An Efficient Adaptive Grid ENO Scheme

Robert Bruce Bauer*

Abstract

Most numerical schemes are implemented on uniform grids. But often uniform grids, in an attempt to resolve fine-scale features, use a fine uniform mesh and over-resolve areas of the domain. Methods which use non-uniform grids usually maintain the same grid for all time steps. These algorithms require an a priori knowledge of the regions which need a smaller mesh size. There are two possible types of problems where these methods fail to provide an optimal grid. The first is when the area needing a smaller mesh is unknown before the computation. The second is when the area needing greater resolution moves throughout the domain. The method provided here will efficiently find the computational grid and adapt the grid so that the grid is always optimal.

Key words: ENO Schemes, Adaptive Grid, Wavelet Analysis.

AMS subject classifications: 63N30,65N13.

1 Introduction

Essentially Non-Oscillatory (ENO) [1, 2] schemes are good high-order methods for problems with sharp gradients and shocks. However, these high order schemes obtain a solution at the cost more *cpu* time. These schemes require multiple *cpu* intensive *if-then* statements to be executed at each data point and each time step. To reduce the number of computations this method was designed to find an optimal grid which gives uniform accuracy using a minimal number of cells. Section 2 describes many of the concepts and terms associated with ENO schemes.

*Division of Applied Mathematics, Brown University, Providence R.I. 02192.

The method purposed here is an Adaptive Grid Essentially Non-Oscillatory (AGENO) scheme. More specifically, AGENO is the application of an adaptive grid to a *finite volume* ENO scheme. The grid adaption component dynamically changes the grid in order to maintain a nearly uniform truncation error throughout the domain. The uniform error enables the scheme to use a minimal number of cells.

Admissible grids are of the form $x_{j+1} = x_j + 2^{-j}\Delta$, where $j = 0..J$. (J and Δ are user parameters.) Based on these grids, refinement and coarsening processes are defined. Refinement is defined as dividing a cell in half; Coarsening is defined as the combination of two adject cells of equal size.

To demonstrate how the decision whether to refine or coarsen the grid is reached, consider the first order central difference approximation to the first derivative of $f(x)$,

$$(1) \qquad f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} + \frac{(x_{j+1} - x_j)}{2} f''(\xi)$$

where $\xi \in [x_j, x_{j+1}]$. Also let $\hat{f}_j$ be an approximation to $|f''(\xi)|$. Now for a given tolerance ϵ, if $(x_{j+1} - x_j) \leq 2\epsilon/\hat{f}_j$, then

$$(2) \left| f'(x_j) - \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j} \right| \leq \epsilon + \mathcal{O}\left((x_{j+1} - x_j)^2\right).$$

Therefore the truncation error is less than ϵ plus a quantity than decays like Δ^2. But if $(x_{j+1} - x_j) \geq 2\epsilon/\hat{f}_j$, then the truncation error is too large and the grid needs to be refined. The requirements for whether to coarsen the grid are similar. While the above construction is for first order finite difference, Section 3 describes an analogous construction for high order finite volume computations.

The tests for refinement and coarsening also require *if-then* statements. Therefore when and where to apply these tests needs to be considered fully. These decisions are based on the characteristics of the PDE being solved. Advection, diffusion, and the non-linear effects associated with the PDE influence these decisions. Proper application of refinement and coarsening tests ensures that *cpu* time is still minimized. Section 4 describes the actual procedures which change the grid and their application.

The results in one dimension demonstrate the considerable *cpu* time that can be saved using AGENO. AGENO applied to Burgers' equation used one eighth the number of cells and took one tenth the amount of *cpu* time compared to ENO (section 5). Applying AGENO to Euler's gas equations with discontinuous initial conditions lead to *cpu* time of one third of the uniform fine grid computation.

