Abstract

BOND, RYAN BOMAR. A Conservative Interblock Communication Algorithm for Dynamically Discontinuous Multiblock Interface Grids. (Under the direction of Dr. D. Scott McRae.)

An algorithm is presented for the conservative communication between grid blocks where point connectivity is not maintained across across the interface. The algorithm is specifically designed for dynamic discontinuities, e.g. those that arise when dynamic r-refinement adaptation is performed on two adjacent blocks without the constraint of boundary point connectivity. r-Refinement adaptation involves moving the points in the computational mesh to better resolve features in the solution field. Discontinuously connected multiblock boundaries are necessary for problems where adjacent blocks have significantly different adaptation requirements or problems where one block is to be adapted while an adjacent block is left stationary. In programs where adaptation is being performed in parallel, discontinuously connected multiblock boundaries allow the adaptation to continue on different processors without the need for communication and subsequent agreement of where the boundary points should be located.

The communication between two adjacent blocks must be conservative to allow the simulation of compressible flow problems. Simply interpolating the boundary data to transfer information from one block to the other violates conservation, so other means of transferring the information must be employed. A technique for generating ghost cells and setting their dependent variable values is presented. The method is conservative, accurate, robust, and computationally efficient.

Tests of the algorithm on both statically and dynamically discontinuous grids are presented, and the computational efficiency of the algorithm is discussed. The algorithm was tested using the diffusion equation, linear wave equation, and the set of Euler equations. Since the algorithm is applicable to any CFD problem where conservative communication is necessary between discontinuous blocks, it can be used when adaptation criteria are different between adjacent blocks, when an adapted block is adjacent to an unadapted block, or when enhanced parallelism can be achieved by relaxing boundary point connectivity.

A Conservative Interblock Communication Algorithm for Dynamically Discontinuous Multiblock Interface Grids

by

Ryan Bomar Bond

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Department of Mechanical and Aerospace Engineering

Raleigh, North Carolina
July 10, 2001

Approved By:

Dr. J. R. Edwards	Dr. H. A. Hassan
Dr. R. E. White	Dr. D. S. McRae, Chair

Biography

Ryan Bomar Bond was born in Fredericksburg, Virginia on January 6, 1976 to David and Celia Bond. Raised in Tullahoma, Tennessee, he became a Christian in 1988 and graduated from Tullahoma High School in 1994. He later received Bachelor of Science degrees in Aerospace Engineering and Mathematics from Mississippi State University in 1998. On May 30, 1998, Ryan married Rucha Sandip Shukla. Rucha is currently pursuing her Doctor of Pharmacy degree at the University of North Carolina at Chapel Hill. Ryan and Rucha currently reside in Durham, North Carolina.

Acknowledgments

I would like to thank my loving wife, Rucha, who has been supportive of me and my work throughout both my undergraduate and graduate educations. I would also like to thank my parents who guided me through my first steps of education, faith, and life. I thank my advisor, Dr. D. Scott McRae, who has guided my academic and professional development for the past three years. I would also like to thank Brent Bates and Bobby Nichols, two men who have influenced my career path more so than anyone else. Thanks also to everyone in 4216 Broughton Hall—my education would not have been complete had we not shared it together.

Contents

List of Tables

List of Figures

List of Symbols

Roman symbols:

A	area, or Jacobian matrix of Euler equations $\frac{\partial \mathbf{F}}{\partial \mathbf{U}}$
$\mathcal{A}$	overlap area
a	sound speed
$\vec{c}$	wave speed and direction for linear wave equation
dV, dS	elemental volume and area
E_t	total energy
$\mathbf{F}$	inviscid ξ or x direction flux vector
$\mathbf{G}$	inviscid η or y direction flux vector
$\mathbf{H}$	inviscid ζ or z direction flux vector
$\bar{h}$	estimate of average cell height used in communication
H	total enthalpy
i, j, k	indices corresponding to ξ, η, and ζ directions
$\hat{i}, \hat{j}, \hat{k}$	Cartesian unit vectors
$I_{2\to1}, I_{1\to2}$	transfer operators for interblock communication
J	Jacobian of coordinate transformation
k	diffusivity
l, m, n	dummy indices
M	Mach number
$\hat{n}$	unit vector normal to a surface
p	pressure
P	polygon
$\mathbf{Q}$	source term vector
$\mathcal{R}$	universal gas constant
s	side
t	time
T, T^{-1}	matrices of left and right eigenvectors of Euler set
u, v, w	Cartesian velocities
u	dependent variable in diffusion and wave equations
$\mathcal{U}, \mathcal{V}, \mathcal{W}$	contravariant velocities
$\mathbf{U}$	vector of conserved variables
$\vec{v}$	velocity vector
$\vec{v_b}$	velocity of boundary element of control volume
$\mathbf{V}$	vector of primitive variables
V	grid cell volume
W	weight function
x, y, z	Cartesian coordinates

Greek symbols:

α_1, α_2	parameters in average cell height estimation
β_1, β_2	parameters in average cell height estimation
γ	ratio of specific heats
$\delta, \delta^+, \delta^-$	central, forward, and backward difference operators
ϵ	small number, error in Newton's method, or term in entropy fix
ϵ_{coinc}	distance for determining coincidence in Ramshaw algorithm
ϵ_s^P	sign of area contribution in Ramshaw algorithm
κ	constant in MUSCL extrapolation
λ	step coefficient in linesearch of Newton's method
Λ	diagonal matrix of eigenvalues for the Euler set
Λ, Ψ	parameters in conservative limiter for SIERRA
ξ, η, ζ	curvilinear coordinates
ρ	density
ω	volume weighting in weight function

1 Introduction

The process of redistributing a fixed number of nodes in a computational grid to improve resolution of a numerical solution is known as r-refinement. This process can be used to adapt a grid dynamically to a changing numerical solution by increasing grid node density in areas of high estimated truncation error. The number of nodes in the mesh remains constant during adaptation. Additionally, r-refinement tends to align grid lines with solution features such as shock waves or other discontinuities. When this happens, Riemann type solvers are applied in the directions relative to solution features that lead to highest accuracy.[1]

To date, r-refinement adaptation algorithms on multiblock grids have been employed with the constraint that the interblock boundaries remain continuously connected. This constraint introduces several problems:

1. When adaptation is employed on one grid block, all adjacent grid blocks must also be adapted in order to maintain the continuous connectivity at the interblock boundaries.

2. The adaptation of one grid block to its associated solution is impeded by the constraint of boundary point connectivity.

3. The degree of coarse-grained parallelization in the grid node movement process is reduced. This is because the processors which are adapting the grid blocks must communicate with each other at every iteration of the node movement algorithm in order to maintain connectivity.

4. Opportunities for acceleration using domain decomposition techniques are subject to connectivity restrictions.

Employing discontinuously connected multiblock boundaries resolves these is-

sues, but it requires an interblock communication algorithm. For problems in computational fluid dynamics, the communication algorithm must be efficient to allow for frequent movement of the grid; it must be conservative to allow computation of compressible flow problems, and it must be accurate to minimize errors introduced at the interblock boundary.

Many methods have been developed which do not attempt to achieve conservation. Most of the better non-conservative methods were developed for chimera (or overset) grid techniques.[2] While these methods achieve efficiency without conservation, other methods achieve conservation without efficiency. In the past, conservative interblock communication algorithms have not focused on the efficiency criterion because efficiency is of lesser importance for static grids than for dynamic grids. Some of the better methods leave a gap between the structured grid blocks which is later filled with an unstructured block.[3] These types of methods, however, have not been shown to be practical for moving grids. This is because the unstructured block would have to be regenerated and its dependent variable values recomputed each time the structured blocks were adapted. The current work seeks to develop an efficient algorithm which can provide conservative interblock communication for discontinuously connected, dynamic multiblock grids.

Section 2 gives the governing equations for the physical laws used to test the algorithm: the diffusion equation, the linear wave equation, and the Euler equations. Section 3 gives and introduction to r-refinement adaptation and the details of the SIERRA algorithm. In Section 4, the communication algorithm is described in detail. Section 5 gives the results of tests on the communication algorithm using both static and dynamic grids and the three physical models from Section 2. Sections 6 and 7 give the conclusions and future work, respectively.

2 Governing Equations

Three different physical models were used in the testing of the communication algorithm: the diffusion equation, the linear wave equation, and the set of Euler equations coupled with the ideal gas law. Each of these models was cast in a finite volume form on curvilinear coordinates.

2.1 The Diffusion Equation

The diffusion equation governs the diffusion of a solute through a stationary solvent. In three-dimensions, the equation is given by

$$u_t = k(u_{xx} + u_{yy} + u_{zz}). \tag{1}$$

2.2 The Linear Wave Equation

The linear wave equation governs the transport of some quantity along a vector $\vec{c}$, where $\|\vec{c}\|_2$ is the wave speed. In three-dimensions, the equation is given by

$$u_t = (\vec{c} \cdot \hat{i})u_x + (\vec{c} \cdot \hat{j})u_y + (\vec{c} \cdot \hat{k})u_z. \tag{2}$$

2.3 The Euler Equations and Ideal Gas Law

2.3.1 The Euler Equations

The Euler equations govern the motion of an inviscid fluid. In Cartesian coordinates with no body forces or external heat addition, the equations are as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} + \frac{\partial \mathbf{H}}{\partial z} = 0 \tag{3}$$

$$\mathbf{U} = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E_t \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (E_t + p)u \end{bmatrix} \mathbf{G} = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (E_t + p)v \end{bmatrix} \mathbf{H} = \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (E_t + p)w \end{bmatrix}.$$

2.3.2 The Ideal Gas Law

The Euler set of 5 equations contains 6 unknowns (ρ, u, v, w, p, E_t), so a sixth equation must be added to close the system. Equations which close this system come from thermodynamics and are known as equations of state. The simplest equation of state is the ideal gas law:

$$p = (\gamma - 1)(E_t - \frac{1}{2}(u^2 + v^2 + w^2)). \tag{5}$$

2.3.3 Roe's Method for Solving the Euler Equations

The Euler equations were implemented using Roe's method with the `minmod` limiter.[4][5] The equation set, variable vector, and fluxes are defined as follows:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial \xi} + \frac{\partial \mathbf{G}}{\partial \eta} + \frac{\partial \mathbf{H}}{\partial \zeta} = 0 \tag{6}$$

$$\mathbf{U} = \frac{1}{J} \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E_t \end{bmatrix}$$

$$\mathbf{F} = \frac{1}{J} \begin{bmatrix} \rho \mathcal{U} \\ \rho \mathcal{U} u + \xi_x p \\ \rho \mathcal{U} v + \xi_y p \\ \rho \mathcal{U} w + \xi_z p \\ (E_t + p)\mathcal{U} \end{bmatrix} \mathbf{G} = \frac{1}{J} \begin{bmatrix} \rho \mathcal{V} \\ \rho \mathcal{V} u + \eta_x p \\ \rho \mathcal{V} v + \eta_y p \\ \rho \mathcal{V} w + \eta_z p \\ (E_t + p)\mathcal{V} \end{bmatrix} \mathbf{H} = \frac{1}{J} \begin{bmatrix} \rho \mathcal{W} \\ \rho \mathcal{W} u + \zeta_x p \\ \rho \mathcal{W} v + \zeta_y p \\ \rho \mathcal{W} w + \zeta_z p \\ (E_t + p)\mathcal{W} \end{bmatrix}.$$

The contravariant velocity components are

$$\begin{aligned} \mathcal{U} &\equiv \xi_x u + \xi_y v + \xi_z w \\ \mathcal{V} &\equiv \eta_x u + \eta_y v + \eta_z w \\ \mathcal{W} &\equiv \zeta_x u + \zeta_y v + \zeta_z w. \end{aligned} \tag{8}$$

Equation 5, the ideal gas law, can be recast as:

$$\rho H = \frac{\gamma p}{\gamma - 1} + \frac{1}{2}\rho(u^2 + v^2 + w^2). \tag{9}$$

When Roe's upwind-biased flux-difference splitting is employed, the cell interface fluxes are given by

$$\begin{aligned}
\mathbf{F}_{i+1/2} &= \frac{1}{2}[\mathbf{F}(\mathbf{U}_L) + \mathbf{F}(\mathbf{U}_R) - |A|(\mathbf{U}_R - \mathbf{U}_L)]_{i+1/2} \\
A &= \frac{\partial \mathbf{F}}{\partial \mathbf{U}} \\
&= T\Lambda T^{-1} \\
&= T(\Lambda^+ + \Lambda^-)T^{-1} \\
\Lambda^{\pm} &= \frac{1}{2}(\Lambda \pm |\Lambda|) \\
|A| &= T|\Lambda|T^{-1}.
\end{aligned} \tag{10}$$

