Parallelization of Unsteady Adaptive Mesh Refinement for Unstructured Navier-Stokes Solvers

Alan M. Schwing *

Applied Aeroscience and CFD Branch
NASA Johnson Space Center, Houston, TX, 77058

Ioannis Nompelis † and Graham V. Candler ‡

Department of Aerospace Engineering and Mechanics
University of Minnesota, Minneapolis, MN, 55455

This paper explores the implementation of MPI parallelization in an unstructured Navier-Stokes solver. It uses dynamic adaptive mesh refinement of hexahedral cells to increase grid density in regions with strong gradients. Implicit and explicit time advancement methods are considered. Distributed implementation of the Data-Parallel Line Relaxation implicit operator is discussed for grids with hanging nodes. Parallel performance of simulations for an unsteady, inviscid flow are examined for both adapted and unadapted meshes in order to provide a baseline for comparison. Relative costs for adaptation and time stepping provide insight into computational bottlenecks. The flow solver and methods presented here are validated with data from a double cone experiment in hypersonic flow. For a given level of accuracy, adapted grids provide predictions that are less expensive than those obtained on unadapted grids for this staple test problem. Unsteady adaptation provides considerable savings for all problems considered.

I. Introduction

Computational fluid dynamics (CFD) calculations over geometries relevant to the design of complex aerodynamic shapes typically involve a large range of length scales. Many flow features are much smaller than the body length, and it is necessary to resolve these features with localized regions of fine grid spacings in order to obtain accurate results for aerodynamic or aerothermodynamic predictions. To accommodate these demands, grid resolution studies help determine the minimum number of grid elements necessary to resolve small-scale features without compromising solution quality. This can reduce computational cost, but comes at the expense of the engineer or researcher’s time.

In order to maintain good grid quality, constraints on grid spacing, stretching ratios, and cell aspect ratio drive many aspects of grid generation. The user is then left with competing requirements:

- Generate a grid that is sufficiently fine to capture necessary small-scale structures in regions where they are important.
- Where small structures do not exist, use coarse cells in order to limit computational expense.
- Maintain grid quality and acceptable stretching ratios throughout the domain.

This typically results in a number of grid design decisions that may compromise solution fidelity or require an exorbitant amount of an engineer’s time to create.

*Engineer, AIAA Student Member.
†Research Associate, AIAA Senior Member.
‡Professor, AIAA Fellow
Adaptive Mesh Refinement (AMR) is an approach that can alleviate some of these constraints. Using AMR, an inexpensive, coarse grid may be generated over the entire geometry. An appropriately instrumented flow solver can then selectively refine portions of the grid based on user-defined parameters. The code presented here uses an unstructured grid and solver. By using an unstructured framework, we allow individual cells to be modified independently of others. In contrast to block-based refinement, this requires additional memory overhead, but affords the solver additional flexibility and more precise adaptation.

Numerical simulations of external aerodynamic flows at high Reynolds numbers present many challenges. Fluid near the body is highly influenced by the no-slip wall and creates a boundary layer with very large velocity gradients. In order to properly simulate flow fields under these conditions, it is imperative that these gradients are adequately resolved. Computational grids resolve these large gradients by clustering many cells in the near-wall region. Typically, the change in flow quantities are dominant in the off-body direction, yielding cells with very small heights and very large aspect ratios.

For viscous, supersonic and hypersonic problems, these resolution requirements present two main difficulties for efficient simulations. The first is that the large number of grid elements required make each iteration of the flow solve computationally expensive. This encourages researchers to take large time steps and seek efficient numerical techniques to converge the numerical simulation. The second main difficulty is that the small cell spacings at the wall create a very stiff problem and severely limit the maximum stable time step. To reduce the impacts of these effects, practitioners turn to implicit time advancement schemes.

The physics in the boundary layer region are strongly coupled in the off-body direction and do not show large amount of unsteadiness. Contiguous ‘lines’ of cells protruding from the body can be identified along which there are strong gradients in flow quantities. For these reasons, a method using implicit line solves is very suitable for aerodynamic problems. By using lines it is possible to build this method into a scalable, parallel computational code. Research at the University of Minnesota has proven the success of this type of implicit method, Data-Parallel Line Relaxation (DPLR).