2 ENO Schemes

This paper describes describes an adaptive grid algorithm which is applied to essentially non-oscillatory uniformly accurate scheme for the solution of one dimensional hyperbolic systems of conservative laws.

$$u_t + F_x(u) = 0 \qquad u(x,0) = u_0(x) \tag{3}$$

Here $u = (u_1, u_2, \ldots, u_m)^T$ is the state vector and $F(u)$, the flux, is a vector valued function with m components. Let

$$\mathbf{A}(u) = \frac{\partial}{\partial u} F(u). \tag{4}$$

Since this paper is only concerned with hyperbolic systems, $\mathbf{A}(u)$ has m real eigenvalues, $\{\lambda_k(u)\}_{k=1}^m$ and a complete set of right and left eigenvectors, $\{\mathbf{r}_k(u)\}_{k=1}^m$ and $\{\mathbf{l}_k(u)\}_{k=1}^m$. Furthermore assume that $\mathbf{r}_j \mathbf{l}_k = \delta_{jk}$.

Examples 2.1

i. Inviscid Burger's equation, $F(x) = x^2$,

$$u_t + \left(u^2\right)_x = 0$$

ii. One dimensional Euler gas equations for conservation of density, momentum and energy

$$\mathbf{w}_t + \mathbf{F}(\mathbf{w})_x = 0$$

with

$$\mathbf{w} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix}$$

where ρ *is the density,* m *is the momentum, and* E *is the energy. Also*

$$\mathbf{F}(\mathbf{w}) = u\mathbf{w} + \begin{pmatrix} 0 \\ P \\ Pu \end{pmatrix}$$

with $u = m/\rho$ *(velocity) and* $P = (\gamma - 1)\left(E - \frac{1}{2}\rho u^2\right)$ *(pressure) and* γ *is the polytropic gas constant.*

2.1 PDE attributes

In order to design an adaptive grid algorithm for a numerical method, it is important to understand the PDE which it is solving. This section briefly describes the different characteristics of PDE, so that refinement and coarsening of the grid can be done intelligently.

<u>Advection</u>–the wave-like propagation of features throughout the domain. Fine-scale features will propagate throughout the domain, requiring either refinement and coarsening of the grid.

<u>Non-linear effects</u>–formation of fine structure from otherwise smooth data. The formation of shocks and contact discontinuities are both examples of non-linear effects. This will may require refinement of the grid anywhere in the domain.

<u>Dissipation</u>–viscous forces smoothing out features in the domain. This will only require coarsening of the grid anywhere in the domain.

2.2 Cell Averages

Instead of solving the partial differential equation (3), the cell averaged formulation of the partial differential equation is solved instead. The cell averaged formulation is obtained by integrating the PDE over any connected subset of the domain. ie. for the problem on $[0,1]$, integrate over $[a,b]$, where $0 \le a < b \le 1$. This yields the cell average form of the PDE

$$\frac{d}{dt}\int_a^b u(x,t)dx = F(u(b,t)) - F(u(a,t)). \tag{5}$$

Now discretize using the points $x_{j+1/2}$, where $x_{-1/2} = 0$, $x_{N+1/2} = 1$, and $x_{j-1/2} < x_{j+1/2}$. Furthermore define

$$\bar{u}_j(t) = \int_{x_{j-1/2}}^{x_{j+1/2}} u(x,t)dx, \tag{6}$$

and

$$\bar{u}_j^0 = \int_{x_{j-1/2}}^{x_{j+1/2}} u(x,0)dx, \tag{7}$$

This produces the more familiar cell average form of the PDE

$$\begin{aligned} \frac{d}{dt}\bar{u}_j(t) + F(u(x_{j+1/2},t)) - F(u(x_{j-1/2},t)) &= 0 \\ \bar{u}_j(0) &= \bar{u}_j^0 \end{aligned} \tag{8}$$

Physically this can be thought of as the quantity within cell j changes by the amount of the quantity entering from the left and exiting from the right.

