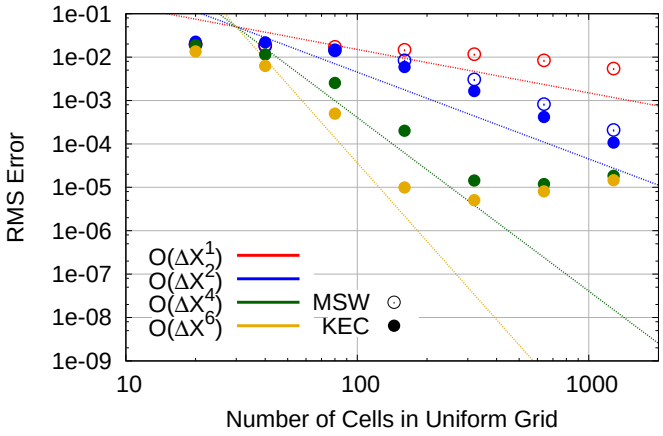
There are additional degrees of freedom introduced when moving beyond the one-dimensional problem. The choice for the second neighbors to a face become non-unique and in order to ensure that the approach chosen here does not adversely impact the accuracy of the methods, the order of error assessment is repeated for a two-dimensional case. In addition, since the motivating problems are inherently two- or three-dimensional, it is necessary to ensure that their accuracy does not suffer as result of the AMR procedure.

This case uses a two-dimensional Gaussian pulse convecting for one cycle on a periodic grid (in the x- and y- direction) with no fluxes in the z-direction. Initial conditions for the 2-D density pulse are identical to those for the 1-D simulation, Eq. (4.1). The grid is comprised of a varying number of cells, uniform in size and number in the x- and y-directions. Again, a CFL of 0.1 is used for all computations and the volume-weighted RMS error is compared.

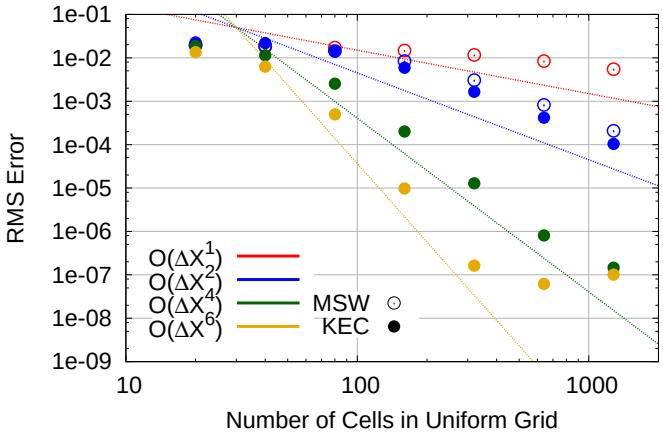
Similar to the 1-D case considered earlier, Fig. 4.4 shows the RMS error for simulations using a uniformly distributed number of cells. The x-axis shows number of cells in the x-direction (identical in the y-direction). As with the one-dimensional case, the numerical fluxes perform to their analytical order of accuracy as the grid is refined.

Figure 4.5 shows the mesh used in an AMR simulation at the initial time, half-way through the simulation, and at the final time. As with the previous case, 5 levels of refinement are allowed. Only half the grid is shown - cell counts are for the entire mesh, as compared to 102,400 (320×320) for a uniformly refined grid.

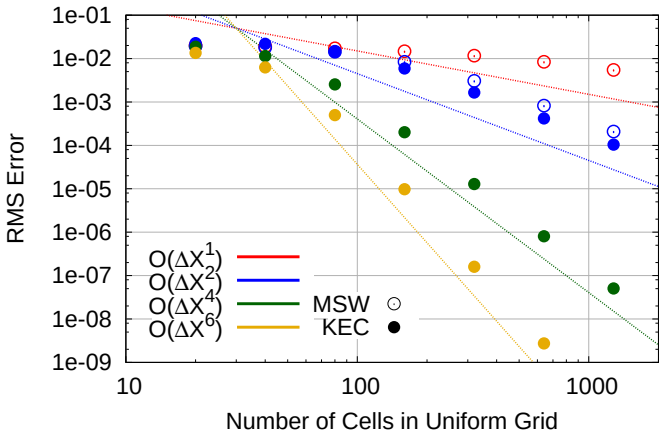
The comparative cases using AMR on a coarse grid with 10 cells in the x- and



(a) ρ_{tol} = 1.0E-4



(b) ρ_{tol} = 1.0E-6



(c) ρ_{tol} = 1.0E-8

Figure 4.3: RMS error for 1-D ρ pulse using AMR and three different values of ρ_{tol}.

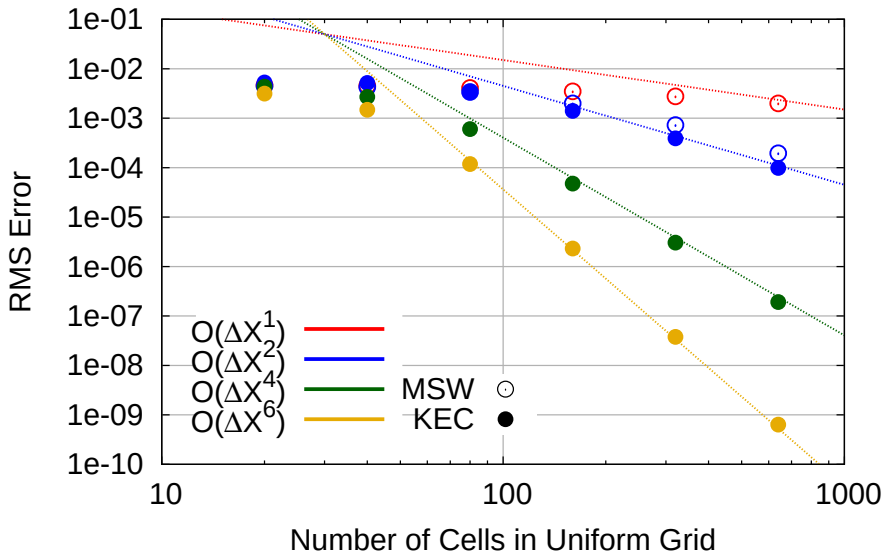
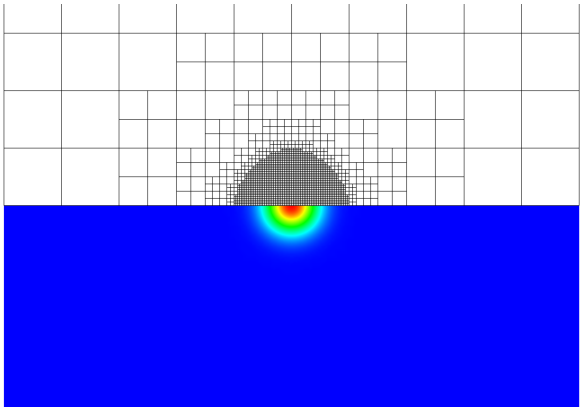


Figure 4.4: RMS error for 2-D density pulse after one cycle using uniform grids.

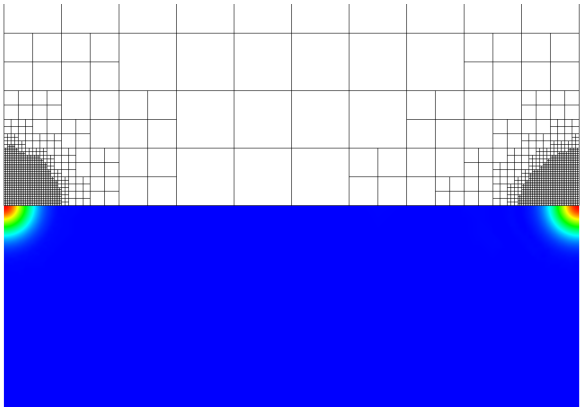
y-directions (100 total elements). A maximum of six grid levels are considered, corresponding to an effective grid with 409,600 elements (640×640). Figure 4.6 shows the results for three values of ρ_{tol}. Again, the larger tolerances result in a minimum error in excess of the desired result seen in the uniform grid case. With a ρ_{tol} of 1.0E-8, the analytical order of error is recovered for the range of grid sizes examined. As seen with the one-dimensional case, with an appropriate value for the refinement tolerance the AMR method is amenable to high-order fluxes.

While promising, these results still leave some ambiguity at the appropriate value for the refinement tolerance necessary for a practical problem. To achieve a specified level of accuracy for this test problem, it seems best to select a value of ρ_{tol} that is at or below the desired RMS of the error. The same guideline might prove useful for non-linear flowfield as well, but it is likely that more complicated problems will require more involved heuristics to drive refinement.

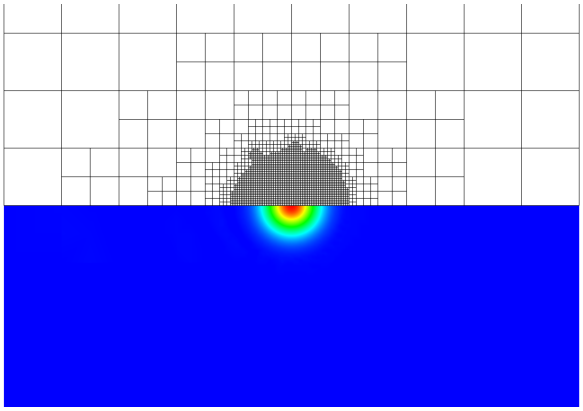
A more laborious alternative is to perform a sensitivity problem for each specific class of problem. Currently, engineers and researchers perform grid sensitivity studies in order to guide grid generation. As implemented in this work, our process requires two such studies: one to determine the tolerance, ϕ_{tol}, and another to determine the



(a) t = 0 s (3,724 Cells)

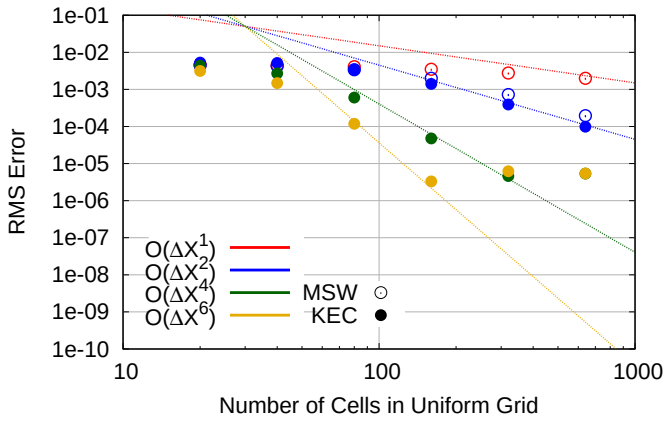


(b) t = 5 s (4,060 Cells)

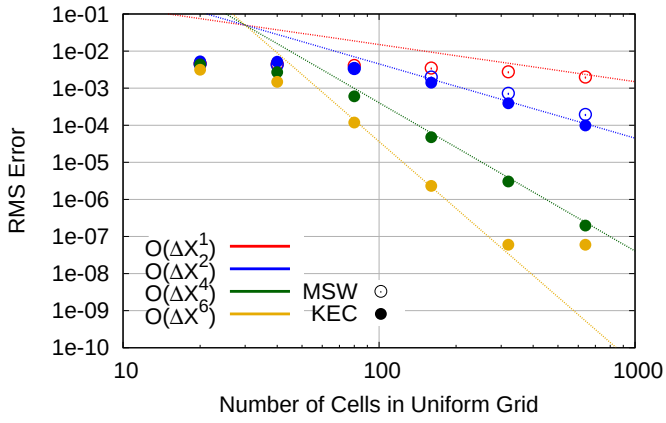


(c) t = 10 s (4,138 Cells)

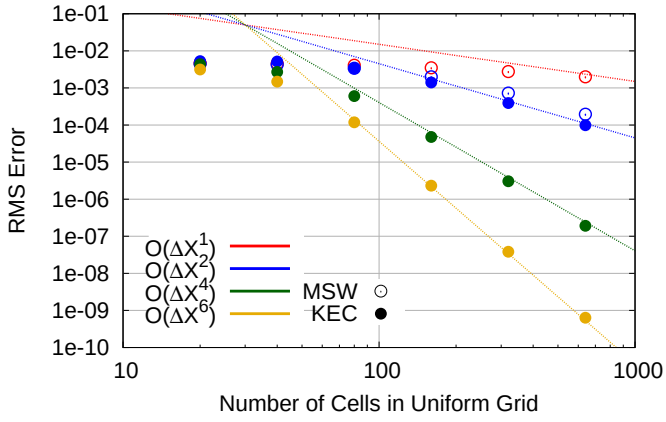
Figure 4.5: Computational mesh, ρ contour, and cell count for 2-D Gaussian pulse at three simulation times using AMR with a refinement tolerance of ρ_{tol}=1E-3. Images cropped in vertical direction.



(a) ρ_{tol} = 1.0E-4



(b) ρ_{tol} = 1.0E-6



(c) ρ_{tol} = 1.0E-8

Figure 4.6: RMS error for 2-D ρ pulse using AMR and three different values of ρ_{tol}.

required minimum cell size. For this reason, more focused work is still required into more advanced metrics for refinement.

4.2 Parallel Performance

Even with implicit methods and AMR, problems that require a large number of grid cells or a long simulation time demand many computer hours to complete. It is common practice to distribute this work across many processors in order to accelerate time-to-solution. For this work, we employ MPI in order to take advantage of large-scale parallel resources. Parallelization enables engineers and researchers to leverage the ever-increasing size of compute clusters to further reduce run times. Proper handling of AMR in the context of distributed computation is complex and was discussed in Chapter 3

4.2.1 Parallel Performance of Unadapted Grids

To measure the parallel performance of the flow solver, an unadapted grid is considered first. This is necessary to first confirm that the numerics involved in time stepping have been implemented in an efficient manner and are understood before adding the additional complexity of AMR. This establishes a baseline for comparison and performance. All start-up and I/O costs are not considered in the timings below. Only time spent in the computational loop reserved for iterations has been included.

A fixed-size problem was selected in order to judge parallel performance of the flow solver. Similar to our previous work, a propagating density pulse is considered. The pulse is allowed to advect for a total of 400 time steps - this provides a fixed unit of work across all cases. Each solution uses a three-dimensional grid that consists of a cube with uniform grid density. Sixth-order accurate spatial fluxes are used with third-order Runge-Kutta time integration. The initial conditions for the density pulse are shown described Eq. (4.1) with $\bar{r}$ being the radius of the computational element from the center of the domain. Simulations were run at a constant CFL of 0.1.

Sixth-order flux evaluations were used for several reasons. Their requirement for gradients of flow variables provides an opportunity to assess the scalability of the gradient calculation. Viscous simulations, regardless of the type of flux evaluation, require flow

gradients, so including them in these timings increases their relevance to the full Navier-Stokes equations. The high-order stencil requires information at second-neighbors which requires a greater exchange of information between processors. Furthermore, these fluxes do not require the (expensive) calculation of flux Jacobians. Since this calculation creates additional work that is internal to the processor, it would help mask the non-blocking data exchange. By selecting the most taxing numerical technique with a reduced amount of local work per time step, this should provide a lower-bound for scalability.

Grids of varying sizes are considered. The expectation is that larger grids will show improved scalability. Cost associated with data exchange at the ghost cells are asynchronous and can be hidden during computational work that involves only internal elements. As the number of internal elements falls (number of ranks increases or cell count decreases), there is inadequate local work to sufficiently mask the exchange. Increasing the number of ranks also increases the available cache size which can act contrary to the above expectation and increase scalability with a decreased number of cells per core. These competing effects are illustrated in the following results.

The original grids were given a decomposition across all ranks by using ParMETIS. While not shown here, the partitions are ideal for the fixed grid and both cell counts and fringe face counts are nearly identical across all ranks regardless of problem size. Since the grids are not adapted, no further costs associated with repartitioning are incurred.

For this study, a compute cluster with Intel Westmere X5650$^{\text{tm}}$ processors was used. Each computational node contains 2 6-core processors (12 cores total) running at 2.67GHz with a 12 MB shared L3 cache. They are equipped with 4GB of memory per core and are linked by QDR Infiniband (40-gigabit).

Figure 4.7 shows the results from the uniform grid scalability study. The speedup is measured relative to the runtime for a one-rank case (r_1) and is calculated as $\frac{r_1}{r_N}$ for a case with N-ranks. The left-hand plot, Fig. 4.7(a), illustrates the speed-up over a range of processor and grid sizes. Quantitatively, the point at which adding additional processors provides diminishing returns is highlighted in Fig. 4.7(b). Shown is the parallel efficiency versus the number of cores for the same set of cases. Parallel efficiency is calculated by multiplying the speedup by the number of ranks, $N\frac{r_1}{r_N}$, and is ideally 1.

Two important observations can be made from the figure:

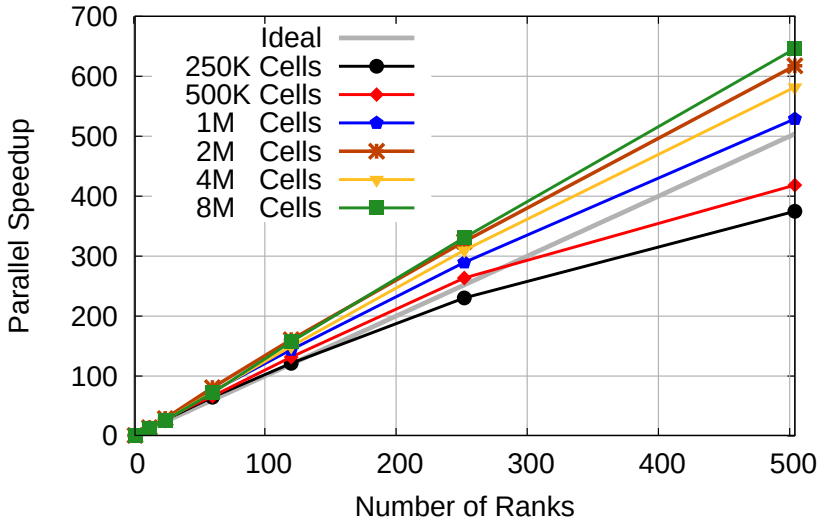
- For small problems (250,000 and 500,000 grid cells), the costs incurred by parallelization prove detrimental after a certain number of processors have been added to the problem. This is typical of a problem that shows strong scaling.
- All problems considered show super-linear behavior for a range of scalings. This is due to an increase of available cache size and the reduction of the amount of cells on each processor. The larger the problem is, the more ranks can be added before increasing parallelization becomes detrimental.

All of the grids show a reduction in parallel efficiency as they are run on a larger number of cores. Another way to view the performance metric is shown in Fig. 4.8. It illustrates the relationship between parallel efficiency and the number of cells on each rank. The behavior for these cases indicates that for grid partitions with more than about 2,500 cells per core, speedup should be super-linear. All of the curves trend towards sub-ideal speedup at a fairly consistent number of cells per rank.

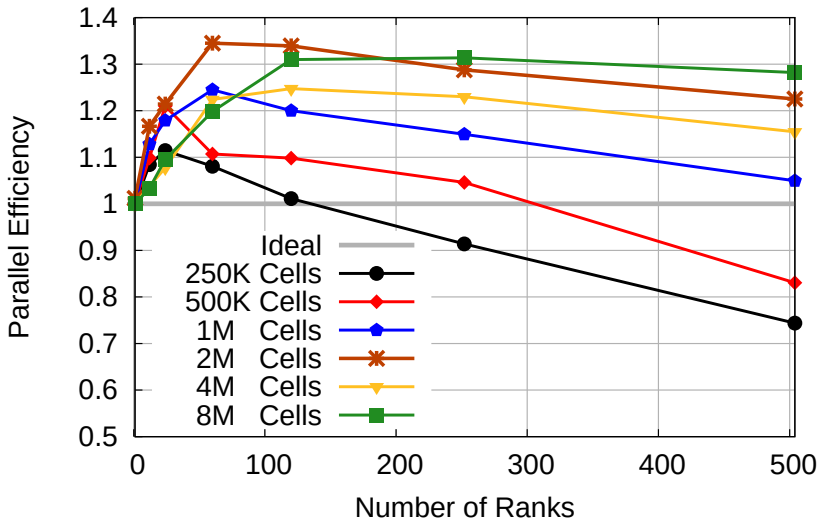
Figure 4.9 show a more detailed breakdown of the scaling. The figure breaks out the time spent for several major routines in the flow solver and presents them as a function of the number of ranks used. The y-axis is normalized to the average cost of a time step and since there are no other costs with unadapted grids, it can be interpreted as a percentage of total runtime.

To orient the reader for this Fig. 4.9 and similar plots later in this document, *Iterating* includes calculation of the inviscid and viscous fluxes, summation over each computational cell, and the explicit/implicit update. *Gradients* includes the time necessary to refactor the linear solve (for adapted grids) and performing the least-squares reconstruction for the gradients. *Exchange* is the combined cost of packing and unpacking the memory buffers for the MPI exchanges and any waiting necessary for the non-blocking receives to return. Time spent handling the boundary conditions is collected in *BCs*. *Waiting* is the total time spent waiting at the end of each iteration at an all-to-all broadcast of residual and Δt values before the next iteration.

For almost all cases, as the number of ranks increases, the overhead associated with *Exchange* and *Waiting* increases. The larger the grid size, the more gradual the increase. For the cases that show the worst scalability in Fig. 4.7, the 250,000 and



(a) Speedup



(b) Parallel Efficiency

Figure 4.7: Parallel performance for uniform, unadapted grids. Work is consistent across all grids at a given size.

500,000 cell cases, these data expose that they become dominated by the exchange time and synchronization at the end of each time step as processor count grows large.

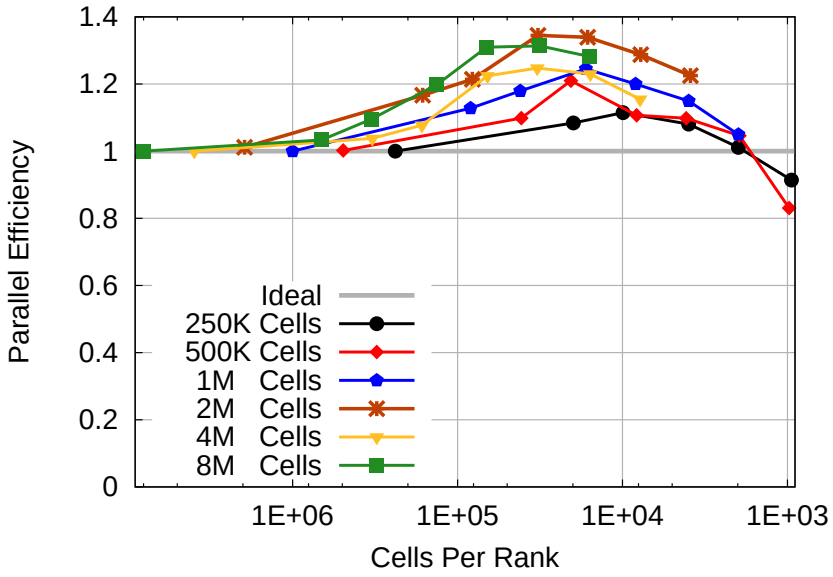


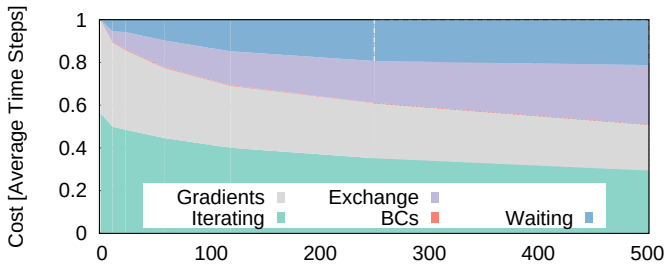
Figure 4.8: Parallel performance for uniform, unadapted grids as a function of cells per rank.

4.2.2 Parallel Performance of Adapted Grids

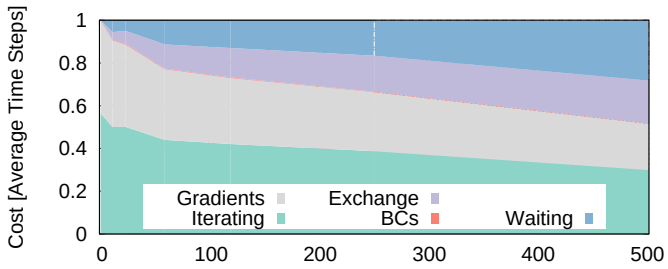
The parallel performance of solutions based on adaptive grids is of primary importance to this work. To investigate parallel performance, the convecting density pulse was again considered. This test case provides an unsteady flow that requires frequent refinement as the pulse moves through the domain. Previous work has shown that with an appropriate refinement tolerance on ρ adapted grids perfectly represent the solution as resolved on uniform meshes.

By selecting an unsteady problem, this provides a taxing environment in which to assess performance. Frequent calls to the AMR and ParMETIS redistribution subroutines highlight inefficiencies and provide suggestions for streamlining the process. Similar to iterating, both the AMR and redistribution processes have many instances of shared communication between ranks and provide opportunities to hide the communication during local computation.

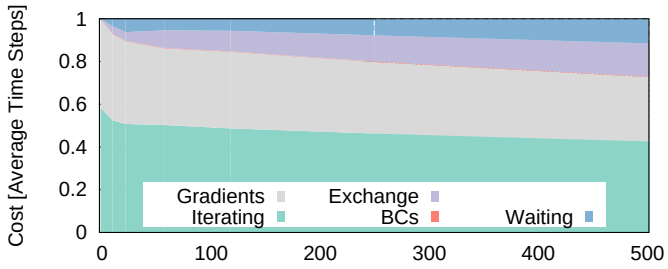
The largest grid presented in the previous results was an 8 million cell mesh of a cubic volume. Each side of the mesh measured 200 cells. With isotropic refinement, a



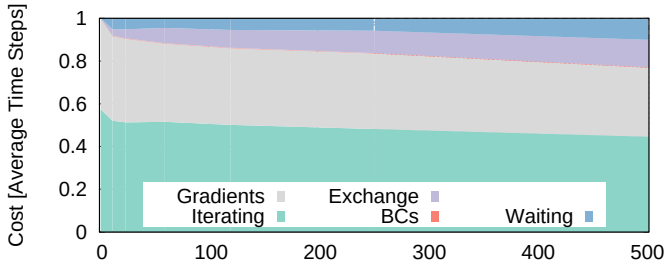
(a) 250,000 Cells



(b) 500,000 Cells



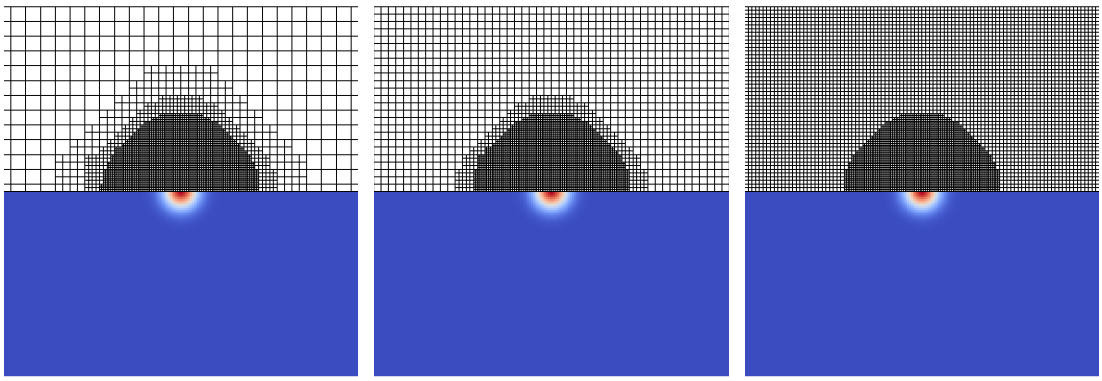
(c) 1 Million Cells



(d) 8 Million Cells

Figure 4.9: Percentage of total runtime in major sections of flow solver for unadapted grids as a function of the number of cores used. Several grid sizes shown.

100 cell-to-a-side mesh that includes 1 level of refined cells will contain cells identical in size to those used on the uniform mesh. Similarly, a 50x50x50 cell computation with 2 levels of refinement and a 25x25x25 grid with 3 levels both provide equivalently fine cells. All of these grids are considered in the subsequent discussion. Figure 4.10 shows a cut through the volume at the initial condition highlighting the extent of adaptive refinement.



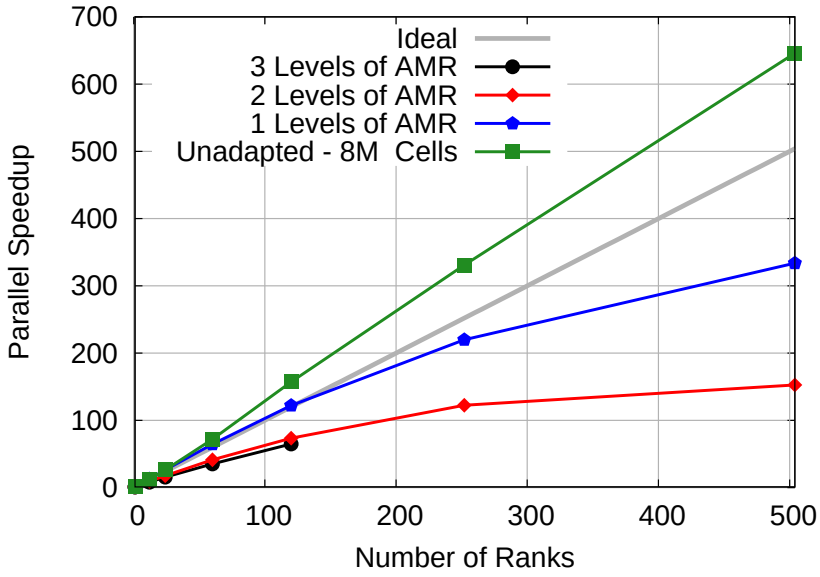
(a) Three levels of refinement (b) Two levels of refinement (c) One level of refinement

Figure 4.10: Cut through the 3-D solution volume showing the initial density pulse and adapted grids.

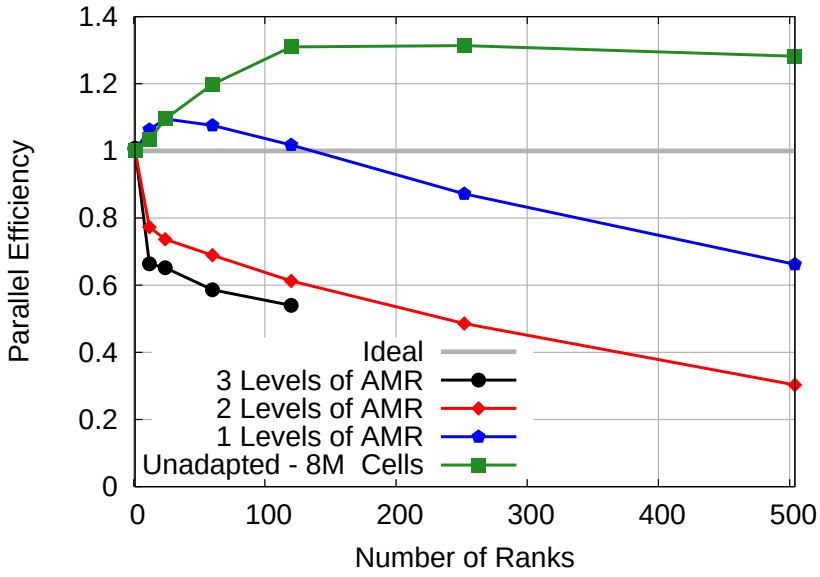
For this problem, there are three important parameters to consider. The first is the refinement criterion. A ρ-based criterion with 1E-8 is used (see Eq. (3.1)). This was confirmed to provide identical accuracy as a uniform, unadapted mesh. The second and third parameters are the frequency for AMR and cell redistribution. For the 400 time steps currently being considered, several combinations are considered.

Figure 4.11 shows results from a scalability study performed with adapted grids. Speedup and parallel efficiency are calculated as described for Fig. 4.7. These data represent a baseline strategy for refinement: AMR performed every 10 iterations, ParMETIS redistribution performed every 50 iterations, and 2 buffer cells. It should be noted that the smallest initial grid contained only 15,625 cells and it was not possible to obtain a partition graph from ParMETIS using more than 120 cores.

The most dramatic result from Fig. 4.11 is that the scalability shown in the adapted grids is far less impressive than was seen previously or the uniform grid results repeated here. Data in the figure indicate that parallel efficiency is reduced as the number of



(a) Speedup



(b) Parallel Efficiency

Figure 4.11: Parallel performance for adapted grids that are as accurate as an unadapted mesh with 8 million cells.

AMR levels increases. Either the overhead associated with AMR and cell redistribution drive these data to show reduced performance or that the reduced number of cells in the grid restricts scalability.

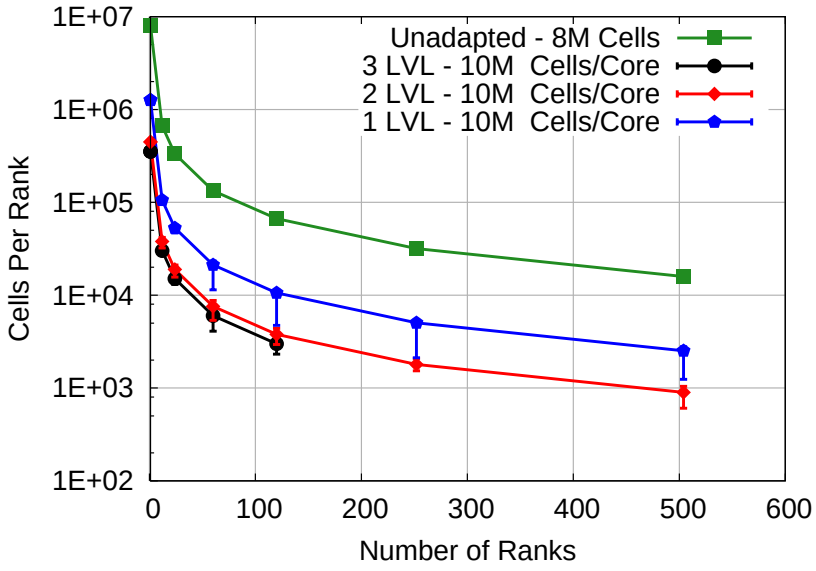
Increasing the number of refinement levels decreases the size of the grid and can result in a less-efficient distribution across processors. Smaller initial grids have fewer degrees of freedom (level zero cells) provided to ParMETIS and the depth of refinement can make cells in important regions disproportionately weighted. In order to investigate this possibility, Fig. 4.12(a) shows the minimum, maximum, and average number of elements in the adapted grids after 400 iterations. In almost all cases, the maximum (represented by the top error bar) and the average (the symbol) are coincident. Many cases show that one rank had a very small cell count (represented by the lower error bar) and indicate a poor distribution where one core has insufficient work and must wait for the others.

The results in Fig. 4.12(a) suggest imperfect partitioning by ParMETIS for some simulations. As the number of ranks increases for a fixed problem size, the imbalance tends to grows. With fewer level zero cells to distribute, it is more challenging to find an ideal work balance. Since the average across all ranks is nearly equivalent to the maximum, only a small subset of the cores are seeing reduced efficiency and this likely plays only a small role in the results in Fig. 4.11.

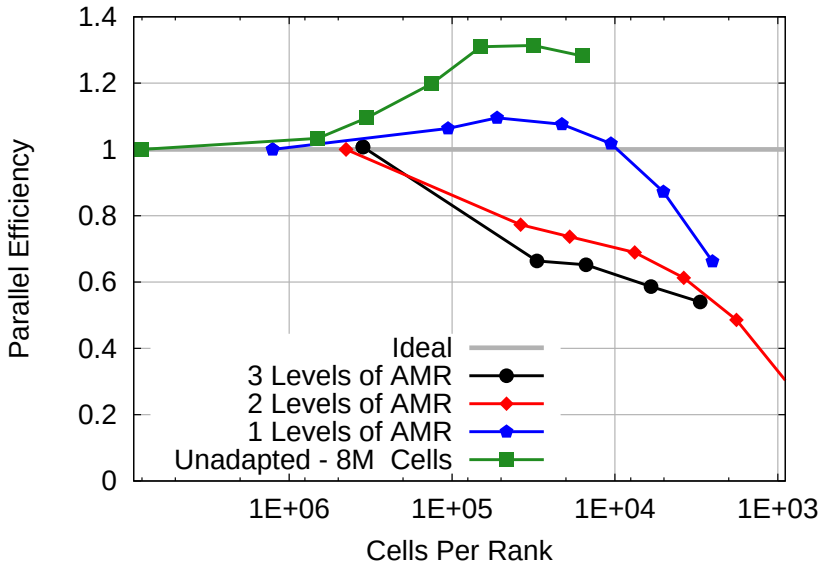
There are diminishing returns with increasing levels of AMR for this problem. Figure 4.12(a) shows nearly identical grid sizes for the cases with 2 and 3 levels of AMR. It is also important to note that each of the adapted grids are an order of magnitude (or more) smaller than the uniform grid. Regardless of the inefficiencies inherent to the partition, this should provide a large savings when absolute runtime is considered.

Figure 4.12(b) directly correlates grid size per rank to the parallel efficiency. Again, there is large difference between the results seen here and the scalability for the uniform grids. The largest of the adapted grids shows a similar trend to the one see for the unadapted results, but it begins to show non-ideal parallel efficiency with 10,000 cells per node instead of the 2,500 seen previously. Smaller grids with additional levels of AMR do not have any super-linear speedup even for equivalent numbers of cells per node.

Figure 4.13(a) presents parallel performance but only includes the time required for



(a) Cells per Rank



(b) Parallel Efficiency

Figure 4.12: Minimun, average, and maximum number of cells across all ranks using adapted grids with varying numbers of cores. Parallel efficiency as a function of cells per rank.

flux evaluation, time stepping, and communication of flow variables at partition fringes. These are the balance of the activities present in an unadapted run. In contrast to Fig. 4.11(a), the scalings are similar to those presented for the uniform grids of similar size (1,262,360, 447,266, and 352,444 for the cases with 1, 2, and 3 levels of AMR). This implies that the reduction in scalability is due to activities present in the adaptation and not a significant reduction in performance by hanging grids and unequal partitions.

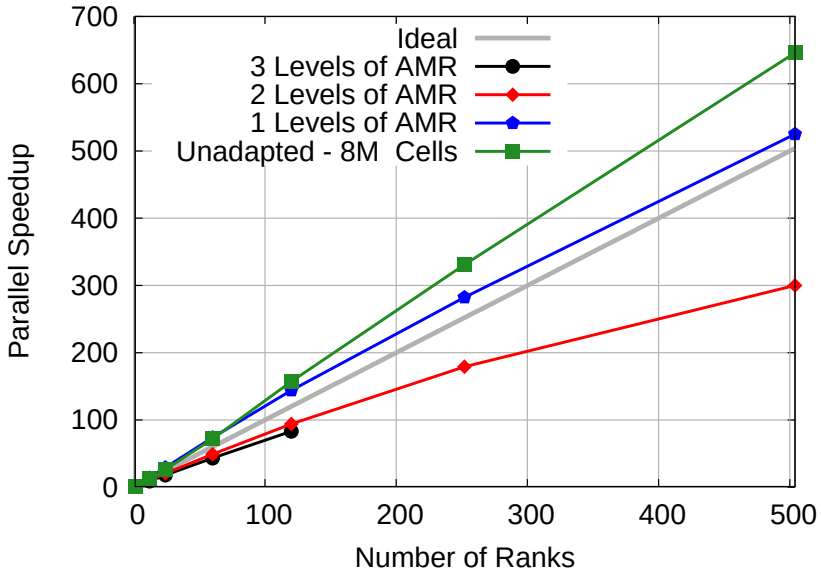


Figure 4.13: Parallel performance of iterating, gradient calculation, and communication on adapted grids and unadapted grids.

Similar to the plots shown for the unadapted grids, Fig. 4.14 presents a breakdown of the relative cost for major routines in the adapted runs. The results are shown in a similar format to Figure 4.9, but these also include the relative cost for all computation and communication used in the AMR and redistribution subroutines. The costs are normalized to the cost of advancing the solution one time step. All are presented as a function of the number of cores used.