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 can be complex and we discuss the specifics of our implementation not only for a fixed partition, but also for dynamic reallocation of the computational domain.

II. Motivational Problems

A model problem that presents many complex interactions seen in hypersonic flow is that of a double-cone geometry, shown schematically in Fig. 12. There are a number of inviscid and viscous flow structures that combine to create a challenging test case. The boundary layer and separation zone require proper handling of the viscous terms and create numerical stiffness that an implicit time advancement scheme must take into account. At certain freestream conditions and cone half-angles, the flow becomes unsteady and require proper resolution in time to capture these effects. The case examined in this paper is steady, but the flowfield does take considerable time to develop, so appropriate handling of transient terms is still important.

Double cones and double wedges are important because they can represent a combination of phenomena common in external fluid flows. A control surface or structural element on the body of a hypersonic vehicle generates shock waves. These protuberances also generate regions of recirculation or separated flow. In order to assess requirements on vehicles, accurate modeling of these influences must be achieved in a design environment sensitive to excessive runtimes. Using solution adapted mesh refinement allows simulation on lightweight grids that can be refined as necessary in order to combine accuracy and efficiency.

Another problem dominated by viscous phenomena are bluff body flows. Capsule designs are important for nearly all planetary reentry vehicles currently under development at NASA and aerospace companies. These flows frequently involve unsteady turbulent structures that include a wide range of length scales. Separation location varies with flight condition and is critical to predicting the surface pressures on the body and the location can be influenced by grid refinement. Targeted refinement with implicit AMR can allow greater fidelity at reduced expense for this problem.

At higher Reynolds numbers, the range of length scales increases. For that reason, very fine spacing is required in portions of the flow to capture the smallest relevant feature. Using uniform grids to capture all
of the wake structures can result in a volume of refinement that is very large. Unsteadiness of the wake and
turbulent phenomena drive onerous grid requirements. Sufficient resolution must be achieved for small scale
features throughout the cycles of unsteadiness.

III. Previous Work

A. Adaptive Mesh Refinement

Adaptive Mesh Refinement is not a new technique and was pioneered by a number of authors over many
years, beginning with the work of Berger, Oliger, and Colella. 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. 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
appear to dovetail nicely into the underlying numerical theory and are well represented in the literature.

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 where sufficient grid
density already existed. Though it is more difficult to implement, anisotropic refinement allows for unequal
subdivision in the primary cell directions. Anisotropic subdivision can yield substantial benefits in terms
of grid size and resulting cost provided that the grid is aligned with the features of interest.

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. 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.

Specific to this work are approaches for handling hypersonic flows in an adaptive framework. The
now-classic problem of the double-cone in hypersonic flow is examined in the following sections. Previous
researchers have shown strong comparisons to test data when using adapted grids with hanging nodes. An active area of research that involves similar flow structures in hypersonic aerodynamics is ramjet and
scramjet inlets. Recent work at relevant conditions on these geometries with multi-resolution-based AMR
has shown compelling results.

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. 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.

B. Implicit Time Advancement Methods

When investigating the viscous Navier-Stokes equations, there are strenuous demands placed on the implicit
solver. As has been shown for unadapted grids, the choice of implicit method strongly influences the conver-
gence properties and robustness of the resulting Navier-Stokes solver. Studies of numerical results included
FMPR and DPLR 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. These methods are parallel and provide a robust
capability for implicit time advancement. Using the PETSc library, researchers can leverage existing modular
Instead of using the PETSc libraries, we choose to implement the Data-Parallel Line Relaxation (DPLR) implicit method. To our understanding, application of the method to adapted grids with hanging nodes has not been performed. The DPLR 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 overhead as compared to the PETSc libraries by using data structures already in the flow solver.\footnote{21}

Our previous work with implicit methods and AMR employed Full-Matrix Point-Relaxation (FMPR).\footnote{20} This unbiased, iterative implicit solve showed promise on simulations involving inviscid, steady-state hypersonic flow. Results obtained by solving the Euler equations using grids created with AMR performed comparable to those created with uniform cell distributions. This work continues to employ FMPR in the domain far from viscous walls and DPLR in the near-wall regions.