2.3 Formalization of Grid

With the discretazition defined above, additional grid definitions are necessary. Let $C_j = [x_{j-1/2}, x_{j+1/2}]$. Restrict the cell sizes such that,

$$|C_j| = 2^{-w_j}\,\Delta \qquad \sum_j |C_j| = 1. \tag{9}$$

where $w_j \in [0, S]$, $\Delta = \frac{1}{R}$. R is the coarsest resolution and S is the number of levels of refinement. Also define $x_j = (x_{j-1/2} + x_{j+1/2})/2$ and $\Delta_j = (x_{j-1/2} - x_{j+1/2})$. The set of all grids which meet these conditions is defined as $\mathcal{G}_0$. Equation (6) can be re-written as

$$\bar{u}_j(t) = \int_{x \in C_j} u(x,t)dx. \tag{10}$$

Throughout Δ will be referred to for the analysis some interpolation errors. To obtain more exact error quantities, products of different Δ_j's should be used instead of the products of Δ. But it is the feeling of the author that it is sufficient to bound the errors using Δ.

2.4 Reconstruction

Equation (8) requires the evaluation of $F(u(x_{j+1/2}, t))$. However only a knowledge of $\bar{u}_j$ is available. Therefore a reconstruction of $u(x)$ from $\bar{u}_j$ is required. Define $R(x; \bar{u})$ as the interpolating polynomial of degree p which reconstructs $u(x)$ from $\bar{u}$. Also let $R_j(x; \bar{u})$ be the interpolation polynomial of degree p which reconstructs $u(x)$ on the cell C_j. Then $R(x; \bar{u}) = R_j(x; \bar{u})$ for $x \in C_j$.

Properties 2.1 *Reconstruction via. ENO Interpolation $R(x; \bar{u})$ satisfies:*

i. At all points x for which there is a neighborhood where $u(x)$ is smooth

$$R(x; \bar{u}) = u(x) + \mathcal{O}(\Delta^p) \tag{11}$$

ii. R is conservative.

$$\overline{R(x_j; \bar{u})} = \bar{u}_j \tag{12}$$

iii. R(x;ū) is essentially non-oscillatory.

$$TV(R(x; \bar{u})) \le TV(u) + \mathcal{O}((\Delta)^p) \tag{13}$$

Property iii ensures that the reconstruction R is essentially non-oscillatory. This guarantees that $R(x)$ does not have Gibbs-like oscillations of $\mathcal{O}(1)$. However $R(x)$ may have oscillation on the order of Δ^p. [2, 3, 4].

2.5 Conservative Numerical Schemes

Equation (8) gives rise to a semi-discrete numerical scheme,

$$\frac{d}{dt}\bar{u}_j(t) = -\left(\hat{f}_{j+1/2} - \hat{f}_{j-1/2}\right) \tag{14}$$

where $\hat{f}_{j+1/2}$ is an estimate of $F(u(x_{j+1/2}))$. Define

$$\hat{f}_{j+1/2} = \mathcal{F}\left[R_j(x_{j+1/2}, t), R_{j+1}(x_{j+1/2}, t)\right]. \tag{15}$$

$\mathcal{F}[u, v]$ is called the numerical flux. There are two conditions on the numerical flux.

i. <u>Consistency</u>

$$\mathcal{F}[u, u] = F(u)$$

ii. <u>Lipschitz Continuous</u>

$$|\mathcal{F}[u, v_1] - \mathcal{F}[u, v_2]| \le K|v_1 - v_2|$$
$$|\mathcal{F}[u_1, v] - \mathcal{F}[u_2, v]| \le K|u_1 - u_2|.$$

2.5.1 Specific Fluxes

Define the first order reconstruction $R_j(x, \bar{u}) = \bar{u}_j$. Using this example of a reconstruction will make examples of specific fluxes easier.