The matrix Λ is the diagonal matrix of eigenvalues of A. T and T^{-1} are the matrices of left and right eigenvectors, respectively. The subscripts L and R are used to denote the left and right interface states given by an appropriate interpolation. Higher order accuracy is sought by interpolating the dependent primitive variable values to the cell interfaces. The vector of dependent primitive variables is given by

$$\mathbf{V} = \begin{bmatrix} \rho \\ u \\ v \\ w \\ p \end{bmatrix}. \tag{11}$$

The interface values of the primitive variables are calculated using MUSCL differencing and the minmod limiter.[6]

$$\begin{aligned}
(\mathbf{V}_L)_{i+1/2} &= \mathbf{V}_i + \frac{1}{4}[(1-\kappa)\overline{\delta^-}\mathbf{V}_i + (1+\kappa)\overline{\delta^+}\mathbf{V}_i] \\
(\mathbf{V}_R)_{i+1/2} &= \mathbf{V}_{i+1} - \frac{1}{4}[(1+\kappa)\overline{\delta^-}\mathbf{V}_{i+1} + (1-\kappa)\overline{\delta^+}\mathbf{V}_{i+1}] \\
\delta^-\mathbf{V}_i &= \frac{\mathbf{V}_i - \mathbf{V}_{i-1}}{\xi_i - \xi_{i-1}} \\
\delta^+\mathbf{V}_i &= \frac{\mathbf{V}_{i+1} - \mathbf{V}_i}{\xi_{i+1} - \xi_i} \\
\overline{\delta^-}\mathbf{V} &= \text{minmod}(\delta^-\mathbf{V}, \delta^+\mathbf{V}) \\
\overline{\delta^+}\mathbf{V} &= \text{minmod}(\delta^+\mathbf{V}, \delta^-\mathbf{V}) \\
\text{minmod}(x,y) &= \max[0, \min(x\text{sign}[y], y\text{sign}[x])]\text{sign}(x) \\
\kappa &= \frac{1}{3}
\end{aligned} \tag{12}$$

The operators δ, δ^+, and δ^- are the central difference, forward difference, and backward difference, respectively. The dissipation term $-|A|(\mathbf{U}_R - \mathbf{U}_L)]_{i+1/2}$ is given by:

$$|A|(\mathbf{U}_R - \mathbf{U}_L)]_{i+1/2} = \begin{bmatrix} \alpha_4 \\ \tilde{u}\alpha_4 + k_x\alpha_5 + \alpha_6 \\ \tilde{v}\alpha_4 + k_y\alpha_5 + \alpha_7 \\ \tilde{w}\alpha_4 + k_z\alpha_5 + \alpha_8 \\ \tilde{H}\alpha_4 + \tilde{\tilde{u}}\alpha_5 + \tilde{u}\alpha_6 + \tilde{v}\alpha_7 + \tilde{w}\alpha_8 - \frac{\tilde{c}^2}{\gamma-1}\alpha_1 \end{bmatrix} \tag{13}$$

where:

$$\begin{aligned}
\alpha_1 &= \left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}}|(\Delta\rho - \frac{\Delta p}{\tilde{c}^2}) \\
\alpha_2 &= \frac{1}{2\tilde{c}^2}\left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}} + \tilde{c}|(\Delta p + \tilde{\rho}\tilde{c}\Delta\bar{u}) \\
\alpha_3 &= \frac{1}{2\tilde{c}^2}\left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}} - \tilde{c}|(\Delta p - \tilde{\rho}\tilde{c}\Delta\bar{u}) \\
\alpha_4 &= \alpha_1 + \alpha_2 + \alpha_3 \\
\alpha_5 &= \tilde{c}(\alpha_2 - \alpha_3) \\
\alpha_6 &= \left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}}|(\tilde{\rho}\Delta u - k_x\tilde{\rho}\Delta\bar{u}) \\
\alpha_6 &= \left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}}|(\tilde{\rho}\Delta v - k_y\tilde{\rho}\Delta\bar{u}) \\
\alpha_6 &= \left|\frac{\nabla\xi}{J}\right| |\tilde{\bar{u}}|(\tilde{\rho}\Delta w - k_z\tilde{\rho}\Delta\bar{u})
\end{aligned}$$

where Δ indicates the right state minus the left state (e.g. $\Delta\rho = \rho_R - \rho_L$).

The $\tilde{}$ superscript indicates the Roe-averaged state of the corresponding variable.

$$\begin{aligned}
\tilde{\rho} &= \sqrt{\rho_L\rho_R} \\
\tilde{u} &= (u_L + u_R\sqrt{\rho_L/\rho_R})/(1 + \sqrt{\rho_L/\rho_R}) \\
\tilde{v} &= (v_L + v_R\sqrt{\rho_L/\rho_R})/(1 + \sqrt{\rho_L/\rho_R}) \\
\tilde{w} &= (w_L + w_R\sqrt{\rho_L/\rho_R})/(1 + \sqrt{\rho_L/\rho_R}) \\
\tilde{H} &= (H_L + H_R\sqrt{\rho_L/\rho_R})/(1 + \sqrt{\rho_L/\rho_R}) \\
\tilde{c}^2 &= (\gamma - 1)[\tilde{H} - (\tilde{u}^2 + \tilde{v}^2 + \tilde{w}^2)/2]
\end{aligned} \tag{14}$$

The geometry terms and normal velocity are defined as:

$$\begin{aligned}
k_x &= \frac{\xi_x}{|\nabla\xi|} \\
k_y &= \frac{\xi_y}{|\nabla\xi|} \\
k_z &= \frac{\xi_z}{|\nabla\xi|}
\end{aligned} \tag{15}$$

$$\bar{u} = k_x u + k_y v + k_z w. \tag{16}$$

2.3.4 Entropy Fix

Unfortunately, Roe's method admits an expansion shock as an appropriate solution to the Euler equations.[7] The simplest way to take care of this problem is to add some numerical dissipation in the calculation of the eigenvalues. The components of $|\Lambda|$ are replaced by $\beta(\lambda)$.

$$\beta(\lambda) = \begin{cases} |\lambda| & |\lambda| \geq \epsilon \\ \frac{\lambda^2+\epsilon^2}{2\epsilon} & |\lambda| < \epsilon \end{cases} \tag{17}$$

Since

$$\begin{aligned} \lambda_1 &= u \\ \lambda_2 &= u + c \\ \lambda_3 &= u - c \end{aligned} \tag{18}$$

$|u|$, $|u + c|$, and $|u - c|$ are replaced by $\beta(u)$, $\beta(u + c)$, and $\beta(u - c)$ in the flux calculations of Roe's method.[8] ϵ is a user-determined small number—ideally large enough to prevent expansion shocks while causing minimal dissipation of shock waves and contact discontinuities.

3 r-Refinement Adaptation

3.1 Introduction to r-Refinement

r-Refinement adaptation seeks to improve the accuracy of a numerical simulation through repositioning the nodes in the computational grid. The improvement in accuracy results from increased resolution and grid alignment. Resolution is increased via the migration of grid nodes toward areas of high estimated error. Grid alignment is the alignment of grid lines with solution features such as shock waves or other discontinuities. This alignment results in more accurate flux calculations, since the dominant fluxes are oriented normal to the solution features.

r-Refinement begins with the calculation of a weight function to use as a representation of a local error estimate over the solution domain. High values of the weight function correspond to high estimated error, and low values of the weight function correspond to low estimated error. The grid nodes are then moved in accordance with the weight function by means of an elliptic grid generation algorithm, such as the center-of-mass algorithm.[9] The nodes move in such a way as to cluster in regions of high weight function. After the nodes have been moved, the solution must be redistributed to the new positions of the cells. In the solver-dependent approach, the additional grid metric terms representing the grid node velocities correct the convective velocity for cell motion. Grid node movement is thereby included in the solver step. In the solver-independent approach, a separate redistribution procedure is performed to redistribute the solution to the new cell locations before the new grid and solution are passed to the flow solver, where the solution is marched forward in time. This uncoupled approach allows any solver to be used in conjunction with the adapter.

3.2 The Solver-Independent, Efficient r-Refinement Algorithm (SIERRA)

The Solver-Independent, Efficient r-Refinement Algorithm (SIERRA) was used in this study. SIERRA employs a weight function based on the Laplacians of the dependent variables and the volumes of the grid cells. The grid nodes are then repositioned using the SIERRA weight function in a center-of-mass algorithm, after which the solution redistribution procedure is applied.[10]

3.2.1 Weight Function

For a single variable, the weight function is given by

$$W_{i,j,k} = |\nabla^2 U_{i,j,k}| V_{i,j,k}^{\omega}. \tag{19}$$

The Laplacian operator is applied in the computational space using either a 7 or 27-point stencil. The term ω is the volume weighting; $0 < \omega < 1$ leads to more adaptation and better resolution of strong solution features, and $\omega > 1$ leads to less adaptation. When a vector of dependent variables is considered, $\|\nabla^2 \mathbf{U_{i,j,k}}\|_\mathbf{2}$ is used as a replacement for $|\nabla^2 U_{i,j,k}|$, or for some problems, a single variable from the vector (such as density) can be used to reduce the cost of calculating the weight function. In this work, the following 27-point stencil was used:

$$\begin{aligned}
\nabla^2 U_{i,j,k} \approx \frac{6}{26}(&U_{i-1,j-1,k-1} + U_{i-1,j-1,k} + U_{i-1,j-1,k+1} \\
&+U_{i-1,j,k-1} + U_{i-1,j,k} + U_{i-1,j,k+1} U_{i-1,j+1,k-1} + U_{i-1,j+1,k} + U_{i-1,j+1,k+1} \\
&+U_{i,j-1,k-1} + U_{i,j-1,k} + U_{i,j-1,k+1} + U_{i,j,k-1} - 26U_{i,j,k} + U_{i,j,k+1} \\
&+U_{i,j+1,k-1} + U_{i,j+1,k} + U_{i,j+1,k+1} U_{i+1,j-1,k-1} + U_{i+1,j-1,k} + U_{i+1,j-1,k+1} \\
&+U_{i+1,j,k-1} + U_{i+1,j,k} + U_{i+1,j,k+1} + U_{i+1,j+1,k-1} + U_{i+1,j+1,k} + U_{i+1,j+1,k+1})
\end{aligned} \tag{20}$$

The L2 norm as shown above was used to derive a scalar from the vector of Laplacians. The vector used to generate the weight function was $\mathbf{V}$, the vector of the primitive

variables, rather than **U**, the vector of conservative variables.

Unfortunately, this formulation produces a weight function which is not very smooth. This can reduce mesh smoothness and increase mesh skewness, so the weight function is typically conditioned after it is calculated. The first stage of the conditioning process is limiting the minimum value of $\nabla^2 U$.[10]

$$(\nabla^2 U)^{\text{limited}} = \max(\nabla^2 U, \text{cut off value}) \tag{21}$$

This process helps reduce the noise in the weight function in regions of uniform flow. In regions of high $\nabla^2 U$, discontinuous derivatives remain; thus, the weight function must be smoothed with an elliptic smoothing algorithm.

$$W_{i,j,k}^{n+1} = W_{i,j,k}^{n} + \nabla^2 W_{i,j,k}^{n} \tag{22}$$

Smoothing the weight function not only reduces the weight function noise; it also speeds up convergence of the grid. Symmetric Gauss-Seidel is used to perform the smoothing. The number of iterations of symmetric Gauss-Seidel is a user determined parameter of SIERRA.