IV. Computational Methodology

A. Adaptive Mesh Refinement

The authors are currently working to develop a robust, AMR capability within a finite-volume framework for use with hypersonic, reacting, and turbulent flows. This work builds on the underlying assumptions and methods outlined previously.\footnote{20} Portions of the strategy and improvements to it are briefly summarized here.

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 as shocks, which frequently accompany hypersonic flows - and have been shown to more precisely predict vehicle surface quantities.\footnote{23, 24} Additionally, hexahedral shapes provide face connectivity structures that enable larger computational stencils. These stencils are required for the calculation of high-order numerical fluxes.

We constrain our grids to include only hexahedral cells, but the flexibility inherent to the finite-volume method provides the underlying grid with many degrees of freedom. The unstructured grid allows for complex grid topologies on the surface and in the domain volume. Furthermore, the method allows hanging nodes without compromising solution integrity, even with the stencils required for the high-order flux evaluations.

Our current approach uses isotropic refinement in each of the solution directions such that grid density in a parent cell is doubled along each edge.

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 in the domain. When a cell or face is refined, its children 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.

Research by many authors has identified a large number of possible refinement criteria. For the purposes of understanding scalability and performance, the choice of refinement criteria is not a primary concern. Our current work uses two basic refinement criteria. The first is an undivided difference of flow variables (density, velocity, pressure, and/or temperature). Undivided differences inform the solver that there is a sufficiently large change in relevant quantities between adjacent cells to merit subdivision. These differences indicate that there is a variation or discontinuity that requires additional spatial resolution.

Equation 1 shows the undivided difference used for refinement of a flow variable, $\phi$, based on the tolerance, $\phi_{tol}$. 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 this work normalizes the difference to the local minimum in the flow variable $\phi$.

$$\phi_{tot} < \frac{|\phi_i - \phi_{i1}|}{\min(\phi_i, \phi_{i1})} \quad (1)$$

Another method used in the following results is a sensor based on the gradient of a flow variable. Shown in Eq. 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 $\phi_{tol}$ is determined.

$$\phi_{tol} < |\nabla \phi|$$ \hspace{1cm} (2)

Our adaptive refinement incorporates ‘buffer cells’. Regions that are flagged for refinement based on the criteria in Eq. 1 or 2 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 at which grid quality is assessed. As will be shown later, excessive buffer cells can reduce the overhead associated with AMR, but comes at the cost of increased grid size.

Since we intend our methods to be applicable to unsteady simulations, grid coarsening is important to remove refinement when it is no longer required. Cells are coarsened when all of their children have no children of their own, have not been flagged for refinement, and are not required to satisfy the requirement for buffer cells. Before the cell is actually removed, the restricted solution value on the parent cell is compared to the refinement criteria. If the restricted value would not trigger refinement of the parent cell, then the child cells are removed. When coarsening cells or faces, the now unnecessary child cells or faces are dropped from memory. If they are needed later in the computation, they are rebuilt.

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. We also desire a method for adaptation that provides a smooth distribution of grid cells away from viscous boundaries and does not cause cell spacing discontinuities in the domain. We use a hyperbolic tangent stretching function away from viscous surfaces to accomplish this goal.\textsuperscript{25}

This method requires a $\Delta x$ at the viscous wall and a second $\Delta x$ in the domain far from the wall. After adaptation, the code traverses each line of nodes emanating from the surface and records a connectivity array. By noting the existing spacing on the far side of the line (opposite the viscous wall) and using a user-specified wall spacing on the other, $\Delta x$ values at both ends are known. 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 $2^{L_{line}-L_{fine}}$ where $L_{fine}$ is the finest grid level prescribed by the user and $L_{line}$ is the current level of nodes in the line.

**B. Finite-Volume Navier-Stokes Solver**

We solve the compressible Navier-Stokes equations with a finite-volume scheme. The compressible Navier-Stokes equations can be written in conservation law form:

$$\frac{\partial \bar{U}}{\partial t} + \nabla \cdot (\bar{F}_c - \bar{F}_v) = 0$$

where $U = (\rho, \rho u, \rho v, \rho w, E)^T$ is the array of conserved variables and $\bar{F}_c$ and $\bar{F}_v$ are the convective and viscous fluxes, respectively. $\rho$ is the density, $\rho u$, $\rho v$, and $\rho w$ are the three-dimensional components of momentum, and $E$ is the total energy per unit volume.