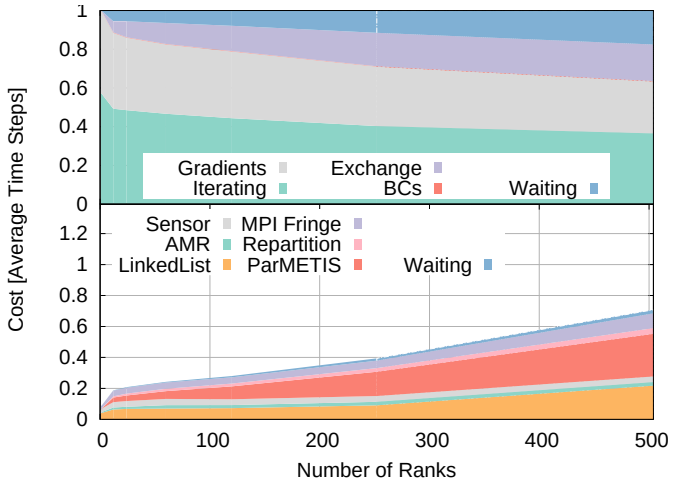
The costs associated with AMR are grouped as follows: *AMR* includes the work involved with creating refined geometry and updating all face and cell metrics. *LinkedList* includes allocation and deallocation of the 1-D arrays, translating from the linked lists

to the 1-D arrays, and memory management of the 1-D arrays during adaptation operations. *ParMETIS* is the cost for building the graph using the required format for ParMETIS and calling the ParMETIS subroutines. *Repartition* combines the costs for transferring exchange elements and recalculation of grid metrics. All of the communication and local work required for determining which cells to refine or coarsen is included in *Sensor*. *MPI Fringe* includes costs required to determine and exchange information associated with the fringe elements and rebuild the required exchange arrays. Finally, *Waiting* is the total time spent waiting for non-blocking receive calls to return during repartitioning and the renumbering of nodes, faces, and cells.

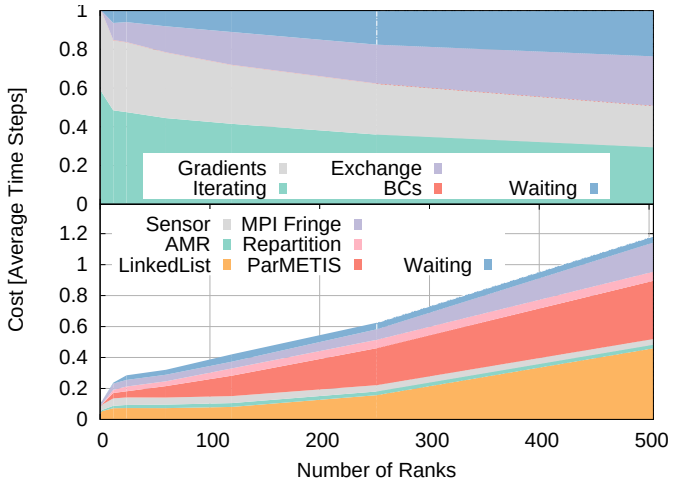
Results in Fig. 4.14 show scalability for the time stepping portion of the code that is similar to what was seen with the uniform grids of equivalent sizes (500,000 when using 2 levels of AMR and 1 million when using 1 level of AMR). There is more idle time due to imbalance between the partitions, though. The largest impact of the AMR is seen in the lower portion of each plot. Costs for adaptation are normalized by the time stepping, so when looking at the 504 core case with 2 levels of AMR, a relative cost of 1.19 indicates that for every second that the code spent advancing the solution, it spend 1.19 seconds handling AMR and repartition events. This cost represents a significant drain on performance and is the reason why the scalability was poor when using AMR across many nodes.

Looking closer, the largest costs associated with adaptation are the mapping to/from linked lists and the 1-D arrays. For this code, the decision to translate between the two was described earlier and this study highlights the consequence of the decision. The other prominent component to adaptation is the call to ParMETIS, which is outside the purview of our work. Both of these costs increase (relatively) with the number of ranks used and represent a fixed cost that does not decrease with the reduced cost of smaller partitions. Fortunately, the costs of AMR, determining where to refine, and repartitioning itself do not grow meaningfully with the number of ranks and represent scalable operations. Recalculation of the fringes increases slightly with ranks, but is not a significant driver, either.

While it is important to understand the scalability of the adaptation and repartitioning, for practical problems it can be more important to understand the absolute cost to solution. To compare the performance of the adapted and uniform grids, Fig. 4.15



(a) 1 Level of Refinement



(b) 2 Levels of Refinement

Figure 4.14: Runtime in major sections of flow solver for adapted grids relative to the cost of time stepping as a function of the number of cores.

shows the total CPU time required to perform 400 time steps for a number of different cases.

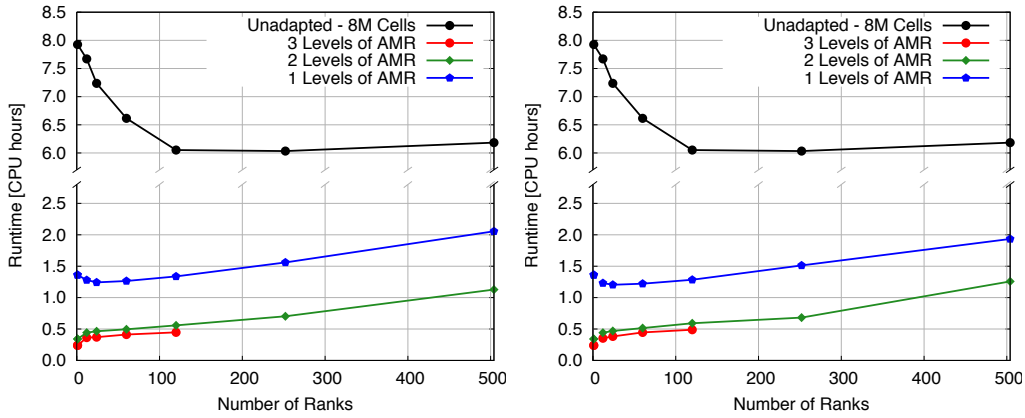
The super-linear scalability seen previously for the 8 million cell uniform grid manifests as a reduction in the CPU time required as additional ranks are used. Ideal scaling would result in a horizontal line on these plots and poor scaling is indicated by an upward trend in CPU time with an increase in the number of ranks for a specific problem.

All of the adapted results show poor scalability as the number of cores approaches 504. For all cases, the savings by using adapted grids is significant.

Figure 4.15(a) illustrates the performance of the baseline adaptation strategy with AMR performed every 10 time steps and redistribution occurring every 50 time steps. Two other strategies are considered, Fig. 4.15(b) attempts to limit costs associated with repartitioning by incurring imbalance and decreasing the frequency of redistribution to once every 200 iterations. This has a small effect, and only for the largest initial grid (with 1 level of refinement) is there a noticeable benefit. Figure 4.15(c) tries another approach and instead greatly increases the buffer size to 11 cells and performs AMR and redistribution every 100 iterations. Unfortunately, this increases the cost of the solution considerably and strongly suggests that frequent refinement is a more efficient approach for unsteady problems.

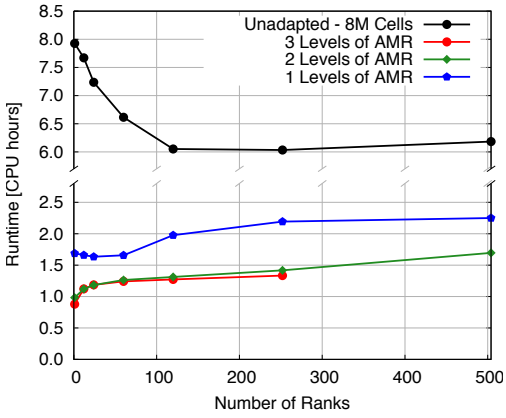
Looking a bit closer to the costs associated with AMR and redistribution for the strategies listed above, Fig. 4.16 shows normalized costs as a function of iteration for the cases using one level of AMR and 252 ranks. The x-axis is iteration number and the AMR and repartitioning events are shown as they occur in time. The costs are normalized to the average cost of the 10 most recent time steps. For the case with the least frequent repartitioning, the growing imbalance for the processors causes significant waiting in Fig. 4.16(b). In general, across all cases, the costs associated with adaptation for this problem are on the order of 2 time steps and redistribution is roughly 10 time steps.

Proper selection for the frequency of adaptation and redistribution are problem specific and the baseline presented here is not a recipe for success across all problem types. However, these results help illustrate the relative cost of adaptation to time advancement and indicate that for moderately sized problems on a large number of cores, AMR can yield considerable benefit for unsteady problems. The reduced parallel efficiency manifested in the adapted results is important to realize and reduce to the extent possible, but does not result in poor performance when compared to the unadapted alternative.



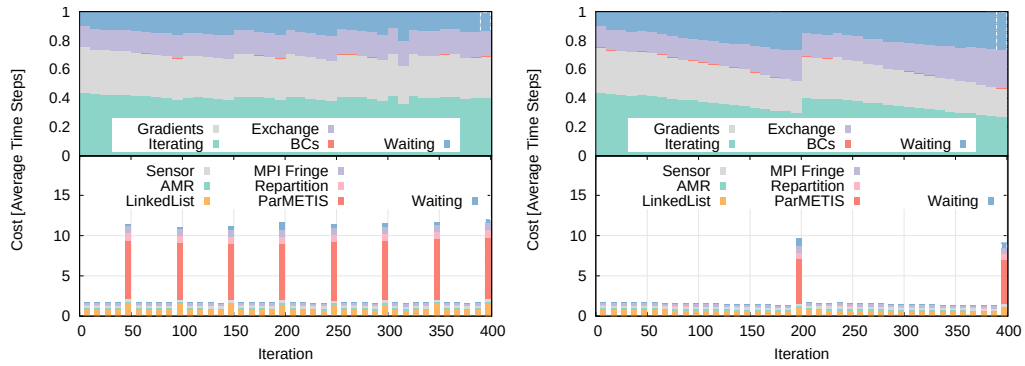
(a) AMR every 10 iterations
Redistribution every 50 iterations

(b) AMR every 10 iterations
Redistribution every 200 iterations



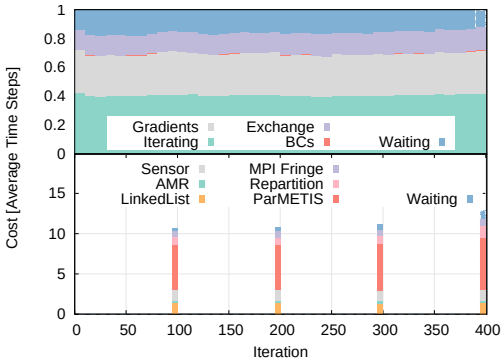
(c) AMR every 100 iterations
Redistribution every 100 iterations

Figure 4.15: Comparison of runtime for adapted grids and unadapted grid.



(a) AMR every 10 iterations
Redistribution every 50 iterations

(b) AMR every 10 iterations
Redistribution every 200 iterations



(c) AMR every 100 iterations
Redistribution every 100 iterations

Figure 4.16: Runtime in major sections of flow solver for adapted grids relative to the cost of time stepping for 400 iterations.

Chapter 5

Validation and Application

The previous chapter showed that the computational methods, with and without adaptation and hanging nodes, accurately represent the governing equations. Having been verified, this work turns to verification and application. There are a number of motivational problems that are interesting and representative of physics seen in more complex flowfields. This chapter includes several numerical simulations that have analytical or experimental data with which to compare.

Application problems were selected to illustrate a range of conditions. There are applications of both the Euler and Navier-Stokes equations - fluid flow with and without viscosity. Steady and unsteady conditions provide interesting challenges for the mesh adaptation and both are considered. This chapter includes investigations of shock-dominated flowfields as well as subsonic, bluff-body flows. One-, two-, and three-dimensional geometries and non-trivial grid topologies provide increasing challenge and afford greater savings through adaptive mesh refinement.

5.1 Sod Shock Tube

The Sod shock tube problem is an unsteady, inviscid flow that principally involves a shock wave, contact surface, and expansion fan across a one-dimensional domain. It was popularized by Sod in 1978 when he examined the resulting flowfield analytically.[63] This problem is frequently used when evaluating numerical flow solvers and is examined here in to compare the performance of uniform and adapted grids.

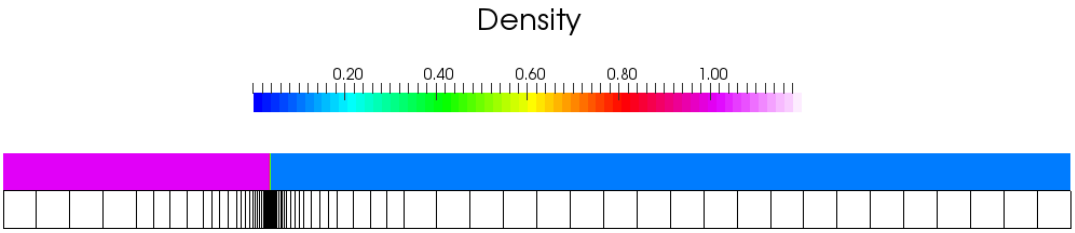
The initial conditions mimic a shock tube with a fictional diaphragm at an x-location of 0.0. At the initial time, the diaphragm ruptures and the high-pressure flow creates a supersonic flow moving into the region of low-pressure flow. There is a contact discontinuity that travels with the high-speed flow and an associated expansion moving in the opposite direction. In a finite domain, both the shock wave and expansion waves reflect off the far ends of the tube and interact. The initial conditions and equation of state for the flow are,

$$x < 0.0 \begin{cases} \rho = 1.0 & [kg/m^3] \\ u = 0.0 & [m/s] \\ p = 1.0 & [Pa] \end{cases}, \qquad x > 0.0 \begin{cases} \rho = 0.1 & [kg/m^3] \\ u = 0.0 & [m/s] \\ p = 0.125 & [Pa] \end{cases}, \qquad \begin{array}{c} \gamma = 1.4, \\ p = \rho RT. \end{array}$$

For this problem, the computational domain extended from -2.5 to 7.5 meters with the addition of solid wall boundary conditions on either end to force wave reflection. The flow was simulated for a relatively long period of time of 20 seconds, in order to observe the interaction of the waves following reflection. RK3 integration in time with a constant timestep, 0.5E-3 seconds, reduced errors associated with the temporal discretization. Due to the presence of strong discontinuities, all simulations use upwinded modified Steger-Warming fluxes.

Adaptation was performed every 10 time steps for the cases run with AMR. There were 2 buffer cells included. A refinement metric based on density gradient ($|\nabla\rho| > \rho_{tol}$) is useful metric for this flow when monitoring the advancing waves. Figure 5.1 shows contours of density at several times throughout the simulation for a solution with a ρ_{tol} = 1E-2. Grid lines are shown to highlight the AMR adaptation. For the initial simulation, a coarse representation of the tube is sufficient to capture the necessary physics.

Figure 5.2 shows a comparisons of the density in the domain for five solutions. One is run on a uniform grid of 1024 cells across the domain. The other four use a range of refinement tolerances and begin with a grid of only 32 cells, but with five levels of AMR. The size of their finest cells in all grids are equal. Several instances in time are shown.



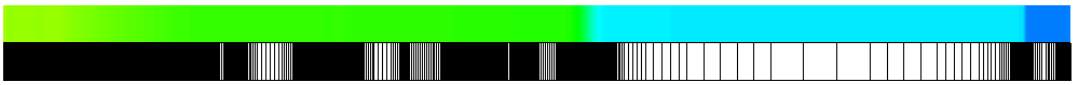
(a) t = 0 s (75 Cells)

(b) t = 1 s (285 Cells)

(c) t = 5 s (820 Cells)

(d) t = 10 s (801 Cells)

(e) t = 15 s (568 Cells)



(f) t = 20 s (600 Cells)

Figure 5.1: Computational mesh, ρ contour, and cell count for 1-D Sod shock tube at six simulation times.

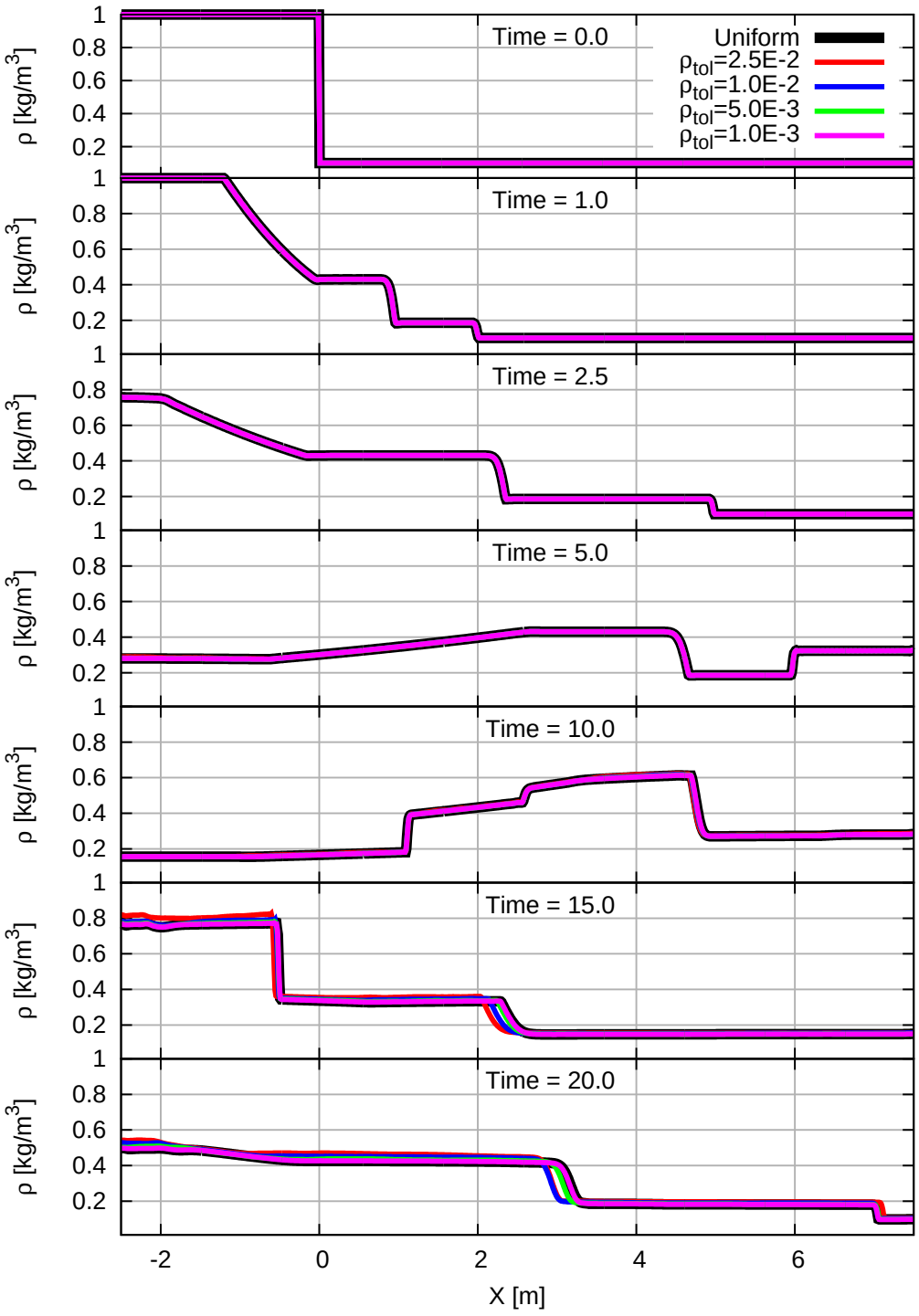


Figure 5.2: Comparison of instantaneous ρ on uniform and adapted grid for Sod shock tube at several times.

From the figure, it is clear that all adapted grids compare favorably for initial propagation of the waves, before reflection off the walls. Afterwards, there begins to be differences near left portion of the figure. Density gradients are small and the adapted grids begin to coarsen their representation of the mesh. At the final time shown here, 20 seconds, all adapted grids capture the character of the flow inside of the shock tube. All but the finest refinement tolerance show deviations form the uniform grid case.

Another way to present the results is as show in Fig. 5.3. The y-axis of the figure is time and the x-axis is x-location in the tube. The contour on the figure is density and there are a number of couture lines plotted as well. The initial waves are identified and labeled - their reflections off the walls of the tube are apparent. Figure 5.4 isolates just the contour lines and compares the results obtained form the uniform grid to the adapted results. Between the data sets, there is very little disagreement until about 10 seconds into the computation.

Figure 5.5 shows the number of cells in each of the solutions as a function of time. As is clear, the adapted case with the smallest refinement tolerance typically uses a full 1,024 cells across the domain - very similar to the uniform grid case. This explains the very good agreement. Choice of refinement tolerance is significant and feature-based tolerances require some tuning in order to ensure that they are relevant for the problem of interest. For this problem, researchers investigating the initial propagation of the shock and expansion waves could use even the largest tolerance here and achieve near-total agreement with the uniform grid case. However, if the aim is to characterize the flow inside of the closed tube after several reflections, 20 seconds, then much more stringent requirements are necessary with this choice of refinement parameter.

For comparison, the time required for each solution to simulate 20 seconds of flow time is shown in Table 5.1. As expected, the less-demanding refinement tolerances require much less computational time but incur additional error as seen previously. The finest tolerance actually requires more computational time than the uniform case because it does not see significant reduction in the number of cells but must calculate gradients in the solution and perform AMR and feature-detection. Costs associated with those portions of the solve are also shown.

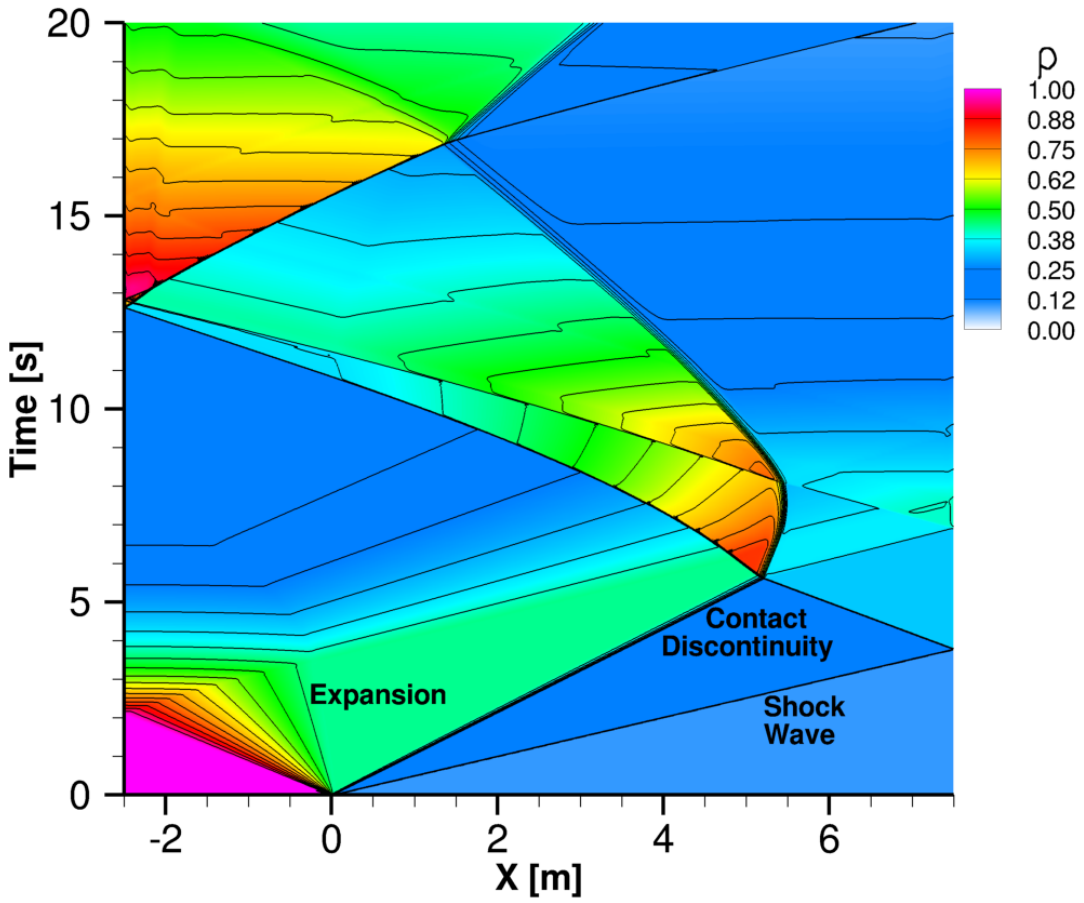


Figure 5.3: Isolines and contour of ρ in time and space for Sod shock tube; uniform grid.

AMR Tolerance	Total Time [s]	Gradients [s]	Sensor / AMR [s]	Linked Lists [s]
2.5E-2	115.5	4.41	3.16	4.95
1.0E-2	149.3	5.28	3.85	6.12
5.0E-3	170.8	5.49	4.46	6.63
1.0E-3	198.3	5.48	5.22	7.12
Uniform	191.0	0.00	0.00	0.00

Table 5.1: Time required to solve 20 seconds of flow time for the Sod shock tube problem.

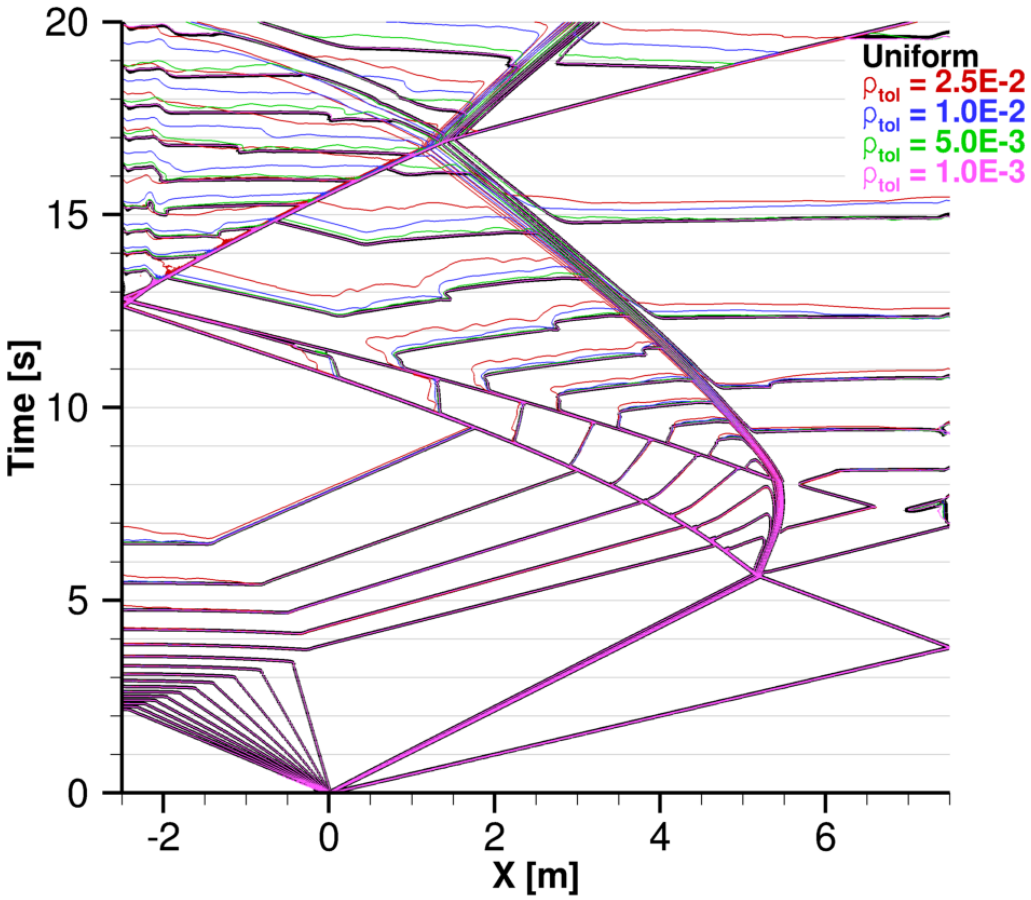


Figure 5.4: Isolines of ρ in time and space for Sod shock tube; uniform and adapted grids.

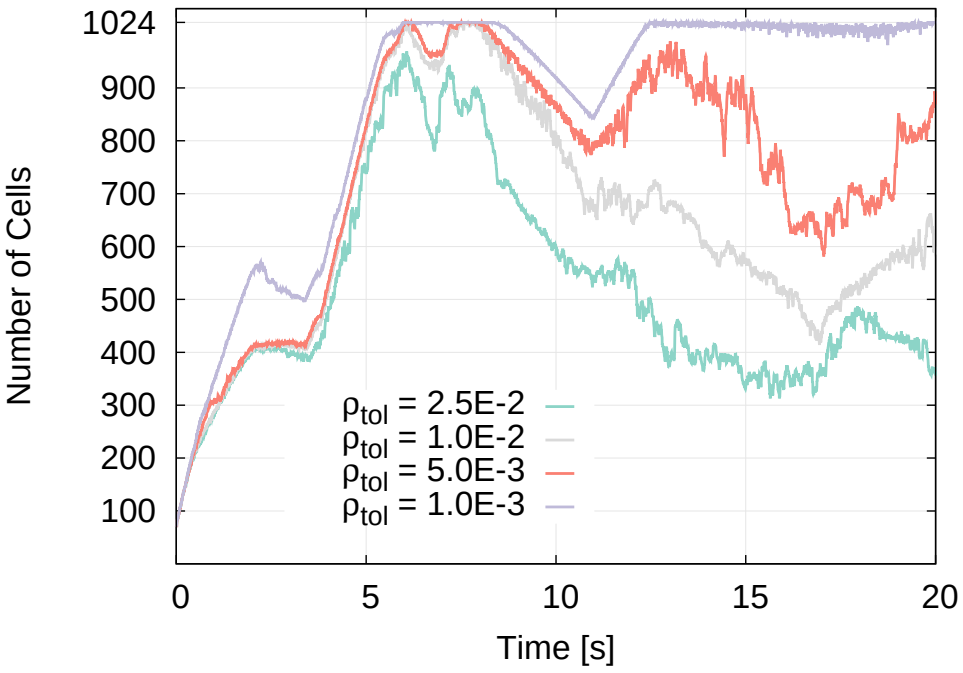


Figure 5.5: Number of cells for adaptively refined solutions as a function of simulation time.

5.2 Hypersonic Double Wedge

An important phenomenon for researchers and engineers in the field of hypersonics is the shock-shock interaction. One of the simplest experimental and computational geometries used to investigate the effects of this interaction is the double wedge. Another common geometry for similar flows will be discussed later, the axisymmetric double cone. By varying the freestream conditions and the wedge or cone angles, an entire family of shock-shock interactions are observed (as classified by Edney).[27] Simulation of an inviscid double wedge provides a steady test case for demonstration of the adaptive mesh strategy at work for a physically relevant flow.

Researchers have performed implicit computational simulations of double wedges using Modified Steger-Warming fluxes similar to those described in this work.[53] Those results provide a guide for expected behavior and aid in selection of an appropriate test case. Several shock-shock interactions create relatively large portions of subsonic flow with unsteady flow characteristics. For the perfect gas solver, a double wedge with cone half angles of 15° and 35° is selected as a steady interaction for analysis here. At a freestream condition of Mach 9.0 and $\gamma = 1.4$ this geometry produces a Type VI shock-shock interaction.

Shock-shock interactions can generate a highly-complex arrangement of flow structures. Figure 5.6 shows a schematic of the important flow features in an interaction of the type examined below. The attached shock creates a complex array of alternating expansion and compression waves that travel down the length of the second wedge. Regions of the flow are identified by numerical references for analytical comparison.

While not accurate inside of the compression or expansion fans, analytical tools can predict the pressure in the regions identified in Fig. 5.6. The pressure in region 1 is easily attained from the wedge geometry, freestream conditions, and the oblique shock relations. Likewise, region 2 is derived from the conditions inside of region 1 and another application of the oblique shock relations. Regions 3 and 4 require a slightly more complex derivation. The relationship between regions 2 and 3 is governed by isentropic, Prandtl-Meyer expansion. Region 4 can be described by the freestream conditions and the oblique shock relations - provided the flow direction in regions 3 and 4 is known. To determine the flow direction, a solution to that maintiains $p_3 = p_4$ must

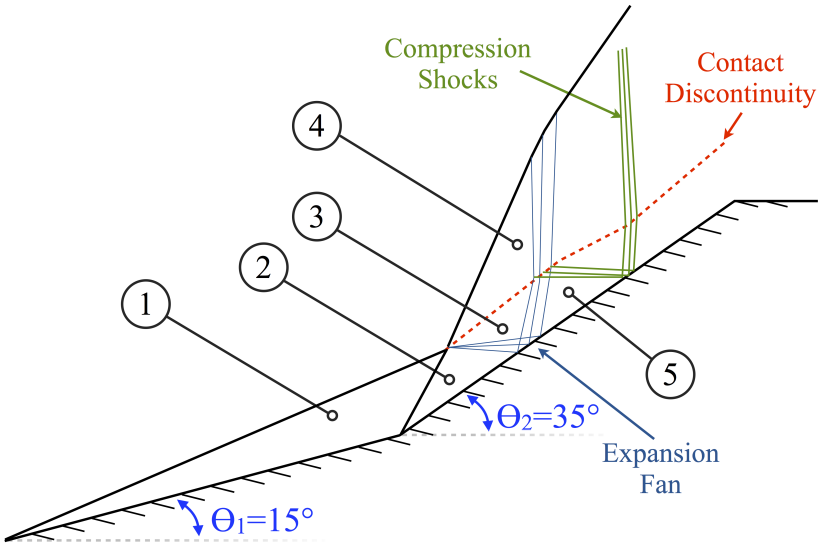


Figure 5.6: Schematic of important flow features in 15°-35° type VI shock-shock interaction.

Region	1	2	3	4	5
Mach	5.04	3.04	3.25	1.73	3.48
$\frac{p}{p_\infty}$	11.24	79.95	58.61	58.61	42.14

Table 5.2: Conditions for regions in double wedge show in Fig. 5.6; Mach 9, $\gamma = 1.4$.

be found. Once it is found, the conditions in regions 3 and 4 is known. Finally, the pressure in region 5 is obtained by isentropic expansion from region 3. Table 5.2 shows conditions for all five regions.

5.2.1 Grid Generation

Work by Olejniczak et al. showed that a two-dimensional grid of 1024×1024 elements was necessary to capture the smallest-scale features in the double wedge simulations.[53] To better estimate grid convergence and allow for comparison on higher density grids, a series of uniform grids were made with the finest being 2048×2048. The grids were similar to Olejniczak's in topology with grid lines normal to the wedge surfaces and generally isotropic cells. While inviscid, there was still a small amount of clustering towards the wall and corner to better resolve the shock reflection.

This finest uniform grid was sequenced in each dimensions in order to to create successively smaller domains with the minimum being only 32×32. This minimum grid was then refined using AMR to recreate a resolution identical to the fine grids. The numerical results from the adapted grids will be compared against uniform grids with similar cell spacings in their finest cells.

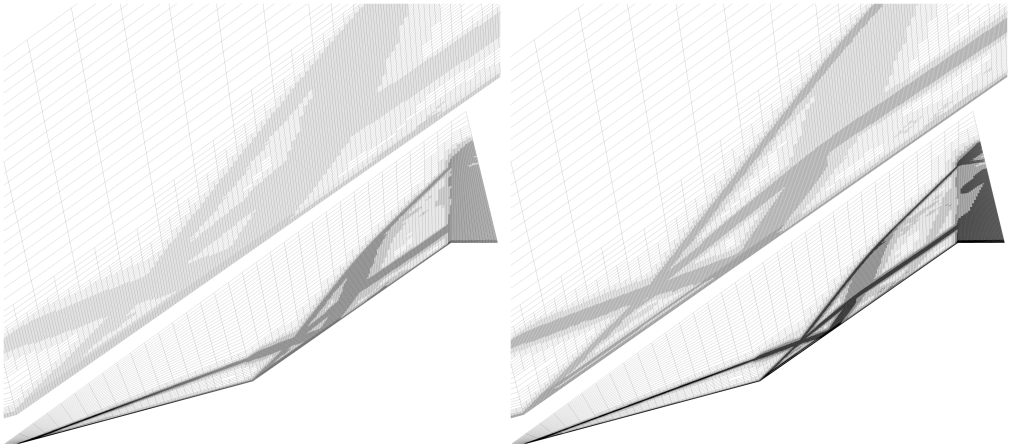
5.2.2 Adaptation Strategy

The double wedge problem is steady and typically requires only one flow time to converge the numerical result. Grid adaptation was performed every flow time. This was repeated until the solution had converged on the finest allowable grid level. For this test case, one flow time (the time it takes for a fluid element on the upstream portion of the wedge to travel to the most downstream point) is slightly less than 0.0016 seconds. Cells were identified for refinement using undivided difference sensors (Eq. (3.1)). Refinement could be triggered by one of two tolerances: ρ_{tol} = 1E-2 and p_{tol} = 1E-2. Each flow time, the maximum grid level across the domain was increased by one. A progression of grid levels is shown in Fig. 5.7 with inset images providing detail near the corner.

One consideration that should be mentioned is grid cell alignment. The uniform grids were smoothed using an elliptic smoother. With the coarser representations of the grid, the AMR routine does not perform any additional smoothing steps and simply subdivides the faces. For this reason, the 1024×1024 grid generated by AMR is not as smooth as the fine grid from the grid generation code. The grid lines are not necessarily normal to the boundary near regions of high curvature (in the 15°-35° corner).

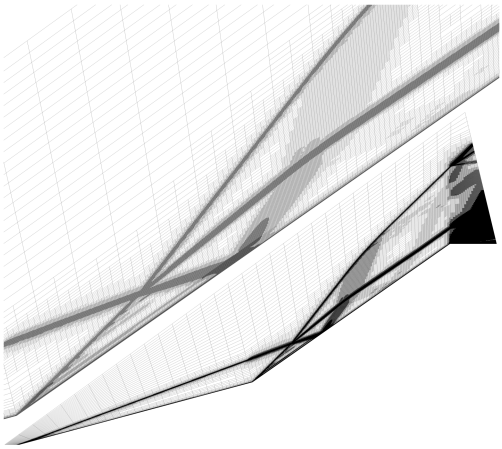
As will be seen below, there is a noticeable difference between the solutions due to these effects. It is only apparent in regions of the flow where the solution is already sensitive to misalignment and where the original grid was highly skewed due to the smoothing operations. Figure 5.8 shows a close-up of the grid near the 15°-35° corner for a uniform mesh of 512×512 points and an initially coarse mesh refined to the same global cell density. Most noticeable is the difference in wall spacing on the 35° ramp.

The impact of these differences in grid quality is not explicitly examined in the following presentation of the results. Subsequent discussion identifies altered pressure predictions between the uniform grid and the grids generated using AMR specifically in this portion of the geometry. Additional simulations (not shown) were performed by



(a) 4 levels (512 ×512)

(b) 5 levels (1024×1024)



(c) 6 levels (2048×2048)

Figure 5.7: Images of the mesh for varying levels of refinement and their effective resolutions.

uniformly refining a grid using AMR in order to remove the possibility of hanging nodes causing these differences. Those simulations agreed with the AMR solutions with hanging nodes, strongly implicating these grid quality issues in the observed discrepancies in region 2.

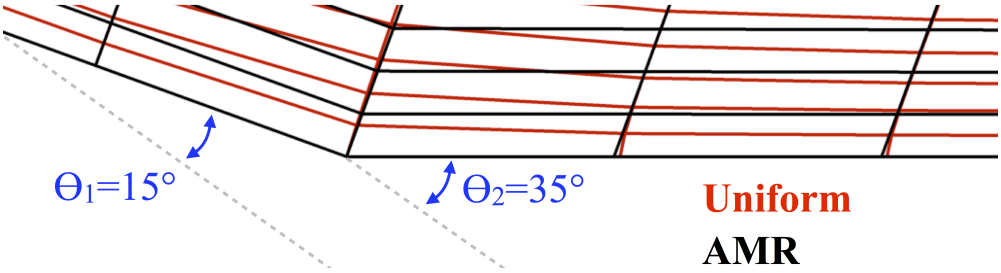


Figure 5.8: View of cells for 512×512 uniform grid and coarse grid refined to 512×512 with AMR.