3.2.2 Grid Node Movement

SIERRA uses the center-of-mass algorithm for grid node movement.[9] The center-of-mass algorithm defines a mass distribution over a cluster of cells near a node and then moves the node to the center-of-mass of the cluster. When the values of the weight function are stored at the cell centers, the center-of-mass algorithm applied in

the computational space is given by

$$\begin{aligned}
\xi_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}\xi_{l,m,n}}{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}} \\
\eta_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}\eta_{l,m,n}}{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}} \\
\zeta_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}\zeta_{l,m,n}}{\sum_{n=k-1/2}^{k+1/2}\sum_{m=j-1/2}^{j+1/2}\sum_{l=i-1/2}^{i+1/2} W_{l,m,n}}.
\end{aligned} \tag{23}$$

Taking the cluster to be the eight cells in computational space for which node i, j, k is a vertex ensures that node i, j, k will remain inside the bounding box defined by cell centers $i \pm \frac{1}{2}, j \pm \frac{1}{2}, k \pm \frac{1}{2}$ in the computational space. However, care must be taken to ensure that the mapping used to calculate the new grid node position in physical space is accurate enough that no grid line crossover occurs in the physical space as well. This formulation of the center-of-mass algorithm is the cheapest form for cell-centered schemes, since it is easy to calculate the Laplacians at the cell centers. If Equation 23 is to be used for a node-centered scheme, then the node-centered Laplacians must be interpolated to the cell centers. Alternatively, the following formulation uses the Laplacians at the nodes directly:

$$\begin{aligned}
\xi_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}\xi_{l,m,n}}{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}} \\
\eta_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}\eta_{l,m,n}}{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}} \\
\zeta_{i,j,k}^{\text{new}} &= \frac{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}\zeta_{l,m,n}}{\sum_{n=k-1}^{k+1}\sum_{m=j-1}^{j+1}\sum_{l=i-1}^{i+1} W_{l,m,n}}.
\end{aligned} \tag{24}$$

Equation 24 can also be used for a cell-centered scheme, but the cell-centered Laplacians must be interpolated to the nodes. Unlike Equation 23, however, Equation 24 does not have the property that the new node will be inside the bounding box defined by the cell centers $i \pm \frac{1}{2}, j \pm \frac{1}{2}, k \pm \frac{1}{2}$ in the computational space. Rather, Equation 24

has the property that the new node will be inside the bounding box defined by the old nodes $i \pm 1, j \pm 1, k \pm 1$, making grid line crossover possible. The bounding box can be made smaller by damping Equation 24, but this is rarely necessary in practice because using a 27-point node-centered stencil tends to produce smoother grids than the 8-point cell-centered stencil anyway.

Both Equation 23 and Equation 24 perform the center-of-mass algorithm in computational space. The new coordinates $(\xi^{\text{new}}_{i,j,k}, \eta^{\text{new}}_{i,j,k}, \zeta^{\text{new}}_{i,j,k})$ for each node must be used to find the new physical coordinates $(x^{\text{new}}_{i,j,k}, y^{\text{new}}_{i,j,k}, z^{\text{new}}_{i,j,k})$. The grid transformation can be performed to varying degrees of precision. The following mapping was used for this work:

$$\begin{aligned}
\Delta x &= x_\xi \Delta\xi + x_\eta \Delta\eta + x_\zeta \Delta\zeta \\
&\quad + \frac{1}{2}(x_{\xi\xi}\Delta\xi^2 + x_{\eta\eta}\Delta\eta^2 + x_{\zeta\zeta}\Delta\zeta^2) \\
&\quad + x_{\xi\eta}\Delta\xi\Delta\eta + x_{\xi\zeta}\Delta\xi\Delta\zeta + x_{\eta\zeta}\Delta\eta\Delta\zeta \\
\Delta y &= y_\xi \Delta\xi + y_\eta \Delta\eta + y_\zeta \Delta\zeta \\
&\quad + \frac{1}{2}(y_{\xi\xi}\Delta\xi^2 + y_{\eta\eta}\Delta\eta^2 + y_{\zeta\zeta}\Delta\zeta^2) \\
&\quad + y_{\xi\eta}\Delta\xi\Delta\eta + y_{\xi\zeta}\Delta\xi\Delta\zeta + y_{\eta\zeta}\Delta\eta\Delta\zeta \\
\Delta z &= z_\xi \Delta\xi + z_\eta \Delta\eta + z_\zeta \Delta\zeta \\
&\quad + \frac{1}{2}(z_{\xi\xi}\Delta\xi^2 + z_{\eta\eta}\Delta\eta^2 + z_{\zeta\zeta}\Delta\zeta^2) \\
&\quad + z_{\xi\eta}\Delta\xi\Delta\eta + z_{\xi\zeta}\Delta\xi\Delta\zeta + z_{\eta\zeta}\Delta\eta\Delta\zeta.
\end{aligned} \tag{25}$$

This mapping prevents grid line crossover for most grids. Higher order methods may be necessary for highly concave surfaces. First order methods may result in grid line crossover.

Alternatively, the center-of-mass algorithm can be performed directly in the physical space. This approach produces smoother grids on the interior of the domain; however, boundary curvature can cause points to be gravitated toward or away from

the boundary and produce crossover at the boundary, even in the presence of a uniform weight function. The physical space formulation is an adaptation of Equation 24 obtained by replacing ξ, η, and ζ with x, y, and z. Because of the boundary problems, the physical space formulation is recommended only for simple geometries.

3.2.3 Solution Redistribution

When r-refinement is performed in a solver-independent procedure, the flow solver does not contain the grid node movement terms to correct for the movement of the control volume. Instead, an additional algorithm, known as the solution redistribution procedure, is performed after the grid movement step and before the flow solver step. The solution redistribution procedure solves the moving control volume problem for a fixed time level. This means that a control volume has fluxes associated with each of its faces, but these fluxes are only dependent upon two things: the volume swept out by each face as the control volume moves and the interpolated values of the dependent variables at the face of the control volume. Ideally, this makes the dependent variable changes due to control volume movement independent of the time integration, and it makes the dependent variable changes due to time integration independent of control volume movement. Such an approach makes grid adaptation more portable because the two integrations can be separated. The time integration is the traditional flow solver step.

The general conservation law for a stationary control volume is

$$\frac{d}{dt}\int_V \mathbf{U}dV + \int_{\partial V} \mathbf{U}\vec{v}\cdot\hat{n}dS = \int_V \mathbf{Q}dV \tag{26}$$

where

dV = a volume element

dS = a surface element

$\hat{n}$ = the unit normal vector for a surface element

$\mathbf{Q}$ = the source vector expressed in terms of the conservative variables.

If the control volume is moving, then the velocity must be expressed relative to the velocity of the control surface, rather than relative to the inertial reference frame. Thus, the conservation law for a moving control volume is obtained by replacing the term $\vec{v}$ in Equation 26 with $\vec{v} - \vec{v_b}$, where $\vec{v_b}$ is the velocity of the surface element.

$$\frac{d}{dt}\int_V \mathbf{U}dV + \int_{\partial V} \mathbf{U}(\vec{v} - \vec{v_b}) \cdot \hat{n}dS = \int_V \mathbf{Q}dV \tag{27}$$

Since the flow solver solves Equation 26, the solution redistribution procedure must solve the difference of Equations 27 and 26. Subtracting Equation 26 from Equation 27 yields the following conservation law for the solution redistribution procedure:

$$\left.\frac{d}{dt}\int_V \mathbf{U}dV\right|_{\text{SRP}} = \int_{\partial V} \mathbf{U}\vec{v_b} \cdot \hat{n}dS. \tag{28}$$

This conservation law can be expressed in terms of a flux balance. The single-step, explicit method is given as follows.

$$\begin{aligned}(\mathbf{U}V)^{\text{new}} &= (\mathbf{U}V)^{\text{old}} + \mathbf{F}_{i+1/2,j,k} - \mathbf{F}_{i-1/2,j,k} \\ &\quad + \mathbf{G}_{i,j+1/2,k} - \mathbf{G}_{i,j-1/2,k} + \mathbf{H}_{i,j,k+1/2} - \mathbf{H}_{i,j,k-1/2}\end{aligned} \tag{29}$$

where

$$\begin{aligned}
\mathbf{F}_{i+1/2,j,k} &= \mathbf{U}_{i+1/2,j,k}\Delta V_{i+1/2,j,k} \\
\mathbf{G}_{i,j+1/2,k} &= \mathbf{U}_{i,j+1/2,k}\Delta V_{i,j+1/2,k} \\
\mathbf{H}_{i,j,k+1/2} &= \mathbf{U}_{i,j,k+1/2}\Delta V_{i,j,k+1/2} \\
\mathbf{U}_{i+1/2,j,k} &= \text{interface value of dependent variables} \\
\Delta V_{i+1/2,j,k} &= \text{volume swept out by cell side movement.}
\end{aligned}$$

For most applications, the single-step explicit method is not accurate enough to handle significant grid movement. High frequency error introduced by the solution redistribution procedure results in a noisy weight function, which in turn, causes random fluctuations in node movement. These random fluctuations in node movement result in more error being introduced by the solution redistribution procedure. This feedback can cause grid and solution quality to deteriorate within only a few iterations of the adapter. To combat this problem, a second or fourth order Runge-Kutta scheme with interval subdivision is usually used.[10] To express this scheme, the following notation is used: $U^{n,m}$ where n is the step (which varies between 1 and N) and m is the RK stage (which varies between 1 and M).

For $n = 1, N-1$
  For $m = 1, M-1$

$$\begin{aligned}
(\mathbf{U}V)^{n,m+1} &= (\mathbf{U}V)^{n,1} + \mathbf{F}^{n,m}_{i+1/2,j,k} - \mathbf{F}^{n,m}_{i-1/2,j,k} \\
&\quad + \mathbf{G}^{n,m}_{i,j+1/2,k} - \mathbf{G}^{n,m}_{i,j-1/2,k} + \mathbf{H}^{n,m}_{i,j,k+1/2} - \mathbf{H}^{n,m}_{i,j,k-1/2} \\
(\mathbf{U}V)^{n+1,1} &= (\mathbf{U}V)^{n,1} + \mathbf{F}^{n,M}_{i+1/2,j,k} - \mathbf{F}^{n,M}_{i-1/2,j,k} \\
&\quad + \mathbf{G}^{n,M}_{i,j+1/2,k} - \mathbf{G}^{n,M}_{i,j-1/2,k} + \mathbf{H}^{n,M}_{i,j,k+1/2} - \mathbf{H}^{n,M}_{i,j,k-1/2}
\end{aligned}$$

where

$$\mathbf{F}^{n,m}_{i+1/2,j,k} = \mathbf{U}^{n,m}_{i+1/2,j,k}\Delta V^{n,m}_{i+1/2,j,k}$$

$$\Delta V^{n,m} = \frac{V^{n+1} - V^n}{M+1-m}$$

The number of interval subdivisions can be set to a fixed number or a variable number which is dependent upon the amount of grid movement. In this work, the number of interval subdivisions was $2N_{\max}\max(\|\Delta\xi\|_\infty, \|\Delta\eta\|_\infty, \|\Delta\zeta\|_\infty, \frac{1}{2N_{\max}})$, where $\Delta\xi$ is

defined as $\xi^{\text{new}} - \xi$. This resulted in between 1 and $N_{\max}$ steps with 1, 2, or 4 RK stages each.

The solution redistribution procedure must also calculate the interface values of the conservative variables. This calculation is an interpolation of nearby cell-centered values of the conservative variables. The formula used in this work is

$$\mathbf{U}_{i+1/2,j,k} = \frac{-\mathbf{U}_{i-1,j,k} + 9\mathbf{U}_{i,j,k} + 9\mathbf{U}_{i+1,j,k} - \mathbf{U}_{i+2,j,k}}{16}. \tag{31}$$

This formula is of order $O(\Delta\xi^3)$. High order approximations such as this can introduce fluctuations into the solution, thus limiting must be performed to ensure that the scheme is total variation diminishing (TVD). A conservative flux limiter which attempts to accomplish this was developed by Laflin[10]:

> Let U_i be a cell centered value, $U_{i+1/2}$ be a cell interface value, V_i be cell volume, and $\Delta V_{i+1/2}$ be the volume swept out by the side of the control volume. The unlimited value for the flux is given by:
>
> $$\mathbf{F}_{i+1/2} = \mathbf{U}_{i+1/2}\Delta V_{i+1/2} \tag{32}$$
>
> The limited value for the flux is given by the following set of equations:
>
> $$\mathbf{F}^{\text{limited}}_{i+1/2} = \min(\mathbf{\Lambda}_{\max}, \mathbf{\Psi}_{\max}, \max(\mathbf{F}_{i+1/2}, \mathbf{\Lambda}_{\min}, \mathbf{\Psi}_{\min})) \tag{33}$$
>
> where:
>
> $$\begin{aligned}
> \mathbf{\Lambda}_{\min} &= \min(\mathbf{U}_i, \mathbf{U}_{i+1})(V_i + \Delta V_{i+1/2}) - \mathbf{U}_i V_i \\
> \mathbf{\Lambda}_{\max} &= \max(\mathbf{U}_i, \mathbf{U}_{i+1})(V_i + \Delta V_{i+1/2}) - \mathbf{U}_i V_i \\
> \mathbf{\Psi}_{\min} &= -\min(\mathbf{U}_i, \mathbf{U}_{i+1})(V_{i+1} - \Delta V_{i+1/2}) + \mathbf{U}_{i+1} V_{i+1} \\
> \mathbf{\Psi}_{\max} &= -\max(\mathbf{U}_i, \mathbf{U}_{i+1})(V_{i+1} - \Delta V_{i+1/2}) + \mathbf{U}_{i+1} V_{i+1}
> \end{aligned}$$

The conservative limiter strictly enforces the conservation property while approximately enforcing the TVD property. The TVD property is only approximately enforced because the derivation of the conservative limiter only considers the movement of one cell face. For a 3-D problem, each cell has six cell sides, all of which might be moving at the same time. Unfortunately, for some flows, strict TVD enforcement is critical. To address this, a non-conservative limiter has also been developed:

$$\mathbf{U}_{i,j,k}^{\text{new,limited}} = \max(\min(\mathbf{U}_{i,j,k}^{\text{new}}, \mathbf{U}_{\max}), \mathbf{U}_{\min}) \tag{34}$$

where

$$\begin{aligned} \mathbf{U}_{\min} &= \min(\mathbf{U}_{l,m,n}^{\text{old}}), l \in [i-1, i+1], m \in [j-1, j+1], n \in [k-1, k+1] \\ \mathbf{U}_{\max} &= \max(\mathbf{U}_{l,m,n}^{\text{old}}), l \in [i-1, i+1], m \in [j-1, j+1], n \in [k-1, k+1]. \end{aligned}$$

In this work, both the conservative and non-conservative limiters were used. By employing the conservative limiter first and then the non-conservative limiter second, the updated solution is kept as close to TVD as possible without violating conservation. Then, the non-conservative limiter is used to strictly enforce the TVD property. This tandem approach strictly enforces the TVD property everywhere while only violating conservation in the case where it is absolutely necessary to maintain the TVD property.