We solve a weak form of the conservation equation and compute cell-averaged values ($\bar{U}$) using numerical time integration from an initial condition. One useful feature of the finite volume formulation is that it allows for arbitrary polyhedra with a fixed volume and arbitrary number of bounding faces. The numerical fluxes are calculated at each of the faces and application of the divergence theorem over one such computational cell yields a discrete representation of the governing equations. Our treatment of the equations will divide the total flux into its viscous and inviscid components:

$$V \frac{\partial \bar{U}}{\partial t} + \sum_{faces} \left[ \bar{F}_c \cdot \hat{n}S \right] - \sum_{faces} \left[ \bar{F}_v \cdot \hat{n}S \right] = 0 \hspace{1cm} \text{or} \hspace{1cm} \frac{\partial \bar{U}}{\partial t} = -\frac{1}{V} \sum_{faces} \left[ \bar{F}_c \cdot \hat{n}S \right] + \sum_{faces} \left[ \bar{F}_v \cdot \hat{n}S \right]$$

with $V$ being the cell volume, $S$ the face area, and $\hat{n}$ the outward pointing unit normal to the face. This form of the governing equations lends itself to unstructured grids. As long as sufficient connectivity exists that link cells to their surrounding faces or vice-versa, it is agnostic to ordering.

For the inviscid fluxes, a useful deconstruction of the face fluxes represents $F^2_c$ as $\bar{F} = \bar{F}_- + \bar{F}_+$ where $\bar{F}_-$ and $\bar{F}_+$ are upwinded components of the inviscid flux in the direction of the positive and negative running...
eigenvalues. With outward pointing normals, as in the equation above, all of the upwinded fluxes \( \vec{F}_+ \) and \( \vec{F}_- \) depend on cell-averaged quantities in the current cell and its neighbors, respectively. The discrete equation representing a first-order accurate method can be written as:

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

Higher-order methods can be developed by augmenting the numerical stencil used to generate \( \vec{F}_+ \) and \( \vec{F}_- \) or by changing the flux construction entirely.

The viscous fluxes are computed at each face, which requires calculation of the spatial derivatives at the face centroids. Our approach uses a weighted least squares method to calculate the gradients at each cell center. Similar to the work described by Nompelis et al., these cell-centered gradients are averaged at the faces and improved with application of deferred correction.\textsuperscript{26,27} For grids with hanging-nodes, this approach is unaltered.

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 all calculated at time level \( n \). This method can be written as:

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

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

### C. Data-Parallel Line Relaxation

Explicit numerical methods are limited in their maximum allowable stable time step by the Courant-Friedrichs-Lewy (CFL) condition. For viscous grids that have very fine near-wall grid spacing, the stable time step is very small and makes many problems intractable. Our motivational problems involve hypersonic 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, we turn to implicit methods.

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:

\[
F_c^{n+1} = F_c^n + \frac{\Delta t}{V} \bar{F}_c^n \left( \bar{U}^{n+1} - \bar{U}^n \right)
\]

\[
= F_c^n + A^n_c \left( \bar{U}_i^{n+1} - \bar{U}_i^n \right) + A^n_+ \left( \bar{U}_{ii}^{n+1} - \bar{U}_{ii}^n \right)
\]

where \( A^n_c \) and \( A^n_+ \) are the right- and left-running flux Jacobians at the face. \( \delta \bar{U}_i \) 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 the face neighbor.

The viscous fluxes are also approximated at the future time level \( n + 1 \). To do this, we introduce an approximate viscous Jacobian, \( \bar{M} \), 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:

\[
F_v^{n+1} = F_v^n + \frac{\Delta t}{V} \bar{F}_v^n + (R^{-1}MR) \frac{\partial}{\partial \eta} \left( N^\eta \delta \bar{U}_i^n \right)
\]

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:

\[
\delta \bar{U}_i + \frac{\Delta t}{V} \sum_{\text{faces}} \left[ (A^n_c \delta \bar{U}_i + A^n_+ \delta \bar{U}_{ii} + (R^{-1}MR) N (\delta \bar{U}_i - \delta \bar{U}_{ii}) ) \cdot \hat{n}S \right] = \Delta \bar{U}_i
\]

By storing the Jacobian matrices and performing an iterative solution procedure, the values for \( \delta \bar{U}_i \) are obtained such that \( \Delta \bar{U}_i \) is known.
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. Both methods are amenable to distributed computation.