5.2.3 Results

For a steady-state problems, there is no requirement to track features as they migrate from areas of higher cell density to coarser regions. Transient events are not important and by moving to coarser cells they are dissipated and removed from the solution. With unsteady calculations, the timestep should be reasonable when considering the convective (or acoustic) speeds. The solutions presented below were run at a timestep that was 500-times greater than the largest stable explicit timestep (CFL=500).

Figure 5.9 shows a grid convergence study using uniform grids. The solutions were run over MPI on ten processors. The figure shows computational results for pressure relative to the freestream value (p_∞) on the surface of the double wedge. Also shown on the figure are the analytical predictions for the pressure ratio in regions 1, 2, and 5 (gray lines). Insets highlight the pressures in regions 2 (upper-left) and 5 (lower-right).

In general, the pressure comparisons are good when compared to the theory. Region 2 shows a non-physical jump in pressure at the shock and a great deal of ringing in the solution downstream. Where the expansion fan between regions 2 and 5 intersect the body, the ringing disappears and the solution compares very similar to what the analytical result. Grid convergence is not entirely realized with this series of grids: the specifics in region 2 are becoming a better match to the theory and the width of the expansion fan and compression shocks are tightening. However, as was shown in previous studies, there is very little change in the gross flow features or comparison to theory after the 1024×1024 grid.

A similar progression of grid is presented for comparison with adapted grids. Several additional resolutions are also considered - these correspond to uniform grids that were

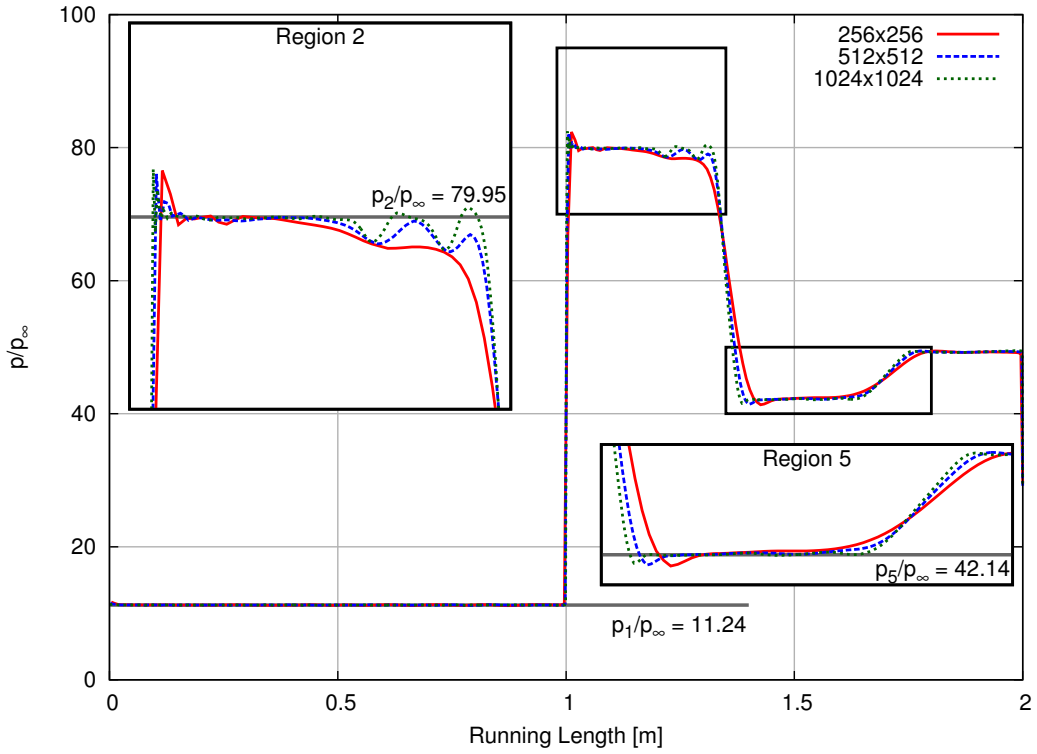


Figure 5.9: Surface pressure on the wedge for a series of uniform grids with varying grid densities.

considered too expensive for this analysis. Figure 5.10 shows results for wall pressure obtained on adapted grids with hanging nodes. The values are very similar to those seen previously. In fact, when plotted together, the only differences that exist when using identical grid resolutions are the pressures in region 2. These results illustrate that grid convergence may be obtained near a resolution of 4096×4096. This is higher than was estimated in the prior work by Olejniczak et al.

Solutions on grids obtained using AMR have a more pronounced peak and present a different pattern of oscillations prior to the impingement of the expansion fan. As was mentioned earlier, this is attributed to the difference in the grids based on simple cell subdivision (see Fig. 5.8). With refinement in the AMR, the oscillations in region 2 become less pronounced and the pressure peak becomes more localized, even though

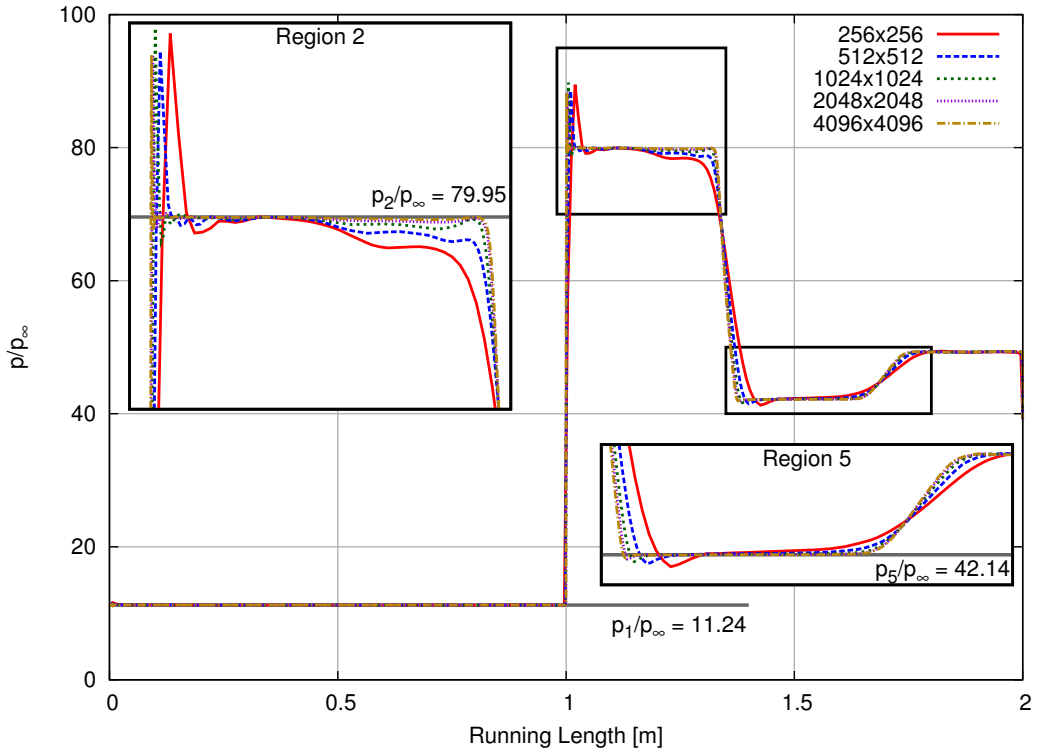


Figure 5.10: Surface pressure on the wedge for a grids with varying levels of AMR.

its magnitude remains unchanged. Downstream of the large pressure peak, the cases with AMR show a more accurate pressure profile in region 2 with reduced oscillations as compared to the uniform grid cases.

5.2.4 Computational Efficiency

The solutions obtained on adapted and unadapted meshes are very similar. All other costs being equal, adapted grids should be much less expensive to iterate on because they use many fewer cells. There are additional costs associated with adaptation that include additional overhead and previous work in this document has shown that they can be significant when compared to the cost of time stepping alone. The refinement frequency and solution duration effect total time-to-solution. Several additional simulations were

run to provide results for timing and sensitivities to these parameters.

The previous adapted results were for simulations that performed AMR every flow time (FT) - they had a refinement frequency of 1 [1/FT], or $f = 1$. To ensure that they an opportunity to fully develop on the finest mesh, the solutions were run for a number of flow times equal to the level of maximum refinement plus one. In addition to this baseline case, a case that had a much higher frequency of refinement was run. It performed adaptation 16 times per flow time ($f = 16$) but still limited the maximum grid level to what was used for the $f = 1$ case. Another set of cases were run that had a refinement frequency of 16 but had no limit on the maximum grid level as a function of simulation time. These solutions were only run for 1 flow time since their results did not change after that point.

Figures 5.11-5.13 shows the number of cells in the adapted grids as a function of simulation time. The number of cells in the unadapted grids with a uniform grid level are also shown as a solid line across the plot; the uniform cases were only run for one flow time. For the adapted grids that were run for more than one flow time, the grid level was fixed each flow time as mentioned previously. As the solution advances, there is a consistent trend in cell count growth. Rapid adaptation over a single flow time ($f = 16$) sees significant growth for the first half flow time and then a plateau of refinement that reduces the mesh size. Regardless of the adaptation strategy, the final number of cells is consistent at the end of the simulation.

Figure 5.14 shows a metric for integrated force on the double wedge geometry over several flow times. The y-axis shows the percent difference in the x-force relative to the final result from the uniform grid. These solutions correspond to a 256×256 uniform grid and the adapted solutions use 3 levels mesh refinement. Between flow times three and four the adapted solutions run for four flow times are on the finest mesh.

The solution run for four flow times see large jumps in their force coefficients when the meshes move to a finer cell size. After these events, the solutions achieve a nearly-steady solution after only half a flow time. After roughly 3.5 flow times they have achieved an error on the order of zero percent. More detailed observations show that the error caused by refinement in the shock near the tip of the geometry dissipates as it passes through coarse cells in region 1. In effect, this reduces the flow time for these simulations and accelerates convergence. The adapted solution with $f = 16$ and no limit

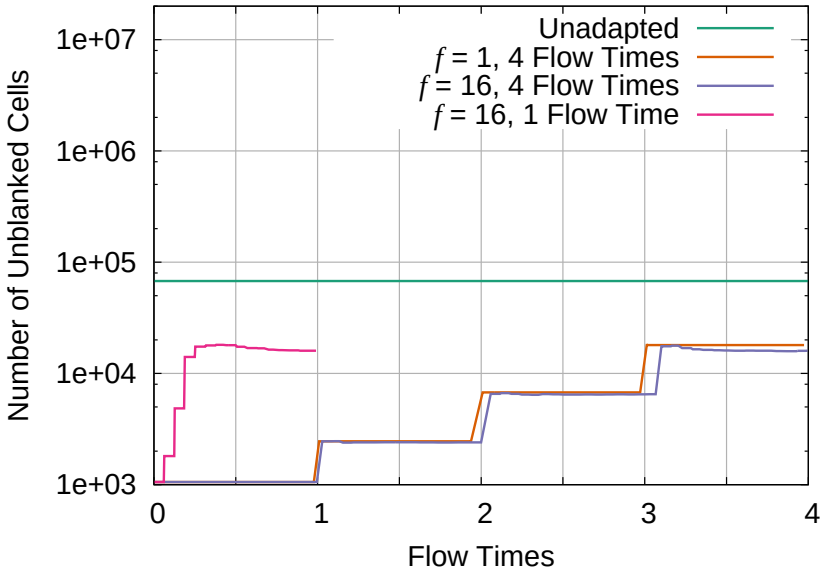


Figure 5.11: Cell count versus solution time for cases with three levels of refinement compared to a 256×256 unadapted reference.

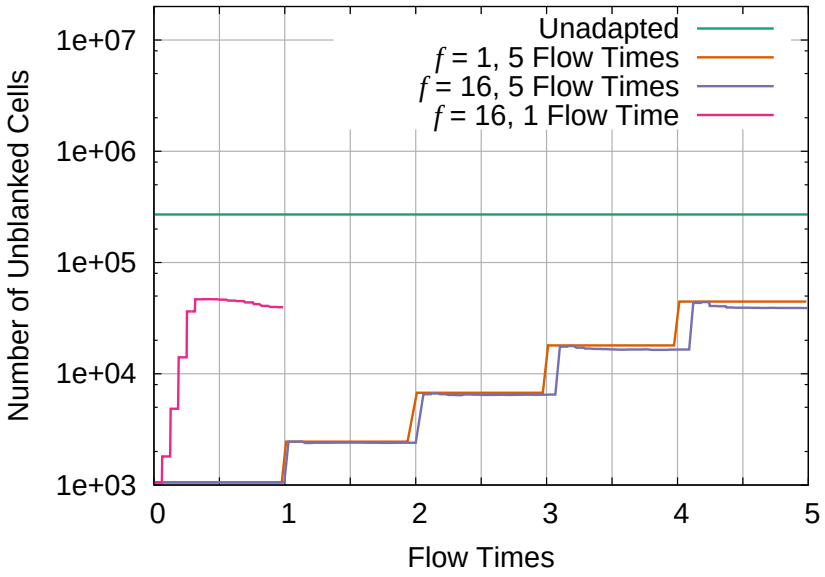


Figure 5.12: Cell count versus solution time for cases with four levels of refinement compared to a 512×512 unadapted reference.

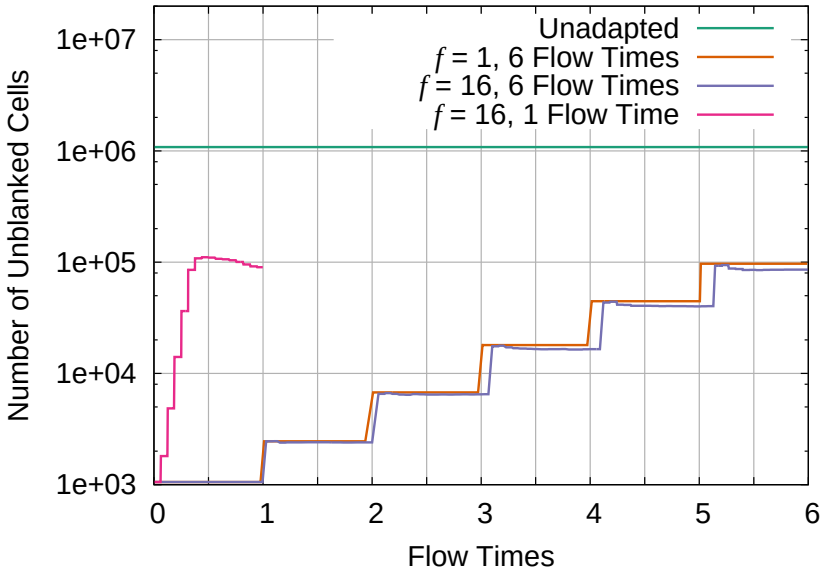


Figure 5.13: Cell count versus solution time for cases with five levels of refinement compared to a 1024×1024 unadapted reference.

on the mesh refinement reaches a converged value identical to the uniform case after only one flow time. It is not surprising that the results reach their final value within one flow time on the final mesh. We expect a steady, supersonic flowfield to converge in roughly one flow time due to the hyperbolic character of the Euler equations.

Table 5.3 shows significant statistics for these solutions. In addition to documenting the amount of time (as a percentage of the total runtime) spend performing each of the major solver functions, it also compares the total walltime and number of cells in the final meshes to those required for the uniform case. All cases were run in parallel on 10 core using MPI. The cell count for the uniform grid is slightly higher than what is expected for a 256×256 mesh because the nominal dimensions (256 cells) refer only to the cells on the two wedge faces and not on the flat portion after the double wedges (which adds additional cells in the stream wise direction).

These results allows several interesting observations. This problem realizes significant benefit by using dynamic adaptation compared to the uniform grid reference. Regardless of the adaptation strategy used, adaptive griding provides a 85-90% savings. The percentage of the runtime for the adaptive runs spent performing repartitioning

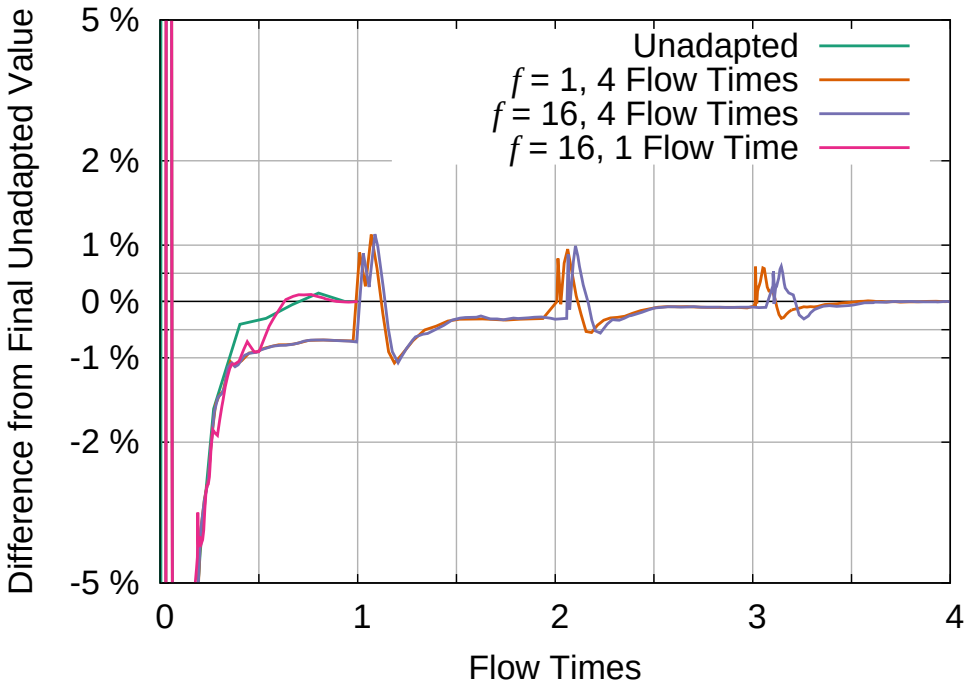


Figure 5.14: Percent difference in integrated x-direction force for 256×256 uniform grid and those with three levels of AMR.

and memory management is related to the frequency of the adaptation. In general, AMR required less than 15% of the runtime for the adapted cases enabled by a nearly 75% reduction in cell count.

The same set of cases were examined for a refined solution using 512×512 cells on the double wedge. Figure 5.15 shows the convergence of the x-direction force with a very similar behavior to what was seen for the coarser grid. The grids with a gradual development of cell refinement require about 4.5 flow times to converge while the uniform mesh and the rapidly refined grid require only a single flow time. Table 5.4 contains timing and cell count statistics for these solutions. As seen previously, use of AMR provides considerable benefit as compared to a uniform grid. Use of additional levels of refinement (four as opposed to three) reduces the relative cost of the adapted grids to only about 10% of the unadapted grid cost.

Similar to the results using three level of adaptation, those obtained for the solutions that used four levels adaptation yields significant savings for this steady-state problem.

	Uniform	Adapted	Adapted	Adapted
Refinement Frequency [1/Flowtimes]	-	1	16	16
Simulation Time [Flowtimes]	1	4	4	1
Time Marching	87.35%	83.84%	78.58%	78.25%
AMR / Feature Detection	0.00%	0.07%	0.59%	0.63%
Repartitioning / Memory Management	0.00%	0.71%	6.19%	5.60%
MPI Exchange / Waiting	10.97%	13.68%	12.84%	13.81%
Walltime [s]	117.90	17.59	17.24	13.98
Walltime (Relative to Uniform)	-	14.92%	14.62%	11.85%
Final Number of Cells	67,584	18,201	15,990	15,957

Table 5.3: Timing and cell count comparisons for uniform 256×256 mesh and equivalent grids with three levels of AMR.

	Uniform	Adapted	Adapted	Adapted
Refinement Frequency [1/Flowtimes]	-	1	16	16
Simulation Time [Flowtimes]	1	5	5	1
Time Marching	87.37%	85.13%	82.14%	80.93%
AMR / Feature Detection	0.00%	0.05%	0.45%	0.60%
Repartitioning / Memory Management	0.00%	0.51%	4.14%	5.34%
MPI Exchange / Waiting	10.93%	12.54%	11.36%	11.34%
Walltime [s]	652.65	66.32	60.81	53.70
Walltime (Relative to Uniform)	-	10.16%	9.32%	8.23%
Final Number of Cells	270,336	45,079	38,935	39,235

Table 5.4: Timing and cell count comparisons for uniform 512×512 mesh and equivalent grids with four levels of AMR.

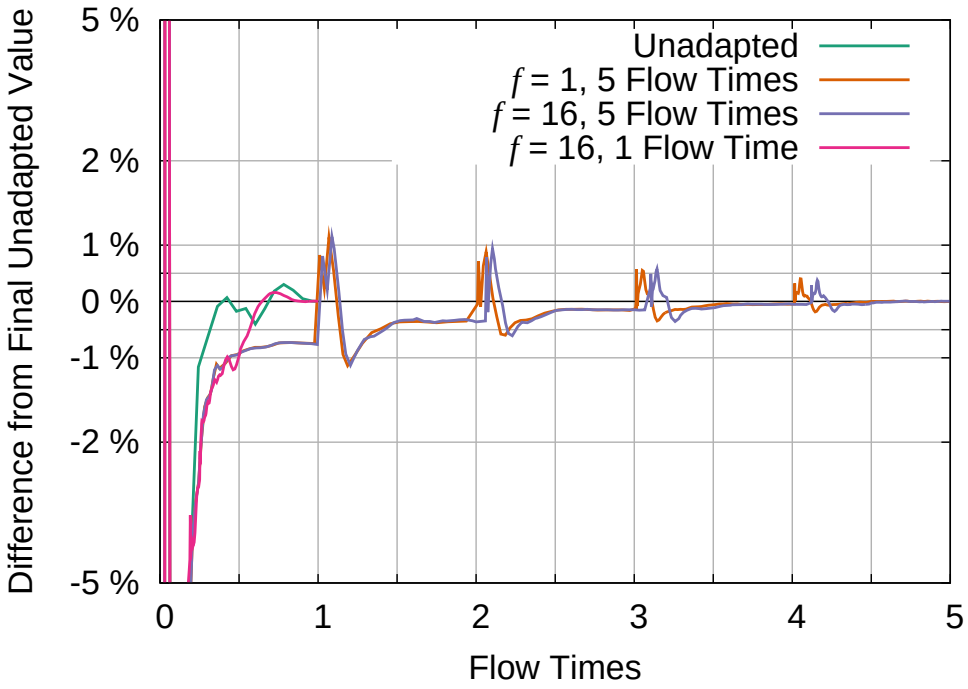


Figure 5.15: Percent difference in integrated x-direction force for 512×512 uniform grid and those with four levels of AMR.

Costs associated with AMR, feature detection, repartitioning, and memory management are similar as a percentage of total runtime which is significant since it illustrates that these costs are not appreciably impacted by using additional levels of refinement. The uniformly refined grid used here has a factor of four times the number of cells in regions of the grid that are not important for capturing the shock and interactions when compared to the 256×256 grid. This favors the adapted grids that now represent a 90% savings in walltime and an 85% reduction in cell count.

Finally, this comparison was repeated for the grids equivalent to the solution using 1024×1024 cells on the double wedge. Figure 5.16 shows the convergence of the x-direction force and Tab. 5.5 contains the timing and cell count statistics. These results continue the trend seen between the previous two cases. Use of AMR, regardless of the adaptation strategy (frequency or solution duration), reduces the runtime by roughly 95% and cell count by 90%. Again, the amount of time spent performing operations related to adaptation are no more expensive than they were with the cases using fewer

levels of refinement.

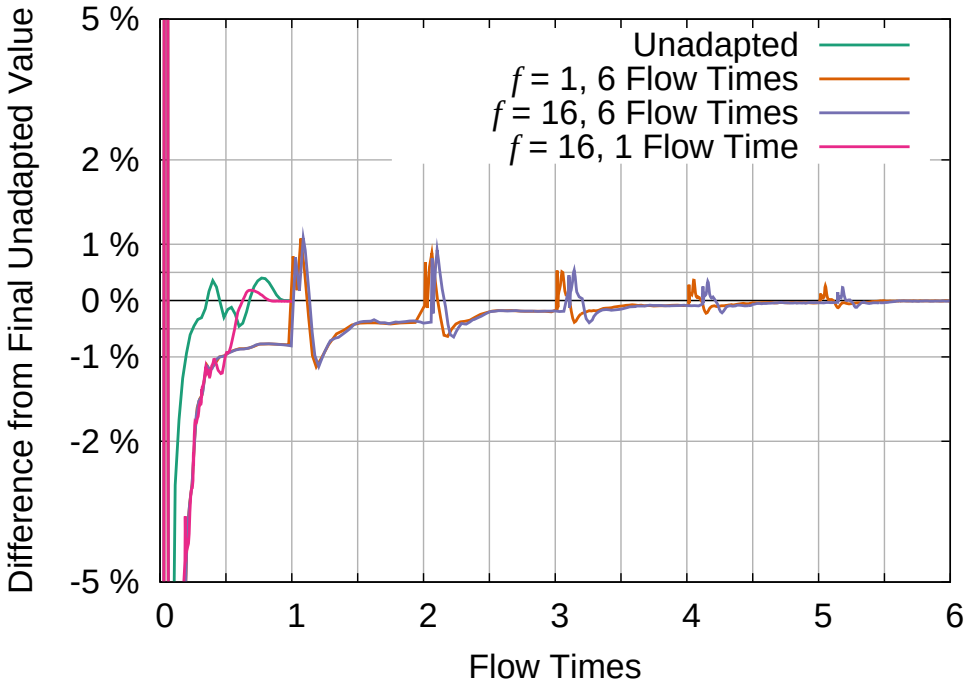


Figure 5.16: Percent difference in integrated x-direction force for 1024×1024 uniform grid and those with five levels of AMR.

These results point to the opportunity for adapted mesh refinement to deliver comparable accuracy at a much reduced computational cost for shock-dominated flows. However, these conditions were inviscid and steady. Additional work is required to explore more complicated grid topologies, numerics associated with viscosity, and flows with elliptic character.

	Uniform	Adapted	Adapted	Adapted
Refinement Frequency [1/Flowtimes]	-	1	16	16
Simulation Time [Flowtimes]	1	6	6	1
Time Marching	88.08%	86.32%	85.00%	82.54%
AMR / Feature Detection	0.00%	0.03%	0.36%	0.50%
Repartitioning / Memory Management	0.00%	0.39%	3.22%	4.09%
MPI Exchange / Waiting	10.18%	11.49%	9.53%	11.10%
Walltime [s]	3954.42	233.57	216.38	188.81
Walltime (Relative to Uniform)	-	5.91%	5.47%	4.77%
Final Number of Cells	1,081,344	97,859	85,694	88,745

Table 5.5: Timing and cell count comparisons for uniform 1024×1024 mesh and equivalent grids with five levels of AMR.

5.3 Hypersonic Double Cone

This section examines another standard problem for the hypersonic community, the hypersonic double cone. Experimental shock tunnel tests were conducted on double cone geometries at CUBRC.[38] These test results have been formed the basis of previous computational comparison by a number of authors working on hypersonic CFD.[25;50] Due to its strong applicability to the field and its history of numerical comparisons it is an important validation case. Researchers working on AMR or solution adapted grids have had success simulating these flow fields in the past.[33;44]. An active area of research that involves similar flow structures in hypersonic aerodynamics is ramjet and scramjet inlets. Recent work at relevant conditions on inlet geometries with multi-resolution-based AMR has shown compelling results.[28]

The problem of the hypersonic biconic is a good test problem use with AMR. Freestream conditions strongly influence separation and shock locations and can dramatically change the character of the interaction. Ideally, a grid made for the problem would incorporate tailored boundaries that would conform to the final shock shapes, but this can be an expensive process and involves iterations between grid generation and simulations of the flow field. With AMR it is possible to use a coarse mesh for initial solutions and allow the grid to refine only near important features. Grid alignment is important, but with sufficient grid density, the error due to misalignment is reduced. Additionally, certain conditions can yield unsteady results. Application of an unsteady-AMR capability can similarly track the movement of the shock and separation region and maintain an inexpensive grid for computation.

One run from a series generated at the CUBRC test facility is explored below. This is a common dataset used for flow solver validation. The flow conditions for the cases are listed in Table 5.6. The double cone model had a length of 0.18 meters with half angles of 25° and 55°. Similar to the double wedge examined previously, a flow time is defined as the time taken by the freestream flow to travel the characteristic length. With the nominal freestream velocity, 2712.7 [m/s], one flow time is 6.635E-5 seconds and is referenced in the results that follow. Figure 5.17 illustrates the flow features on a double cone and shows an image of density gradient magnitude from a numerical simulation from the following results.

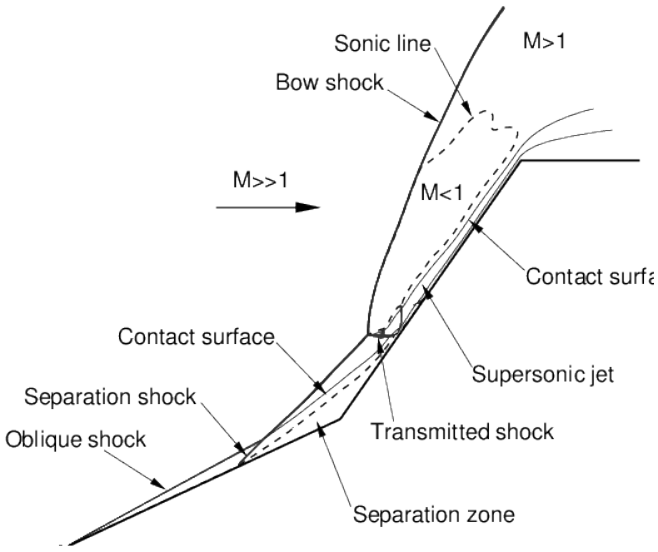
Mach	11.30	[-]
ρ_∞	5.515E-4	[kg/m^3]
u_∞	2712.7	[m/s]
T_∞	138.89	[K]
T_{wall}	296.11	[K]

Table 5.6: Freestream conditions for run 35.

These solutions model the freestream as a non-reacting flow in thermodynamic equilibrium. Nompelis *et al.* have shown that accounting for vibrational non-equilibrium by modeling the flow inside the nozzle preceding the test section can greatly improve the prediction of peak heating.[50] Comparisons between the performance of unadapted and adapted grids are the focus of this work. By choosing not to model these physics, this work accepts the resulting discrepancy between the solution and the test data, but provide a consistent set of assumptions across the numerical results. Resolution of the flow features is strongly dependent on the dissipation in the numerical model used in the flux evaluation. Based on the previous work by Druguet, the modified Steger-Warming fluxes are sufficient.[25]

The hypersonic double code is a pseudo-steady problem. Flow separation in the corner where the two cones meet can take a significant number of flow times to develop. Previous work has found that it requires on the order to 100 flow times to converge.[30] Since the flowfield is strongly dependent on resolution of the viscous boundary layer, using small cell spacings dictates the maximum stable explicit time step. Due to the disparate length scales introduced by very small cells on the relatively large model, implicit time integration is required.

Included numerical results employ a coupled DPLR-FMPR implicit operator. The line solves extend from the viscous wall into the domain and are truncated by hanging-node or the opposite boundary as outlined previously in Sec. 2.5.2. For the problem of interest, the flow will establish a steady flow state and time accuracy is not required. A maximum CFL of 4,000 was selected to quickly advance the results and mitigate the extremely small time step prescribed by the viscous wall spacing. This points to another motivation for using DPLR; employing FMPR alone creates violent oscillations in the solution at CFL values of this order. With DPLR, the ability to use higher CFL values decreases time to convergence by more than an order of magnitude.



(a) Description of flow features near double cone. [50]

(b) Contour of $|\nabla\rho|$ from double cone simulation.

Figure 5.17: Illustration of flow features and magnitude of density gradient in numerical result.

5.3.1 Grid Generation

Grid requirements are well understood for this double cone problem. Previous research has demonstrated that a grid of 524,288 cells (1024×512) is sufficient to resolve all relevant flow structure and properly predict the length of the separation region. Figure 5.18 presents the results from one such study. For nearly a third of the domain, upstream of the separation shock, the solution remains unchanged with the addition of cells. It is likely that additional resolution was unnecessary in this region. Shown below, targeted refinement with AMR avoids adding cells in this portion of the domain and instead focus them towards the latter two-thirds of the geometry.

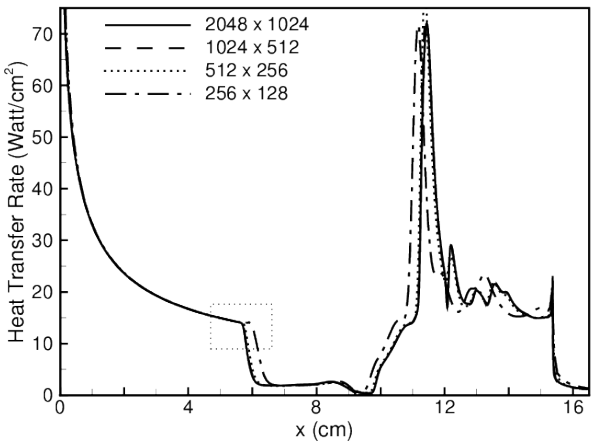
A grid convergence study on uniform meshes was performed as part of this work (not shown). It also found that characteristics of the flow did not change with additional refinement. The length of the separation region and the size and shape of the shock structure was not affected by doubling the resolution past 1024×512.

This is a relatively small problem with only half a million cells. For this reason, initial grid sizes are very small for the adapted cases. When using 4 levels of refinement (the most considered here) the domain includes only 64 cells along the double cone and 32 cells normal to the surface of the model; 2,048 cells in total. Due to the constraints in using ParMETIS for partitioning, this limits the number of processors for which an appropriate parallel distribution can be found and is discussed later.

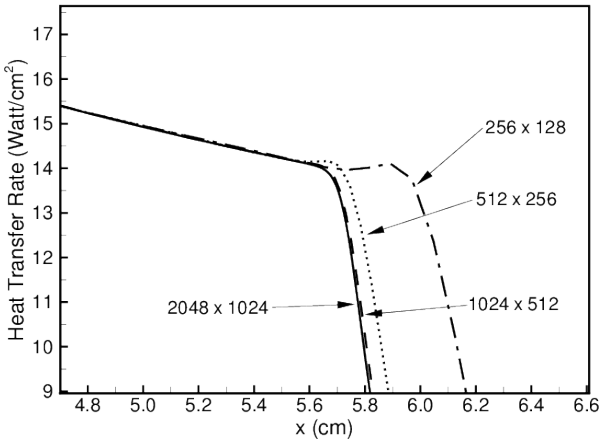
Sensitivity to the viscous wall spacing was also examined. These results indicated that an initial off-body spacing of 0.5μm was sufficient to capture the boundary layer and that further refinement was unnecessary. Both the adapted and unadapted grids use grid stretching away from the surface. For the adapted grids, the initial off-body spacing is much coarser and it is halved with each successive level of refinement. It is only when adapted to the finest level (equivalent to the uniform grid of 1024×512) that the viscous spacing reaches 0.5μm.

5.3.2 Adaptation Strategy

Unlike the double wedge examined previously, this flow requires significant time to develop and allowing the solver to adapt to the finest grid level initially is not ideal. The shock structure grows slowly off of the double cone and the separated region expands



(a) Heat transfer over entire doube-cone.



(b) Subset near separation shock.

Figure 5.18: Illustration of grid convergence study on uniformly refined meshes. [50]

until it reaches its steady-state size. For this reason, the adaptation was restricted to a maximum level of refinement. While the maximum level was fixed, refinement at that maximum level could migrate through the domain. This enabled fine cells to propagate to where they were needed and a pseudo-steady solution at a given resolution could be obtained.

After a specified number of flow times, the maximum level of refinement was increased. The fine cells could again propagate as the flow developed on the finer grid. This gradual stepping of refinement increased until 60 flow times had elapsed. After 60 flow times, the finest cells were equivalent to those found in the unadapted grid. Adaptation with propogation continued every flow time until the solution had advanced 150 flow times.

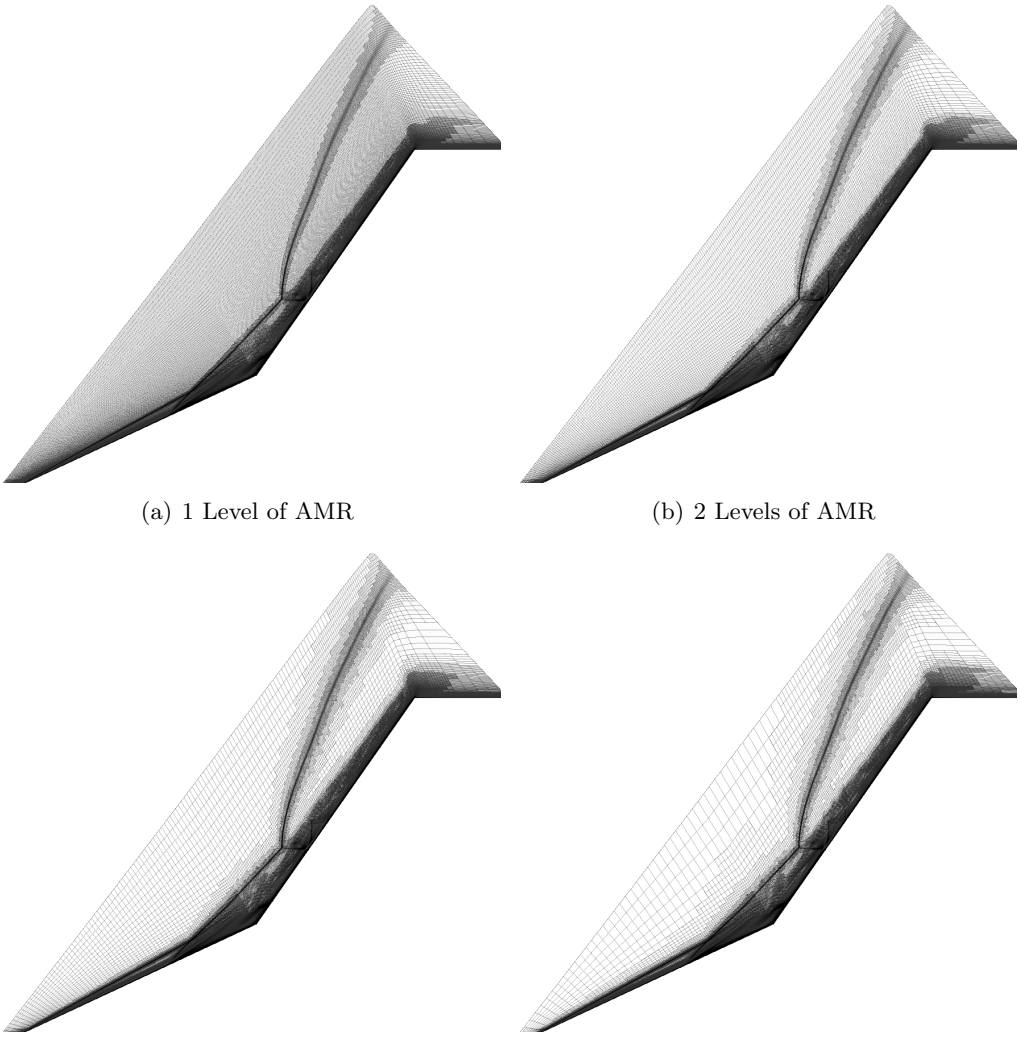
An exhaustive search for the optimum refinement criterion and frequency was not conducted. Previous work has shown success by refining based on the magnitude of the density gradient, $|\nabla\rho|$. Greenshields *et al.* established that subdividing cells with $|\nabla\rho| > 0.2$ [kg/m^4] identified cells in the shock regions and near the surface.[33] The shock causes a strong gradient in density due to compressibility. Near the surface, the temperature difference between the wall and the freestream flow also cause a manful ρ gradient. This work adopts an identical refinement sensor (Eq. 3.2).