4 Interblock Communication

Many multiblock solvers use ghost cells (overlap cells taken from the adjacent block) to provide the boundary conditions for computations within each block. These ghost cells are chosen or generated so that they can provide data for the computation of block interface cell fluxes. The goal is to have the flux computation proceed as if the block boundary did not exist. A communication algorithm is used to transfer data from the interior cells of the adjacent block to the ghost cells. For continuously connected blocks, the communication algorithm simply involves copying dependent variables from the interior cells of the adjacent block to the ghost cells. In order for two discontinuously connected blocks to communicate, interior data of the adjacent block must be transferred to the ghost cells using a more complex algorithm. First, the ghost cells must be generated by locating their vertices, known as the ghost nodes. Next, the dependent variable values must be determined using interior data from the adjacent block.

4.1 Ghost Node Generation

For two 3-D discontinuously-connected blocks with arbitrary positioning of ghost nodes, data transfer is generally a 3-D problem since the patterns of overlap between the ghost cells and the interior cells of the adjacent block vary in all three dimensions. This results in a mapping to the ghost cells from the interior cells of the adjacent block with six degrees of freedom. However, if the ghost cells are constructed appropriately, this data transfer can be reduced to a 2-D problem. This special form of ghost cell generation is referred to as "tiling." Reducing the number of varying spatial dimensions to two reduces the number of degrees of freedom of the data transfer to four. A description of the "tiling" technique follows.

The interface of two 3-D blocks is defined by two surface grids, known as the

block boundary grids, which are defined on the boundaries of their parent blocks. If the grid blocks are continuous across the interface, then the block boundary grids are identical. If the grid blocks are not continuous across the interface, then a four-sided cell face belonging to the boundary grid on one side of the interface is divided into N polygons by the boundary grid on the other side of the interface. Figure 1 shows an example of a cell being cut into 4 polygons. In the general case, a ghost cell would

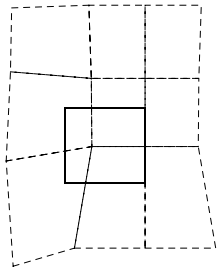

Figure 1: Boundary Cell Face Being Cut by Opposing Boundary Grid

be cut into an arbitrary number of polyhedra by the interior cells of the adjacent block, but if ghost cells are generated appropriately, then the ghost cells would be divided into N polyhedra, just as their faces are divided into N polygons. This is accomplished by locating a boundary point in the (ξ, η) transformed computational space of the boundary grid on the opposing side of the interface. The ghost nodes are then created by "tiling" the point, which involves using the same ξ and η coordinates while translating by ± 1 in the ζ direction. The (ξ, η, ζ) coordinates are then mapped into the (x, y, z) space to give the physical location of the node. Figure 2 shows a point "tiled" from a surface grid onto the next two ζ constant surfaces in the adjacent block. Once all of the nodes have been tiled, the cells are defined by the nodes using either a cell-centered or node-centered finite volume discretization. For this work, a cell-centered discretization is considered.

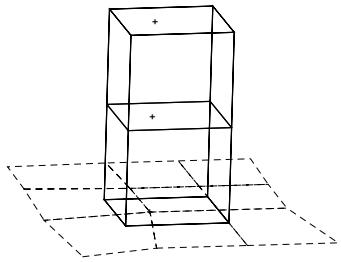

Figure 2: Process of "Tiling" to Form Ghost Nodes

4.2 Ghost Cell Dependent Variable Value Determination

In order to use ghost cells for communication, data must be transferred into them from interior cells of the adjacent block. As previously mentioned, this data transfer is simpler if the ghost nodes are tiled in order to generate the ghost cells. If this is done, then a single mapping can be derived which transfers dependent variable values to ghost cells of one block from interior cells of an adjacent block. If one of the blocks is labeled block one, and the adjacent block is labeled block two, then the mapping which transfers data from block two to block one is $I_{2\to1}$. The transfer mapping from block one to block two is given by $I_{1\to2} = I_{2\to1}^T$. The mapping $I_{2\to1}$ is generally sparse, since the mapping entry linking interior cells of block two to ghost cells of block one is zero except when the cells overlap. An example use of the mapping $I_{2\to1}$ to fill a ghost

cell with its proper value of the dependent variable u is given in Equations 35-40:

$$\mathbf{U}_{i_1,j_1,k,1} = \frac{1}{V_{i_1,j_1,k,1}} \sum_{i_2,j_2} \mathbf{U}_{i_2,j_2,k,2} \bar{h}_{i_1,j_1,i_2,j_2,k} \mathcal{A}_{i_1,j_1,i_2,j_2} \tag{35}$$

$$\bar{h}_{i_1,j_1,i_2,j_2,k} = \beta_1 \frac{V_{i_1,j_1,k,1}}{A_{i_1,j_1,1}} + \beta_2 \frac{V_{i_2,j_2,k,2}}{A_{i_2,j_2,2}} \tag{36}$$

$$\beta_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2} \tag{37}$$

$$\beta_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2} \tag{38}$$

$$\alpha_1 = \left(\frac{\mathcal{A}_{i_1,j_1,i_2,j_2}}{A_{i_1,j_1,1}} + \epsilon \right) \left(1 - \frac{\mathcal{A}_{i_1,j_1,i_2,j_2}}{A_{i_2,j_2,2}} + \epsilon \right) \tag{39}$$

$$\alpha_2 = \left(\frac{\mathcal{A}_{i_1,j_1,i_2,j_2}}{A_{i_2,j_2,2}} + \epsilon \right) \left(1 - \frac{\mathcal{A}_{i_1,j_1,i_2,j_2}}{A_{i_1,j_1,1}} + \epsilon \right) \tag{40}$$

where:

$\mathbf{U}$ = dependent variable vector of cell

A = area of the cell face lying on the block boundary

$\mathcal{A}$ = overlap area of two cell face areas

V = volume of cell

ϵ = some small positive number.

$\mathcal{A}$ can be generated using a Ramshaw algorithm, which calculates the overlap areas of all cell faces between the two blocks.[11] $\mathcal{A}$, A, and V need to be generated each time the grid is moved. When the mapping is to be used, the dependent variable values in the ghost cells are set using $I_{2\to1}$ and $I_{1\to2}$. The number of layers of ghost cells necessary for each block depends upon the differencing method. For second order upwind methods, two layers of ghost cells are generally necessary (although more layers of cells can be used to decrease communication frequency in parallel applications).

4.3 Cell Face Intersection Area Calculation using the Ramshaw Algorithm

The cell face intersection areas are calculated using a Ramshaw algorithm.[11] The Ramshaw algorithm stems from the general equation for the area of a polygon P:

$$A_P = \frac{1}{2}\sum_{s=1}^{N_s} \epsilon_s^P (x_1^s y_2^s - x_2^s y_1^s) \tag{41}$$

where

$$\begin{aligned}
N_s &= \text{the number of sides} \\
x_1^s, y_1^s &= \text{the } x,y \text{ coordinates of vertex 1 of side } s \\
x_2^s, y_2^s &= \text{the } x,y \text{ coordinates of vertex 2 of side } s \\
\epsilon_s^P &= \begin{cases} 1 & \text{if polygon } P \text{ lies to the left of side } s \\ -1 & \text{if polygon } P \text{ lies to the right of side } s. \end{cases}
\end{aligned}$$

This formula allows the area of any polygon to be calculated given the locations of its vertices. Thus, if it was easy to determine all of the polygons created like the one in Figure 1, all of the overlap areas could be easily calculated. Unfortunately, the identification of these polygons is a daunting task, so an alternative method is needed.

Equation 41 has the advantage that the sides of the polygon need not be traversed in any particular order. Thus, it is possible to simply traverse the ξ-constant and η-constant mesh lines of both of the block boundary grids defining the interface. As long as this is done line segment by line segment and the left and right polygons for each line segment are identified correctly, all of the overlap areas will be calculated without the need for any polygon identification. Thus, the problem is reduced to marching along each grid line, detecting intersections with other grid lines, and keeping track of which cells faces are to the left, right, or enclosing the line segment. Any line segment which is not coincident with a line segment of the opposing block

boundary grid will have two adjacent cell faces (one on its left and one on its right) belonging to its own block boundary grid and one cell enclosing it belonging to the opposing block boundary grid. In this case, the line segment would contribute to the area defined by the overlap of two cells in its own block boundary grid with one cell of the opposing block boundary grid. Any line segment which is coincident with a line segment of the opposing block boundary grid will have two adjacent cell faces belonging to its own block boundary grid and two adjacent cell faces belonging to the opposing block boundary grid. In this case, the line segment would contribute to the area defined by the overlap of two cells in its own block boundary grid with two cells of the opposing block boundary grid. The coincidence case must be detected because it results in area contributions being counted twice, since coincident line segments exist in both block boundary grids. This problem is handled by only taking one half of the contribution when a coincident line segment is detected. When the contribution of its corresponding line segment in the other block boundary grid is calculated, the other half of the contribution is recovered.

The line segments of a grid line are defined by the intersection of that mesh line with all other lines in its own block boundary grid and all lines in the opposing block boundary grid. Thus the nodes are detected by calculating intersections of grid lines. Calculating the intersection of two straight lines is a fairly straightforward process, since the equations of the two lines can be solved for the intersection point. However, when the intersection of two line segments is being considered, the intersection point must be examined to make sure that it is actually on the line segments, i.e. that it is between the vertices defining the line segments. Thus, a parameter t is defined along the lines where $t = 0$ at the starting node of the line segment and $t = 1$ at the ending node of the line segment. If the t value of the intersection point is not inside the interval $[0, 1]$, then the calculated intersection is not valid. These criteria allow a grid

line to be divided into line segments defined by a set of nodes which are intersections of the grid line with other grid lines.