For our application with adapted grids and hanging nodes, the basic strategy for DPLR is augmented simply by only considering cells that are of the same level of refinement as the faces at viscous surfaces. Figure 1 has a simple illustration showing refinement of near-body cells and its effect on the implicit lines. An initial grid without any refinement, Fig. 1(a), grows DPLR 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.

Once the near-body grid has been adapted, the DPLR lines only extend so far as there is an unbroken string of cells at the same grid level as the viscous boundary faces. Figures 1(b) and (c) demonstrate a successively refined mesh and the resulting DPLR lines for grids with hanging nodes. The lines do not extend to the refined region in the top-left corner because there is not an unbroken group of refined cells from the viscous wall.

Figure 1. Illustration of DPLR lines (gray) away from viscous boundary for an initial grid and after some refinement.

D. Distributed Computation

The goal of distributed computation is to spread the problem across a large number of computation cores, or ranks, to more quickly iterate. Doing this incurs fixed costs for communication between the ranks and it is important to structure the code in such a way to minimize the cost of communication. Furthermore, 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. Our flow solver uses the standard OpenMPI library in order to achieve distributed computation.

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 each rank and frequent communication between the two processes. MPI communication as implemented in our code predominantly uses non-blocking send and receive calls such that numerical work is performed as the 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.

When constructing the list of fringe faces, our 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. For this reason, we use ParMETIS to derive an appropriate loading for each of the ranks in the computation and rebalance the grid as indicated. When using unadapted grids, this rebalancing is done once at the beginning of the run. Grids generated using AMR provide unique challenges in this regard.

AMR can quickly cause previously load-balanced simulations to fall out of balance. This is mainly due to unsteady or transient phenomena or to a naive initial partitioning. 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 AdaptiveRepout routine. Initial partitions and adaptive partitions are refined with successive calls to RefineKway.

Our strategy for load balancing and redistribution operates on the level zero cells in the mesh. By operating on the coarsest cells in the mesh, we ensure that a cell and all of its descendants are colocated 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.

The procedure builds the partition graph of level zero cells and weights each cell by the number of children that it has (for adapted grids). It also provides edge weightings for each level zero face equal to the number of children that the face has. 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 DPLR line are partitioned together. This is easily accomplished by collapsing each line into a single element and appropriately weighting the cells and faces.

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, dependent on measured runtimes, or a relevant 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 a fixed repartition frequency in order to provide a controlled environment for study of the algorithm.

E. Derived Datatypes

For the nodes, faces, and cells, our code uses FORTRAN’s derived datatypes in order to avoid a large number of arrays that must be updated or increased during adaptation. Furthermore, these datatypes are combined into linked lists. The flexibility inherent to linked lists allows for much simpler management of the dynamic mesh and elements can be easily dropped or added as necessary.

Our code uses a dual datatype approach. Linked lists of the 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 one-dimensional arrays.

Initially, all computational data are 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 this purpose. The portions of the code responsible for time stepping, gradient calculation, and I/O depend on these arrays. Before an adaptation event or repartitioning, the code stores all flow variables in the linked lists and deallocates the 1-D arrays. Following the adaptation or repartitioning, 1-D arrays are once again allocated and updated with the most current information from the linked lists.

The most immediate issue with this strategy is that it requires much 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 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 eliminate the savings afforded by linear loops, but can be designed to eliminate reallocation events.

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 rewrite of the current software. Understanding the performance of this approach can validate its approach (or similar) for those who might abstract it in this fashion.