Figure 5.19 shows the final adapted grids after 150 flow times have elapsed. The meshes are superimposed over a plot of the gradient density magnitude in order to illustrate the underlying flow structure. Refinement closely matches the bow shock shape, fills the separated flow region, and importantly leaves much of the domain coarse. Regions of freestream flow are refined only to the extent necessary to provide an appropriate buffer for the finest cells.

This approach refines the developing solution without knowledge of the final feature locations. By tracking features as the evolve, the researcher needs little knowledge about the final flowfield and can apply an identical refinement criterion to similar problems. This work uses an absolute value for the refinement criterion, so there may still be adjustment necessary for dramatically different flows, however.

5.3.3 Results

All solutions were run on 36 cores except for the case with four levels of refinement. It had only 2,048 cells which presented difficulty for ParMETIS when finding an acceptable partition on that many cores. For the coarsest initial grid, there were 64 initial lines - effectively only 64 elements for ParMETIS to partition. That solution used only 24 cores. This is one limitation when using adaptive grids as implemented here. With very coarse initial grids that are partitioned across the level zero cells, the maximum number



(a) 1 Level of AMR

(b) 2 Levels of AMR

(c) 3 Levels of AMR

(d) 4 Levels of AMR

Figure 5.19: Image of adapted grids superimposed over magnitude of density gradient.

of ranks is limited. The implicit method makes the problem slightly more difficult since DPLR lines must remain together.

Figure 5.20 shows measured pressure and heat flux from the CUBRC experiment (circles) as well as the numerical results for the unadapted grids and the adapted grids. All solutions have had 150 flow times to develop. The agreement between the computational results is absolute and the use of adaptation does not impart error in the results.

Comparison between the experimental data and the output from the solver compare well to with previous published results.[38]

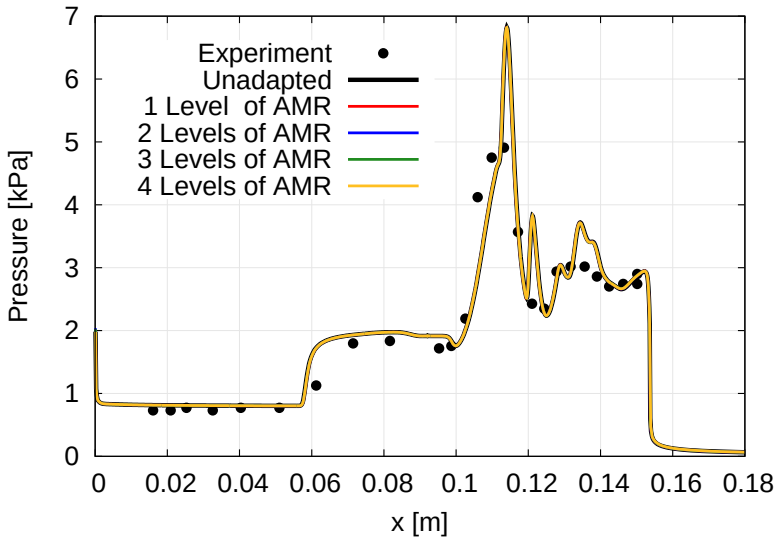
5.3.4 Computational Efficiency

The final adapted meshes using one, two, three, and four levels of refinement include 441K, 405K, 390K, and 382K cells, respectively. These point to moderate savings when compared to the uniform grid with 524K cells. Computational savings are significant because the solutions develop for 60 flow times on very coarse representations of the final mesh. As it shown in Fig 5.21, the required CPU time for the unadapted grid is more than double that required for the cases with two, three, or four levels of refinement. Figure 5.21(b) uses a logarithmic y-axis in order to highlight the portion of the simulations that use adaptation.

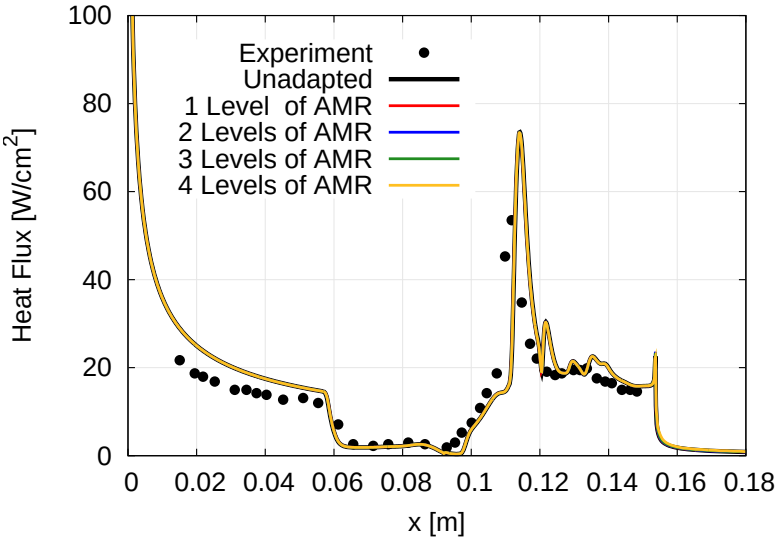
For this problem, the relative cost of adaptation is trivial compared to the costs associated with time stepping the problem. Figures 5.22 and 5.23 show the normalized costs of time advancement and adaptation/redistribution for problem using four, three, and two levels of refinement. These timings are shown as a function of flow time to illustrate the effects of refinement as the flow develops. Each bar on the plot is normalized to the average of the previous 10 time steps with adaptation and redistribution occurring once every flow time.

The costs for adaptation and repartitioning are very small during the propagation cycles, when the finest grid spacing in the domain is not changing. When the level of refinement increases there are relatively large costs for all portions of the adaptive procedures. Local refinement of the grid creates imbalance and there is dramatic load balancing that much occur. Even so, it remains on the order of 10 times the cost of a single time step. At a CFL value of 4,000, these simulations require roughly 100 time steps per flow time with most AMR cycles requiring 3 iterations worth of computation time. The cost of AMR, is on the order of 3% of the total runtime and provides at least a 50% savings overall.

AMR is well suited for simulation of this psuedo-steady problem. There are well-identified flow structures and refinement based on flow features ($|\nabla\rho|$) correctly capture regions of importance. The sparse use of grid points for initial computation while the flow develops saves considerable computational time.

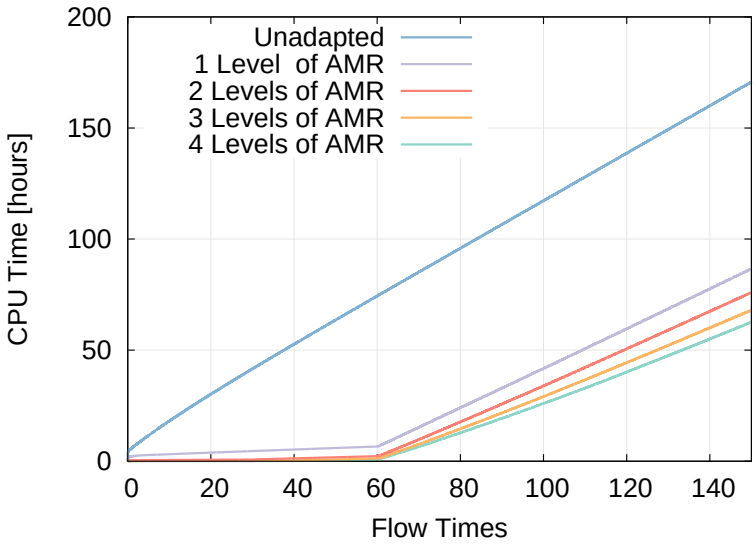


(a) Surface pressure

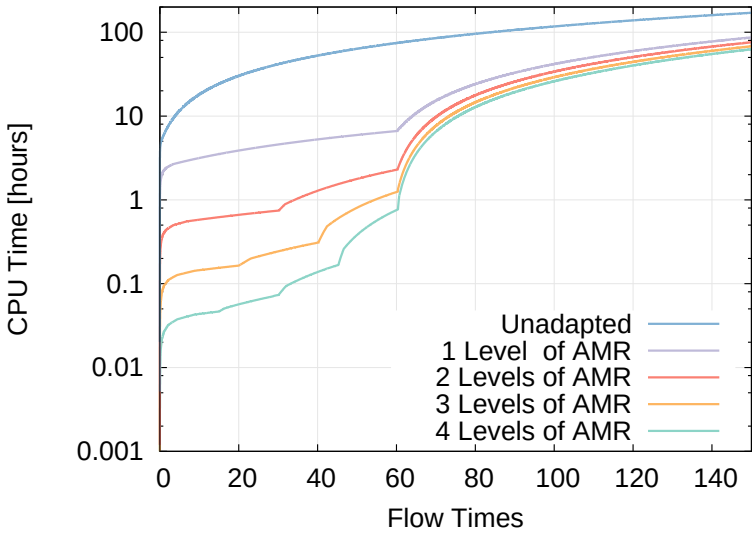


(b) Wall heat flux

Figure 5.20: Comparison of wall pressure and heat flux to experimental results.

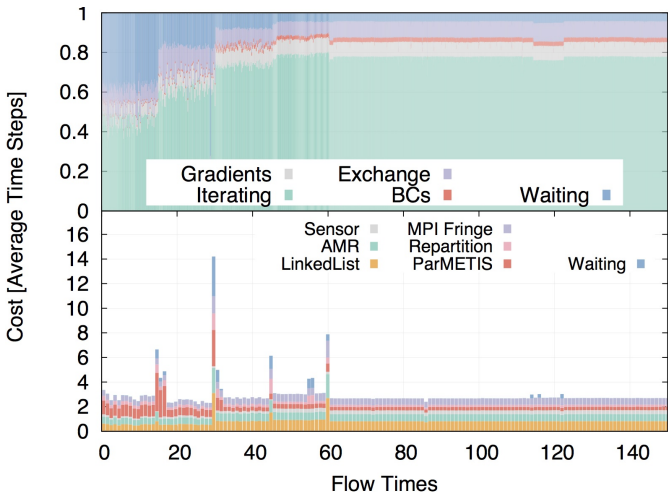


(a) Linear y-axis

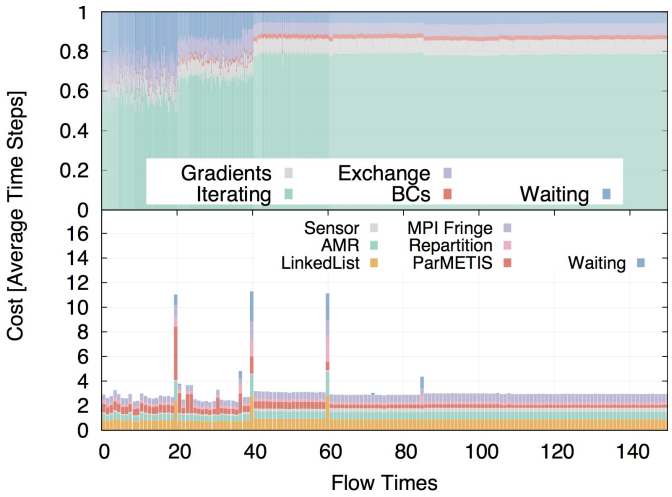


(b) Logarithmic y-axis

Figure 5.21: Comparison of absolute CPU cost for uniform and adapted grids. Both linear and logarithmic y-axis shown.

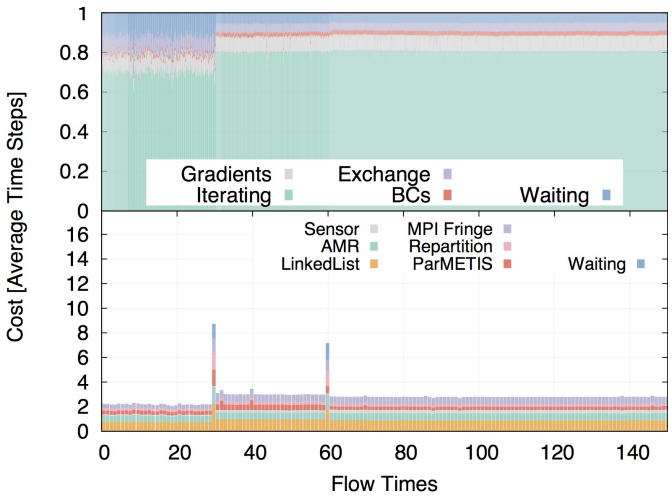


(a) Four levels of refinement

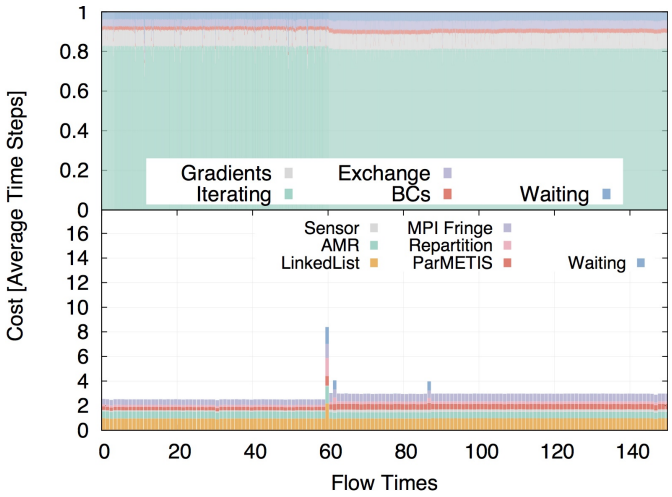


(b) Three levels of refinement

Figure 5.22: Runtime in major routines for adapted grids relative to the cost of an average time step as a function of flow time.



(a) Two levels of refinement



(b) One level of refinement

Figure 5.23: Runtime in major routines for adapted grids relative to the cost of an average time step as a function of flow time.

5.3.5 Parallel Performance

For efficient parallel computation, the graph of distributed work should be equipartitioned across all ranks involved with the simulation. As the grid cells are refined, previously balanced partitions require rebalancing. Figure 5.24 shows the computational mesh colored by the rank ID that owns the cells. Due to the strategy that was outlined earlier based on partitioning the level-zero cells, even on the final mesh the partitions are distributed based on the coarse cells.

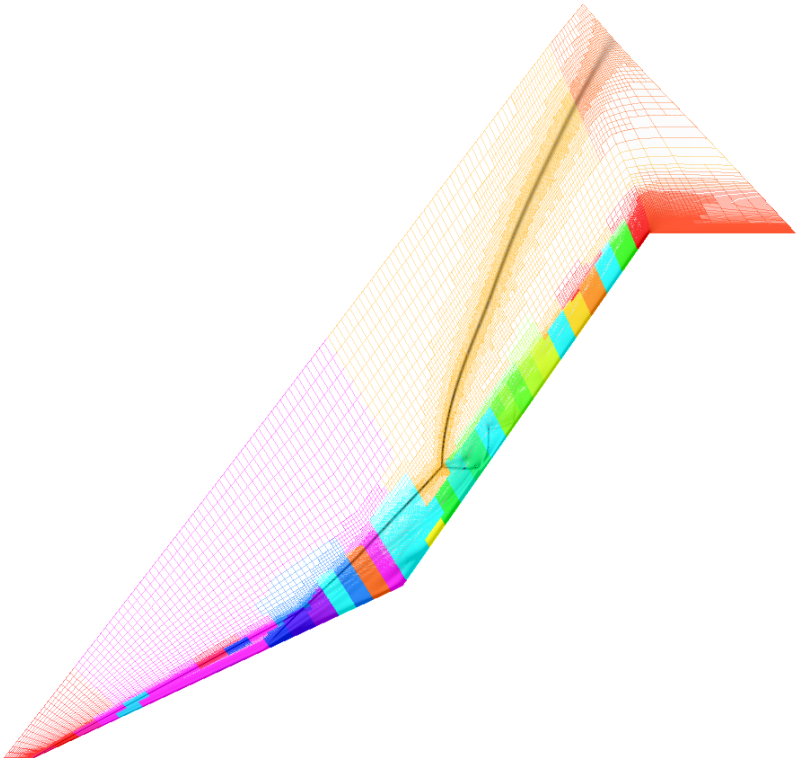
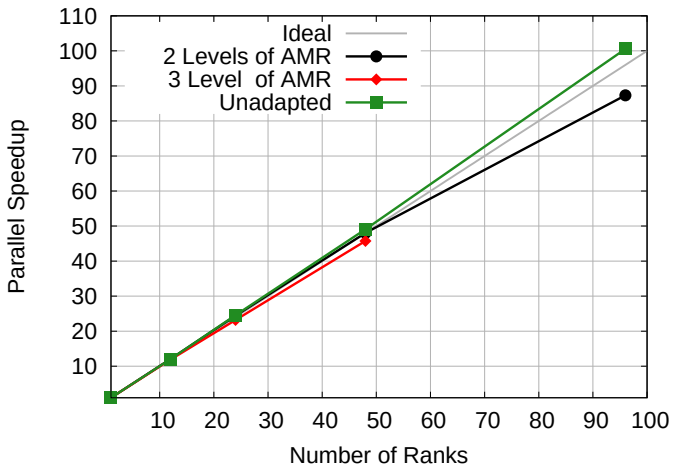


Figure 5.24: Computational mesh colored by partition rank; contour of $|\nabla\rho|$ in black and white.

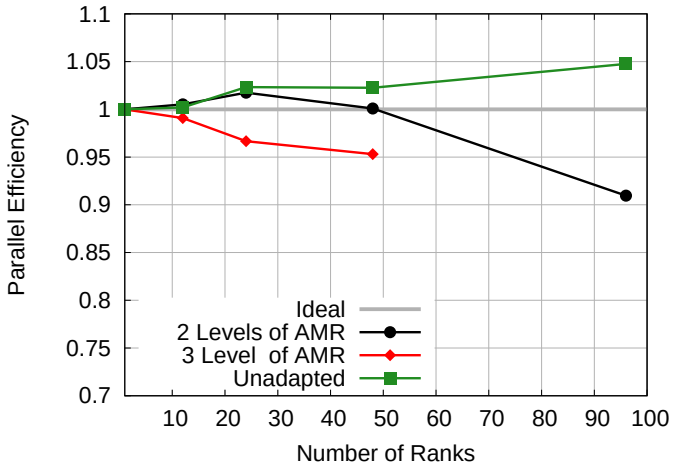
Figure 5.25 shows a scalability study for the double cone problems for two levels of AMR as well as for the unadapted grid case. For these timings, the flow was simulated for a total of 80 flow times. The unadapted grid shows strong scaling for the range of cores examined. Adapted grids have a reduced parallel efficiency and are not as robust in their scaling.

These data present better scalability than the results seen in Fig. 4.11 for an equivalent number of refinement levels. Differences between the two figures are expected. The double cone problem uses implicit time stepping which has different requirements for interprocessor data exchange and local calculation. This problem also calls the AMR and repartitioning subroutines less frequently and the refinement criteria were very different.

Similar to what was seen previously, additional levels of AMR reduce the speedup seen in the results. This is important to note, but using additional levels of AMR did reduce the absolute time required to iterate as seen in Fig. 5.21. The ideal number of levels of AMR is case-specific and to achieve a certain size in the finest cells, there are diminishing returns in the reduction of grid cells when increasing the depth of refinement.



(a) Speedup



(b) Parallel Efficiency

Figure 5.25: Parallel performance for double cone simulations. Adapted and unadapted grids shown.

5.4 Type IV Shock-Shock Interaction

In Edney's original work, shock-shock interactions were observed between an oblique shock interacting with the detached bow shock ahead of a test geometry.[27] He categorized these interactions into six types based on the relative location of an impinging shock and the bow shock. Of the six interactions, the type IV interaction is the most severe and results in aerodynamic heating that is significantly larger than what is observed for an undisturbed body (no impinging shock). On a real vehicle, shock-shock interactions are common and the intensity of the resulting environment must be accounted for when performing design.

Type III and IV interactions are caused by the oblique shock meeting the bow shock where it is nearly normal to the flow and velocities behind the bow shock are subsonic. Depending on the freestream conditions and the strength of the shocks there is variation of the specifics, but he major feature in the type IV interaction is a supersonic jet impinging on the surface of the body. This creates localized, small features in the form of a corridor of compression shocks and expansion waves.

This work focuses on analysis of the type IV interaction. There have been several ground test experiments that provide a wealth of data for code validation. Two of the most commonly cited are the tests performed at ONERA in the R5Ch blowdown wind tunnel and those performed at CUBRC's LENS facility.[39;57] The ONERA test resulted in laminar, perfect gas experimental data with measurements taken on the surface of the test cylinder as well as inside the flowfield. It has been investigated and compared to by a number of other researchers and the numerics in the flow solver developed in this work should be sufficient for accurate simulation. Finite-volume computations performed by D'Ambrosio, DSMC simulations by Moss *et al.*, and finite-element simulations by Kirk have showed good agreement with experiment.[21;44;46]

Figure 5.26 shows a schematic for the bodies in the test section for the ONERA wind tunnel test. This test used a triangular prism as a shock generator upstream of an instrumented circular cylinder. The cylinder had a radius of 8 [mm] and both the prism and the cylinder had a width of 100 [mm]. Due to the high width-to-diameter ratio of the cylinder, 6.25, the experiment is considered to be two-dimensional. Relative displacement between the two bodies was variable, but the majority of the data was

collected with a distance of 110 [mm] between the forward-most point of the shock generator and the cylinder's center. The shock generator at the cross section of an isosceles triangle with a 10° leading edge. It was inclined 10° to the flow creating a 20° ramp to turn the oncoming flow.

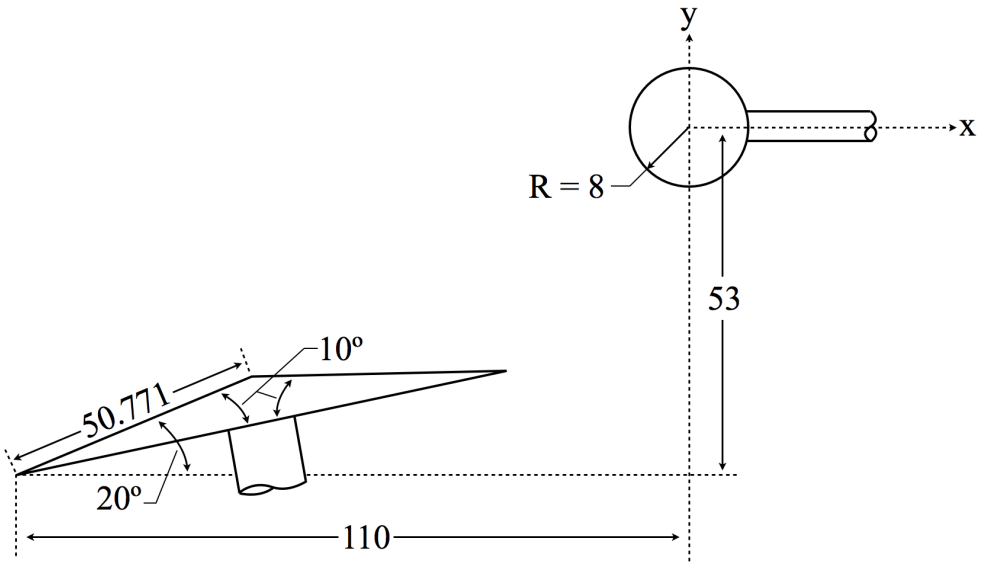


Figure 5.26: Test configuration for ONERA shock-shock interaction test; all measurements in millimeters.

The freestream conditions for the flow are tabulated in Table 5.7. Due to the low Reynolds number, the flow is laminar and no turbulence model is used for the following results. Researchers have documented unsteady type IV interactions for some of the CUBRC runs,[49] but for the conditions and set-up in the ONERA tunnel the interaction should remain steady. Furthermore, these experiments are perfect gas and use air as the test gas. The wall temperature on the shock generator and the cylinder were held at a constant temperature of 300 [K].

5.4.1 Grid Generation

For a Mach 10 flow with a 20° turning angle, shock relations predict a 25.82° shock angle. However, due to the low Reynolds number (significant viscous effects) and the

Mach	10.00	[-]
p_∞	5.9	[Pa]
T_∞	52.5	[K]
Re_D	2,672	[-]

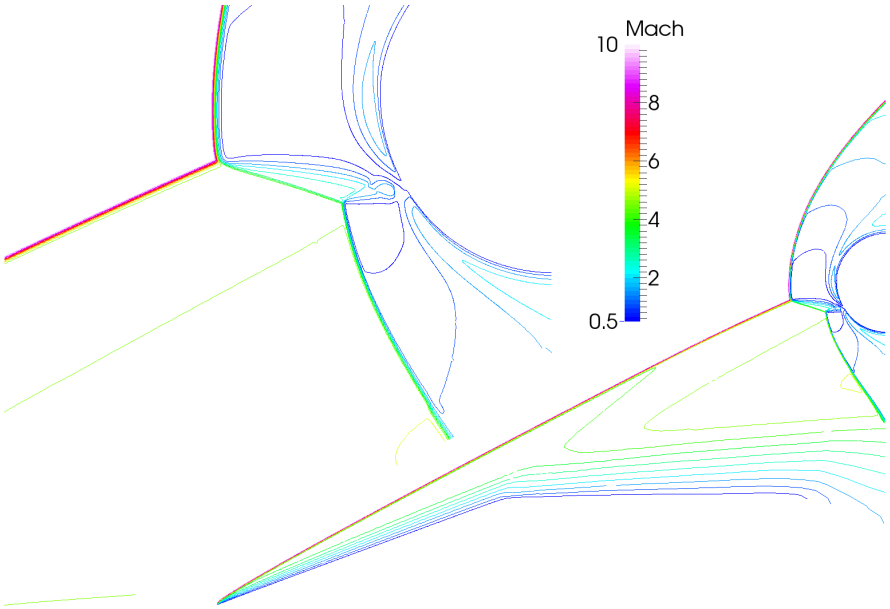
Table 5.7: Freestream conditions for ONERA experiment.

large region for expansion downstream of the initial ramp, it is incorrect to simply model the cylinder, incoming freestream, incident shock, and post-shock conditions calculated with oblique shock relations. The flow is very non-uniform and viscous effects on the shock generator impart uncertainty on the initial location of the shock and result in initial curvature of the upstream oblique shock wave. Figure 5.27 illustrates this by showing Mach number and temperature contours over the computational domain for a grid-refined solution. For this reason, the complete test apparatus is modeled from the shock generator to the cylinder's center.

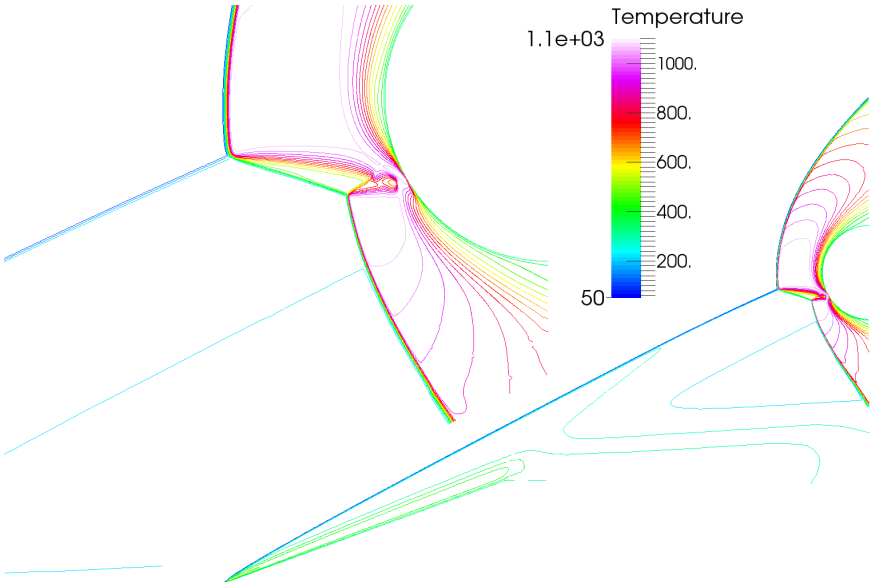
Sufficient resolution of the impinging shock, the bow shock, and the post-shock states are required in order to achieve accurate prediction of surface quantities. Figure 5.31 shows the initial, coarse mesh used for these simulations generated in Pointwise.[56] The applied boundary conditions are shown and portions of the domain are shaded for easy reference. The shock generator and cylinder are modeled as viscous walls with surface fluxes described explicitly. Downstream boundaries are set to extrapolation outflow as they are supersonic. Upstream boundaries are set to constant conditions shown in Tab. 5.7.

Region one in Fig. 5.31 includes grid lines closely aligned to the ideal shock angle. This is intended to allow the incident shock to travel the length of the computational domain with a minimum of grid alignment error. Since the final location of the oblique shock was unknown, this region of the flow was made relatively large. Region two is a body-fitted grid that surrounds the cylinder. It uses a hyperbolic tangent grid clustering to the surface with the finest grid (4 levels of AMR) having an initial off-body cell height of 2.0E-6 [m]. Region three is included to complete the domain and is not aligned or clustered in any particular manner. Its effect is secondary to the success of the computation and is very skewed to facilitate simple grid generation.

The initial grid was uniformly refined four times in order to create five different grid systems. These systems are referred to as follows: Baseline, Coarse, Medium, Fine, and



(a) Mach number



(b) Temperature (K)

Figure 5.27: Mach number and temperature contours for converged simulation of the ONERA experiment.

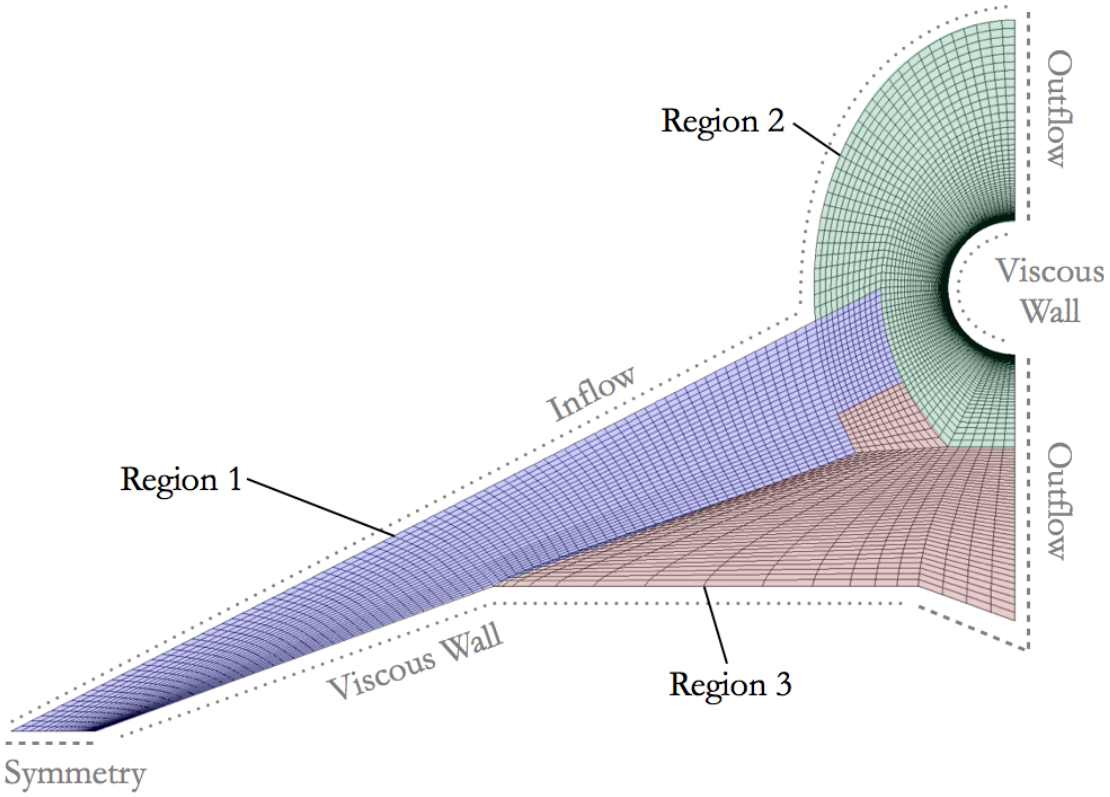


Figure 5.28: Base mesh with boundary conditions for simulations of ONERA test.

Super-Fine, each representing a doubling of the number of grid cells in each direction. As will be shown below, the super-fine system is considered to be grid refined with very little change in the results between fine and super-fine (surface and flowfield values). Table 5.8 shows metrics for grid size and cell spacings in each of these grids. The number of cells in the radial direction away from the cylinder are measured from the wall to the closest part of region one in Fig. 5.31. For reference, the grid used by D'Ambrosio had 586×150 cells in the region near the cylinder (radial×circumferencial).

5.4.2 Adaptation Strategy

Many of the prevailing flow features in this solution are similar to those examined in the double cone previously. Both are hypersonic flows with strong shocks and viscous

Mesh Description	Cells in Cylinder Circumference	Cells in Radial Direction	Total Number of Cells
Baseline	64	64	7,453
Coarse	128	128	29,812
Medium	256	256	119,248
Fine	512	512	476,992
Super-Fine	1024	1024	1,907,968

Table 5.8: Grid metrics of computational domains for ONERA simulations.

boundary layers. Regions of importance are highlighted by density gradients and a criteria based on its magnitude is sufficient. Specifically, this work uses a tolerance on the magnitude of the density gradient of 0.5 [kg/m^4] when using local refinement. In addition to this criteria, the grid was also locally refined in a region within one millimeter of the cylinder. The most refined cases uses four levels of isotropic cell refinement with a grid resolution that is comparable to the super-fine uniform grid.

Similar to the previous work with the double cone, results computed on uniformly refined grids are compared to those obtained by using adaptive refinement. The solutions require on the order of thirty flow times to converge to a steady value on the finest grids. One flow time is normalized by the length between the ramp and the center of the cylinder, 110 [mm]. AMR was performed every flow time (approximately) or 7.5E-5 [s]. When performing adaptive simulations, the maximal level of refinement in the grid was fixed for four flow times and then incremented. After another four flow times, the maximum level was increased again. This continued until the simulations were at the maximum level for the grid at which point the solution was terminated once the elapsed time totaled thirty flow times.

In addition to refinement based on the gradient of density, a simple grid sequencing adaptation was also used. It followed the same refinement schedule described above, but it uniformly refined the entire domain during adaptation. This is sometimes referred to as grid sequencing. This approach requires much less time than if a fine grid was used the the entire simulation. It will be shown to be more expensive than the results with the $|\nabla\rho|$ refinement sensor, but provides a compromise between the methods for problems where choice of refinement sensor is ambiguous. Grid sequencing is included in several flow solvers currently in widespread use.[22;55]

Figure 5.34 shows the final mesh for a solution that used four levels of AMR. It is clear that cells near both the oblique and bow shocks targeted for refinement. Furthermore, the boundary layer on the shock generator and cylinder are refined using the density gradient criteria as is the supersonic jet characteristic of the type IV interaction. While not evident here, faces on the cylinder boundary were projected to an analytic definition, grid lines away from the wall enforced normal spacing to the wall, and they were redistributed during adaptation to ensure wall spacings consist with a cell height of 2.0E-6 [m] on the super-fine grid.

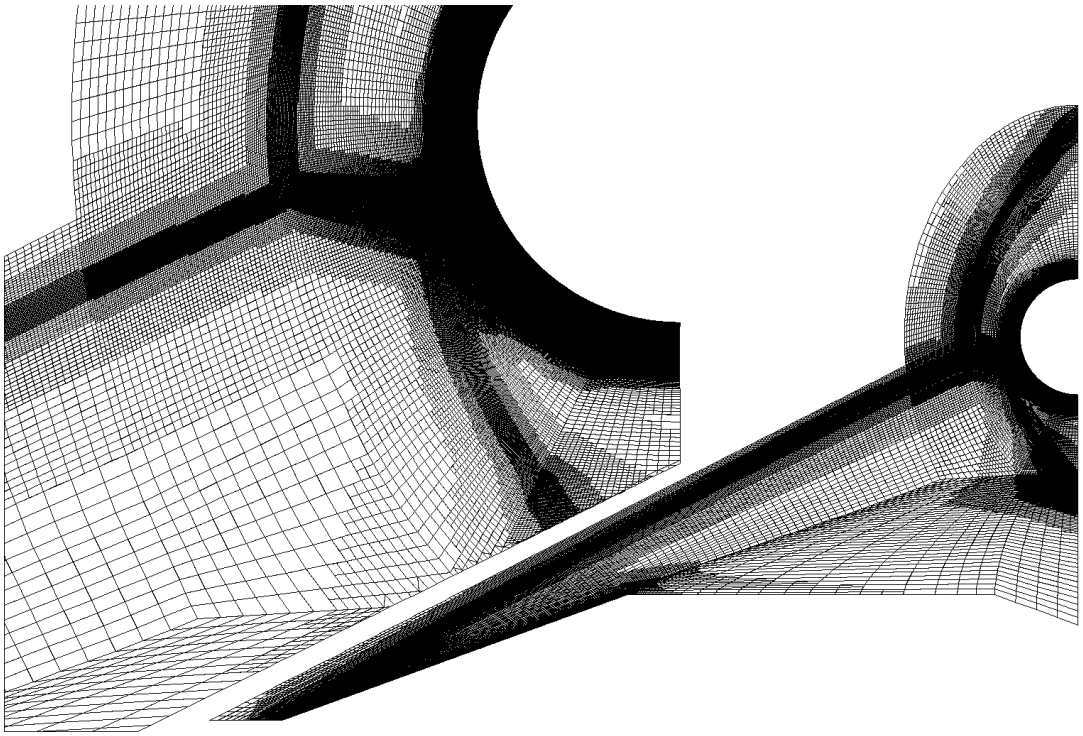


Figure 5.29: Final mesh for adapted grid using 4 levels of AMR (867,961 cells).

5.4.3 Results

Solutions were steady and generally took several flow times to converge. Figure 5.30 shows the force in the x-direction on the cylinder as a function of simulation time for computations using uniformly refined grids. The values are relative to the final force measurement on the super-fine grid. As the grid is refined, it takes longer to reach a converged value and the super-fine requires the longest to converge. For consistency, all solution were run for thirty flow times.

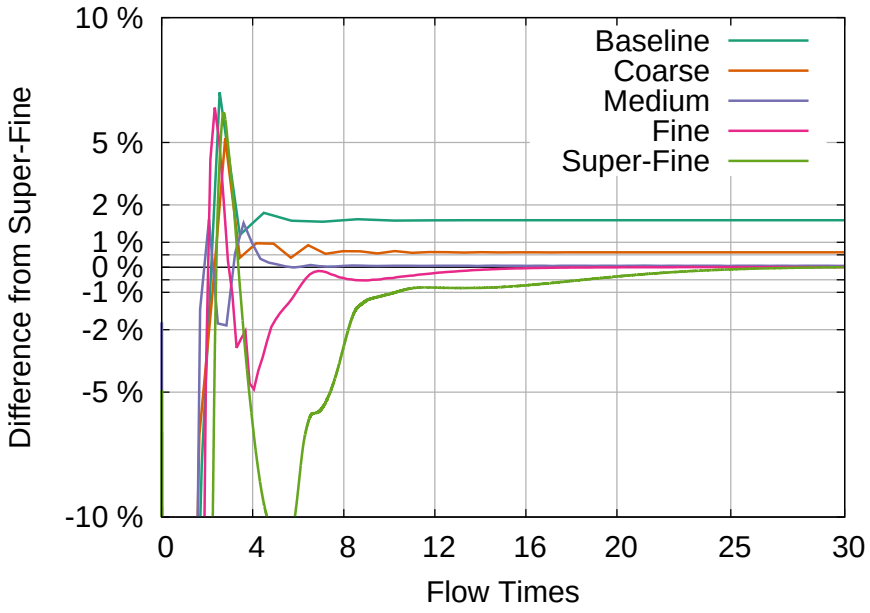


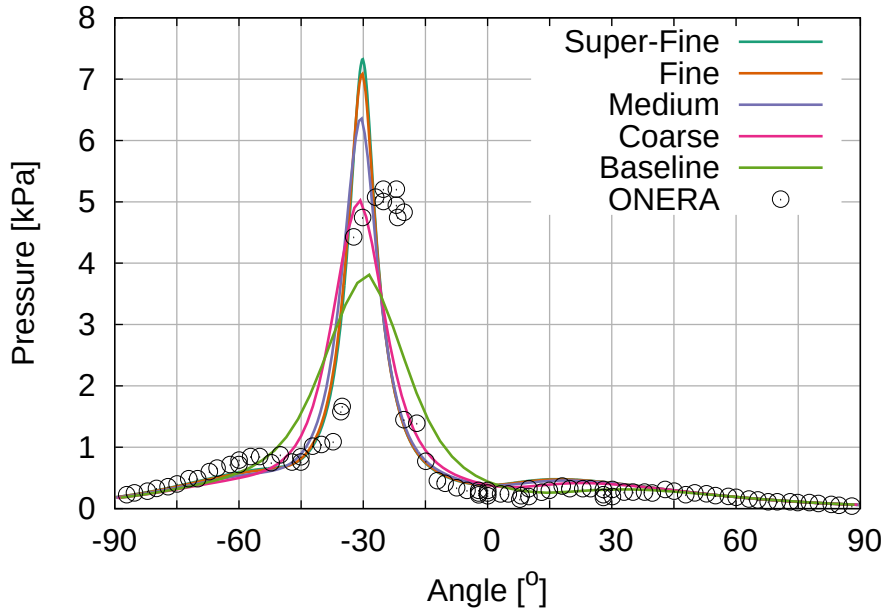
Figure 5.30: Comparison of x-direction force in time for uniformly refined grids.