Once a line segment is defined, it is necessary to determine which cell faces are either adjacent to or enclosing it. This requires that any node defined by physical coordinates (x, y) be located in the (ξ, η) space of both of the block boundary grids. This location is performed by inverse linear transfinite interpolation using Newton's method with line search.[12] The algorithm is given as follows:

Let $\vec{x} = (x, y)$ and $\vec{\xi} = (\xi, \eta)$.
Let $\vec{x}_p$ be the point of interest.
Define $\vec{F}(\vec{\xi}) = \vec{x}_A + (\vec{x}_B - \vec{x}_A)(\xi - \lfloor\xi\rfloor) + (\vec{x}_C - \vec{x}_A)(\eta - \lfloor\eta\rfloor)$
$+(\vec{x}_D + \vec{x}_A - \vec{x}_B - \vec{x}_C)(\xi - \lfloor\xi\rfloor)(\eta - \lfloor\eta\rfloor) - \vec{x}_p.$
Where vertices A, B, C, D are defined as follows:
$\vec{x}_A \leftarrow (x_{\lfloor\xi\rfloor,\lfloor\eta\rfloor}, y_{\lfloor\xi\rfloor,\lfloor\eta\rfloor})$
$\vec{x}_B \leftarrow (x_{\lfloor\xi\rfloor+1,\lfloor\eta\rfloor}, y_{\lfloor\xi\rfloor+1,\lfloor\eta\rfloor})$
$\vec{x}_C \leftarrow (x_{\lfloor\xi\rfloor,\lfloor\eta+1\rfloor}, y_{\lfloor\xi\rfloor,\lfloor\eta\rfloor+1})$
$\vec{x}_D \leftarrow (x_{\lfloor\xi\rfloor+1,\lfloor\eta\rfloor+1}, y_{\lfloor\xi\rfloor+1,\lfloor\eta\rfloor+1})$
Start checking at location $\vec{\xi} = \vec{\xi}_0$.
Do until ϵ is less than some tolerance.
$\Delta\vec{\xi} \leftarrow -(F'(\vec{\xi}))^{-1}\vec{F}(\vec{\xi})$
$\lambda \leftarrow 1$
Do while $\|\vec{F}(\vec{\xi} + \lambda\Delta\vec{\xi})\|_2 > (1 - 1 \times 10^{-4}\lambda)\epsilon$.
$\lambda \leftarrow \frac{\lambda}{2}$
$\Delta\vec{\xi} \leftarrow \lambda\Delta\vec{\xi}$
$\xi \leftarrow \xi + \max(\min(\Delta\xi, 1), 0)$
$\eta \leftarrow \eta + \max(\min(\Delta\eta, 1), 0)$
$\epsilon \leftarrow \|\vec{F}(\vec{\xi})\|_2$

$\lfloor\rfloor$ is the floor function, which is the greatest integer less than or equal to its argument.

The final step is determining when lines are coincident and when they are not. This seems like a trivial task, but the limitations of floating point arithmetic can make it difficult to consistently determine coincidence. The easiest place to determine coincidence is along the edges of the block boundary grids, where it is simply assumed. However, on the interior of the block boundary grids, a consistent method must be used for determining coincidence. The penalty for inconsistently determining

coincidence is severe. If a line segment belonging to one block boundary grid is labeled as being coincident with a line segment of the opposing block boundary grid, the same coincidence must be detected during the sweep through the opposing boundary grid. If this does not happen, the communication operators $I_{2\to1}$ and $I_{1\to2}$ will not only be wrong, but they could possibly even contain negative overlap areas.

The test for coincidence consists of calculating the perpendicular distance between the end points of one line segment with the other line segment. If this distance is less than some specified tolerance ϵ_{coinc}, then the line segments are deemed to be coincident. However, when the distance is very close to the tolerance, the possibility arises that this criterion could pass when sweeping through one of the block boundary grids and fail when sweeping through the other block boundary grid. Thus, a line segment $\overline{AB}$ belonging to side one of the interface could be determined to be coincident with line segment $\overline{CD}$ belonging to side two of the interface during the sweep through the ξ-constant and η-constant lines of side one, whereas line segment $\overline{CD}$ would not be determined to be coincident with line segment $\overline{AB}$ during the sweep through the ξ-constant and η-constant lines of side two.

The problem is remedied by a process called "snapping." "Snapping" prevents nodes from being in localities where the test for coincidence could produce inconsistent results. If a node on one block boundary grid is found to be within a distance $2\epsilon_{\text{coinc}}$ of the lines on the opposing block boundary grid, then that node is projected onto the line of the opposing block boundary grid. This projection, however, also moves the mesh lines, so the snapping process must be repeated for both block boundary grids until all nodes within $2\epsilon_{\text{coinc}}$ of the lines on the opposing block boundary grid

have already been projected. The algorithm is given as follows.

Do until no more nodes need to be projected.
  Do for both block boundary grids.
    Do for all nodes $(x, y)_{i,j}$.
      If the distance between $(x, y)_{i,j}$ and the closest ξ and η lines
      in the opposing block boundary grid is less than $2\epsilon_{\text{coinc}}$,
        Project the node onto the line(s).

The closest lines mentioned in the algorithm are determined using the inverse TFI algorithm. Grid snapping does move the nodes of the meshes slightly, but as long as ϵ_{coinc} is small, the movement of the nodes will not introduce significant error into the grid or solution.

4.4 Determination of ϵ_{coinc}

One common problem in r-refinement algorithms is that they contain many parameters which are determined by the user, rather than the programmer. In the current implementation of SIERRA, there are multiple user-determined constants for each of the three stages of the algorithm (weight function calculation, node movement, and solution redistribution). This moves the burden of tuning SIERRA to the user, requiring a significant amount of trial and error. To introduce a new user-determined parameter in the communication algorithm would complicate this process further, and might even serve to make r-refinement more difficult to implement for a real problem, rather than easier. As a result of this, it is desirable for the computer to calculate ϵ_{coinc}, making its effect on the algorithm transparent to the user. This is accomplished by determining $\epsilon_{\text{machine}}$, and then drawing conclusions about its affect on the inverse TFI algorithm.

Newton's method in floating point arithmetic cannot be expected to reach a tolerance of lower than $\sqrt{\epsilon_{\text{machine}}}$, so the tolerance for the Newton algorithm is set as such. This is estimated to be the smallest possible distance between two points that can be detected. The coincidence distance, ϵ_{coinc}, must be significantly larger

than this, so ϵ_{coinc} is set to $\sqrt[3]{\epsilon_{\text{machine}}}$. Since the algorithm was implemented in FORTRAN 95, the following line of code was used to calculate ϵ_{coinc}:

```
epsilon_coinc = (nearest(1.,1.)-1.)**(1./3.).
```

The `nearest(x,y)` function in FORTRAN 90/95 calculates the closest number to `x` in the direction of `y` which can be represented by the machine.[13] In 8-byte arithmetic, this results in a ϵ_{coinc} of approximately 5×10^{-6}.

5 Tests of the Algorithm

5.1 Static Grid Test Using the Diffusion Equation

Testing of the communication algorithm on a static two-block grid with a high degree of interface discontinuity gave promising results. Figure 3 shows a 3-D plot of the grids; while Figure 4 shows a 2-D plot of the interface.

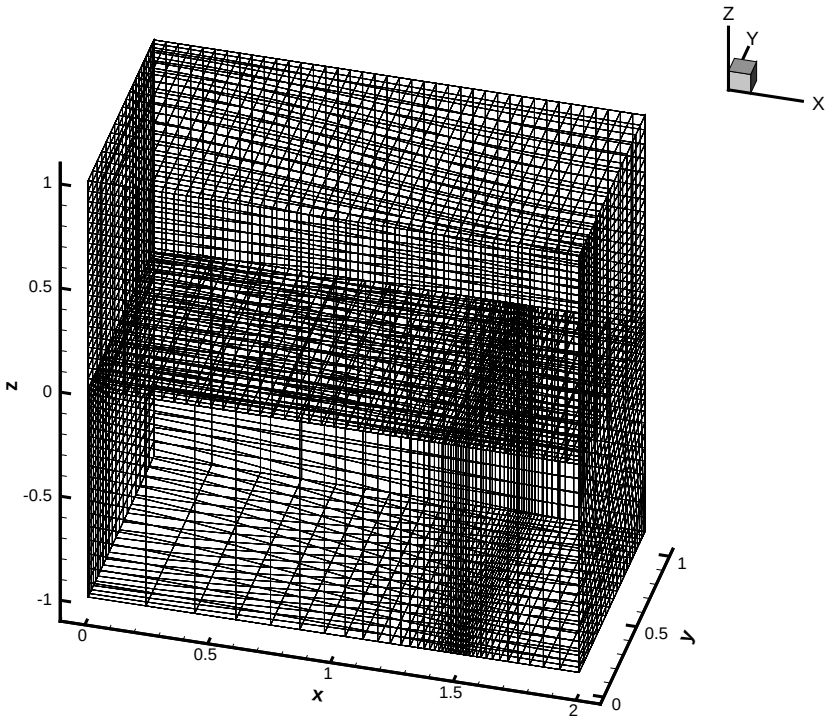

Figure 3: Multiblock Grid Used in Preliminary Testing

The interblock communication algorithm was implemented in a diffusion equation code using a cell-centered finite volume formulation. The algorithm was tested for conservation by setting up a diffusion problem with a given initial mass distribution and impermeable walls. This problem allowed conservation to be tested simply by comparing the initial total mass in the system with the mass in the system at any given iteration. The initial iterate had a total system mass of 14.5712280421577.

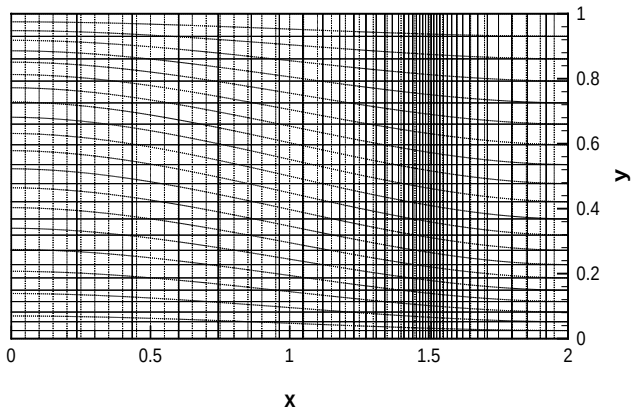

Figure 4: Overlay of Surface Grids Defining the Multiblock Interface ($z = 0$)

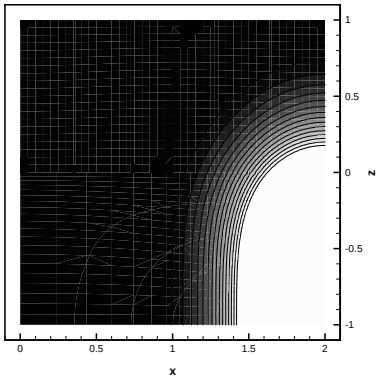

Figure 5: Contour Plot of Solution to Diffusion Equation at $y = \frac{1}{2}$

After 10000 iterations, the total mass in the system was 14.5712282575963, indicating conservation accuracy to 8 decimal places. Figure 5 shows a contour plot on a cut plane at $y = \frac{1}{2}$. The plot shows that the interblock boundary (the $z = 0$ plane) is invisible in the solution.

5.2 Dynamic Grid Test Using the Linear Wave Equation

Tests of the algorithm on adaptive grids were performed to demonstrate that the algorithm is robust enough to handle dynamic discontinuities. The communication algorithm was implemented in a linear wave equation code using SIERRA to adapt the grids.[10] The block topology was the same as that of the static grid tests as shown in Figure 3. The test problem was the movement of a planar wave in the $(1, 1, 1)$ direction. Figure 6 shows the two block boundary grids defining the block interface. The plot shows how the grid blocks adapt to the wave front as it passes through the interface.

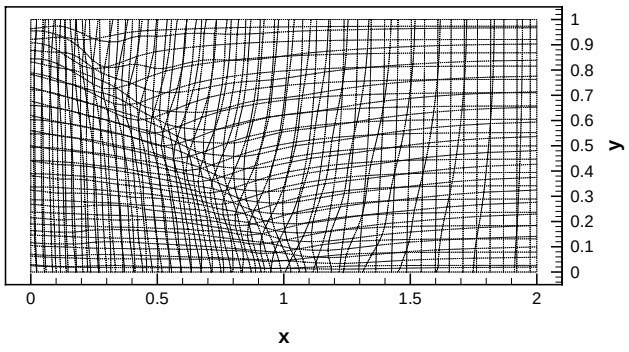

Figure 6: Overlay of Surface Grids Defining the Multiblock Interface ($z = 0$)

Figure 7 shows a contour plot of the wave as it passes through the interface. The cut plane is the $y = \frac{1}{2}$ plane. Figure 8 shows a close-up of Figure 7 so that the block interface is visible in the wave front. The plot shows that the interface is visible in the solution, but the discrepancy at the interface is actually smaller than the

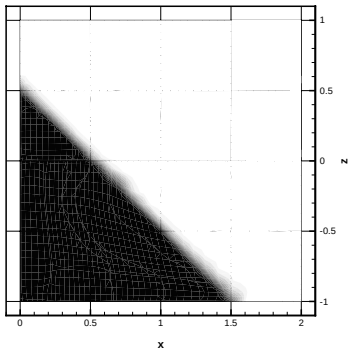

Figure 7: Contour Plot of Solution to Wave Equation at $y = \frac{1}{2}$

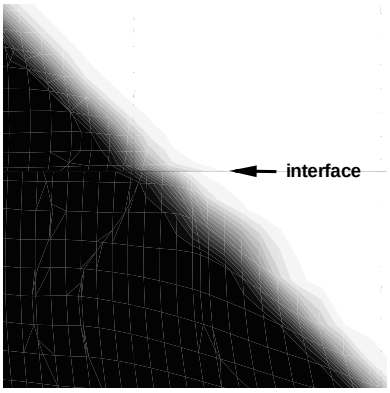

Figure 8: Close-up of Figure 7 Showing the Wave Front at the Interface

discrepancies elsewhere along the wave front due to differences in grid node density (the wave front is better resolved where the grid spacing is smaller).