V. Parallel Performance

A. 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.\textsuperscript{30,31}

The initial conditions for the density pulse are shown in Eq. 3 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.

\begin{equation}
\begin{aligned}
\rho &= 1.0 + \frac{1}{10} \exp\left(-\frac{(\bar{r} - 5.0)^2}{2}\right) \\
\rho \mathbf{v} &= 0.0 \\
\rho \mathbf{w} &= 0.0 \\
\end{aligned}
\end{equation}

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\textsuperscript{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 2 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 $r_1/r_N$ for a case with N-ranks. The left-hand plot, Fig. 2(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. 2(b). Shown is the parallel efficiency versus the number of processors for the same set of cases. Parallel efficiency is calculated by multiplying the speedup by the number of ranks, $N r_1/r_N$ 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. 3. 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 2. Parallel performance for uniform, unadapted grids. Work is consistent across all grids at a given size.

Figure 3. Parallel performance for uniform, unadapted grids as a function of cells per rank.

Figure 4 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 undated grids, it can be interpreted as a percentage of total runtime.

To orient the reader for this Fig. 4 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 $\Delta t$ 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
Figure 4. 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.

scalability in Fig. 2, the 250,000 and 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.

B. 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 convection 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 \( \rho \) 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 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 5 shows a cut through the volume at the initial condition highlighting the extent of adaptive refinement.

For this problem, there are three important parameters to consider. The first is the refinement criterion. A \( \rho \)-based criterion with \( 1\times10^{-8} \) is used (see Eq. 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 6 shows results from a scalability study performed with adapted grids. Speedup and parallel efficiency are calculated as described for Fig. 2. These data represent our 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. 6 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 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. 7(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. 7(a) suggest imperfect partitioning by ParMETIS. 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. 6.

There are diminishing returns with increasing levels of AMR for this problem. Figure 7(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 7(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 8(a) presents parallel performance but only includes the time required for 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. 6(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 8. Parallel performance of iterating, gradient calculation, and communication on adapted grids and unadapted grids.

Similar to the plots shown for the unadapted grids, Fig. 9 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, 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.

Figure 9. 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.

Results in Fig. 9 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 our 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. 10 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 10(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. 10(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 10(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.

![Graphs showing runtime comparison for adapted grids and unadapted grid.](image)

Looking a bit closer to the costs associated with AMR and redistribution for the strategies listed above, Fig. 11 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 codes 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. 11(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.
Figure 11. Runtime in major sections of flow solver for adapted grids relative to the cost of time stepping for 400 iterations.

VI. Hypersonic Double Cone

In addition to analytical and theoretical problems, we focus in this section on results obtained using our parallel code to a standard problem for the hypersonic community, the hypersonic double cone. The experiment was conducted at CUBRC. Due to its strong applicability to the field and its history of numerical comparisons we consider it an important validation case. Researchers working on AMR or solution adapted grids have had success simulating these flowfields in the past.

The problem of the hypersonic biconic is a good test problem for use with AMR. Freestream conditions strongly influence the behavior of the flow. Ideally, a grid made for the problem would incorporate tailored boundaries that would conform to the final shock shapes. This can be an expensive process and involves several initial simulations of the flowfield. 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 can be 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.

We look at a common run from the CUBRC test data, Run 35. The flow conditions for the case are listed in Table 1.

The double cone model had a length of 0.18 meters with half angles of 25° and 55°. 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 will be referenced in the results that follow.
Table 1. Freestream conditions for Run 35.

Mach        11.30  [-]  
$\rho_\infty$  5.515E-4  [kg/m$^3$]  
$u_\infty$    2712.7  [m/s]  
$T_\infty$    138.89  [K]  
$T_{\text{wall}}$  296.11  [K]  

Figure 12 illustrates the flow features on a double cone and shows an image of density gradient magnitude from a numerical simulation.

Figure 12. Illustration of flow features and magnitude of density gradient for converged solution.

Our 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. Comparisons between the performance of unadapted and adapted grids are the focus of this work. By choosing not to model these physics, we accept 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, we select a modified Steger-Warming flux.

The hypersonic double code is a pseudo-steady problem. Flow separation in the corner where the two cones meet can take a significant amount of time to develop. Previous work has found that it requires on the order to 100 flow times to converge. 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.