A grid sensitivity study was performed using the uniformly refined grids. The results of this study are shown in Fig. 5.31 and compared to the experimental data collected in the ONERA experiment for both pressure and heat flux. In both pressure and temperature, the convergence is very similar. In general, the location of the interaction on the cylinder does not change, but the magnitude for peak pressure and heat flux increases as the grid is refined. There is very little change in the character of the curves between fine and super-fine, but the peak values are slightly higher for the super-fine grid. Due to the small magnitude in the change between the two finest grids relative

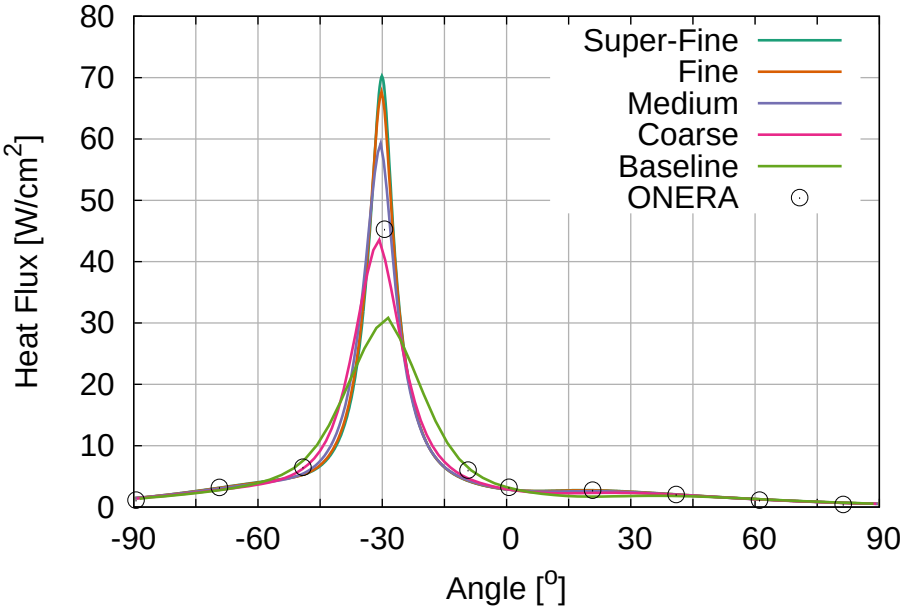
to the agreement to the test data, the super-fine grid is considered well resolved for the following comparisons.

Impingement location as measured by the surface pressure appears to be further down (negative angle) than what was observed in the experiment. The results presented here are in-family with the findings from other researchers. The difference in the peak pressure measurements was examined by D'Ambrosio who found that spatially averaging the computational results over a distance comparable to the experimental gauge size saw improvement. Unresolved is the difference in the peak locations, however this disagreement is consistent with results from other researchers. This problem is very sensitive to the experimental inflow conditions and relative location of the shock generator and the test body and it's possible that the experimental values were slightly different than what was reported.

The experiement also took measurements using a Dual-line Coherent Anti Stokes Scattering (DL-CARS). This enabled them to perform surveys in the flowfield ahead of the cylinder. Three surveys were made; one below the type IV interaction, one above it, and one through it. The locations for the surveys are shown in Fig. 5.32(a). Comparisons between the three finest uniform grids and the experimental data are shown in Figs. 5.32(b)-5.32(d). Similar to the effect of refinement on the surface quantities, these flowfield measurements indicate that the super-fine grid is sufficiently resolved. It is almost impossible to see differences between the fine and super-fine results.

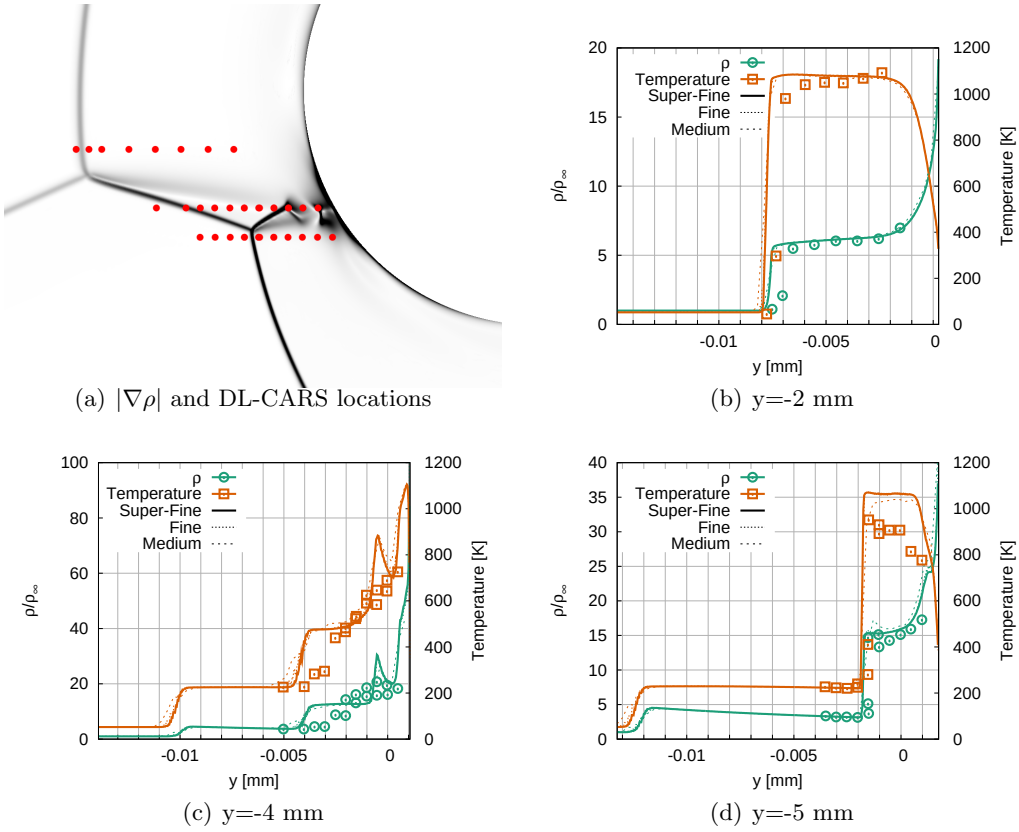


(a) Cylinder Surface Pressure



(b) Cylinder Surface Heat Flux

Figure 5.31: Results from uniform grid refinement study.



(a) $|\nabla\rho|$ and DL-CARS locations

(b) y=-2 mm

(c) y=-4 mm

(d) y=-5 mm

Figure 5.32: DL-CARS comparisons between experiment (symbols) and solutions obtained on uniformly refined grids (lines).

The discussion now shifts to results obtained using adapted grids. Only three computational data sets are shown in the following plots. The first is the super-fine grid looked at previously as a reference. Co-plotted with it is a solution obtained by using grid sequencing ("4-Level Seq") and one obtained by using the $|\nabla\rho|$-based adaptation. Both of the adapted grids have an identical resolution as the super-fine grids after 16 flow times. The super-fine grids has nearly 2 million cells and the sequenced grid has the same on its finest mesh. However, the final grid with four-levels of mesh refinement based on feature adaptation has half that amount: 867,961 cells.

Figure 5.33 shows the convergence of the integrated force on the cylinder in the x-direction. Both of the adapted simulations show sharp peaks and dramatic oscillations every four flow times. This corresponds to the increase in global level of refinement and the inclusion of finer cells into the mesh. After 16 flow times, all grids are uses cells identical in size to those in the super-fine grid and there are no more jumps in resolution. In general, even the adapted grids need on the order of thirty flow times to remove large oscillations. All solutions agree very well once a steady state is reached.

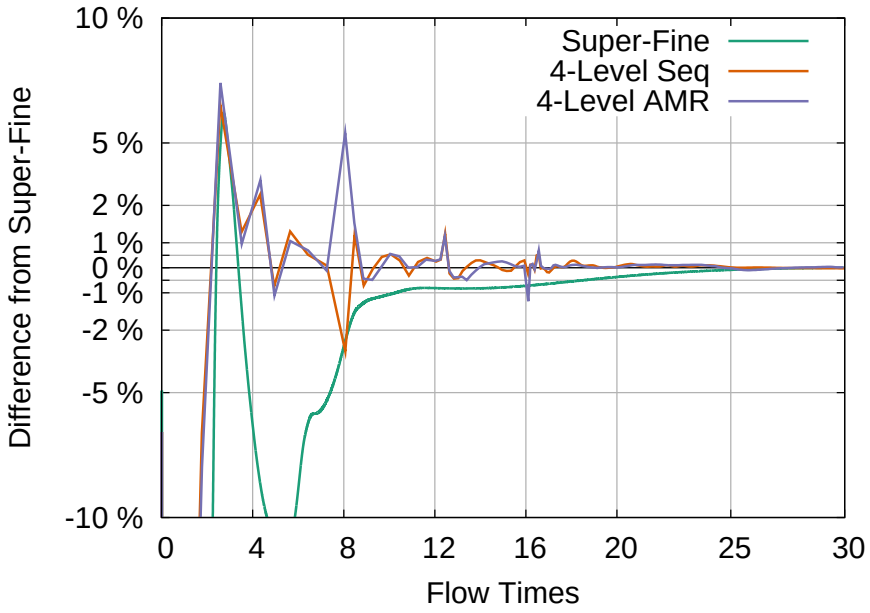
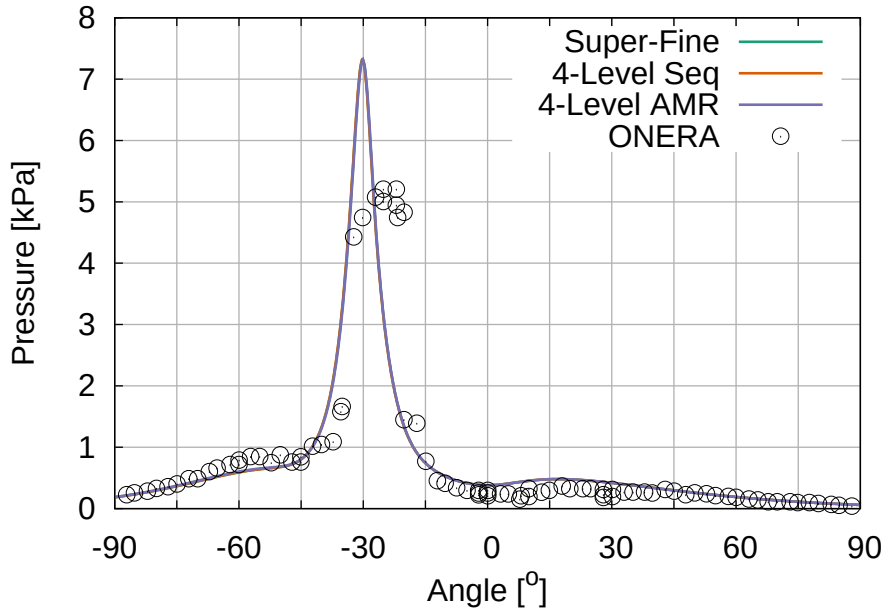


Figure 5.33: Comparison of x-direction force in time for adapted grids.

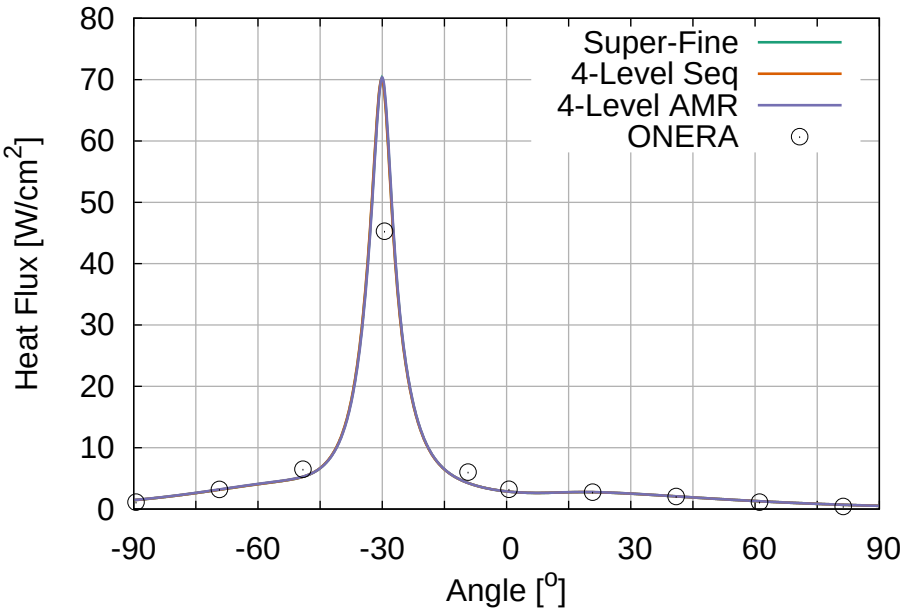
Comparisons of the pressure and heat flux on the cylinder solution are shown in

Fig. 5.34. Agreement between the super-fine uniform grid and the adapted grids is very good and it is nearly impossible to discern that there are three computational curves on the figure. It should be no surprise that the sequenced results agree with the unadapted results since they are obtained on an identical grid. Similar to what has been shown previously in the document, with an appropriate refinement tolerance grids with hanging nodes achieve identical results to those on uniform grids. This results illustrates that the refinement tolerance selected here is sufficient to capture all necessary physics in this problem.

A similar comparisons exists in the flowfield comparisons, Fig. 5.35. In this figure, quantities from the mesh adapted with $|\nabla\rho|$ feature detection are plotted against the super-fine results. Results obtained with grid sequencing are excluded because they provide no additional insight. The super-fine results are in black, behind the adapted results. Only in very high gradient regions is the black line visible which illustrates that they agree to a profound extent.

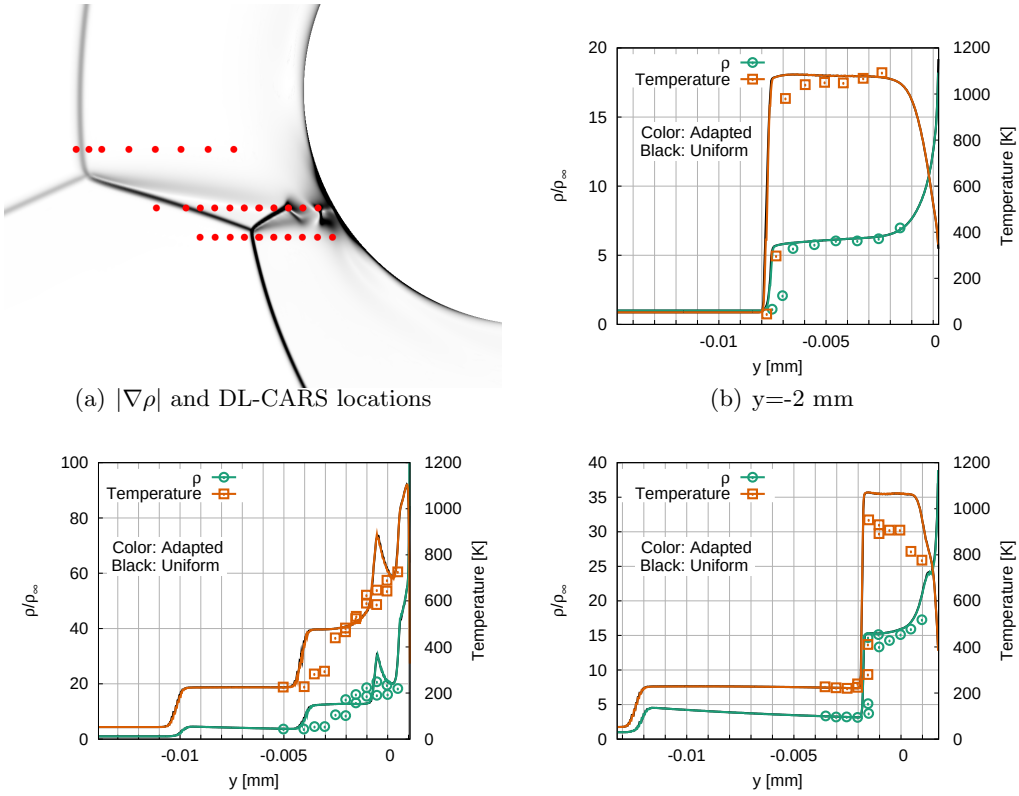


(a) Cylinder Surface Pressure



(b) Cylinder Surface Heat Flux

Figure 5.34: Results from uniform grid refinement study.



(a) $|\nabla\rho|$ and DL-CARS locations

(b) y=-2 mm

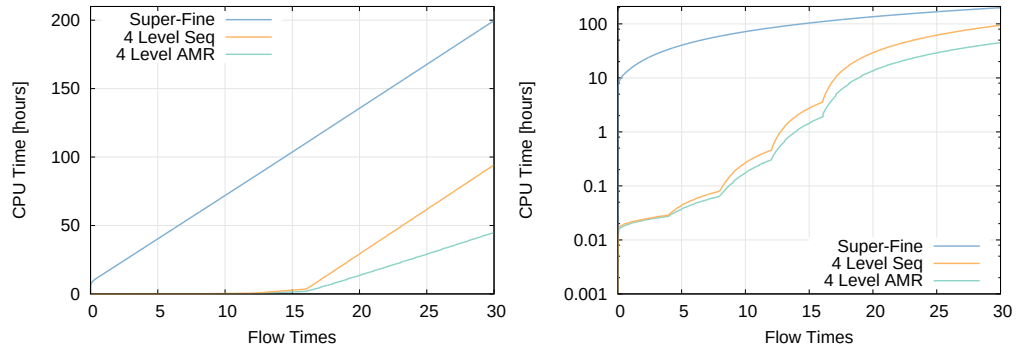
(c) y=-4 mm

(d) y=-5 mm

Figure 5.35: DL-CARS comparisons between experiment (symbols) and solutions using uniform and adapted grids (lines).

5.4.4 Computational Efficiency

Similar to the double cone examined previously, the reduced cell count in the grids generated using AMR should results in significant computational savings when compared to the uniformly refined counterpart. Using grid sequencing, more than half of the simulation was obtained on a coarse representation of the domain, so it should also show a reduction in cost when compared to a solution that iterated on a 2 million cell mesh for the entire thirty flow times. Figure 5.36 shows a comparison of the total CPU time required to perform the simulations shown previously.



(a) Linear y-axis (b) Logarithmic y-axis

Figure 5.36: Comparison of absolute CPU cost for uniform and adapted grids. Both linear and logarithmic y-axis shown.

For this problem, using coarse grids to initially set-up the flow appears to provide a dramatic reduction in the cost of the solution. Grid sequencing reduces the computational time required be about half when compared to using a super-fine grid for the entire computation. If feature-based adaptation is used, it reduces the cost by about half again, or to one quarter of the cost of the super-fine grid. The AMR grid has about only half the number of points when at the finest grid level, so the savings become greater as the number of flow times increases since the computational cost per flow time is linearly related to the grid size.

With a pseudo-steady problem with refinement every flow time, the adaptation routine is invoked only 29 times over the 30 flow time simulation time. This means that the cost for adaptation should be minimal compared to the cost of time stepping the

Mesh	Super-Fine	4-Level MG	4-Level AMR
Total Runtime [CPU hours]	199.65	94.17	44.74
Time Marching	77.56%	75.55%	71.89%
Calculating Gradients	9.13%	8.90%	8.46%
AMR / Feature Detection	0.00%	0.09%	0.12%
Repartitioning / Memory Management	0.00%	1.99%	5.90%
Node Redistribution	0.00%	1.15%	2.19%
MPI Exchange / Waiting	10.07%	10.08%	9.45%

Table 5.9: Percentage of run time in significant subroutines for simulations on ONERA test.

problem. Table 5.9 shows a comparison of the percentage of the total run-time spent in significant subroutines for the adapted and super-fine simulations. For reference, at a CFL of 100 there were approximately 600 time steps every flow time.

As illustrated by the table, the most expensive addition when adding adaptation to the solution procedure are the costs associated with memory management and repartitioning. Next, as a proportion of total run time, is redistribution of the nodes away from the viscous surfaces. The costs associated with the AMR operation and feature decoction is negligible. Even with these additional operations, the very large majority of the simulation is spend iterating. Even moderate reductions in the cell count can offset the minor costs associated with refinement for problems with refinement frequencies similar to this one.

Use of grid sequencing alone appears to provide considerable benefit for pseudo-steady problems. In case where additional simulation time is required on the finest mesh, use of selectively refined grids yields further benefit by greatly reducing the region of refinement in the grid. Selective refinement depends on the researcher's ability to find a criteria for refinement that is robust and captures the most important phenomena in the flow. Dynamic grid systems and flow solvers flexible enough to handle them offer large performance benefits without a reduction in accuracy.

5.5 Low-Reynolds-Number Cylinders

Aerodynamics of bluff bodies are important to aerospace engineers. Re-entry capsules currently being used for manned missions to-and-from the international space station are bluff bodies. So, too, are several of the vehicles under design by public and commercial entities. Other vehicles frequently include include bluff-body protuberances. One of the most basic bluff bodies used in analysis is a two-dimensional circular cylinder. This section will examine numerical results of a cylinder in a uniform low-Reynolds number flow. Results comparing several adaptation strategies to uniformly refined grids are shown.

This problem is very different than previous examples and includes several new challenges. The resulting flow is unsteady and requires a more demanding schedule for adaptation. Similar to the previous problem, the resulting flow field is typically unsteady and is strongly influenced by viscous forces. The solution requires over one-hundred characteristic flow times for accurate measurement of mean aerodynamic forces. Unlike the double cone solutions, this flow does not converge to a psuedo-steady state and exhibits a persistent oscillatory behavior. This application requires projection and grid redistribution to maintain adequate resolution of the viscous walls during cell subdivision and refinement.

As discussed below, even though the geometry is simple, the circular cylinder in uniform flow presents a wide range of behaviors. Even the two-dimensional geometry begins to have strong three-dimensional effects at higher Reynolds numbers. There are multiple modes of unsteadiness that need resolved for accurate resolution of the time-averaged results; as compared to experiments. Mittalt and Balachandar showed that computational domains with limited span-wise length are insufficient for good agreement with measured results at a Reynolds number of 300.[45]

Figure 5.37 is reproduced from the work of Williamson.[72] It shows the strong sensitivity of the pressure in the wake of a cylindrical body as measured from several experiments over a wide range of Reynolds number. The wide variation in base pressure is due to a strong sensitivity to the trailing vortices behind the cylinder. For Reynolds numbers less than 50 the flow is laminar and steady. From a Reynolds number of 50 to near 188.5 the flow is two-dimensional. Higher than 188.5, as found by Barkley

and Henderson, the flow is three-dimensional and require resolution of the mode-A and mode-B spanwise instabilities.[5] Above a Reynolds number of 200,000, the flow over the cylinder transitions to turbulent and there is a sharp character change in base pressure coefficient.

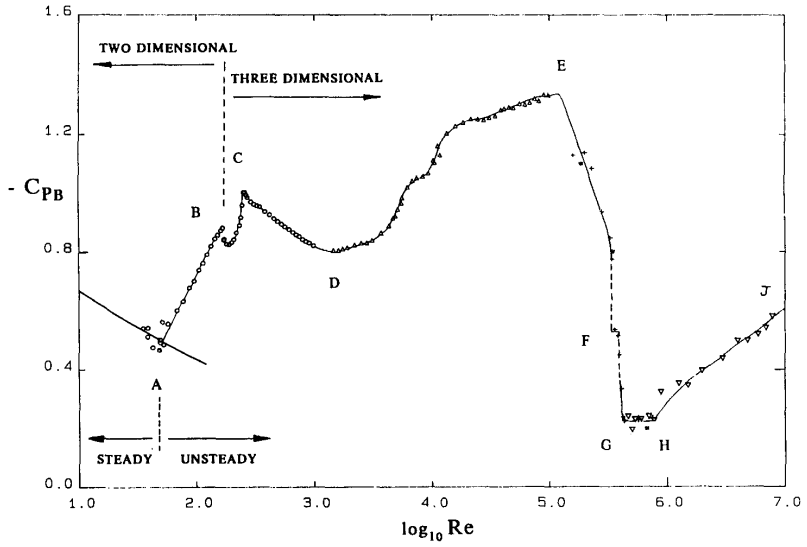


Figure 5.37: Cylinder base pressure as a function of Reynolds number.[72]

This work looks at solutions with freestream Reynolds numbers, based on cylinder diameter, of 80 and 160. These values are selected in order to remain in the laminar regime and show adaptation of vortex-dominated flow. Specific freestream conditions are shown in Table 5.10. Reynolds numbers were varied by adjusting density. The cylinder is modeled as having a diameter of 1 [m]. Based on the freestream velocity (V_∞) and cylinder diameter, a characteristic flow time is 1.43E-2 seconds. The freestream Mach number for all cases is 0.20 in order to minimize effects of compressibility. All cases are laminar and no turbulence model is used. To eliminate influence of transients and ensure that fully developed wake-vortex shedding, solutions were run for nearly 300 flow times.

Re_D	V_∞	ρ_∞	μ_∞
[-]	[m/s]	[kg/m^3]	[kg/m· s]
80	70	2.1103E-5	1.8466E-5
160	70	4.2206E-5	1.8466E-5

Table 5.10: Freestream conditions for cylindrical test cases.

Mesh Description	Levels of Adaptation	Azimuthal Spacing	Wake Cell Size	Off-Body Distance
Coarse	0	0.1000D	0.40D × 0.40D	0.01D
Medium	1	0.0500D	0.20D × 0.20D	0.005D
Fine	2	0.0250D	0.10D × 0.10D	0.0025D

Table 5.11: Grid spacings for two-dimensional cylinder grids, with and without AMR.

5.5.1 Grid Generation

A two-dimensional grid was made using the structured grid generator GridPro.[34] It was smoothed for an excessive number of iterations in order to ensure steady-state in the elliptic solver. The grid was coarse with the expectation of refining it by a factor of four in the non-trivial directions. This results in a fine grid with 16-times as many cells as the coarse, initial grid. The off-body spacing was controlled in order to ensure that all grids had y^+ value of less than one in attached flow. Since the fine grid has a wall spacing that is four times smaller than the coarse grid's, its resulting y^+ value is significantly less than 1.0. Figure 5.38 shows an image of the coarse grid. The image illustrates the extent of the domain, the near-body grid clustering, and the near-isotropic cells downstream of the cylinder designed to capture the wake (flow from left-to-right).

To avoid the influence of the far-field boundaries, the domain extends at least 30 diameters away from the cylinder. Mahesh *et al.* used boundaries that were 20D away from the cylinder for their low-Reynolds number cases ($\leq$300) and 25D for a high-Reynolds number solution (3,900). Grids used in this work had increased cell spacings near the boundaries to dissipate any error introduced by their location. Inspection of the results did not show visible reflection or influence of the boundaries. The final spacings for grids used in this analysis are tabulated in Table 5.11.

Mahesh *et al.* cite an azimuthal spacing of 0.01-0.012D for their DNS and LES grids. This translates to approximately 315 cells around the cylinder for their finest

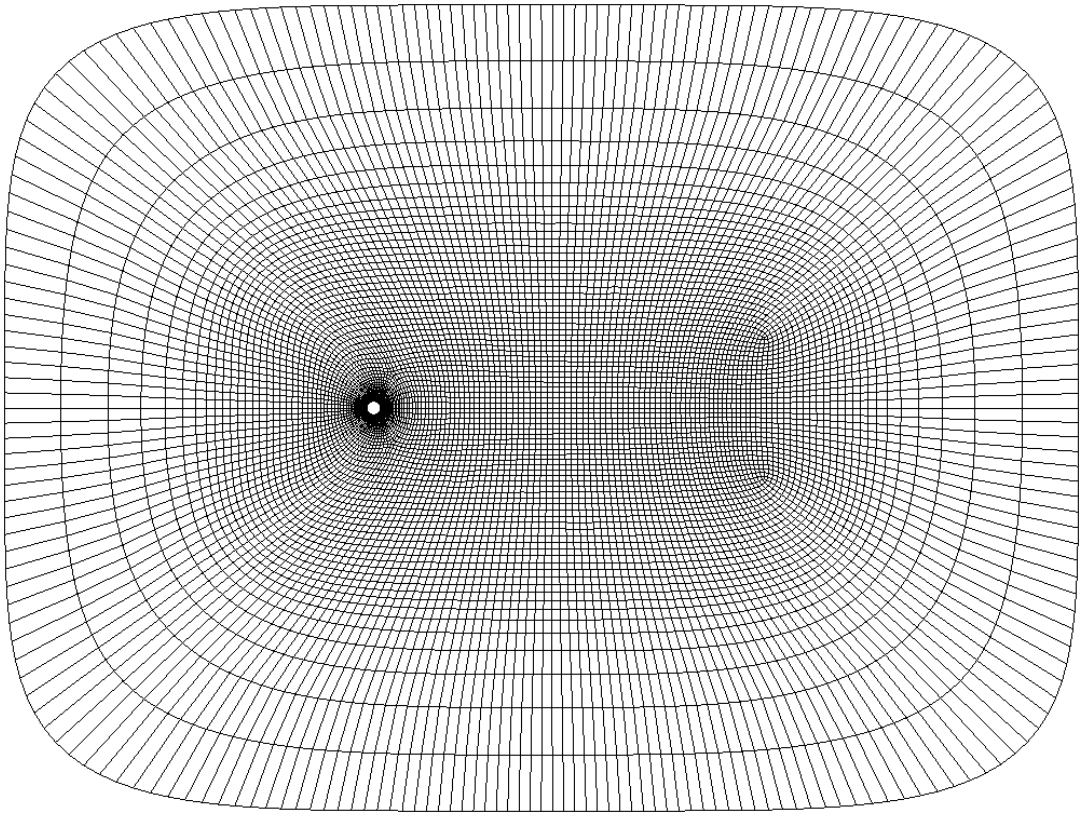
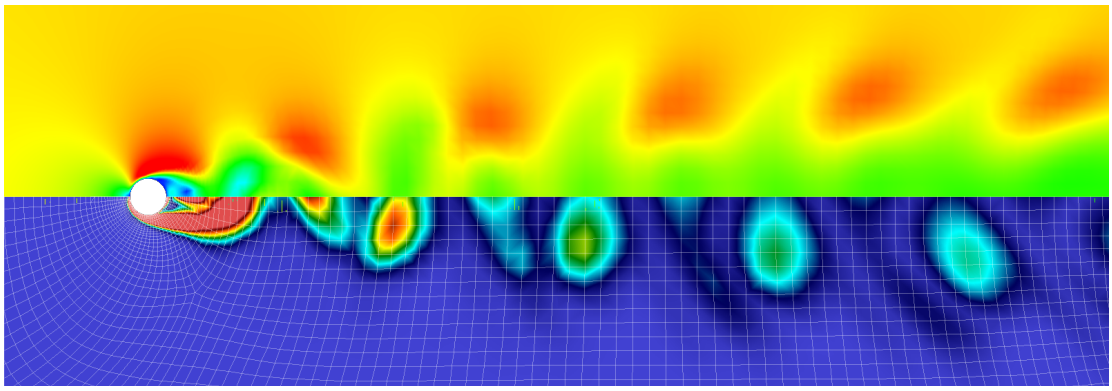


Figure 5.38: Entire coarse mesh for cylinder simulations.

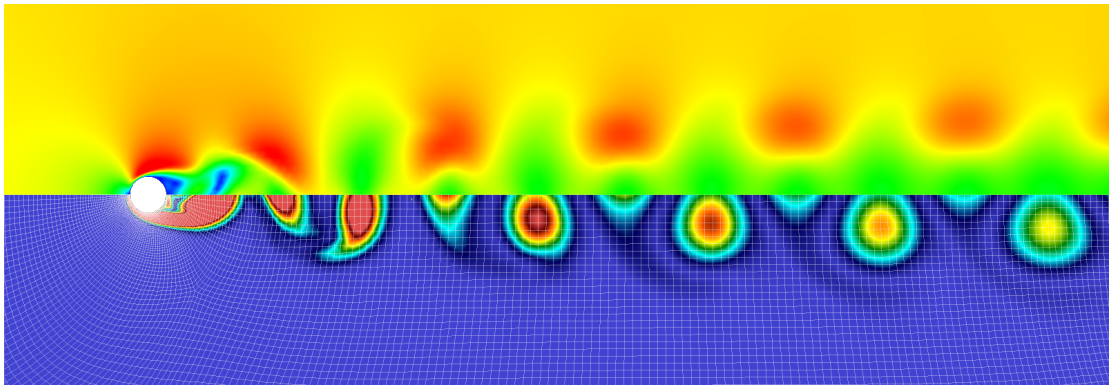
grid. Their finest spacing in the wake near the cylinder is roughly 0.04D × 0.04D. The off-body distance for the cell adjacent to the wall was approximately 0.0025D. Mittalt and Balachandar use a coarser spacing near the cylinder. After performing a grid convergence study, their final mesh includes 160 cells azimuthally or a spacing of nearly 0.02D.

Previously reported computational results by these authors show good agreement with cylindrical experiments. The grids used in this analysis are comparable and should be adequate for the problem at hand. Figure 5.39 shows example solutions obtained with a range of grid levels at a Reynolds number of 160. Contours of velocity magnitude (top) and vorticity (bottom) are shown for qualitative comparisons. All are extracted at an identical simulation time, but the shedding is no in sync across these solutions.

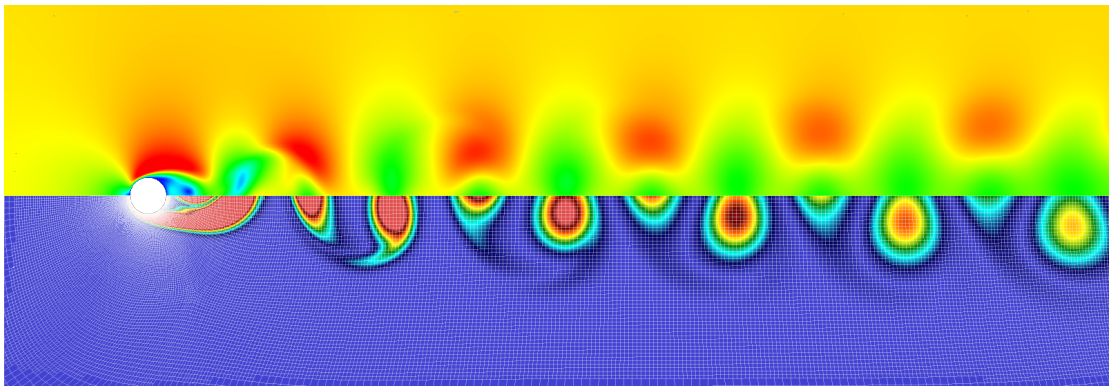
What is important is the increase in detail with grid refinement.



(a) Coarse



(b) Medium



(c) Fine

Figure 5.39: Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $\mathrm{Re_D} = 160$ cases at various grid resolutions.

5.5.2 Adaptation Strategy

There are several viable adaptation strategies for this problem and several are explored below. The first is the most straight-forward and is a geometric criteria. Two are based on vorticity and involve feature detection and repeated calls to adaptation and repartitioning routines. In addition, one additional refinement technique included as a target for comparison in this section: uniform refinement across the entire domain.

The first adaptation strategy refines all cells from the original coarse grid inside of a box domain that spans 4D on either side of the cylinder, 3D ahead of it, and 30D downstream. This is designed to be similar in character to the singularity-based grid, but was created with significantly less expense. Computationally, it should be as efficient as a conventional grid with singularities and is used in this work as a comparison for the feature-based refinement methods. Figure 5.40 shows the resulting grid after two levels of refinement have been performed in the region mentioned.

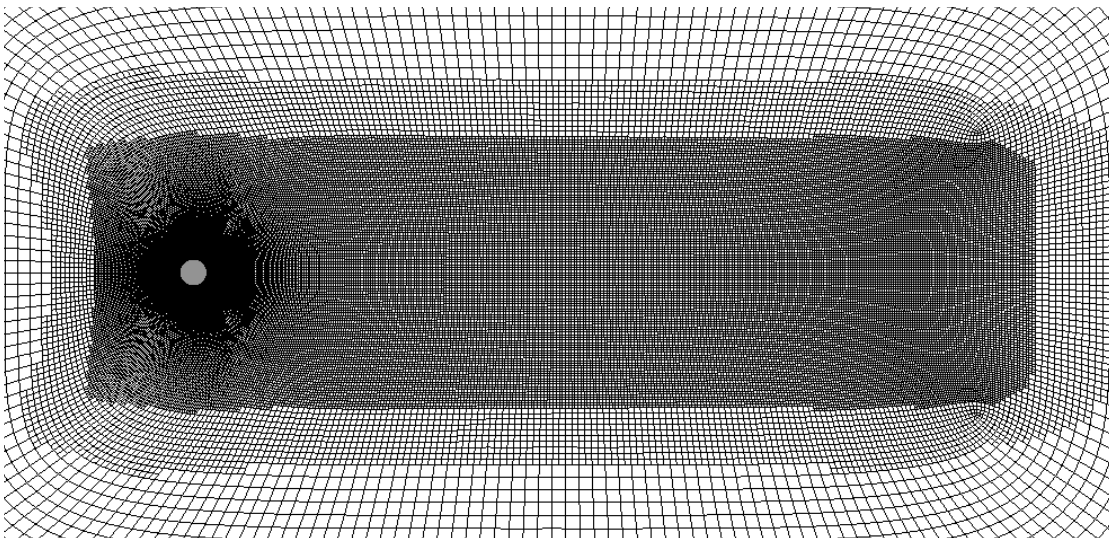


Figure 5.40: Illustration of extents of 'Geometric' AMR, two levels of refinement shown.

Voticity-based (ω) mesh refinement is logical for flow around a cylinder due to the shear layers and vortical roll-up that dominate important regions of the domain. Both dimensional and non-dimensional criteria were used. The dimensional refinement tolerance targeted cells whose vorticity were greater than 10 [1/s]. This value was determined by inspection of a completed, coarse simulation. Also shown are results that use the

non-dimensional Q-criterion described by Kamkar *et al.* and mentioned in Sec. 3.6. This includes a non-dimensional tolerance of unity and a Q-criterion magnitude limit on refinement. To maintain a uniform refinement near the cylinder geometry, all cells within one diameter of the cylinder are also refined for these cases.