Examples 2.2 (Scalar Flux Functions)

- *Lax Friedrich*

$$\mathcal{F}[\bar{u}_j, \bar{u}_{j+1}] = \frac{1}{2}\left(F(\bar{u}_{j+1}) + F(\bar{u}_j) + \alpha(\bar{u}_{j+1} - \bar{u}_j)\right)$$
$$\alpha = \max_u |F(u)|$$

- *Local Lax Friedrich*

$$\mathcal{F}[\bar{u}_j, \bar{u}_{j+1}] = \frac{1}{2}\left(F(\bar{u}_{j+1}) + F(\bar{u}_j) + \alpha_j(\bar{u}_{j+1} - \bar{u}_j)\right)$$
$$\alpha_j = \max_{x \in C_j} |F(u(x))|$$

- *Godunov*

$$\mathcal{F}[\bar{u}_j, \bar{u}_{j+1}] = \begin{cases} \min_{\bar{u}_j \le u \le \bar{u}_{j+1}} F(u) & \text{if } \bar{u}_j < \bar{u}_{j+1} \\ \max_{\bar{u}_j \le u \le \bar{u}_{j+1}} F(u) & \text{if } \bar{u}_j \ge \bar{u}_{j+1} \end{cases}$$

- *Roe*

$$\mathcal{F}[\bar{u}_j, \bar{u}_{j+1}] = \frac{1}{2}\left(F(\bar{u}_{j+1}) + F(\bar{u}_j) + |\overline{\alpha_j}|\,(\bar{u}_{j+1} - \bar{u}_j)\right)$$
$$\overline{\alpha_j} = \begin{cases} \frac{F(\bar{u}_{j+1}) - F(\bar{u}_j)}{\bar{u}_{j+1} - \bar{u}_j} & \text{if } \bar{u}_j \neq \bar{u}_{j+1} \\ F'(\bar{u}_j) & \text{if } \bar{u}_j = \bar{u}_{j+1} \end{cases}$$

The computation of the flux function for a system of equations is not much different than for scalar equations. The most common flux used for systems is the Roe flux. The Roe flux requires that the absolute value of the Jacobian Matrix $\mathbf{A}(u)$ to be computed. The first example is the extension of Roe's scalar flux function to systems. This flux function does violate entropy conditions of the PDE. Therefore Roe's scheme with an Entropy fix is also presented. For further information see [4, 3, 5].

Examples 2.3 (System Flux Functions)

- *Roe (without Entropy fix)*

  For Roe's scheme a value $\hat{u}(u_1, u_2)$ is computed such that $\mathbf{A}(\hat{u}(u_1, u_2))$ satisfies the mean value relation

  $$F(u_2) - F(u_1) = \mathbf{A}(\hat{u}(u_1, u_2))(u_2 - u_1) \tag{16}$$

  Then the flux function can be define as

  $$\mathcal{F}[u_1, u_2] = \frac{1}{2}\left[F(u_1) + F(u_2) - \sum_{k=1}^{m} \alpha_k\, |\lambda_k(\hat{u})|\, r_k(\hat{u})\right]$$
  $$u_2 - u_1 = \sum_{k=1}^{m} \alpha_k r_k(\hat{u})$$

- *Roe (with Entropy fix)*

  $$\mathcal{F}[u_1, u_2] = \frac{1}{2}\left[F(u_1) + F(u_2) - \sum_{k=1}^{m} \alpha_k \Lambda_k r_k(\hat{u})\right]$$
  $$u_2 - u_1 = \sum_{k=1}^{m} \alpha_k r_k(\hat{u})$$
  $$\Lambda_k = \max(\,|\lambda_k(\hat{u})|, \beta_k)$$

  β_k is a non-negative quantity measuring the violation of the entropy condition in the k^{th} characteristic [4, 3, 5].

2.5.2 Time Integration

Equation (14) is a system of ordinary differential equations for the variables $\bar{u}_j$. To solve these systems any of the many ODE solvers can be used. The third order Total Variation Diminishing Runge-Kutta [6] was used for all computations performed for this paper.