For this problem, we rely on a coupled DPLR-FMPR implicit operator. Our line solves extend from the viscous wall into the domain and are truncated by coarser cells or the opposite boundary as outlined previously. 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. Grid Generation

Grid requirements are well understood for this double cone problem. Previous research has demonstrated that a grid of 524,288 cells (1024x512) is sufficient to resolve all relevant flow structure and properly predict the length of the separation region. Figure 13 presents the results from one such study.

![Grid requirements](image)

Figure 13. Illustration of grid convergence study on uniformly refined meshes.33

We performed a grid convergence study of our own (not shown) and 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.

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, or 2,048 cells in total. This limits the number of processors for which an appropriate parallel distribution can be found and will be discussed later.

Sensitivity to the viscous wall spacing was also examined. Our results indicated that an initial off-body spacing of 0.5\(\mu\)m was sufficient to capture the boundary layer and that further refinement was unnecessary. Both the adapted and unadapted grids use hyperbolic tangent 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 1024x512) that the viscous spacing reaches 0.5\(\mu\)m.

B. Adaptation Strategy

Unlike the density pulse, 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 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 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 \(|\nabla \rho| > 0.2 \text{ [kg/m}^4\text{]}\) identified cells in the shock regions and near the surface.16 We adopt an identical refinement sensor for this work (Eq. 2).

Figure 14 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.

![Figure 14. Image of adapted grids superimposed over magnitude of density gradient.](image)

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

C. Numerical Results

Figure 15 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.\(^\text{32}\)

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 of ranks is limited. Our problem is made slightly more difficult since DPLR lines must remain together.

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 16, 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 16(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. Figure 17 shows 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 pseudo-steady problem. There are well-identified flow structures and refinement based on flow features (|∇ρ| in our case) correctly capture regions with important flow features. The sparse use of grid points for initial computation while the flow develops saves a great deal of computational time.

### D. Parallel Performance

Figure 18 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. 6 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. 16. 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.

![Graphs showing speedup and parallel efficiency](image)

(a) Speedup  (b) Parallel Efficiency

Figure 18. Parallel performance for double cone simulations. Adapted and unadapted grids shown.

VII. Conclusions

Our results show important metrics for parallel performance with a Navier-Stokes solver using AMR. Unadapted grids have strong scalability and the underlying finite-volume method considered here effectively hides parallel communication except in cases with very low per-core node counts. Similar studies with adapted grids for an unsteady problem show a much reduced scalability. This is due to several factors:

- Grids generated using adaptive refinement generally contain fewer grid cells. Fewer cells incur a lower cost for local computation and increase the relative cost of communication and other fixed requirements.
- In our implementation, the grid cells are partitioned based on the coarsest grid. This can constrain the partitioning library and create imperfect load balance between cores manifesting as longer periods of idle time.
- Costs associated with memory management require a non-trivial amount of time. For the dual memory approach outlined here, with both linked-lists and arrays used, the cost does not scale and generally exceeds those associated with the refinement operation.
- Repartitioning incurs a relatively low cost, but determining an efficient partition can be expensive. This is a cost only associated with multi-core runs and reduces the efficiency by creating additional overhead as the number of ranks increase.

For the pseudo-steady double cone, the scaling was improved due to a larger cell count and reduced AMR frequency. However, mesh adaptation yields significant savings in absolute computational cost for all problems addressed in this paper. Opportunities for further improvement exist in addressing the factors mentioned previously, but the use of adaptive grids methods has potential to greatly reduce the expense of CFD solutions for complex aerodynamic flows. Adaptive grids also reduce the significant costs associated with grid generation as well. Proper grid tailoring and generation of a grid topology that efficiently clusters cells near small-scale features is expensive. AMR offloads much of this work to the compute cluster and the flow solver.

We also validated our numerical method against a classical problem in the hypersonic community. Predicted surface pressure and heat transfer compare well to the experimental data and are in family with previously published results from other researchers. Solutions obtained on grids that were adapted over the
course of the computation and included hanging nodes were identical to those computed on more traditional, unadapted grids.

VIII. Acknowledgments

The authors would like to thank Dr. Benjamin Kirk for his advice during this effort. This work was supported in part by the Department of Defense National Security Science and Engineering Faculty Fellowship.

References