Figure 5.41 shows an example of the resulting mesh for a solution after several hundred flow times. The flow has developed oscillatory wake shedding and the ω-based refinement follows the pattern of the shed vortices (more comprehensive images shown later). Both the ω- and Q-criterion-based refinement are limited to being no further downstream than 30 cylinder diameters, similar to the extents of the box refinement.

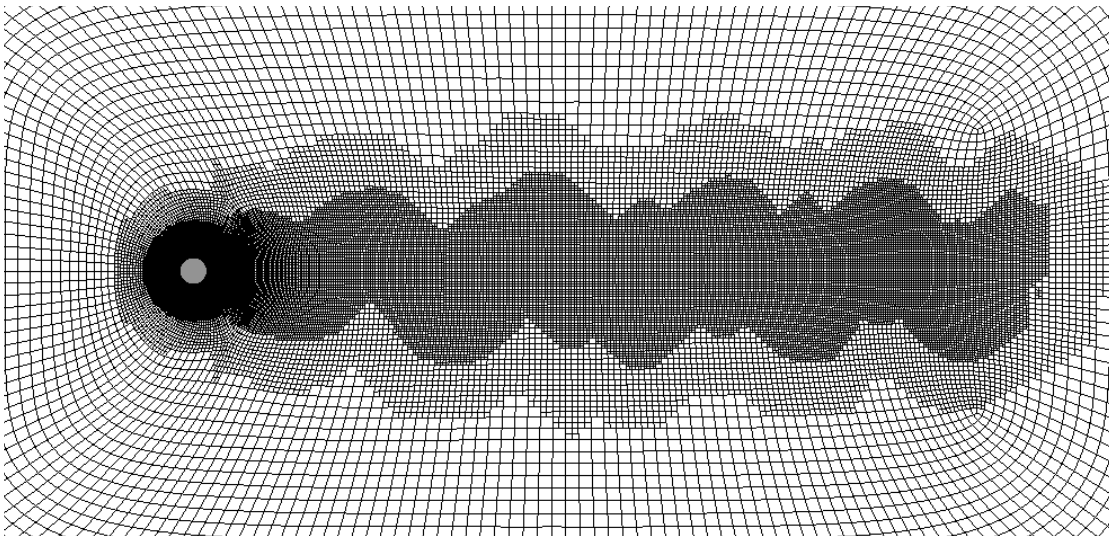


Figure 5.41: Illustration of extents of ω-based AMR, two levels of refinement shown.

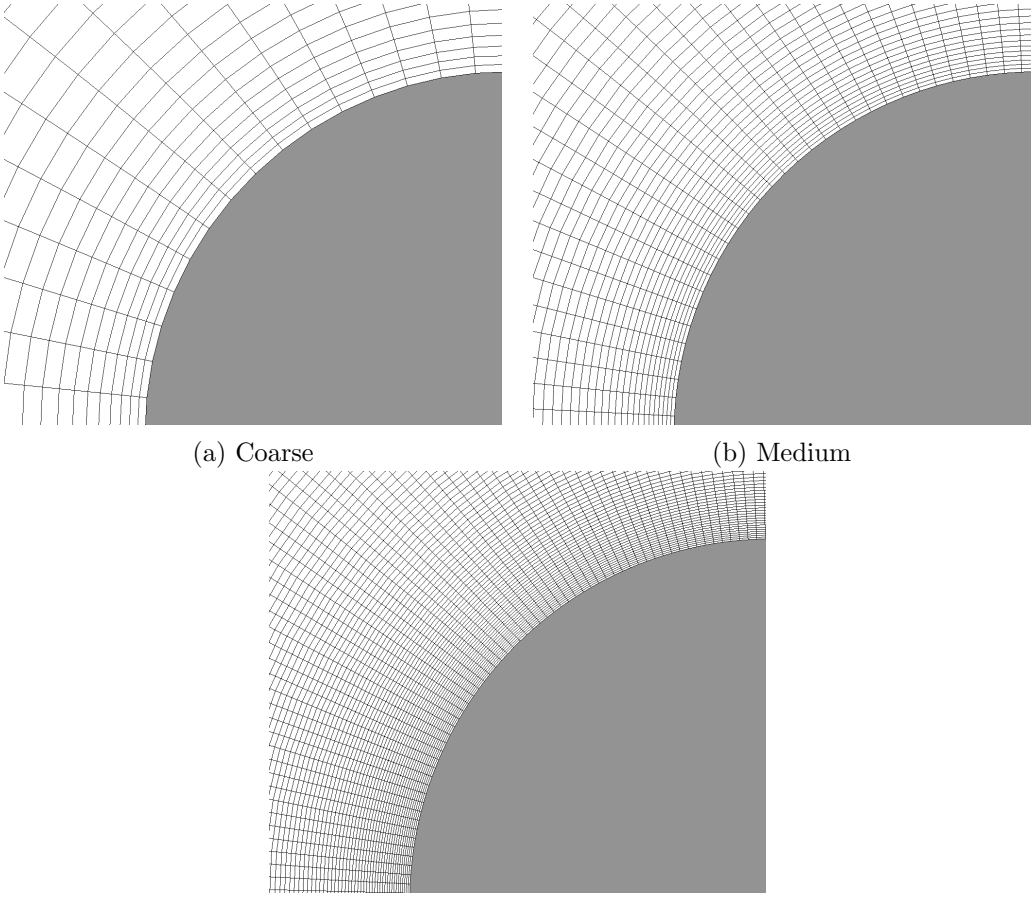
Refinement frequency was based on the time required for the freestream to convect through a typical cell in the fine wake. For those cases, they were all run at time step such that the maximum value of the convective-CFL in the mesh was 1.0,

$$CFL_{convective} = \frac{\Delta t\sqrt{u^2 + v^2 + w^2}}{\Delta x}.$$

In the wake of the cylinder, the convective-CFL is less than 0.1 for all cases. This implies that refinement every ten iterations would ensure that a feature triggering refinement does not move into a coarser region of grid before the next refinement pass. In actuality, these solutions use five buffer cells which refine into a ten fine cells (see Sec. 3.8). For

this reason, refinement every 100 iterations ensures that small-scale features remain well resolved. For conservatism, adaptation for the non-geometry criteria is perfumed every 80 iterations. Geometric criteria do not require refinement during time-stepping and do not use a refinement frequency.

As the initially coarse mesh is refined, the nodes on the surface of the cylinder are projected to maintain a radius of 0.5 [m]. Hyperbolic tangent smoothing along the lines of nodes near the body maintains good stretching ratios away from the viscous surface. Furthermore, grid lines are constrained to be orthogonal to the cylinder. Figure 5.42 shows three close-up views of the grid near the cylinder. They illustrate the effects of projection and node redistribution in the near-body region of the domain.



(a) Coarse (b) Medium

(c) Fine

Figure 5.42: Close-up of near-body mesh of cylinder at several grid resolutions.

5.5.3 Results

A range of solutions at two different Reynolds numbers are compared to comment on the accuracy and efficacy of AMR for this unsteady, subsonic application. For these flow conditions, the behavior is dominated by two-dimensional effects and the grids described previously should be sufficient to capture all relevant physics. All of the following cases were run with sixth-order, central, low-dissipation fluxes using second-order implicit time advancement with point-implicit relaxation.

Solutions were run for a total of 280 flow times, or four simulated seconds. Depending on the level of refinement, some solutions required between 160 and 200 flow times to develop steady vortex shedding behind the test body. Metrics for comparison between cases are average integrated drag coefficient and Strouhal number. The Strouhal number is a non-dimensional shedding rate defined as

$$St = \frac{fL}{U_\infty},$$

where f is the frequency of the vortex shedding, L is the characteristic length (cylinder diameter), and U_∞ is the freestream velocity. The last 70 flow times were used when computing the average drag and Strouhal number. This ensured that flow was fully developed and that a statistically significant sample size was used. Figure 5.43 shows the evolution of the lift coefficient for solutions at $\mathrm{Re_D} = 80$. Four refinement criteria are considered - 'All' is uniform refinement of the entire domain and 'Geometric' that is based on a bounding box. All four eventually show oscillatory behavior with a steady frequency and have an identical magnitude.

$\mathbf{Re_D = 80}$

The primary goal of this section is to compare the cost associated with frequent mesh refinement to the cost of conventional, un-refined grids. This comparison is only relevant if the computations obtain similar measures accuracy for important flow parameters. Table 5.12 contains results for a number of meshes using the previously described range of adaptation techniques for solutions at the lower Reynolds condition.

Percent error is calculated for each solution relative to the results from the coarse grid that was uniformly twice-refined ('All' in the table). It is considered the reference

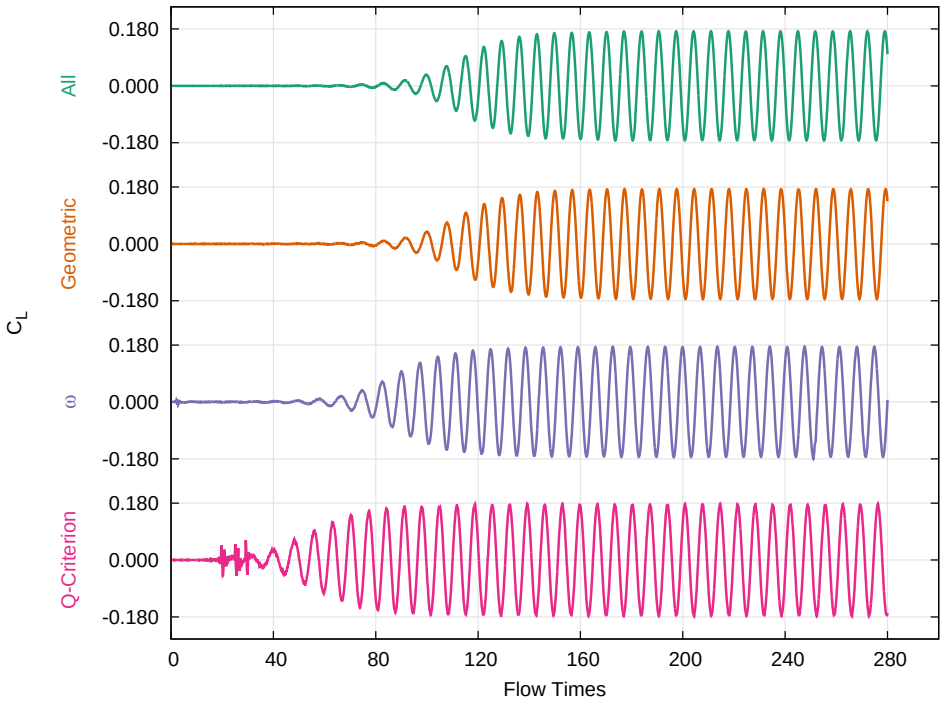


Figure 5.43: Comparison of integrated lift coefficient for cylinders at $\mathrm{Re_D} = 80$ with different refinement criteria.

to which the other solutions are compared because it eliminates any ambiguity about adaptation frequency, quantity of interest, or refinement tolerance. The uniformly refined case also does not included any hanging nodes.

As expected, the coarse grid without any refinement performs poorest and has the highest drag in drag coefficient when compared to the reference case. It does, however, obtain the correct shedding frequency. All other cases show agreement that is less than one percent of the reference value for both drag coefficient and Strouhal number. The low magnitude of error for the solution on the Medium grid indicates that at this Reynolds number, the solutions do not necessarily benefit considerably with the additional resolution afforded by the Fine grid.

When comparing the simulations at the finest refinement level, there is very little difference. Both the drag coefficient and the Strouhal number agree to within a percent.

Mesh Resolution	Adaptation Criteria	C_D	St	Error in C_D	Error in St
Coarse	-	1.336	0.1473	1.13%	0.00%
Medium	Geometric	1.351	0.1458	0.00%	0.96%
Fine	Geometric	1.349	0.1468	0.19%	0.30%
Fine	Q-Criterion	1.352	0.1459	0.07%	0.91%
Fine	ω	1.350	0.1473	0.09%	0.00%
Fine	All	1.351	0.1473	-	-

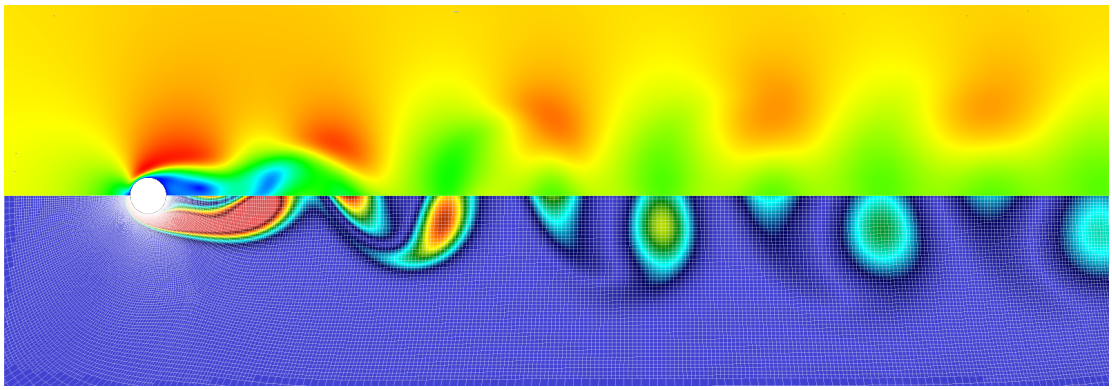
Table 5.12: Drag and Strouhal number results for $Re_D = 80$.

Similarly, the two refinement criteria that depend on feature detection also agree with the reference case to within a percent. For the purposes of comparison, the adapted grids present strategies that are as accurate as more conventional, fixed grids.

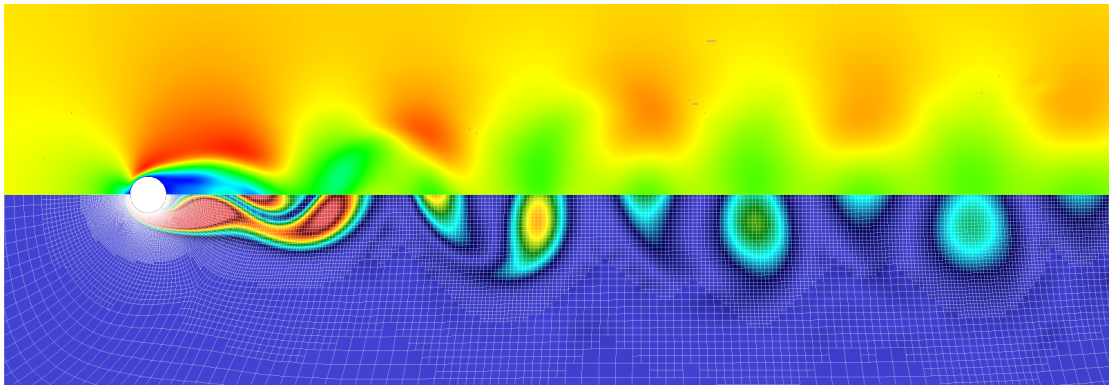
The expected results shown in Tab. 5.10 are different than the values presented here. Differences are on the order of 5% of less. While good to note, they point to an issue with the numerical methodology and not the adaptation. By using a computational reference case, these results are internally consistent and provide insight into the effect of adaptation and hanging nodes alone.

For a qualitative comparison, Fig. 5.44 shows the velocity magnitude (top) and vorticity (bottom) for the three adapted meshes. They agree well and it is obvious that the ω and Q-criterion adaptation methods achieve a smaller cell count than the geometric box refinement. At the cost of more frequent refinement and coarsening, they better contour the grid elements to the features of interest. Refinement based on non-dimensional Q-criterion has regions of poor refinement between the vortex cores that are obvious when looking at these images. This does not appear to impact the prediction of the shedding frequency or the drag, but highlights this sensors focus on strong vortexes and not all regions with vorticity.

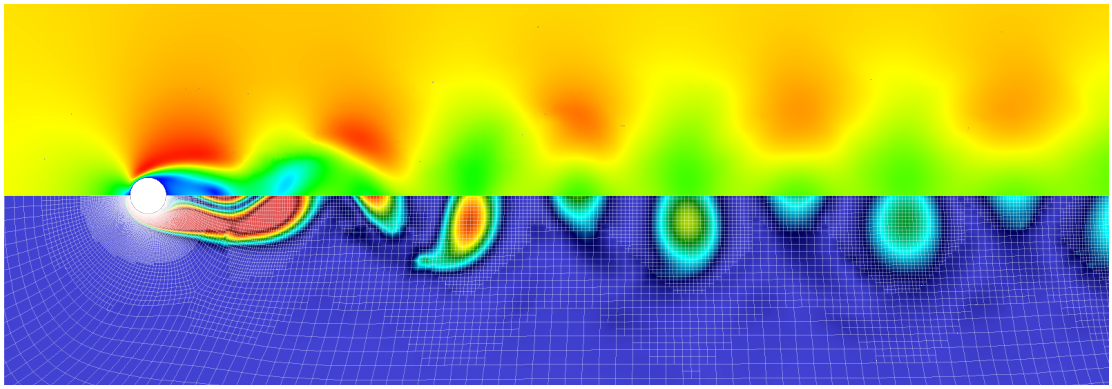
Use of adaptation dramatically reduces the number of cells in the simulation when comparing the uniformly refined mean to the geometric, or feature-driven meshes. Figure 5.45 presents the number of cells in the simulation as a function of time for all methods. Using the geometric criteria, the number of cells is halved. With either the ω or Q-criterion criteria, even fewer cells are required since the refinement conforms to the solution as was seen earlier. The reduction in cells also reduces the walltime required



(a) 2 level AMR - Geometric



(b) 2 level AMR - ω



(c) 2 level AMR - Q-Crit

Figure 5.44: Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $\mathrm{Re_D} = 80$ cases using several adaptation criteria.

for each iteration.

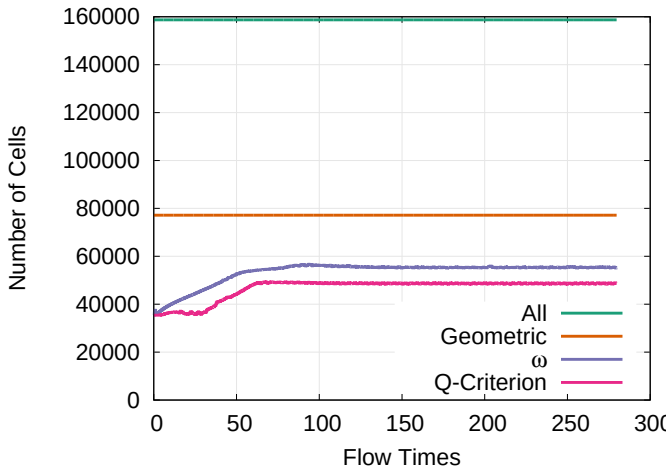
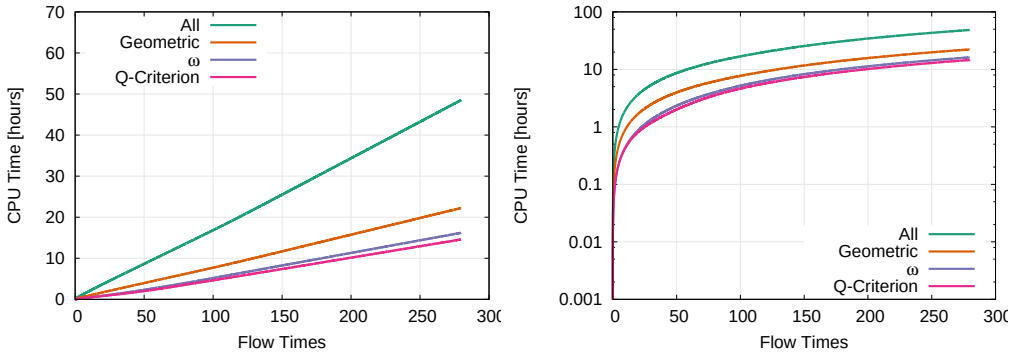


Figure 5.45: Comparison of cell counts for adapted grids ($\mathrm{Re_D} = 80$).

Figure 5.46 shows a comparison of the runtime required for these solutions in order to compare the overhead associate with AMR. Similar to prior examples, the uniformly refined grid is the most expensive, the solution on the grid refined only in a box containing the wake is less expensive, and the least expensive cases are those that were refined using feature detection. This is significant since the vorticity-based sensors required frequent adaptation and repartitioning unlike the other methods. It should be noted that the cost savings shown between the geometric and feature-based adaptation is less significant than has been reported for other cases. One reason is that the region of refinement is limited and geometric refinement does not require significantly more cells than feature detection. A more detailed analysis of these timings is shown in Tab. 5.13.



(a) Linear y-axis

(b) Logarithmic y-axis

Figure 5.46: Comparison of absolute CPU cost for adapted grids. Both linear and logarithmic y-axis shown ($Re_D = 80$).

Refinement Method	Geometric	Q-Crit	ω	All
Total Runtime [CPU hours]	22.23	14.58	16.19	48.52
Time Marching	81.49%	76.23%	76.35%	79.71%
Calculating Gradients	9.78%	8.87%	8.99%	9.72%
AMR / Feature Detection	0.00%	0.57%	0.55%	0.00%
Repartitioning / Memory Management	0.00%	2.90%	2.59%	0.00%
Node Redistribution	0.00%	0.63%	0.65%	0.00%
MPI Exchange / Waiting	6.48%	8.56%	8.63%	8.27%

Table 5.13: Percentage of run time in significant subroutines for cylinder solutions ($Re_D = 80$).

$\mathbf{Re_D = 160}$

The results for the cylinder at a higher Reynolds number are similar for those at a Reynolds number of 80. Comparison of the quantitative results for solutions at these conditions are shown in Tab. 5.14. All simulations agree to the uniformly refined case to within one percent and indicate that the they each resolve the flow field with similar accuracy. Stouhal numbers are increased and the drag has decrease when compared to the lower Reynolds number case. This is consistent with experimental observations. Unlike the previous results, the medium grid still has more significant percent error when compared to the fine grids.

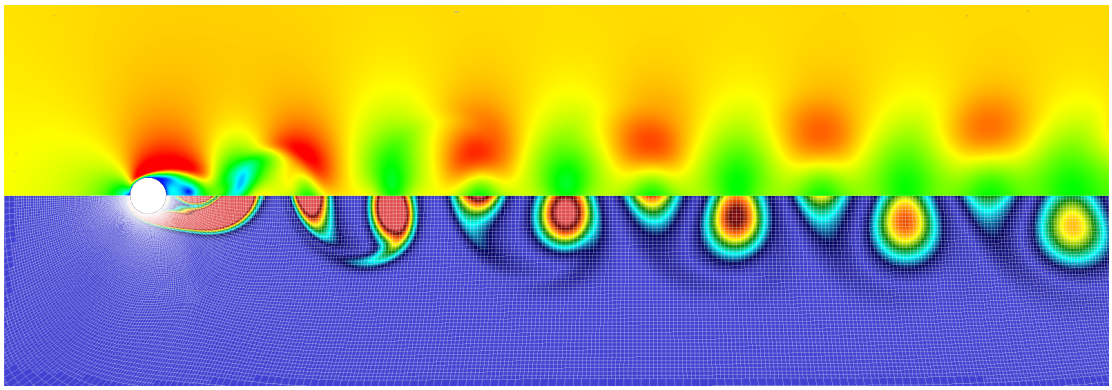
Mesh Resolution	Adaptation Criteria	C_D	St	Error in C_D	Error in St
Coarse	-	1.302	0.1684	1.45%	8.16%
Medium	Geometric	1.314	0.1793	0.55%	2.24%
Fine	Geometric	1.317	0.1834	0.29%	0.00%
Fine	Q-Criterion	1.322	0.1821	0.09%	0.73%
Fine	ω	1.316	0.1829	0.41%	0.24%
Fine	All	1.321	0.1834	-	-

Table 5.14: Drag and Strouhal number results for $\mathrm{Re_D} = 160$.

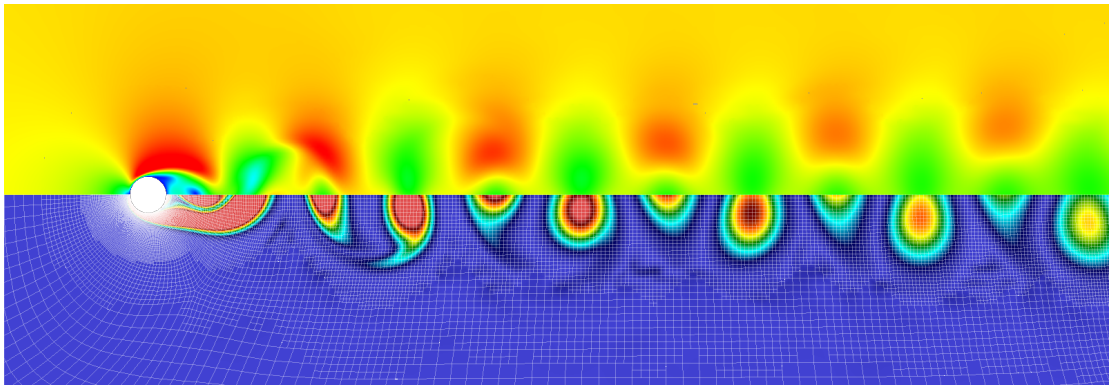
Figure 5.47 provides a qualitative comparison of the refinement techniques. Contours of Velocity magnitude (top) and vorticity (bottom) are shown. Major flow features are resolved across all three grids. Q-criterion adaptation again shows a preference for refining the vortex cores and leaves regions of less-refined cells between the vortices and also between the cylinder and the wake.

Figures 5.48 and 5.49 show comparisons of cell count and walltime between these solutions. Similar to what was seen before, feature-driven adaptation provides the lowest cell count. These solutions require more time than the $\mathrm{Re_D} = 80$ solutions because the time step is lower. The adapted grids still present considerable cost savings when compared to the strategy of uniform refinement.

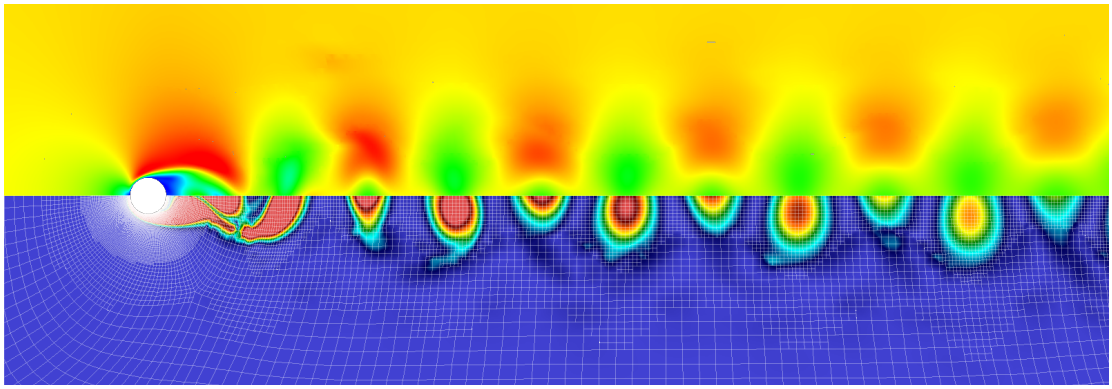
Table 5.15 shows a break-down of the time spent in several of the major subroutines in the solution. Roughly 5% of the runtime is used for AMR, feature-detection, smoothing, node redistribution, and repartitioning. These results are nearly identical to what was seen in for the $\mathrm{Re_D} = 80$ results.



(a) 2 level AMR - Geometric



(b) 2 level AMR - ω



(c) 2 level AMR - Q-Crit

Figure 5.47: Qualitative comparison of velocity magnitude (top) and vorticity (bottom) for $\mathrm{Re_D} = 160$ cases using several adaptation criteria.

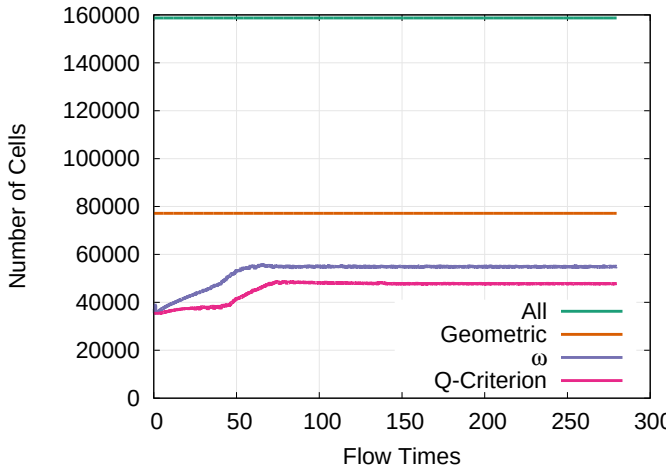
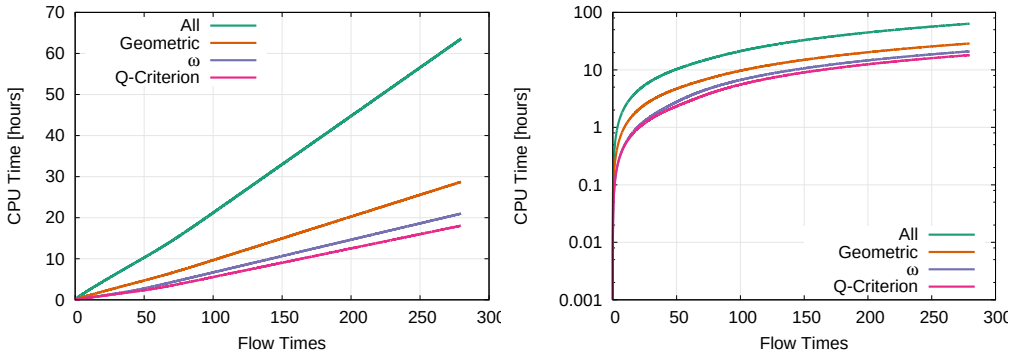


Figure 5.48: Comparison of cell counts for adapted grids ($\mathrm{Re_D} = 160$).



(a) Linear y-axis

(b) Logarithmic y-axis

Figure 5.49: Comparison of absolute CPU cost for adapted grids. Both linear and logarithmic y-axis shown ($\mathrm{Re_D} = 160$).

Refinement Method	Geometric	Q-Crit	ω	All
Total Runtime [CPU hours]	28.73	18.06	20.99	63.58
Time Marching	81.63%	76.29%	76.08%	79.17%
Calculating Gradients	9.77%	8.93%	8.89%	9.60%
AMR / Feature Detection	0.00%	0.59%	0.56%	0.00%
Repartitioning / Memory Management	0.00%	2.53%	2.76%	0.00%
Node Redistribution	0.00%	0.64%	0.65%	0.00%
MPI Exchange / Waiting	6.37%	8.83%	8.86%	9.05%

Table 5.15: Percentage of run time in significant subroutines for cylinder solutions ($\mathrm{Re_D}$ = 160).

5.6 Capsule Aerodynamics

NASA is currently working on developing a new Multi-Purpose Crew Vehicle (MPCV) to support manned missions beyond low-Earth orbit. During two phases of flight, abort and re-entry, the manned component of the MPCV is a crew capsule similar to the Apollo command module used in the 1960's. As mentioned in the previous section, several aerospace companies are also pursuing bluff-body capsules for crew and payload return missions from orbit. Methods for accurate characterization of the environment surrounding the capsule and its aerodynamic performance, especially at transonic and subsonic velocities, are important for the success of these and future programs.

This section includes simulations of the MPCV capsule using the solver outlined in this document. The geometry and conditions combine elements of the previous cases into a much larger, more challenging problem. At supersonic and transonic simulations, there are shock waves that require adequate resolution (a box shock for supersonic flow and a recompression shock in transonic flow). Similar to the cylinder problem, there is unsteady motion in the wake that requires frequent invocation of grid adaptation and graph repartitioning. Unlike the previous applications, the resulting flow is three-dimensional. This necessitates a much larger domain and grid sizes far in excess of those examined previous.

Accurate aerodynamic analysis of bluff bodies depends on meaningful representation of the large- and small-scale turbulent structure in the wake. For this reason, additional equations are required in the flow solver for this problem. In order to model the turbulent behavior of the flow, the Spalart-Allmaras Reynolds-averaged Navier-Stokes (RANS-SA) turbulence model is included.[64] The implementation includes the Catrips-Aupoix[19] compressibility correction and the SA-Neg[3] addition as described in the NASA Langley modeling resource.[20]

A hybrid of the RANS-SA model and large-eddy simulation, detached eddy simulation (DES97)[65], was also added to the solver in order to provide a more physically relevant turbulence model for comparison. The specifics of the derivation are not included here and are considered tangential to the evaluation of adaptive mesh refinement. Combined with the low-dissipation KEC numerical fluxes, DES has been used in prior work to provide good agreement to experimental measurements on capsule geometries.[60]

This section performs simulations for a small subset of the conditions included in the prior work and compares to experimental data from wind tunnel testing. The primary comparison is between results obtained using adaptively refined grids focused on feature refinement and those targeting a typical wake refinement region.

5.6.1 Wind Tunnel Test

For comparison to the simulations, wind tunnel test data from the 05-CA series are included.[7;8] 05-CA employed a model capsule that was 7.66% full-scale and included test points from Mach 0.5 to 1.4 and angles of attack from 140° to 170° . The conditions ranged from Reynolds numbers of 1.89 to 5.30 million (based on CM diameter). Point pressure measurements were collected by 168 static pressure taps. There were 11 unsteady pressure transducers and a six-component balance. Only the integrated loads from the balance are used in the subsequent discussion.

Table 5.16 shows the simulation parameters used for the subsequent analysis. Freestream conditions as well as vehicle attitude (angle of attack (α) and sideslip (β)) are identified. To simplify the analysis, the simulations were run at the nominal angle of attack, 155°.

Mach Number	α	β	Re_D	V_∞ [m/s]	ρ_∞ [kg/m^3]	T_∞ [K]
0.95	154.36°	0.12°	5.30E6	303.28	0.73920	255.57

Table 5.16: Wind tunnel conditions examined in this analysis.

The computational domain extended beyond the location of the wind tunnel walls and no attempt was made to account for the finite nature of the experimental test section. Additionally, the wind tunnel sting is not modeled and only the capsule geometry is present. Previous work using the OVERFLOW flow solver with a RANS turbulence model showed a small combined effect due to the tunnel walls and sting[54].

5.6.2 Grid Generation

The baseline mesh used for this analysis was generated using GridPro.[34] To eliminate any bias from an immature solve, the elliptic mesh solver was run until a steady-state was reached. Grid singularities were used to provide a region of localized refinement targeting the location of the wake for the α =155° solution. Similar to the grids created

for the low-Reynolds number cylinders, the boundaries of the grid extend 30 capsule diameters away from the test geometry. Figure 5.50 shows a cut through the initial mesh prior to refinement.

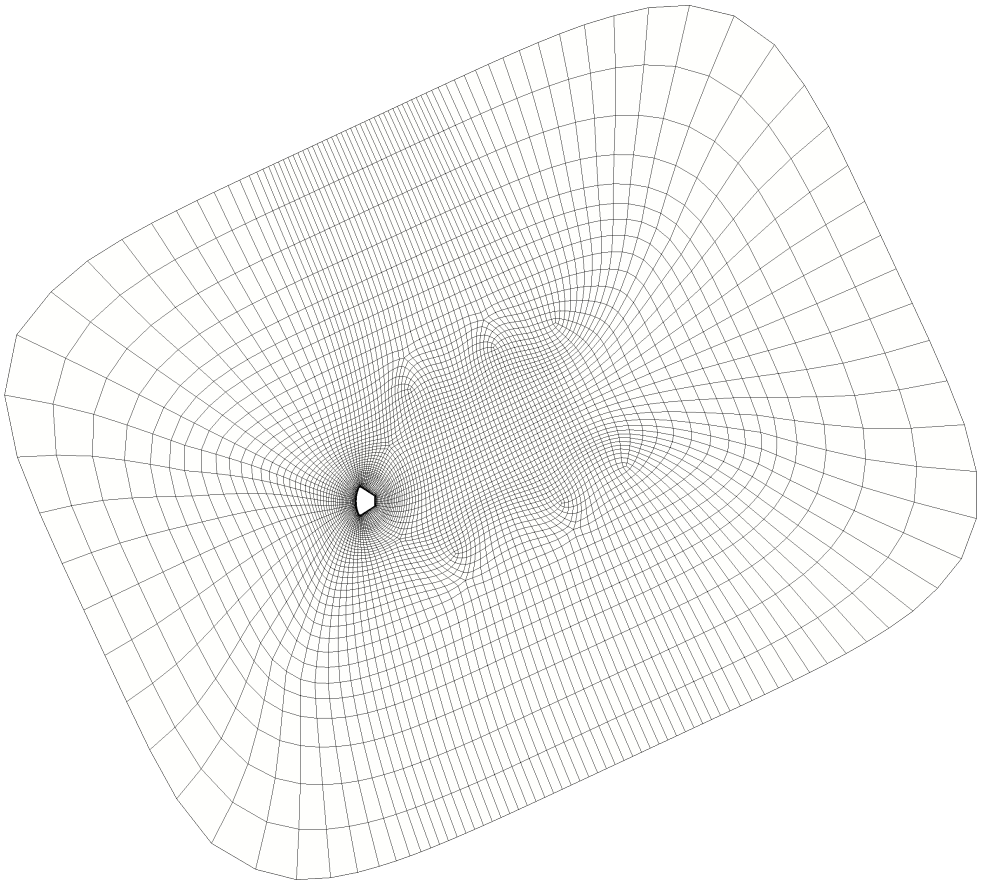


Figure 5.50: Image of pitch plane in baseline mesh with no refinement.

The baseline mesh is comprised of 103,000 cells. The spacing from the wall to the first first grid line is held constant for all grids in order to ensure a y^+ value of less than one in the region of attached flow. This wall spacing was maintained for all simulations, regardless of the level of adaptation.

5.6.3 Adaptation Strategy

Two strategies are included in the results below, one based one refinement in a specified region and one based on flowfield quantities. Similar to the previous case, comparison of these two approaches show that use of dynamic adaptation yields a solution that is as accurate as a simulation with a more conventional, fixed grid. Both approaches use projection to a fine representation of the capsule and node redistribution in the off-body direction to ensure a specified initial cell spacing. Each also uses grid sequencing during start-up to initialize the domain from the baseline mesh.

The strategy using geometric refinement is designed to further refine the region of the domain that contains the vehicle wake. It is in the shape of a cylindrical region that is rotated with the incoming freestream flow. Figure 5.51 shows the resulting grid after three levels of refinement have been performed in the region mentioned. The cylinder extends 2D ahead of the capsule, 9D downstream, and has a radius of 2D. Cylindrical volume refinement is a surrogate for a customary fixed grid and is used as a basis for comparison because the region of refinement is similar to what would be generated using a priori knowledge of the flowfield.

The turbulent wake contains significant vorticity. For this reason, adapting to a specified value of vorticity similar to what was done in Section 5.5 allows targeted refinement for the separated region of flow. Due to the presence of density gradients in the flow and a recompression shock in the wake, refinement based on ρ is also included. The joint tolerance for refinement used here is $\omega \geq 5.0\text{E}3\ [1/s]$ or $|\nabla\rho| \geq 1.0\ [kg/m^4]$. These tolerances were selected by investigation of baseline solutions; little additional work was performed in optimizing these values for the Mach 0.95 flow conditions. Figure 5.52 shows an illustration of the region of the domain refined using such a tolerance.

Figure 5.53-5.56 show images of the discretized capsule surface and near-body mesh in the pitch plane for several levels of refinement. Some observations can be made by looking at these figures. While initially coarse with clear faceting, the surface of the capsule is much more faithfully resolved with refinement and projection of the surface nodes. In the finest grids, the extents of the node redistribution can be seen. This implies that the hyperbolic tangent method had issues establishing a seamless transition between the near-body portion of the domain.