5.3 Dynamic Grid Test Using the Euler Equations

Since the primary goal of developing the conservative interblock communication algorithm was to implement it in a CFD code, a final test of the algorithm using an Euler solver coupled with SIERRA was performed. This not only demonstrated that the algorithm was stable and accurate for use in CFD applications, it also allowed the behavior of the solution to be observed.

5.3.1 Geometry for the Euler Test

The geometry for the dynamic grid test using the Euler equations was a two block grid, each block having dimensions $161 \times 41 \times 5$. The grids represented a 2-D channel configuration extruded in the z direction to form a 3-D channel. The inlet plane was the $x = 0$ plane, and this corresponded to the $I_{\min}$ plane in the computational space. The outlet plane was the $x = 1$ plane, which corresponded to the $I_{\max}$ plane in the computational space. The interface between the two grids was the $z = 0$ plane, which corresponded to the $K_{\max}$ plane of Zone 1 and the $K_{\min}$ plane of Zone 2. The $J_{\min}$ plane of both zones had a 14.5° wedge from $(x, y) = (0.108696, 0.000000)$ to $(x, y) = (0.239130, 0.033759)$. The $J_{\max}$ plane of both zones was the $y = 0.195652$ plane. The $K_{\min}$ plane of Zone 1 and the $K_{\max}$ plane of Zone 2 were the $z = -0.25$ and $z = 0.25$ planes, respectively. Figure 9 shows the geometry.

5.3.2 Initial and Boundary Conditions for the Euler Test

The boundary conditions were implemented using ghost cells. The inlet conditions are given in Table 1. These numbers correspond to a Mach number (M) of 2. The

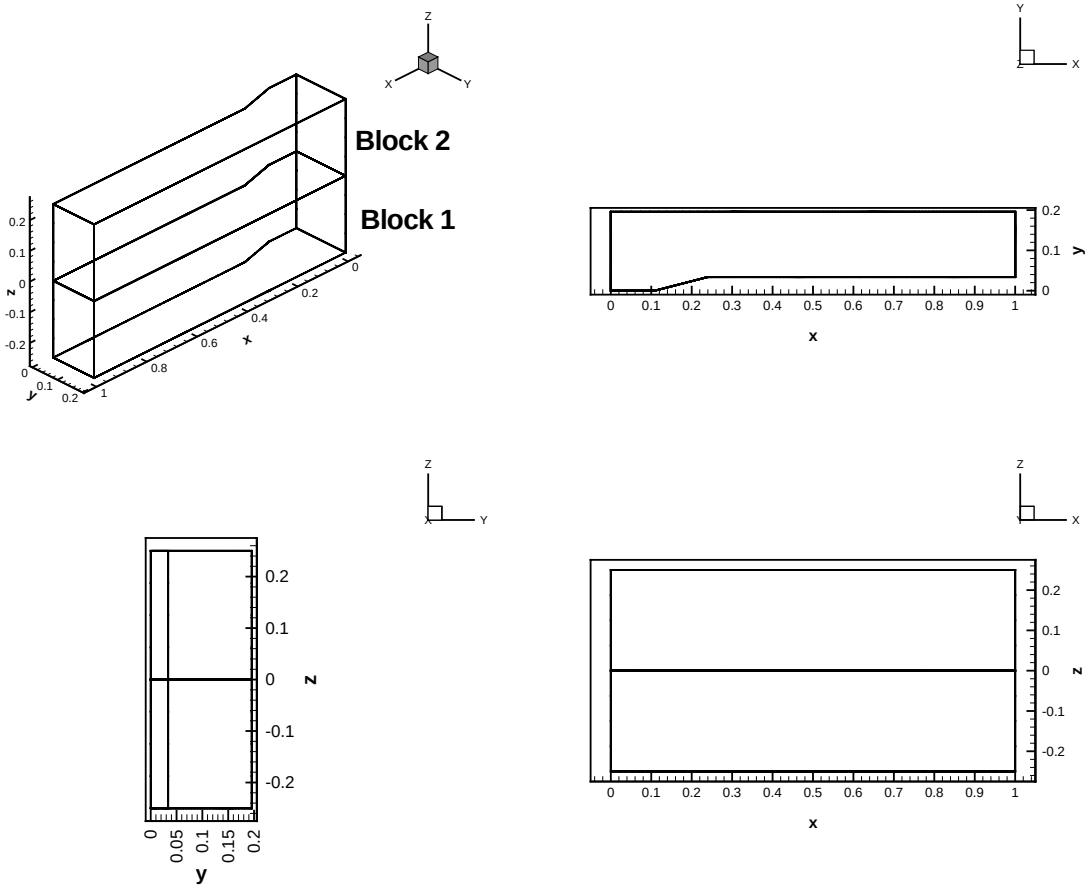

Figure 9: Geometry for the Tests Using the Euler Equations

γ	ratio of specific heats	1.4000
ρ	non-dimensionalized density	1.0000
u	non-dimensionalized velocity in x-direction	1.0000
v	non-dimensionalized velocity in y-direction	0.0000
w	non-dimensionalized velocity in z-direction	0.0000
p	non-dimensionalized pressure	0.1786

Table 1: Inlet Conditions

outlet conditions were set using first-order extrapolation

$$\mathbf{U}(\xi + \Delta\xi) = 2\mathbf{U}(\xi) - \mathbf{U}(\xi - \Delta\xi), \tag{44}$$

since the flow was supersonic at the exit. The boundary conditions along the interface between the two blocks was determined by the communication algorithm. All other block faces were given the slip boundary condition. The slip condition was implemented by using zeroth order extrapolation for ρ and p

$$\begin{aligned} \rho(\xi + \Delta\xi) &= \rho(\xi) \\ p(\xi + \Delta\xi) &= p(\xi), \end{aligned} \tag{45}$$

while reflecting $\vec{v}$ across the cell face where the slip condition is being applied. Figure 10 shows this reflection. The initial conditions for all interior cells were set to be the same as the inlet conditions.

5.3.3 Parameters for SIERRA

The adapter was only operating on Zone 1 of the grid. This ensured a high degree of mismatch between the two block boundary grids in order to test the communication algorithm. (Adapting both blocks with the same criteria would have resulted in closely matched block boundary grids.) Additionally, adapting one block but not the other demonstrated the usefulness of the adapter in applications where only some of the blocks need to be adapted. Table 2 gives the SIERRA parameters for Zone 1.

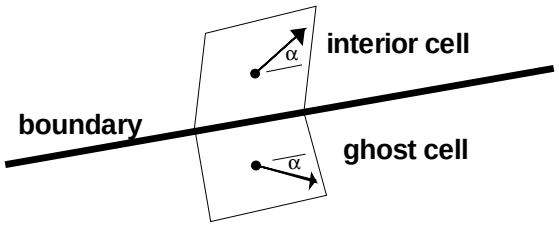

Figure 10: Slip Condition Being Applied to Velocity Vector

minimum allowed value of $\|\nabla^2 \mathbf{U}\|_2$	1×10^{-3}
number of adapter iterations per flow solver step	1
subdivisions per cell in the solution redistribution procedure	1
number of RK stages per cell subdivision	2
exponent for volume weighting (ω)	1
number of smoothing passes for weight function	8
limiter selection	conservative and non-conservative
variable set for non-conservative limiter	conservative variables ($\mathbf{U}$)

Table 2: SIERRA Parameters for Zone 1

5.3.4 Results of the Euler Test

A steady-state solution for the problem was obtained. As is typically a problem with r-refinement adapters coupled with explicit solvers, the grid continued to oscillate slightly even after the shock structure was established and stationary. This caused the norm of the residual to stagnate. To prove that the residual stall was not a result of the communication algorithm, the adapter was turned off after 50000 iterations. Over the next 50000 iterations, the residual dropped to four orders of magnitude lower than the initial residual, as is shown in Figure 11. The CFL number was set

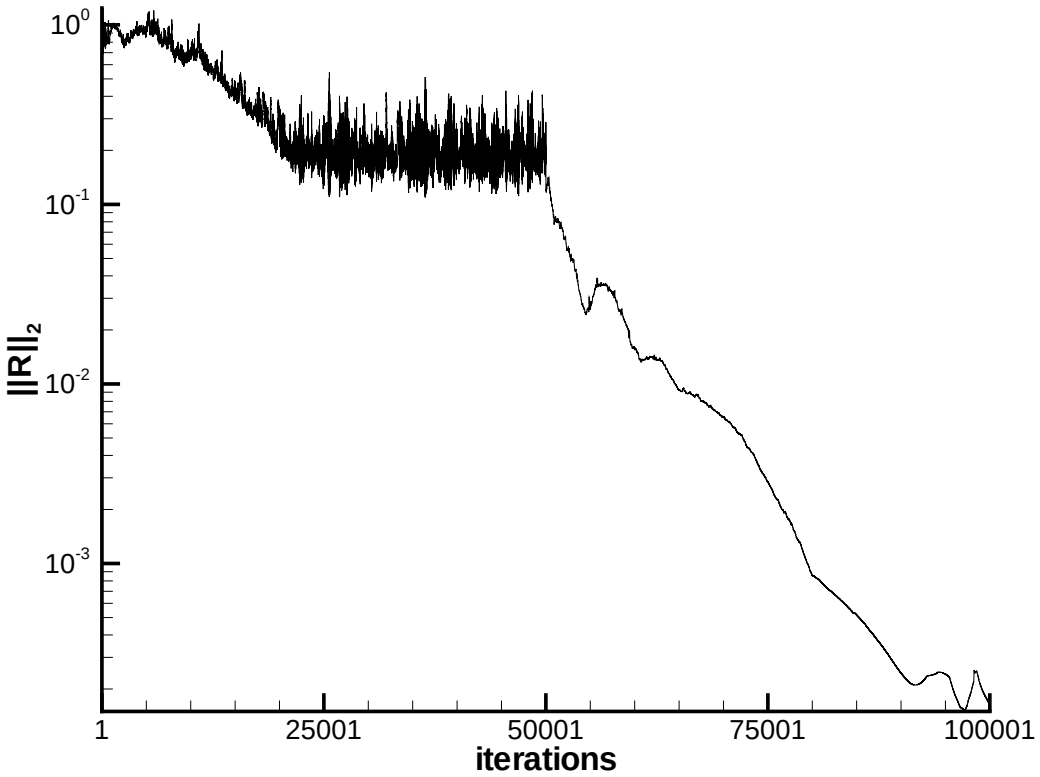

Figure 11: Convergence History for Euler Test

to 0.4, with the time step calculated as

$$\Delta t = \mathrm{CFL} \min_{i,j,k} \left(\left(\frac{|u| + a}{\Delta x} + \frac{|v| + a}{\Delta y} + \frac{|w| + a}{\Delta z} \right)^{-1} \right) \tag{46}$$

Figure 11 is also a good demonstration for why real r-refinement applications require implicit solvers, since the increased resolution results in a very low time step for fixed CFL number.