One final note on the mesh requires a discussion of Fig. 5.57. While the surface nodes

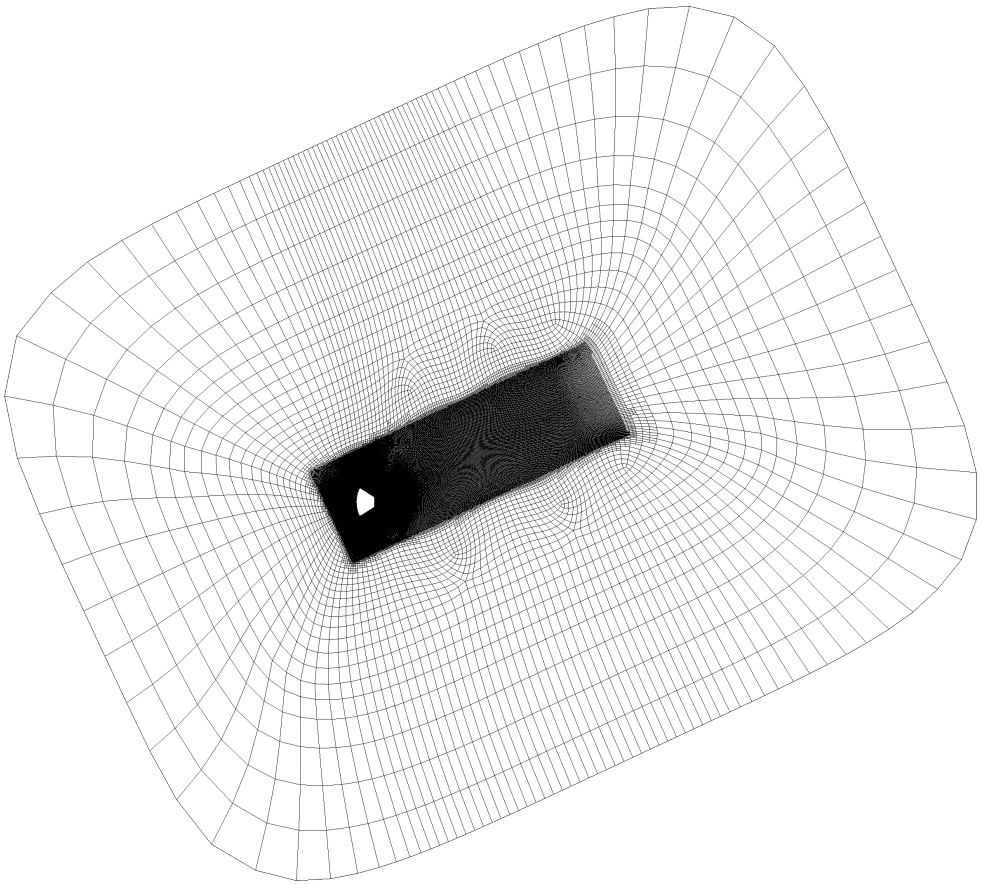


Figure 5.51: Image of cylindrical volume refinement in the pitch plane; three levels of refinement shown.

are projected to a very-fine database of the capsule geometry, there is no smoothing performed on the mesh. For this reason, the refined grid still presents evidence of the original, coarse discretization. In the figure, notice the clustering of points near singularities and the evidence of linear interpolation between the original, baseline grid lines. It is still possible to see the location of the baseline grid even after the solution has been refined three times. Smoothing of the mesh on the surface and in the volume and improving node redistribution away form the surface should be considered for future work, but it is likely not a significant source of error here.

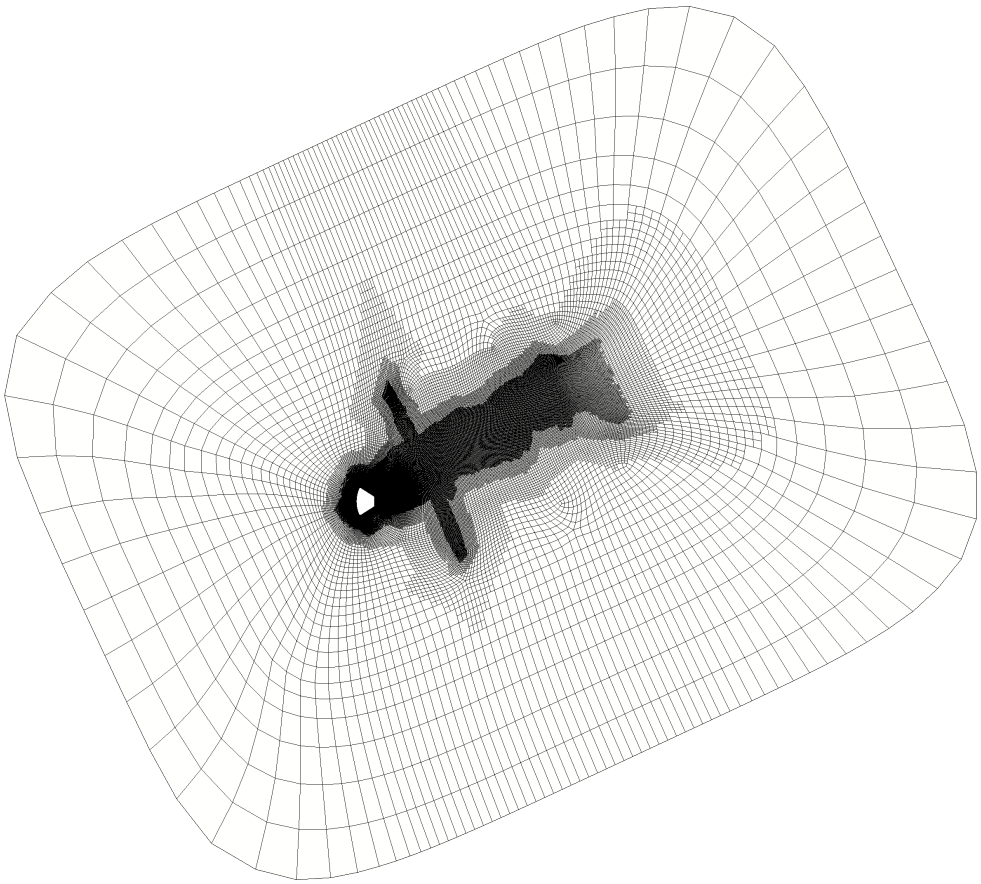


Figure 5.52: Image of feature-based refinement in the pitch plane; three levels of refinement shown.

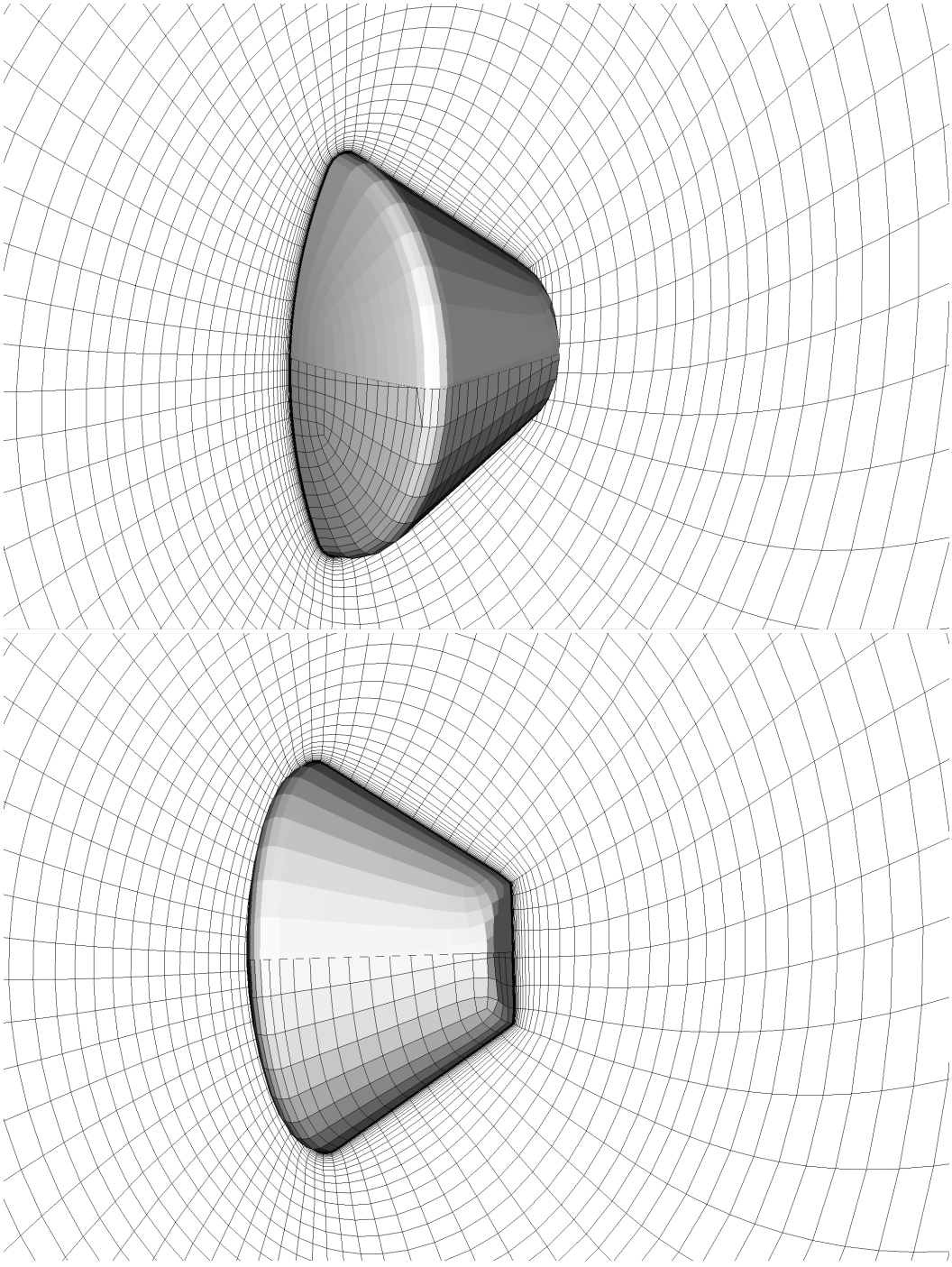


Figure 5.53: Near-capsule grid and capsule surface for initial grid (no AMR).

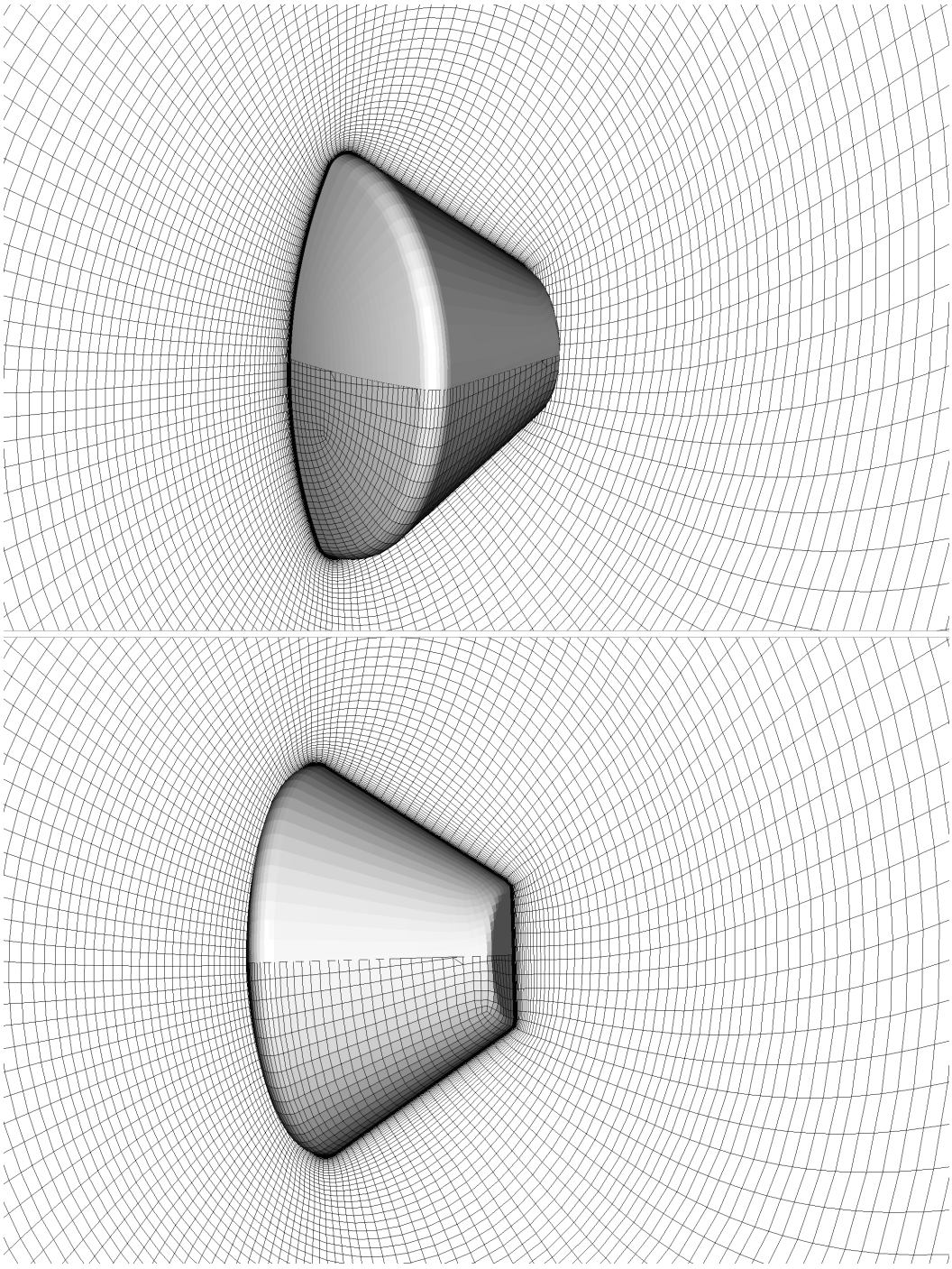


Figure 5.54: Near-capsule grid and capsule surface for coarse grid (1 level AMR).

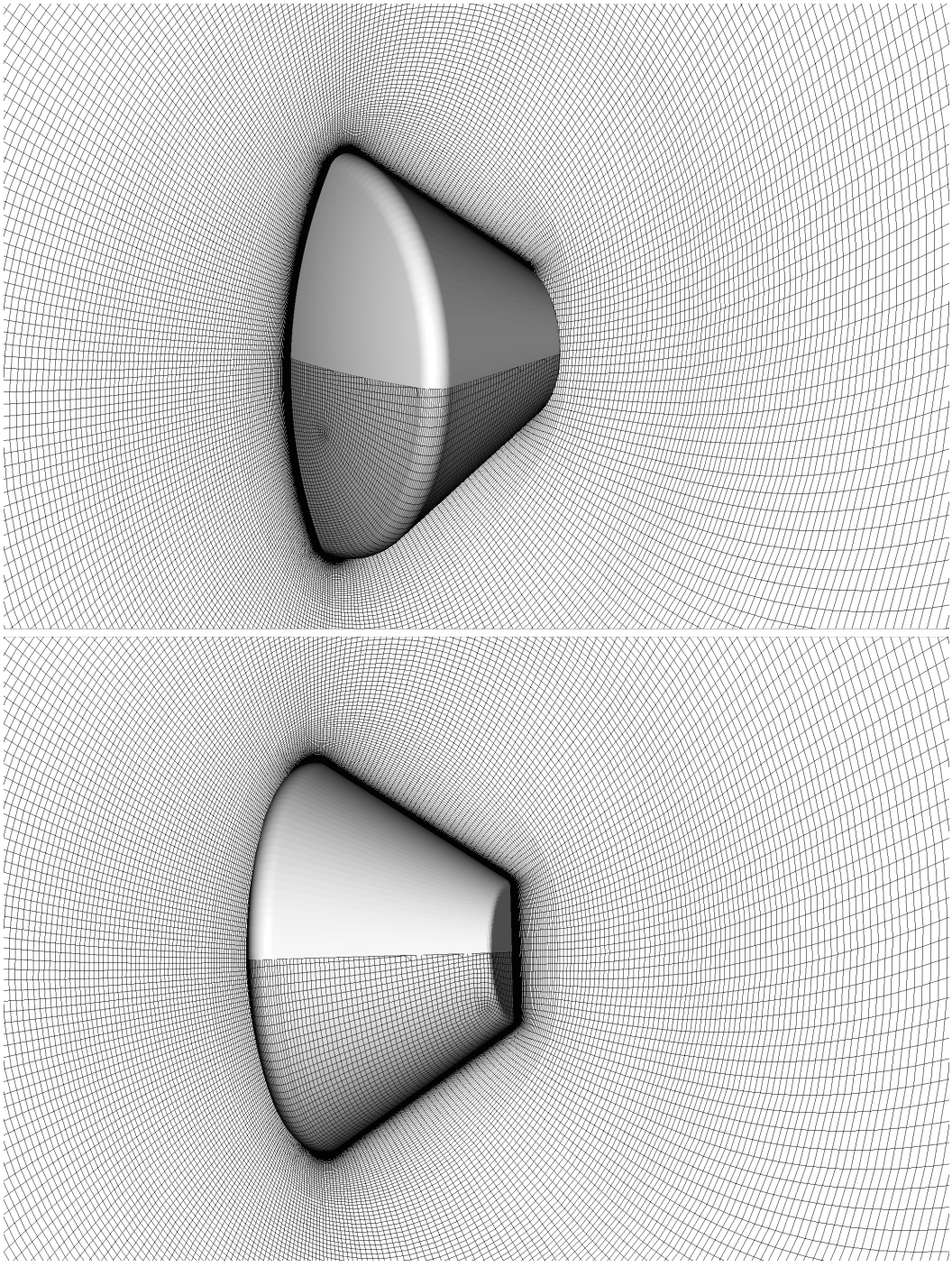


Figure 5.55: Near-capsule grid and capsule surface for medium grid (2 levels AMR).

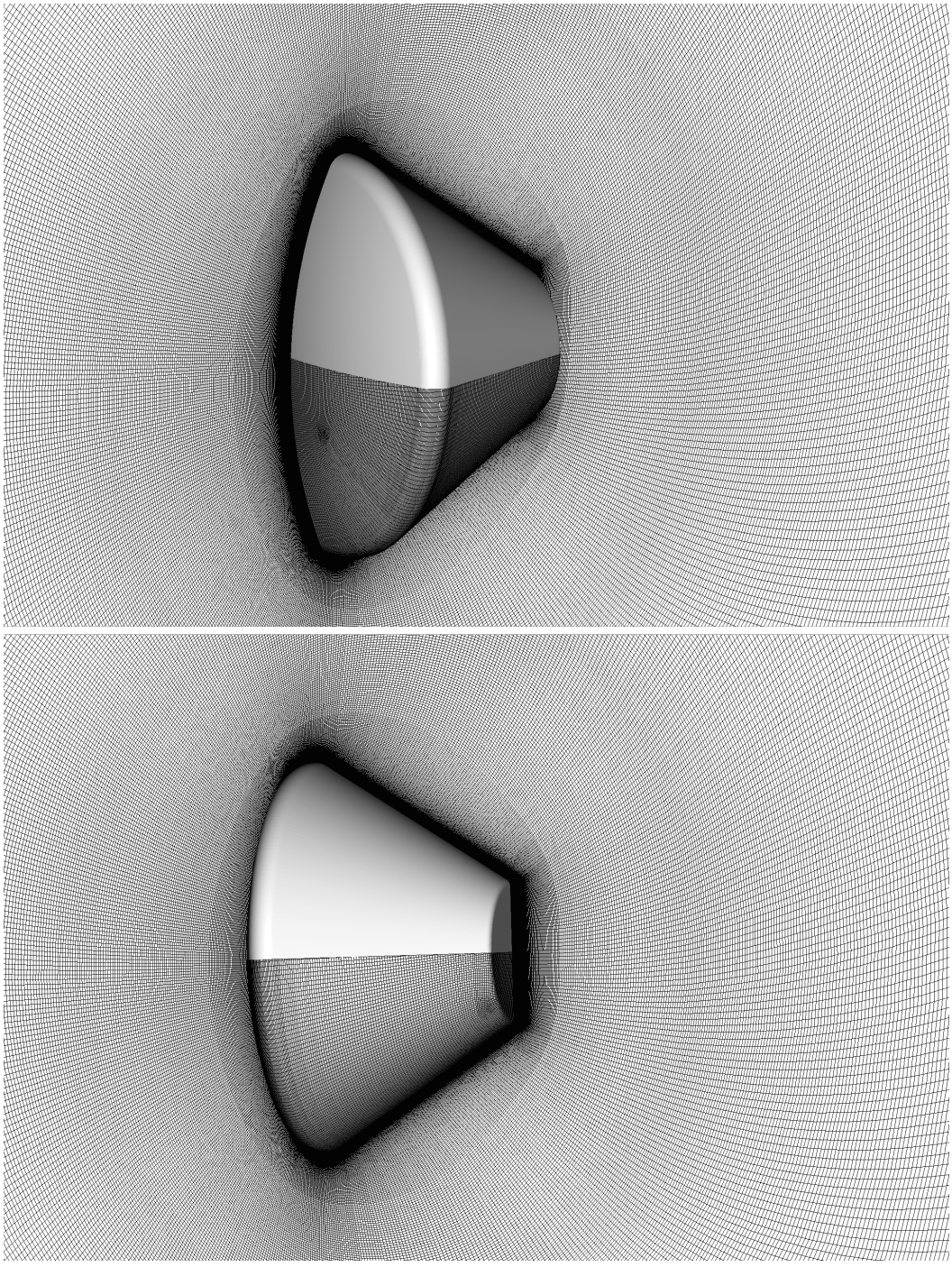
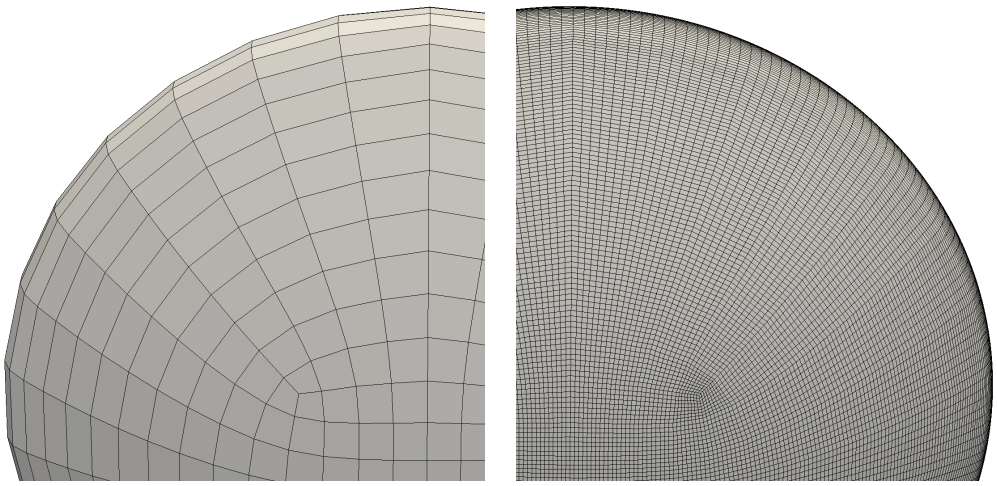


Figure 5.56: Near-capsule grid and capsule surface for fine grid (3 levels AMR).



Baseline Fine

Figure 5.57: Evidence of interpolation between baseline grid nodes in fine grid.

5.6.4 Results

Simulations using zero, one, two, and three levels of adaptive refinement were performed in order to gauge grid sensitivity for this problem. Both the cylindrical volume and feature-based refinement tolerances were considered. In order to ensure that the simulations had reached a fully-developed state, the cases were run for 275 flow times. Once completed, forces on the capsule were time-averaged over the last 50 flow times. The standard deviation of the fluctuations was calculated over the same period.

Figure 5.58 shows the instantaneous drag as the solution developed for a range of grid resolutions. Curves representing all four grid levels are included with solid curves represent solutions using cylindrical volume refinement and dashed lines represent those using feature-based refinement. On the right-hand side of the figure, points represent the average drag for each case. There is an error bar on the average value representing the plus and minus one standard deviation from the averaged value.

Agreement between the two methods of adaptation is very good. Specifics in the fluctuations differ, but after about 150 flow times their character agrees quite well. Time-averaged values also show good agreement with with respect to the standard deviations. The traces for the cases uses one and three levels of adaptation begin to differ during the last half of the averaging window and create slightly more pronounced differences. It is unclear if these is merely a temporary fluctuation or indicative of actual disagreement. Lift coefficient behaves similarly and is shown in Fig. 5.59.

To more closely compared the integrated aerodynamics obtained using adaptation, Fig. 5.60 compares the time-averaged quantities (points) with the experimental data (line) as a function of refinement. It is clear that as the domain is refined, the computational results begin to more closely agree with the experiment. Further, it appears that the grid with two levels of refinement is nearly grid converged and agrees very closely with the simulation using three levels of refinement.

Another important highlighted by Fig. 5.60 is the close agreement between the two refinement strategies. For the resolved flow (two and three levels of refinement) the time-averaged values agree on the order of their standard deviations and appear to be equally representative of the imperial data from the wind tunnel. This satisfies the purpose of this final application by showing that adaptation based on the evolving flowfield is as accurate as one generated with a more conventional approach.

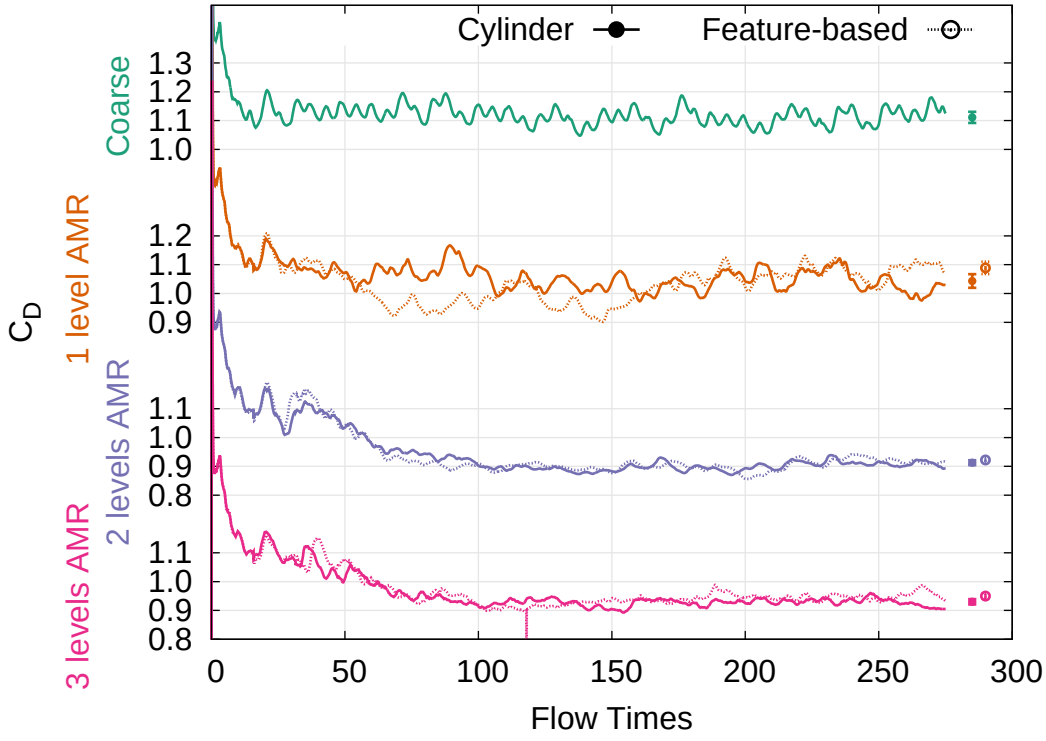


Figure 5.58: Instantaneous and average drag coefficient for capsule simulations using several grid resolutions.

Images showing the progression of flowfield quantities are shown for both adaptation techniques in Figures 5.61-5.63. A contour of Mach number in the capsule pitch plane has an overlay of the computational grid to show the region of refinement. Also colored by Mach number is a translucent isosurface of Q-criterion to identify vortical structures in the unsteady wake. While all images were created at identical physical times in the simulation, there are differences in the wake and location of the recompression behind the capsule even for grids with identical levels of refinement due to solution unsteadiness.

Figure 5.61 shows the solution on the baseline grid without any refinement. The vortical structure is large and resolution is understandably poor. The progression in Fig. 5.62 illustrates increased resolution of small-scale structures in the wake due to increased grid refinement. Between the two finest grids, there is relatively little gross

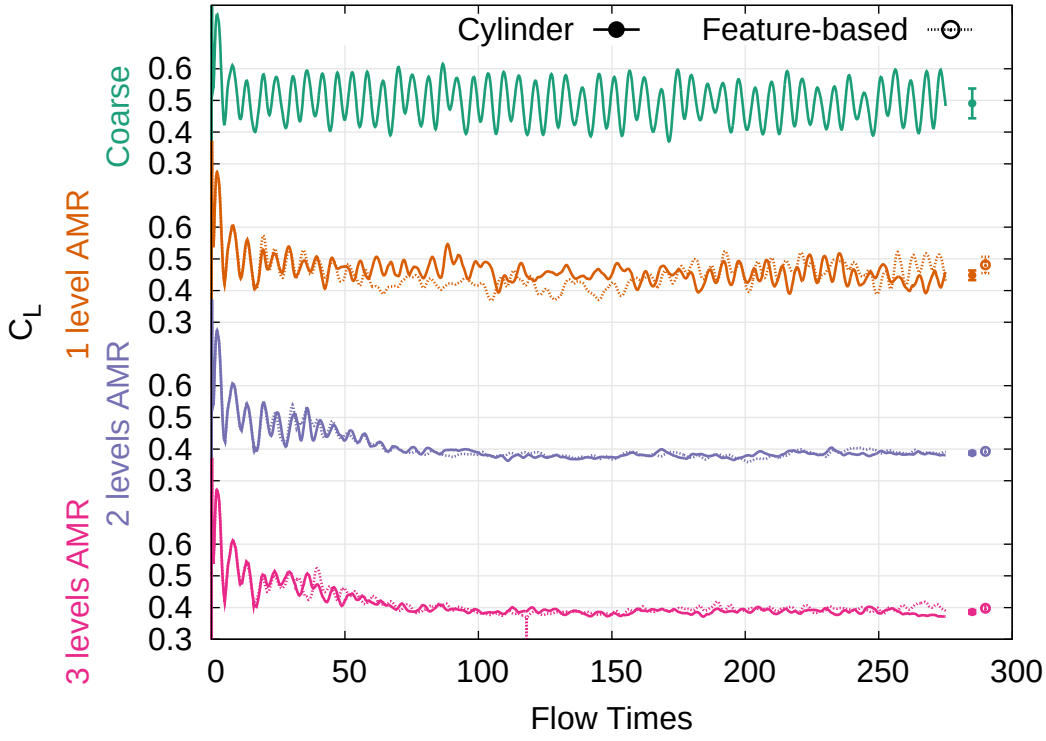


Figure 5.59: Instantaneous and average lift coefficient for capsule simulations using several grid resolutions.

change in the wake and may imply why there was good agreement in the forces and moments.

The images of the mesh when using feature-based adaption (Fig. 5.63) show behavior expected based on past performance. Adaptation conforms to the wake structure and, due to adaptation on $|\nabla\rho|$, the recompression region as well. For the most part, the feature-based adaption refines a smaller portion of the domain then the cylindrical refinement did. The exception is in the area near the recompression region which is refined far outside of the cylinder flagged for regional refinement.

In summary, the results obtained with feature-based refinement agree very well to those obtained using conventional grids. The region of refinement targeted by feature

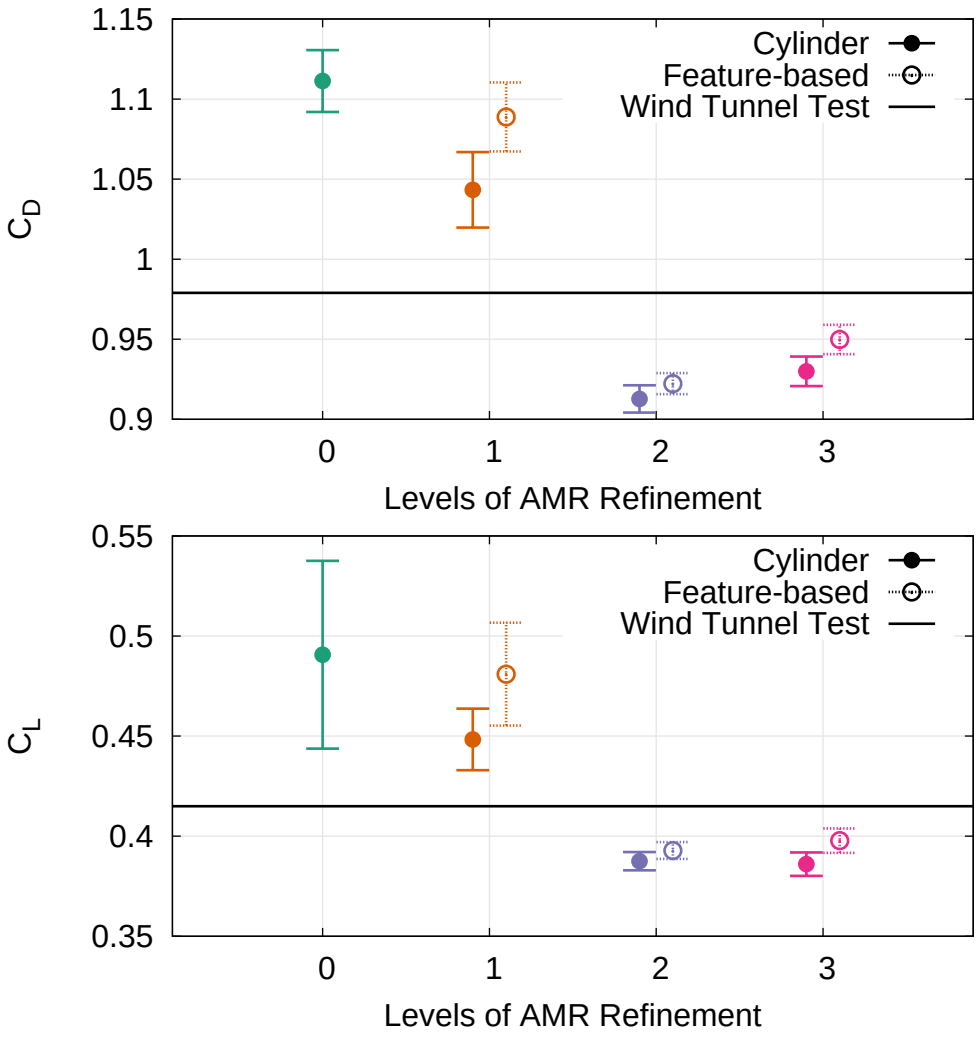


Figure 5.60: Convergence of drag and lift coefficients with additional refinement.

adaptation expand beyond the bounds identified using a priori identification and illustrate the usefulness of refinement and detection during the course of a simulation. While not explored here, there are further advantages to using adaptation for this problem (cylindrical or feature-based). The grid can be refined at run-time to include appropriate refinement downstream of the body of interest. With a more conventional grid generation strategy, modifications to the topology would be necessary to obtain a similarly tailored mesh.

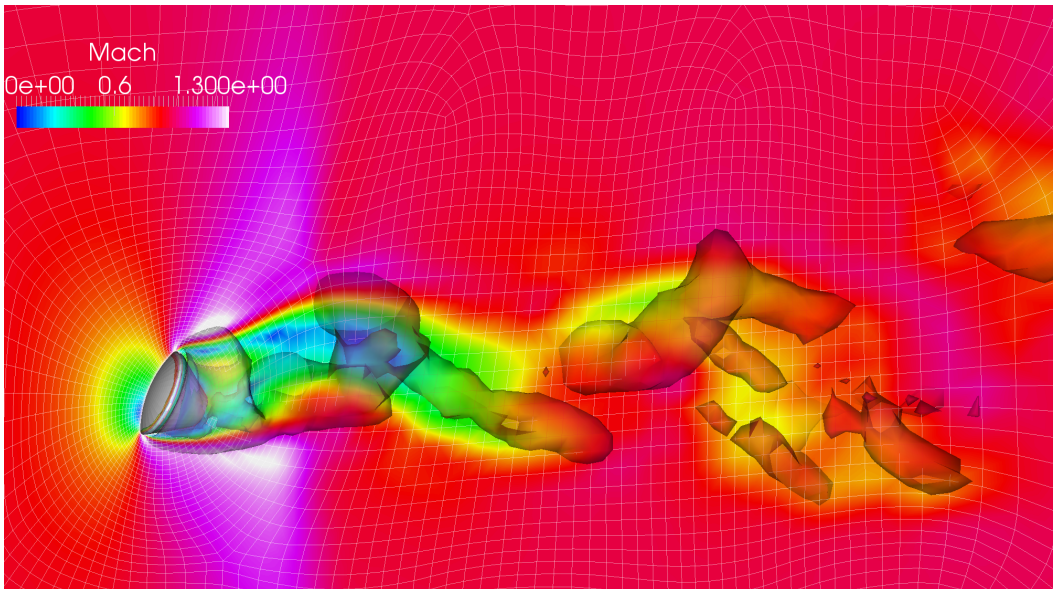


Figure 5.61: Visualization of coarse grid solution.

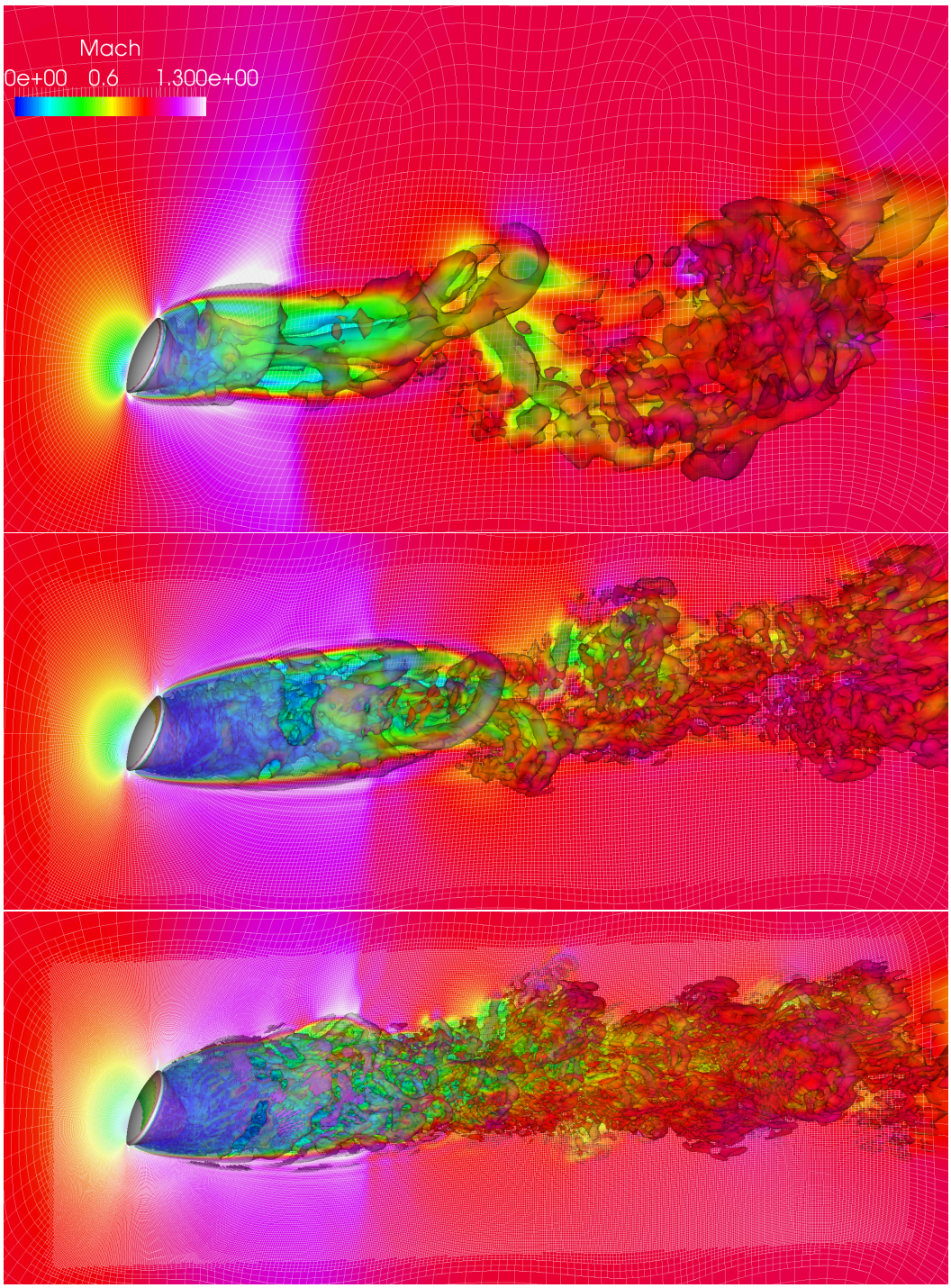


Figure 5.62: Progression of solutions on finer grids with one, two, and three levels of geometric refinement.

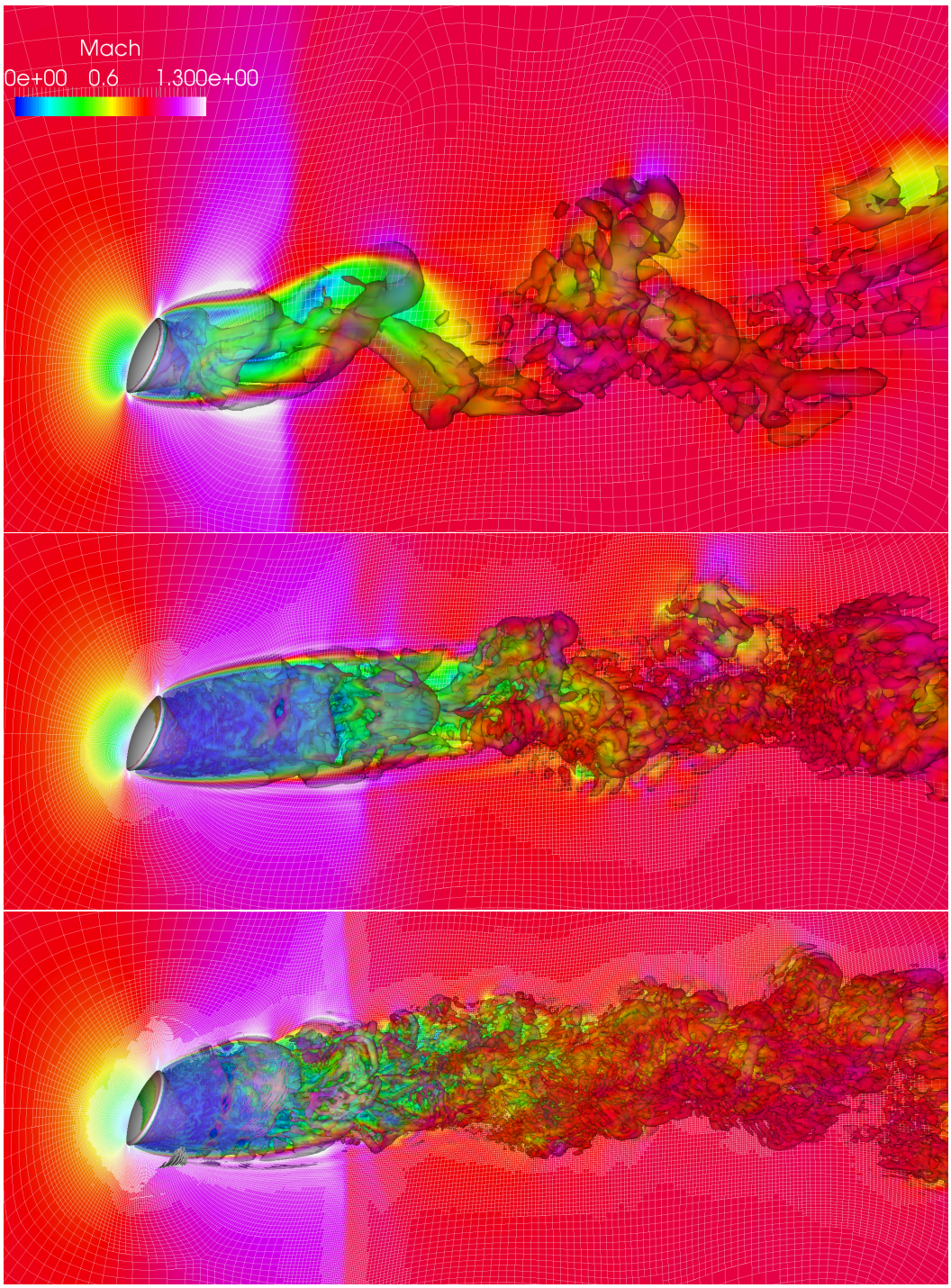


Figure 5.63: Progression of solutions on finer grids with one, two, and three levels of feature-based refinement.

5.6.5 Computational Efficiency

Similar to the previous sections, the computational efficiency of the two methods is compared. The results in this section point to anomalies not seen in the previous cases. Possible reasons for inefficiencies are discussed, but the details associated with them are reserved for future work.

Figure 5.64 shows a comparison of the number of computational cells used in the seven simulations considered for this problem. On the left-hand axis, measured in millions of cells, the sizes of the grids adapted two and there times as shown. The smaller problems with zero and one level of refinement are shown on the right-hand axis and are measured in thousands of cells. The unadapted case and those with region-based refinement are shown in solid lines. Mesh sizes ranged from 103,000 to 29,130,000 on either extreme.

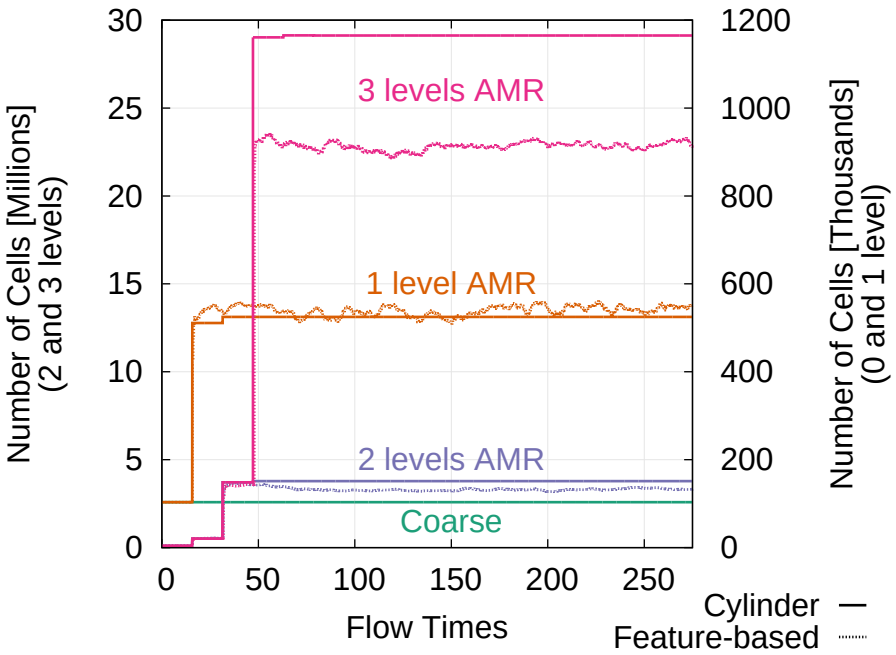


Figure 5.64: Cell count for capsule simulations using both techniques.

When using only one level of refinement, there is no net change in the number of cells

Refinement Method	Cylindrical	Feature-based
Total Runtime [CPU hours]	206.79	232.46
Time Marching	75.88%	71.32%
Calculating Gradients	5.60%	5.43%
AMR / Feature Detection	0.00%	1.42%
Repartitioning / Memory Management	1.04%	2.24%
Node Redistribution	0.01%	0.01%
MPI Exchange / Waiting	9.80%	12.46%

Table 5.17: Run time and percentage spent in subroutines for simulations using one level of AMR - Sandy Bridge system.

refined with either of the two adaptation strategies. For two and three level of AMR, there is appreciable reduction in cell count for the grids using feature-based adaptation (dashed lines). The figures showing visualizations of the mesh illustrate why there is not as pronounced a reduction of cells in this case then was seen in the previous example problems. Feature-based refinement follows the vortical structures in the wake and employ buffer cells to ensure that unsteady features are appropriately resolved. This results in a region of refinement that can be quite a bit larger than what is observed with cylindrical refinement. Even so, the finest grids show a nearly 25% savings in cell count when using feature-based criteria as opposed to refinement more typically seen in conventional grid topologies.

These solutions were run on one of two ICE systems: one with Intel Westmere (2.67GHz) processors linked via DDR and QDR Infiniband or one with Intel Sandy Bridge (2.60GHz) linked via FDR infiniband. Table 5.17 shows the timing information for the simulations that used one level of refinement. Using feature-based adaptation requires more computational time than the static grid with a fixed region of refinement. Since the number of cells are nearly identical for these cases, the added cost associated with adaptation and feature detection and a slightly increased percentage of idle time result in a more expensive simulation.

The trend in Table 5.18 is very similar. Feature-based refinement requires slightly fewer cells than a conventional grid according to Fig. 5.64. Again, cylindrical region refinement is about 10% leash expensive than feature detection. Feature-based refinement spends an increased percentage of time performing exchanges, waiting, and adapting the

Refinement Method	Cylindrical	Feature-based
Total Runtime [CPU hours]	1,210.35	1,305.77
Time Marching	77.36%	67.53%
Calculating Gradients	5.67%	5.32%
AMR / Feature Detection	0.01%	2.35%
Repartitioning / Memory Management	0.88%	2.35%
Node Redistribution	0.15%	0.10%
MPI Exchange / Waiting	8.15%	15.69%

Table 5.18: Run time and percentage spent in subroutines for simulations using two levels of AMR - Sandy Bridge system.

Refinement Method	Cylindrical	Feature-based
Total Runtime [CPU hours]	1,565.31	1,560.47
Time Marching	78.75%	70.81%
Calculating Gradients	5.29%	5.21%
AMR / Feature Detection	0.01%	2.52%
Repartitioning / Memory Management	0.82%	2.15%
Node Redistribution	0.14%	0.11%
MPI Exchange / Waiting	9.51%	14.14%

Table 5.19: Run time and percentage spent in subroutines for simulations using two levels of AMR - Westmere system.

grids. The approximately 7% increase in the time spend doing MPI operations is larger than what has been seen before and is noteworthy.

As a numerical experiment, both cases were run on a different processor architecture using the same number of processors. While the same number of processors were used, the Westmere nodes have 12 processors each as opposed to the 16 processors for Sandy Bridge. This means that more of the rank-to-rank communication was intra-node with the Sandy Bridge processorsand had a reduced latency. Table 5.19 shows the results from those two case run on the Sandy Bridge system. The results echo those shown in Tab. 5.18. Both strategies use a similar distribution of work between the subroutines as they did before. Even with the same number of cores, the time-to-solution is increased using the Westmere chips.

The final set of comparisons are the costs associate with three levels of adaptation,

Refinement Method	Cylindrical	Feature-based
Total Runtime [CPU hours]	8,990.27	9,738.15
Time Marching	74.55%	60.24%
Calculating Gradients	5.44%	4.83%
AMR / Feature Detection	0.01%	4.93%
Repartitioning / Memory Management	1.22%	2.78%
Node Redistribution	1.26%	1.61%
MPI Exchange / Waiting	9.50%	19.47%

Table 5.20: Run time and percentage spent in subroutines for simulations using three levels of AMR - Sandy Bridge system.

Tab. 5.20. These solutions were run using 96 Sandy Bridge processors. Between the two cases, there is a nearly 10% increase in the cost of the simulations when using feature-based refinement instead of a more conventional approach - similar to previous results. The discrepancy between the communication efficiency is more pronounced for these cases and feature-based adaptation requires 10% more CPU time exchanging information then the reference case. This encourages future work and analysis with more detailed study into partitioning across the computational ranks. There are likely possibilities for code efficiency improvements by partitioning or communication optimization.

In this application unlike previous ones, savings in cell count does not equate to a dramatic a savings in execution time. It suggests inefficiencies in the specifics of the parallelization that are beyond the scope of this analysis. For a three-dimensional problem using adaptation with three levels of refinement, there is a factor of 512 between the number of degrees of freedom in a coarse cell and one that has been refined three times. The partitioning strategy employed here builds a weighted graph based on the level-zero cells and is sensitive to such a large disparity in vertex weights. Future work could relax this constraint in order to assess if it provides relief and more efficient communication. It would have an associated cost in memory manage complexity, however.

Clearly, this is not the final word on computational expense using these techniques on large, unsteady problems. Further work is required in identifying if there is an issue with the manner of assembling the partition graph and the limitations placed upon it (partitioning only on level-zero cells), interfacing between the solver and ParMETIS, or in another facet of optimization, compiling, or execution. It is interesting that cases

with an average of more than 200,000 cells per node show an almost 20% idle time (Tab. 5.20) . This is in stark contrast to the timing study performed previously in Sec. 4.2.

The results using feature-based adaptation required, on-average, 10% more computational time than those that used cylindrical region refinement. However, feature-based refinement removes most of the subjectivity in deciding where to place fine grid cells in the domain. Also, in these solutions it refined regions that were outside of the cylindrical region that a more conventional topology might use. For solutions where the ideal placement of refined grid cells is unknown, accurate grid generation requires several iterations between topology generator and solver. If a single solution at 110% of the cost of an unadapted one replaces these expensive iterations, then it still represents a significant savings in total time to solution.

Chapter 6

Summary and Conclusions

In this document, considerations for adaptive mesh refinement in the context of a high-order, parallel, flow solver were explored. Use of adapted grids with hanging nodes did not contribute additional error to the simulations. Furthermore, it provided a means for achieving accurate results using a fraction of the number of computational elements required in conventional grids. During the course of the discussion, limitations and avenues for further work were proposed. In addition to summarizing the results from this work, this section will offer conclusions and suggest avenues for further work.

The first portion of this document outlined the underlying finite-volume methods used in this work. They were presented in the context of conformal grids without adaptation. Discretization of the governing equations in space and time using several approaches were shown. Most important to subsequent sections were the low-dissipation KEC fluxes and specifics for implicit time advancement.

This discussion was followed by a description of the AMR strategy used here and a definition of the terms used when referring to grids with hanging nodes and multiple levels of grid elements. Necessary augmentations to the numerical methods shown previously for use in a non-conformal meshes were presented. The implementation is designed with a dual-memory approach to facilitate implementation in already-developed flow solvers. Also shown were methods for parallelization, grid projection strategies, and handling of dynamic grid events in a distributed computation. Significant routines were presented as generalized algorithms.

Verification of the high-order fluxes with adapted grids was a major goal for this

work. An entire section explored the requirements for achieving accurate results with fourth- and sixth-order accurate numerical fluxes with unsteady AMR. Provided that a significantly small tolerance for refinement/coarsening is employed, adapted grids were as accurate as grids with a uniformly fine grid spacing. This showed that use of AMR did not contribute additional error to the solution and that it was compatible with spatial fluxes of order higher than two.

For modern simulations, it is rare to run a simulation on only one processor. Scalability of the methods described in this paper out to 512 processors was explored using a number of refinement and repartitioning frequencies. Parallel performance was shown for uniform and selectively refined meshes up to eight million cells. There was a significant reduction in the parallel performance of adapted meshes due to the reduction in the number of cells per processor and the additional communication required by repartitioning the mesh following refinement. However, even with these inefficiencies, the amount of time required to do a fixed amount of work was significantly (factor of three) less than what was required for the more-efficient, I uniform grids.

Once the underlying flow solver was verified, AMR was applied to successively more complicated applied problems. These applications were representative of several flows seen in more complicated simulations and a range of flow conditions. A broad range of test geometries were included from the trivial (one-dimensional shock tube) to the realistic (three-dimensional re-entry capsule). For all cases, the adapted simulations agreed very well to comparison cases with more conventional grids that included fixed regions of uniform refinement.

Steady and pseudo-steady, shock-dominated problems showed near perfect agreement between the results obtained using AMR and those obtained with fixed grids. Solutions using adapted grids required (generally) half the walltime as conventional grids. For problems with strong shocks, AMR dramatically reduces the number of cells required in the domain and enables the flow solver to accurately refine to the features of interest without requiring complex grid topologies or a priori knowledge of shock and shock interaction location.

The unsteady applications of AMR for bluff bodies in this document also showed good agreement with the baseline computations. Bluff-body flows at a given attitude have a well-understood wake location and knowledge.Even so, grid generation was made

much simpler by use of AMR targeting a specified region of the flow. Since the resulting meshes with regional refinement and grid sequencing were already quite efficient with cell placement, adaptation did not have as significant a cost savings for the simulations of flow over a cylinder. Use of feature-based adaptation still provided a benefit on the order of 25 percent. The simulations of flow around a capsule showed accurate results with feature-based refinement. It also identified possible avenues for future work in improving parallel efficiency in the solver or mesh partitioning.

As has been stated throughout this document, the usefulness of AMR is limited by the selection of relevant refinement criteria. The tolerances used in this work were informed by interrogating coarse grid solutions and leveraging understanding of the flow physics involved in the simulation. Oftentimes, some iteration was required as the character of the flow changes with refinement. Using feature-based refinement on an unfamiliar problem is not fool-proof. AMR replaces required expertise in grid generation with required expertise in refinement-criteria selection. However, it is frequently easier to iterate on the AMR criteria than it is on topological modifications to the grid.

AMR can dramatically simplify the grid generation processes and offload to the computer creation of solution-specific grids. This can enable more broad use of coarse, generalized grids for analysis sets or allow for simplified grid topologies for complex test geometries. Furthermore, it simplifies global or local grid refinement studies, necessary for understanding a simulation's accuracy with respect to grid spacing.

Left unexplored are exhaustive studies into refinement criteria, anisotropic refinement of cells, and further improvement to the grid via smoothing. These are obvious extensions to the methods described here and the limitations of the current process were mentioned. Improvements in scalability and memory management are also possible and were outlined. They remain as future work.

Adaptive mesh refinement is a useful tool in computational fluid dynamics. Like many tools used in modern numerical simulation, it is an active area of research. The methods and simulations described in this document satisfy the three goals of this work by showing that the high-order kinetic-energy consistent numerics are comparable with grids with hanging-nodes, outlining a scalable implementation, and applying these techniques to a range of motivating problems.

References

[1] Aftosmis, M. J. Surface Triangulation File Formats. *http://people.nas.nasa.gov/aftosmis/cart3d/cart3dTriangulations.html*.

[2] Aftosmis, M. J., Berger, M. J., and Murman, S. M. (2004). Applications of Space-Filling Curves to Cartesian Methods for CFD. Number AIAA 2004-1232.

[3] Allmaras, S. R., Johnson, F. T., and Spalart, P. R. (2012). Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model. Number AIAA 2013-2446.

[4] Balay, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., and Zhang, H. (2013). {PETS}c Users Manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory.

[5] Barkley, D. and Henderson, R. D. (1996). Three-Dimensional Floquet Stability Analysis of the Wake of a Circular Cylinder. *Journal of Fluid Mechanics*, 322:215–241.

[6] Bartkowicz, M. D. (2012). *Numerical Simulations of Hypersonic Boundary Layer Transition*. PhD thesis, University of Minnesota.

[7] Bell, J. H. (2006). EG-CEV-06-19: Test 05-CA Final Report. Technical report, NASA Johnson Space Center.

[8] Bell, J. H. (2007). Transonic/Supersonic Wind Tunnel Testing of the NASA Orion Command Module. Number AIAA 2007-1006.

[9] Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517.

[10] Berger, M. J. and Colella, P. (1989). Local Adaptive Mesh Refinement for Shock Hydrodynamics. *Journal of Computational Physics*, 84:64–84.

[11] Berger, M. J. and Oliger, J. (1984). Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512.

[12] Bramkamp, F., Lamby, P., and Müller, S. (2004). An adaptive multiscale finite volume solver for unsteady and steady state flow computations. *Journal of Computational Physics*, 197(2):460–490.

[13] Bramkamp, F. D. (2003). *Unstructured h-Adaptive Finite-Volume Schemes for Compressible Viscous Fluid Flow*. PhD thesis.

[14] Brix, K., Mogosan, S., M, S., and Schieffer, G. (2009). Parallelisation of Multiscale-Based Grid Adaption Using Space-Filling Curves. *ESAIM: Proc.*, 29:108 – 129.

[15] Buning, P. and Pulliam, T. H. (2011). Cartesian Off-Body Grid Adaption for Viscous Time- Accurate Flow Simulation. Number AIAA 2011-3693, pages 1–11.

[16] Candler, G. V., Barnhardt, M. D., Drayna, T. W., Nompelis, I., Peterson, D. M., and Subbareddy, P. K. (2007). Unstructured Grid Approaches for Accurate Aeroheating Simulations. Number AIAA 2007-3959.

[17] Candler, G. V. and MacCormack, R. W. (1991). The Computation of Hypersonic Ionized Flows in Chemical and Thermal Nonequilibrium. *Journal of Thermophysics and Heat Transfer*, 5(3):266–273.

[18] Candler, G. V., Subbareddy, P. K., and Brock, J. M. (2015). Advances in Computational Fluid Dynamics Methods for Hypersonic Flows. *Journal of Spacecraft and Rockets*, 52(1):17–28.

[19] Catris, S. and Aupoix, B. (2000). Density Corrections for Turbulence Models. *Aerospace Science and Technology*, 4:1–11.

[20] Chris Rumsey and Brian Smith and George Huang. *http://turbmodels.larc.nasa.gov/*. NASA Langley Research Center.

[21] D'Ambrosio, D. (2003). Numerical Prediction of Laminar Shock / Shock Interactions in Hypersonic Flow. *Journal of Spacecraft and Rockets*, 40(2):153–161.

[22] Data Parallel Line Relaxation (DPLR). NASA Ames Research Center.

[23] Derlaga, J. M., Roy, C. J., and Borggaard, J. (2013). Adjoint and Truncation Error Based Adaptation for 1D Finite Volume Schemes. Number 2013-2865, pages 1–11.

[24] Ding, K., Fidkowski, K. J., and Roe, P. L. (2013). Adjoint-Based Error Estimation and Mesh Adaptation for the Active Flux Method. Number AIAA 2013-2942, pages 1–25.

[25] Druguet, M.-C., Candler, G. V., and Nompelis, I. (2005). Effect of Numerics on NavierStokes Computations of Hypersonic Double-Cone Flows. *AIAA Journal*, 43(3):616–623.

[26] Ducros, F., Ferrand, V., Nicoud, F., Weber, C., and Darracq, D. (1999). Large-Eddy Sim ulation of the Shock / Turbulence Interaction. *Journal of Computational Physics*, 152:517–549.

[27] Edney, B. (1968). Anomalous Heat Transfer and Pressure Distributions on Blunt Bodies at Hypersonic Speeds in the Presence of an Impinging Shock. Technical report.

[28] Frauholz, S., Behr, M., Reinartz, B. U., and Siegfried, M. (2012). Numerical Simulation of Hypersonic Air Intake Flow in Scramjet Propulsion Using a Mesh-Adaptive Approach. Number AIAA 2012-5976, pages 1–22.

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

[30] Gaitonde, D. V., Canupp, P. W., and Holden, M. S. (2002). Heat Transfer Predictions in a Laminar Hypersonic Viscous/Inviscid Interaction. *Journal of Thermophysics and Heat Transfer*, 16(4):481–489.

[31] Gnoffo, P. A. (2009). Multi-Dimensional , Inviscid Flux Reconstruction for Simulation of Hypersonic Heating on Tetrahedral Grids. (January).

[32] Gottlieb, S. and Shu, C.-w. (1998). Total Variation Diminishing Runge-Kutta Schemes. 67(221):73–85.

[33] Greenshields, C. J., Weller, H. G., Gasparini, L., and Reese, J. M. (2009). Implementation of semi-discrete, non-staggered central schemes in a colocated, polyhedral, finite volume framework, for high-speed viscous flows. *International Journal for Numerical Methods in Fluids*.

[34] GridPro v4.4 (2004). http://www.gridpro.com, Program Development Company.

[35] Groth, C. P. T. and Northrup, S. A. (2005). Parallel Implicit Adaptive Mesh Refinement Scheme for Body-Fitted Multi-Block Mesh. Number AIAA 2005-5333, pages 1–17.

[36] Ham, F., Lien, F., and Strong, A. (2002). A Cartesian Grid Method with Transient Anisotropic Adaptation. *Journal of Computational Physics*, 179(2):469–494.

[37] Harten, A. (1994). Adaptive Multiresolution Schemes for Shock Computations. *Journal of Computational Physics*, 115:319–338.

[38] Harvey, J. K., Holden, M. S., and Wadhams, T. P. (2001). CodeValidation Study of Laminar Shock/Boundary Layer and Shock/Shock Interactions in Hyper- sonic Flow. Part B: Comparison with NavierStokes and DSMC Solutions. Number AIAA 2001-1031B.

[39] Holden, M. (1998). Shock interaction phenomena in hypersonic flows. In *29th AIAA, Plasmadynamics and Lasers Conference*, number AIAA 1998-2751 in Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics.

[40] Kamkar, S., Wissink, A., Sankaran, V., and Jameson, A. (2011). Feature-driven Cartesian adaptive mesh refinement for vortex-dominated flows. *Journal of Computational Physics*, 230(16):6271–6298.

[41] Karypis, G. and Kumar, V. (1998). A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.

[42] Kennel, M. B. (2004). KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space. *ArXiv Physics e-prints*.

[43] Kim, S.-e., Makarov, B., and Caraeni, D. (2003). A Multi-Dimensional Linear Reconstruction Scheme for Arbitrary Unstructured Grids. Number AIAA 2003-3990.

[44] Kirk, B. S. (2007). *Adaptive Finite Element Simulation of Flow and Transport Applications on Parallel Computers*. PhD thesis.

[45] Mittalt, R. and Balachandar, S. (1997). On the Inclusion of Three-Dimensional Effects in Simulations of Two-Dimensional Bluff-Body Wake Flows. In *1997 ASME Fluids Engineering Division Summer Meeting*.

[46] Moss, J. N., Pot, T., Chanetz, B., and Lefebvre, M. (1999). DSMC Simulation of Shock/Shock Interactions: Emphasis on Type IV Interactions. In *22nd International Symposium on Shock Waves*.

[47] Nemec, M., Aftosmis, M. J., and Wintzer, M. (2008). Adjoint-Based Adaptive Mesh Refinement for Complex Geometries. Number AIAA 2008-0725, pages 1–23.

[48] Nompelis, I., Bender, J. D., and Candler, G. V. (2011). Implementation and Comparisons of Parallel Implicit Solvers for Hypersonic Flow Computations on Unstructured Meshes. Number AIAA 2011-3547.

[49] Nompelis, I. and Candler, G. V. (2002). Computational investigation of high enthalpy flows past a finite cylinder in HEG. *West East High Speed Flow Fields*.

[50] Nompelis, I., Candler, G. V., and Holden, M. S. (2003). Effect of Vibrational Nonequilibrium on Hypersonic Double-Cone Experiments. *AIAA Journal*, 41(11):2162–2169.

[51] Nompelis, I., Drayna, T. W., and Candler, G. V. (2004). Development of a Hybrid Unstructured Implicit Solver for the Simulation of Reacting Flows Over Complex Geometries. Number AIAA 2004-2227.

[52] Nompelis, I., Wan, T., and Candler, G. V. (2007). Performance Comparisons of Parallel Implicit Solvers for Hypersonic Flow Computations on Unstructured Meshes. Number AIAA 2007-4334.

[53] Olejniczak, J., Wright, M. J., and Candler, G. V. (1997). Numerical study of inviscid shock interactions on double-wedge geometries. *Journal of Fluid Mechanics*, 352:1–25.

[54] Olsen, M. E. (2007). Sting Interference Analysis for CA-05 Test: CEM CM in AUPT. Technical report, NASA Ames Research Center.

[55] OVERFLOW (1991–2015). NASA Langley Research Center, Hampton, VA.

[56] Pointwise (1996–2015). http://www.pointwise.com/, Pointwise, Inc., Fort Worth, Texas.

[57] Pot, T., Chanetz, B., Lefebvre, M., and Bouchardy, P. (1998). Fundamental study of shock/shock interference in low density flow. In *21st International Symposium on Rar- efied Gas Dynamics*.

[58] Roy, C. J. (2009). Strategies for Driving Mesh Adaptation in CFD. Number AIAA 2009-1302.

[59] Roy, C. J. (2010). Review of Discretization Error Estimators in Scientific Computing. Number AIAA 2010-126.

[60] Schwing, A. M. and Candler, G. V. (2015). Detached-Eddy Simulation of Capsule Wake Flows and Comparison to Wind-Tunnel Test Data. *Journal of Spacecraft and Rockets*, 52(2):439–449.

[61] Schwing, A. M., Nompelis, I., and Candler, G. V. (2013). Implementation of Adaptive Mesh Refinement in an Implicit Unstructured Finite-Volume Flow Solver. Number AIAA 2013-2446.

[62] Schwing, A. M., Nompelis, I., and Candler, G. V. (2014). Parallelization of Unsteady Adaptive Mesh Refinement for Unstructured Navier-Stokes Solvers. Number AIAA 2014-3080, pages 1–24.

[63] Sod, G. A. (1978). A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws. *Journal of Computational Physics*, 27:1–31.

[64] Spalart, P. R. and Allmaras, S. R. (1992). A One-Equatlon Turbulence Model for Aerodynamic Flows. Number AIAA 92-0439.

[65] Spalart, P. R., Jou, W.-h., Strelets, M., and Allmaras, S. R. (1997). Comments on the Feasibility of LES for Wings, and on a Hybrid RANS/LES Approach. In *First AFOSR International Conference on DNS/LES*.

[66] Steger, J. L. and Warming, R. (1981). Flux Vector Splitting of the Inviscid Gasdynamic Equations with Applications to Finite-Difference Methods. *Journal of Computational Physics*, 40:263–293.

[67] Subbareddy, P. K. and Candler, G. V. (2009). A fully discrete, kinetic energy consistent finite-volume scheme for compressible flows. *Journal of Computational Physics*, 228(5):1347–1364.

[68] Venditti, D. a. and Darmofal, D. L. (2003). Anisotropic grid adaptation for functional outputs: application to two-dimensional viscous flows. *Journal of Computational Physics*, 187(1):22–46.

[69] Vinokur, M. (1983). On one-dimensional stretching functions for finite-difference calculations. *Journal of Computational Physics*, 50(2):215–234.

[70] Waltz, J. (2003). Parallel Adaptive Refinement for 3D Unstructured Grids. Number AIAA 2003-1115.

[71] Williamschen, M. J. and Groth, C. P. T. (2013). Parallel Anisotropic Block-Based Adaptive Mesh Refinement Algorithm For Three-Dimensional Flows. Number AIAA 2013-2442, pages 1–22.

[72] Williamson, C. H. K. (1996). Vortex Dynamics in the Cylinder Wake. *Annual Review of Fluid Mechanics*, 28:477–539.

[73] Wissink, A. M., Kamkar, S., Pulliam, T. H., Sitaraman, J., and Sankaran, V. (2010). Cartesian Adaptive Mesh Refinement for Rotorcraft Wake Resolution. (July):1–18.

[74] Wright, M. J., Candler, G. V., and Prampolini, M. (1996). Data-Parallel Lower-Upper Relaxation Method for the Navier-Stokes Equations. *AIAA Journal*, 34(7):1371–1377.

[75] Zhang, Z. J. and Groth, C. P. T. (2011). Parallel High-Order Anisotropic Block-Based Adaptive Mesh Refinement Finite-Volume Scheme. Number AIAA 2011-3695.

Appendix A

Derived Datatypes

There are several derived datatypes used by the flow solver in order to maintain connectivity between computational cells, faces, and nodes, store solution quantities during dynamic grid events, and serve as linked list elements. Tables A.1, A.2, and A.3 enumerate the contents of these derived datatypes and provide descriptions of their contents. There are other derived datatypes used in the solver, but these are the most important.

The tables indicate which variables are required from the derived datatypes when creating the one-dimensional arrays used by the flow solver. These variables are indicated by a • to the left of the entry. While time stepping, memory for these variables is duplicated and held by both the 1-D array and the derived datatype array. The flow solver is agnostic to the multiple levels of refinement in the grid, and none of the AMR parent/child relationships are maintained outside of the derived datatypes. in the arrays.

As documented here, the relationship from the parents to the children and vise-versa use integers referencing the element's PID, or the location in the oversized array mentioned in Sec. 3.4. As mentioned in that section, these could easily be replaced by pointers if implementation used only linked lists and not the array of derived datatype elements. The pointers 'prev' and 'next' are used to maintain connectivity in the 'Active' and 'Empty' linked lists that help manage and efficiently traverse the array of derived datatypes.

The special pointers 'q' and 'q_n' reference a derived datatype that contains the solution state vector for a given cell at the current or previous time level. This derived

datatype is not shown and was used only to simplify the allocation of several time levels or the inclusion of solution quantities tied to other derived data types. Ghost cells are not maintained in any linked list due to the limited number of boundary conditions implemented for the flow solver used in this work. Solution quantities at the boundaries are sufficient to populate the ghost cells. To incorporate boundaries where solution quantities mature with the solution, an additional solution pointer could be added to each of the face elements and the face portal.

Not shown are derived datatypes that contain information concerning the boundary conditions, CFL schedule, fringe cells from other processors, and triangulated files for surface projection. These were considered specific to the flow solver developed in this work and are likely code-specific.

	Name	**Type (Dimension)**	**Description**
	GID	INTEGER	Node's global number
	PID	INTEGER	Node's local number (in derived datatype array)
	ID	INTEGER	Node's location in 1-D array
	owner	INTEGER	Rank that 'owns' the node
•	xcn	REAL(3)	Current node location
	xco	REAL(3)	Node's original location
	glvl	INTEGER	Node's grid level
	parent	INTEGER(2)	Parents of the node
	child	INTEGER(0:7)	Number of children (0) and their PIDs (1:7)
	partner	INTEGER(7)	PIDs for partners to the children
	users	INTEGER	Number of active faces currently using node
	sing	INTEGER	Is this node a singularity - binary
	projdone	INTEGER	Has surface projection been performed - binary
	prev	POINTER	Pointer to the previous node in the linked list
	next	POINTER	Pointer to the next node in the linked list

Table A.1: Node Derived Datatype

	Name	Type (Dimension)	Description
	GID	INTEGER	Face's global number
	PID	INTEGER	Face's local number (in derived datatype array)
	ID	INTEGER	Face's location in 1-D array
	ID_ghost	INTEGER	Ghost cell's location in 1-D array
•	cent	REAL(3)	Face centroid location
•	norm	REAL(3)	Face normal vector
•	tan1	REAL(3)	Face tangent vector
•	tan2	REAL(3)	Face normal vector; normal to norm and tan1
•	ifn	INTEGER(4)	PIDs for four bounding nodes
•	ife	INTEGER(2)	PIDS for neighboring cells i and ii
	gfe	INTEGER(2)	GIDs for neighboring cells i and ii
	rfe	INTEGER(2)	Rank that owns neighboring cells i and ii
•	ihop	INTEGER(2)	PIDs for second-neighbors ih and iih
	ghop	INTEGER(2)	GIDs for second-neighbors ih and iih
•	dhopv	REAL(3,2,2)	(1:3,1,1) Vector from face centroid to cell i (1:3,2,1) Vector from face centroid to cell ih (1:3,1,2) Vector from face centroid to cell ii (1:3,2,2) Vector from face centroid to cell iih
•	bc	INTEGER	Face boundary condition type
•	bcopt	INTEGER	Optional value used for certain boundary conditions
	ibcID	INTEGER	Location of additional BC information in BC linked list
	glvl	INTEGER	Face's grid level
	parent	INTEGER	Parent of the face
	child	INTEGER(0:4)	Number of children (0) and their PIDs (1:4)
	blanked	INTEGER	Entry is uninitialized (-1), active (0), or blanked (1)
	prev	POINTER	Pointer to the previous face in the linked list
	next	POINTER	Pointer to the next face in the linked list

Table A.2: Face Derived Datatype

	Name	Type (Dimension)	Description
	GID	INTEGER	Cell's global number
	PID	INTEGER	Cell's local number (in derived datatype array)
	ID	INTEGER	Cell's location in 1-D array
•	cent	REAL(3)	Cell centroid location
•	vol	REAL(2)	Cell's volume (1) and its inverse (2)
•	len_scale	REAL(2)	Cell's minimum edge length (1) and its inverse (2)
•	dw	REAL	Distance from centroid to nearest wall
•	ien	INTEGER(8)	PIDs for eight bounding nodes
	ief	INTEGER(6)	PIDS for six bounding faces
	glvl	INTEGER	Cell's grid level
	parent	INTEGER	Parent of the cell
	child	INTEGER(0:8)	Number of children (0) and their PIDs (1:8)
	blanked	INTEGER	Entry is uninitialized (-1), active (0), or blanked (1)
	iref_dof	INTEGER	Identified for the number of degrees of freedom cell has for refinement
	coarsen	INTEGER	Previous number of attempts to coarsen cell
	irefine	INTEGER	Most recent grid level determined for this cell during AMR call
•	q	POINTER	Pointer to type containing solution primitives
•	q_n	POINTER	Pointer to type containing solution primitives - previous time level
	prev	POINTER	Pointer to the previous face in the linked list
	next	POINTER	Pointer to the next face in the linked list

Table A.3: Cell Derived Datatype

Name	Type (Dimension)	Description
GID	INTEGER	Node's global number
xcn	REAL(3)	Current node location
xco	REAL(3)	Node's original location
glvl	INTEGER	Node's grid level
parent	INTEGER(2)	Parents of the node (GID)
child	INTEGER(0:7)	Number of children (0) and their GIDs (1:7)
partner	INTEGER(7)	GIDs for partners to the children
sing	INTEGER	Is this node a singularity - binary
projdone	INTEGER	Has surface projection been performed - binary

Table A.4: Node Portal Derived Datatype

As was mentioned in Sec. 3.4.2, additional derived datatypes were used when passing information during repartitioning events. Tables A.4, A.5, and A.6 identify these datatypes. The included values were selected to be the minimum required to represent the current grid and solution with the remainder of the values in Tables A.1, A.2, and A.3 able to be easily recalculated on the destination rank after the MPI send/recieve.

When adding parent elements to the portals, all children are added immediately after the parent and then removed from memory. The references between them are converted to global identifiers (GID) in order to enable un-packing and reconnection via local number (PID) at the destination rank. Blanking information is not required for the cells of faces - if they have children, then they are marked as blanked.

In addition to repartitions, the portals are used when reading or writing restart files. When writing a restart file, the portals as allocated in order to include all elements and filled. Reading a restart files is nearly identical to initializing the domain after a repartition.

Name	Type (Dimension)	Description
GID	INTEGER	Face's global number
ifn	INTEGER(4)	GIDs for four bounding nodes
ife	INTEGER(2)	GIDS for neighboring cells i and ii
rfe	INTEGER(2)	Rank that owns neighboring cells i and ii
bc	INTEGER	Face boundary condition type
bcopt	INTEGER	Optional value used for certain boundary conditions
ibcID	INTEGER	Location of additional BC information in BC linked list
glvl	INTEGER	Face's grid level
parent	INTEGER	Parent of the face (GID)
child	INTEGER(0:4)	Number of children (0) and their GIDs (1:4)

Table A.5: Face Portal Derived Datatype

Name	Type (Dimension)	Description
GID	INTEGER	Cell's global number
glvl	INTEGER	Cell's grid level
ief	INTEGER(6)	GIDs for six bounding faces
parent	INTEGER	Parent of the cell (GID)
child	INTEGER(0:8)	Number of children (0) and their GIDs (1:8)
ncoarsen	INTEGER	Previous number of designations to coarsen cell
q	POINTER	Pointer to type containing solution primitives
q_n	POINTER	Pointer to type containing solution primitives - previous time level

Table A.6: Cell Portal Derived Datatype