SIERRA produced a smooth, well-adapted grid in Zone 1, which resulted in a highly mismatched interface between Zones 1 and 2. Figure 12 shows the $K_{\min}$ and $K_{\max}$ faces of Zones 1 and 2. Density contours on the $K_{\min}$ and $K_{\max}$ faces

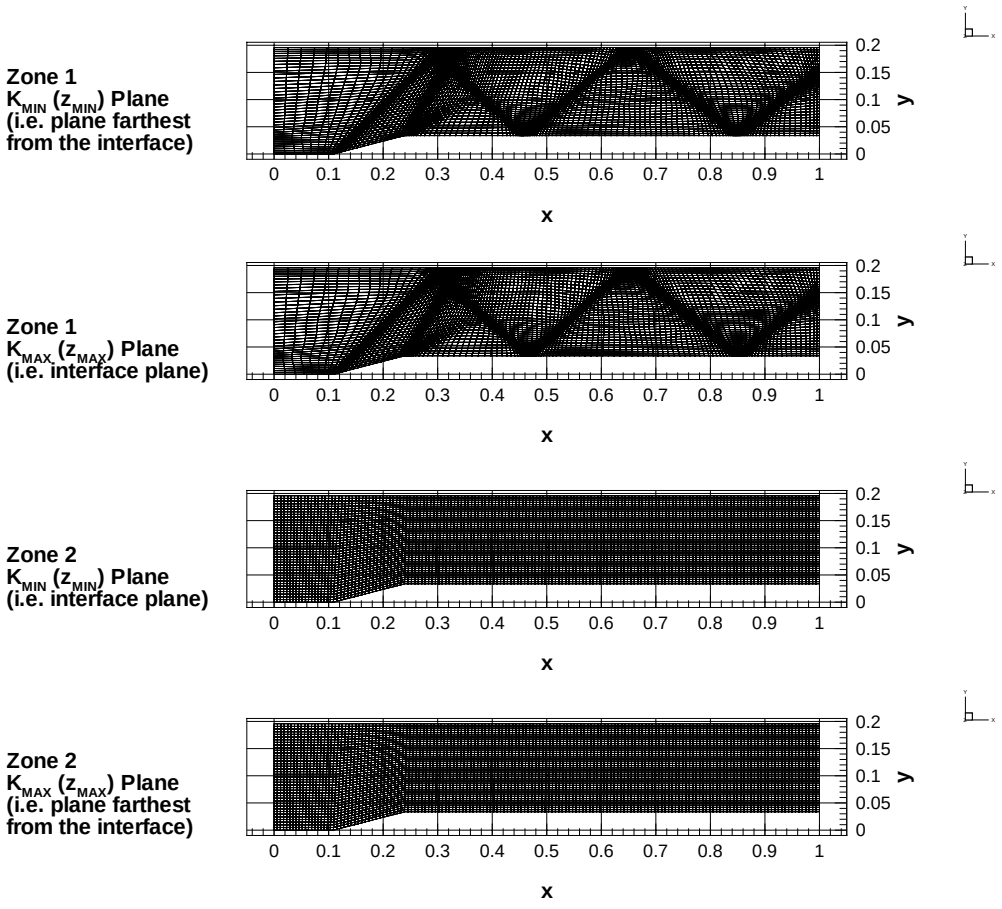

Figure 12: Surface Grids on $K_{\min}$ and $K_{\max}$ Planes of Both Blocks

of Zones 1 and 2 are shown in Figure 13. The solution consists of a shock wave

starting at the bottom of the ramp, reflecting off of the top wall in a Mach reflection, interacting with the expansion fan originating at the top of the ramp, and then going through a series of regular reflections off of the top and bottom walls before impinging on the outlet. The shock waves are very sharp on the $K_{\min}$ plane of Zone 1 because it is far enough away from the interface block that it is hardly affected by the unadapted block. The $K_{\max}$ plane of Zone 1, however, has slightly smeared shock waves because, even though it has the ability to resolve the shocks, it is communicating with a grid that does not have the ability to resolve the shocks. The shocks are excessively smeared on the $K_{\max}$ plane of Zone 2 because of its lack of resolution, but the $K_{\min}$ plane of Zone 2 shows slightly sharper shocks because of the influence of Zone 1. Figures 14 and 15 are close-ups of the Mach reflection in Figures 12 and 13.

The influence of the communication between an adapted and unadapted block on the solution is best seen by slicing the solution in planes normal to the block interface. Figure 16 shows a composite view of density contours on six x-constant planes. With the exception of the inlet plane ($x = 0$), the planes are shown individually in Figure 17.

As can be seen in the figures, there are shock waves in Zone 1 which are smeared close to the interface. Likewise, there are smeared shock waves in Zone 2 which are slightly sharper close to the interface. Because of the large discrepancies in cell size across the interface near these shock waves, the contour lines are not smooth across the interface. However, the centers of the shock waves are clearly at the same place, so the discontinuous contour lines are a result of discrepancies in resolution and not a result of faulty communication between the two blocks.

5.3.5 Computational Costs in the Euler Test

Table 3 gives the percentage of CPU time taken up by the communication algorithm, the adapter, and the remainder of the code (mostly the flow solver). The numbers are a reasonable estimate for the overhead of the communication algorithm

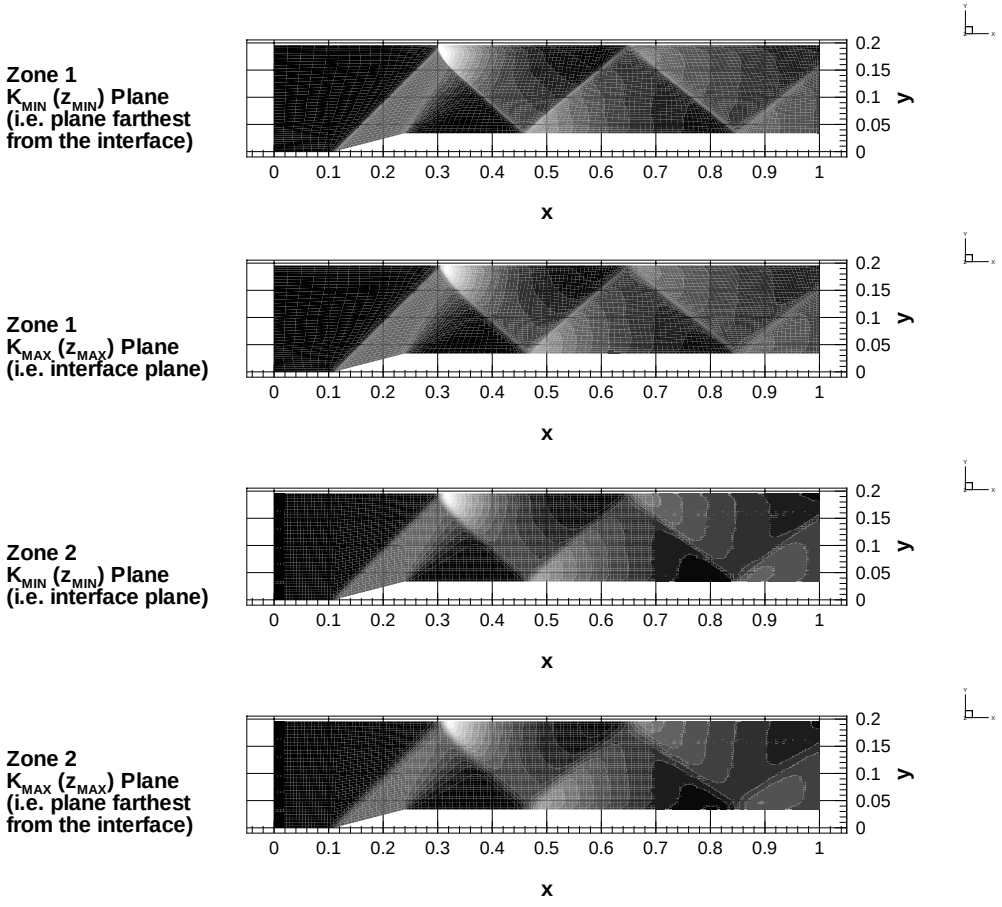

Figure 13: Density Contours on $K_{\min}$ and $K_{\max}$ Planes of Both Blocks

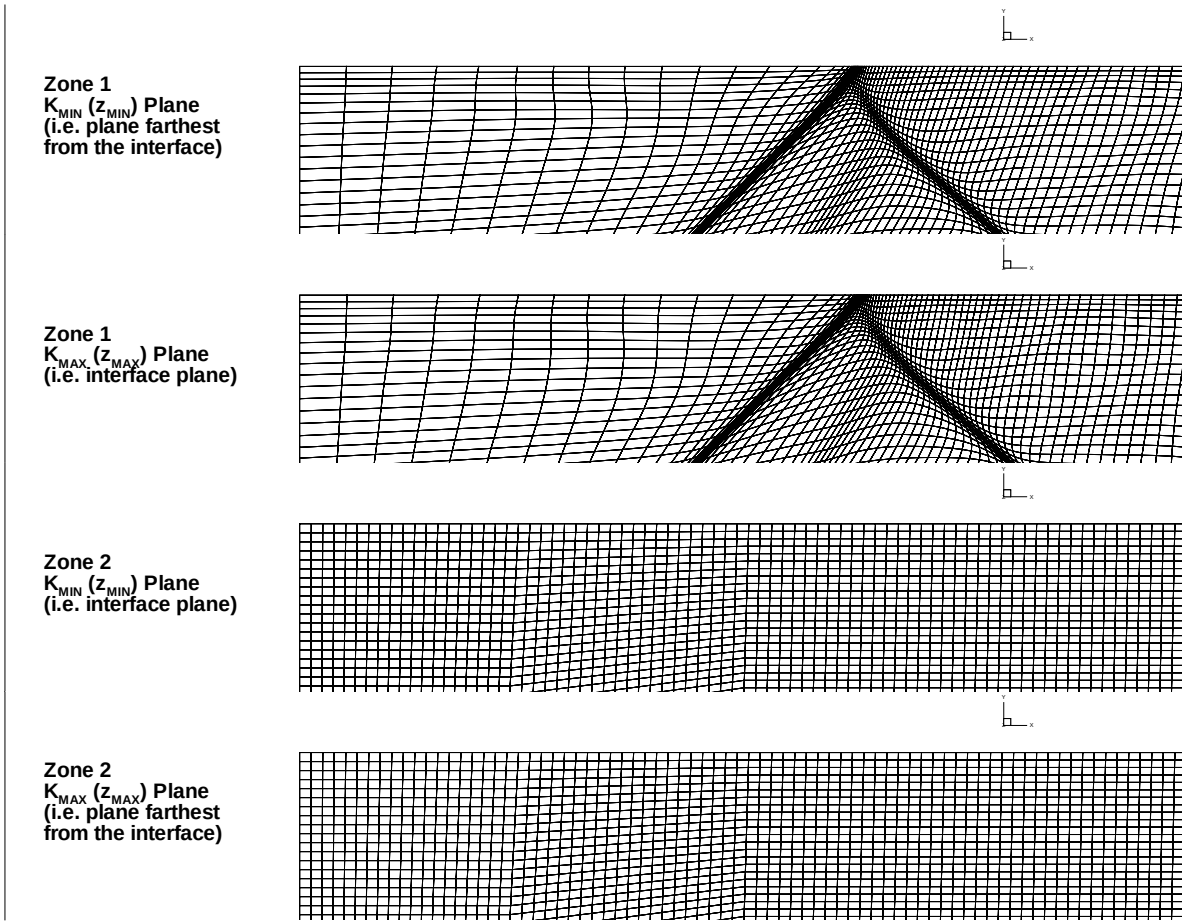

Figure 14: Close-up View of Surface Grids on $K_{\min}$ and $K_{\max}$ Planes of Both Blocks

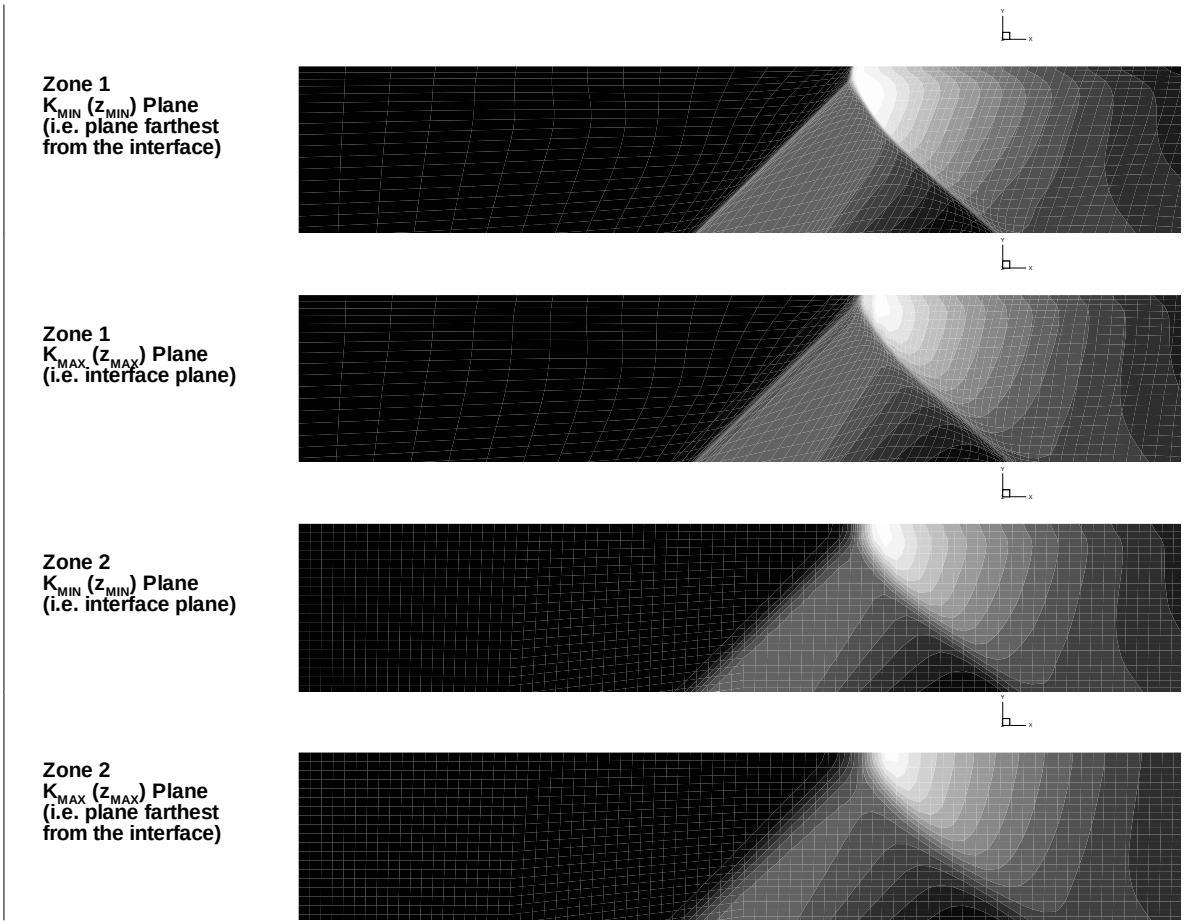

Figure 15: Close-up View of Density Contours on $K_{\min}$ and $K_{\max}$ Planes of Both Blocks

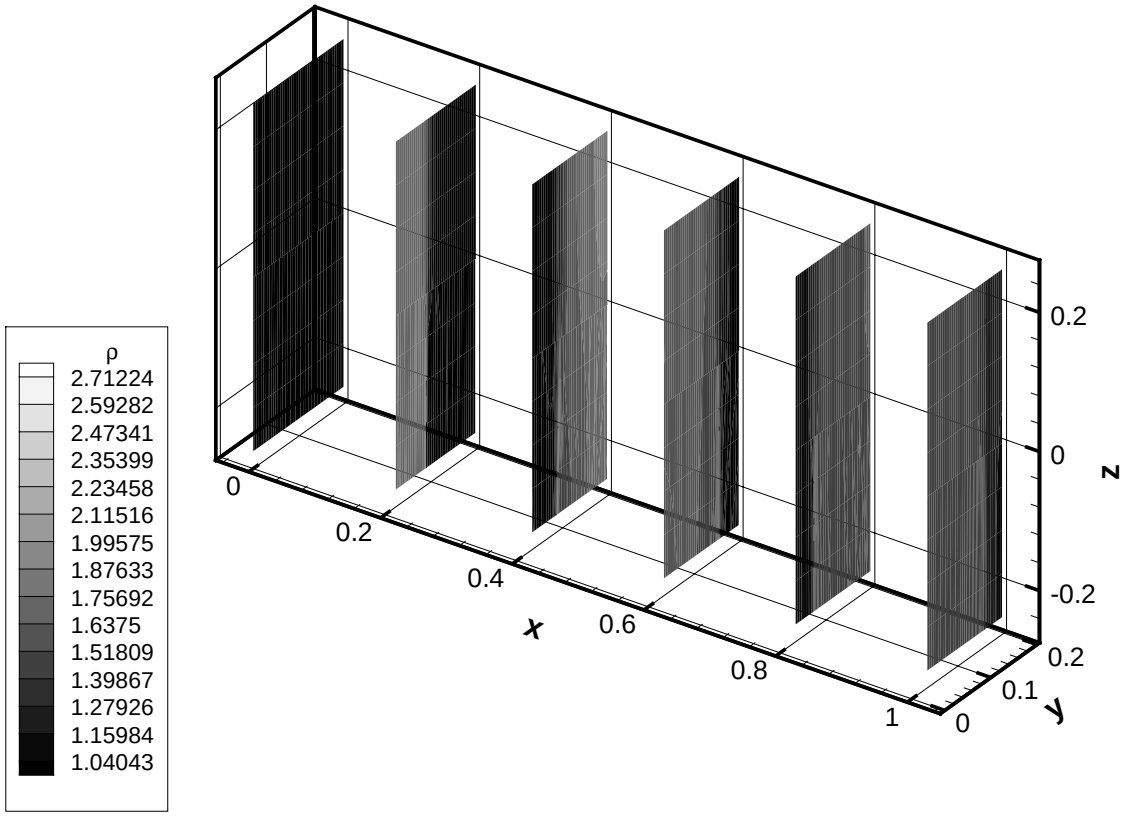

Figure 16: Density Contours on x-constant Slices

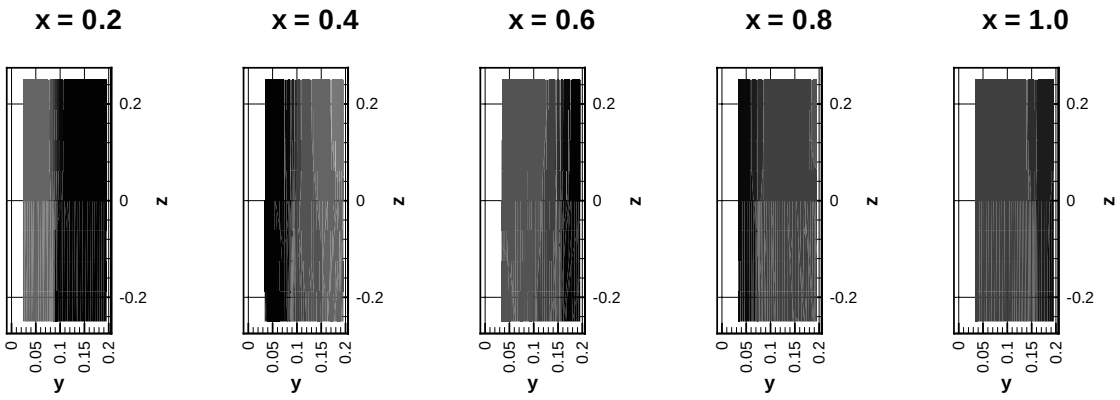

Figure 17: Close-up View of Density Contours on x-constant Slices

portion of code	CPU time (percent)
communication algorithm	30.75
adapter	28.60
remainder	40.65

Table 3: Percentages of CPU Time Expended by Different Parts of Program

and the adapter. However, there would still be communication overhead without the discontinuous boundaries, so it is not reasonable to say that reverting to continuously connected boundaries would completely eliminate this overhead. Likewise, costs in calculating grid metrics and volumes are associated with the adapter, but these would still need to be computed (although only one time) even if the grids were stationary. The costs for the adapter and flow solver scale with the volume (in the computational space) of the grid blocks. The costs for the communication algorithm, however, scale with the area (in the computational space) of the interface between the blocks. Because of this, the communication overhead (as a percentage of the total CPU time) scales with $\frac{\text{total surface area of interfaces}}{\text{total volume of blocks}}$. Thus applications with lower interface area to volume ratios would have less overhead than the test case. This is good, because the overhead of the communication algorithm scales with interprocessor communication for message passing programs, so minimizing the overhead of the interblock communication algorithm and minimizing the message passing overhead are equivalent. Blocks which are cubic (in the computational space) are ideal.

6 Conclusions

A conservative, efficient interblock communication algorithm for dynamically discontinuous block boundary grids has been developed and tested. The algorithm is conservative and efficient. It is also accurate and stable for a number of different applications. For the diffusion equation problem, contour lines remained smooth enough at the interface that the interface location was not distinguishable in the contour plots. For the linear wave equation, that was not the case, but the discrepancies at the interface were no worse than variations in wave front thickness due to variable grid node density elsewhere along the wave front. The interface was also distinguishable in the contour plots of the Euler equations, because only one block was being adapted, creating a huge discrepancy in resolution across the boundary.

The algorithm should work for any conservation-law type PDE, and has been demonstrated on the diffusion equation, linear wave equation, and Euler equations. It can be used for adjacent blocks with differing adaptation requirements, adjacent blocks with one block adapting and the other stationary, or applications where relaxing the continuous connectivity requirement could enhance parallelism. These new developments make r-refinement algorithms such as SIERRA more portable.

The communication algorithm also addresses a fundamental limitation that exists when field equations are mapped onto a parallel computer using domain decomposition. Load leveling would normally require equal subdomain sizes on all processors. Unfortunately, this cannot always be achieved without using non-regular subdivisions of the domain. The resulting subdomains present a difficult problem when information is transferred between processor field updates. The communication algorithm allows the conservative transfer of information between blocks on different processors regardless of the dimension and grid continuity of the individual subdomains that have been assigned to each processor.

7 Future Work

There are two areas of extension for the current work:

1. allowing the use of non-planar interfaces, and

2. allowing communication over part, but not all, of the interblock boundary.

The first area is a fairly simple extension. The Ramshaw algorithm for the current work operates in the physical space. However, any space which is common to the two block boundary grids will suffice, so a parametric space (e.g. s-t space) could be defined and used in the communication algorithm. This would involve the generation of the space, additional terms in the estimation of $\bar{h}$ (the average height of an overlap polyhedron), and some extra bookkeeping and memory management. Additionally, the physical definition of the surface must be defined as some geometric entity (such as a NURBS surface), so that the nodes in the interface grids can be moved along it. Allowing for communication on part but not all of the interface would be more difficult. Such capability would be necessary, for example, in the case of a splitter plate separating two adjacent bays of a supersonic/hypersonic inlet (communication boundary condition upstream of the splitter plate and no-slip boundary condition downstream of the splitter plate). Without this capability, the zone inside each inlet would have to be broken at the start of the splitter plate to handle the change in boundary condition. Breaking the zone would restrict node movement at the splitter plate tip, thus decreasing the amount of adaptation possible. (Accurate simulation of inlet unstart in multiple bay, ganged hypersonic inlets is the final goal for this research.)

References

[1] Laflin, K. R. and McRae, D. S., "Solution Dependent Grid Quality," Proceedings, 6th International Conference on Numerical Grid Generation in Computational Field Simulation, University of Greenwich, London, England, July 6–9, 1998, pp. 119–130.

[2] Meakin, R. L., "A New Method for Establishing Intergrid Communication among Systems of Overset Grids," AIAA 91-1586, June 1991.

[3] Klopfer, G. H. and Molvik, G. A., "Conservative Multizonal Interface Algorithm for the 3-D Navier Stokes Equations," AIAA Paper 91–1601, 1991.

[4] Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors and Difference Schemes." *Journal of Computational Physics*, vol. 43, pp. 357–372.

[5] Vatsa, V. N., et. al. "Navier-Stokes Computations of a Prolate Spheroid at Angle of Attack," *Journal of Aircraft*, vol. 26, pp. 986–993.

[6] Van Leer, B., "Towards the Ultimate Conservative Difference Scheme. V. A Second Order Sequel to Godunov's Method." *Journal of Computational Physics*, vol. 32, pp. 101-136.

[7] Tannehill, J. C., Anderson, D. A., and Pletcher, R. H., *Computational Fluid Mechanics and Heat Transfer, Second Edition*, Taylor and Francis, Washington DC, 1997, p. 397.

[8] Harten, A. and Hyman, J. M., "Self Adjusting Grid Methods for One-Dimensional Hyperbolic Conservation Laws," *Journal of Computational Physics*, vol. 18, pp. 235-269.

[9] Benson, R. A. and McRae, D. S., "A Solution-Adaptive Mesh Algorithm for Dynamic/Static Refinement of Two and Three Dimensional Grids," *Proceedings of the Third International Conference on Numerical Grid Generation in Computational Field Simulations*, Barcelona, Spain, June 1991, pp. 185–199.

[10] Laflin, K. R. and McRae, D. S., "Solver-Independent, Efficient r-Refinement Algorithm (SIERRA)," Proceedings, 6th International Conference on Numerical Grid Generation in Computational Field Simulation, University of Greenwich, London, England, July 6–9, 1998, pp. 109–118

[11] Ramshaw, J. D., "Conservative Rezoning Algorithm for Generalized Two-Dimensional Meshes," *Journal of Computational Physics*, Vol. 59, 1985, pp. 193–199.

[12] Kelley, C. T., *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995, pp. 71–91,135–151.

[13] Chapman, S. J., *FORTRAN 90/95 for Scientists and Engineers*, McGraw-Hill, Boston, 1998, p. 850.