2.6 ENO Interpolation

The goal is to create an interpolation algorithm which satisfies properties 2.1. This has been accomplished by Harten el al. [3, 2, 4]. Since only the cell averages of the function $u(x)$ are known, the reconstruction must be carried out using $u(x)$'s primitive function. Define

$$U(x) = \int_0^x u(y)dy. \tag{19}$$

Using the values $\bar{u}_j$, $U(x)$ is know at the distinct points $x_{j+1/2}$,

$$U(x_{j+1/2}) = U(x_{j-1/2}) + \bar{u}_j. \tag{20}$$

Interpolation is carried out on the points $U_{j+1/2} = U(x_{j+1/2})$.

Now consider the interpolation which is carried out on cell j. For obvious reasons the points $x_{j-1/2}$ and $x_{j+1/2}$ must be used. Using just these two points, the interpolation of $U(x)$ is second order and the reconstruction of $u(x)$ is first order. For a p^{th} order reconstruction, a $(p+1)^{st}$ order interpolation of $U(x)$ is required. Therefore for a p^{th} order reconstruction, $p+1$ points of $U_{j+1/2}$ are required. Adding the requirement that the $p+1$ points are all next to each other, there are p possible stencils.

The algorithm for determining the smoothest polynomial is defined recursively. Once the smoothest q^{th} order polynomial has been found, the $(q+1)^{st}$ order polynomial is easily determined.

i. Given a q^{th} order polynomial $V_j(x, q, l_q(j))$ defined on the cell C_j which interpolates $U(x)$ at the points $\{x_{j+l_q(j)-1/2}, \ldots x_{j+l_qj+q+1/2}\}$, then

   (21a) $U_s = V_j(x_s, q, l_q(j))$
   (21b) $s = j + l_q(j) - 1/2, \ldots, j + l_q(j) + q + 1/2$

   $l_q(j)$ is a pointer to the left-most cell used for the q^{th} order interpolating polynomial on cell j.

ii. Compute

   (22a) $U_L = U[x_{j+l_q(j)-3/2} \cdots x_{j+l_q(j)+q+1/2}]$
   (22b) $U_R = U[x_{j+l_q(j)-1/2} \cdots x_{j+l_q(j)+q+3/2}]$,

where U[...] are the standard un-even divided differences.

(23a) $U[x_j \dots x_{j+k}] = \dfrac{U[x_j \dots x_{j+k-1}] - U[x_{j+1} \dots x_{j+k}]}{x_{j+k} - x_j}$

(23b) $U[x_j] = U(x_j)$

U_L is the smoothness of the $(q+1)^{st}$ derivative if the next point added to the stencil is $x_{j+l_q(j)-3/2}$. Similarly U_R is the smoothness of the $(q+1)^{st}$ derivative if the next point added to the stencil is $x_{j+l_q(j)+3/2}$.

iii. If $|U_L| < |U_R|$ then the algorithm will add the point $x_{j+l_q(j)-3/2}$ to the stencil. If $|U_R| \geq |U_L|$ then the algorithm will add the point $x_{j+l_q(j)+q+3/2}$ to the stencil. Therefore

$$(24) \qquad l_{q+1}(j) = \begin{cases} l_q(j) - 1 & \text{if } |U_L| < |U_R| \\ l_q(j) & \text{if } |U_R| \geq |U_L| \end{cases}$$

Defining $l_{q+1}(j)$ defines the polynomial $V_j(x, q+1, l_{q+1}(j))$.

Once the desired order has been reached, the function $u(x)$ can easily be recovered.

$$(25) \qquad R_j(x, \bar{u}) = \frac{d}{dx} V_j(x, p, l_p(j))$$

Step 3 will choose a stencil which will avoid crossing a shock if possible. If $U(x)$ is in $\mathcal{C}^\infty$ then

$$(26a) \qquad U[x_j, \dots, x_{j+k}] = \frac{1}{k!} \frac{d^k}{dx^k} U(\xi)$$

$$(26b) \qquad \xi \in [x_j, x_{j+l}]$$

If $U^{(q)}(x)$ is discontinuous within the interval $[x_j, x_{j+l}]$, then for $k \geq q$

$$(27) \qquad U[x_j, \dots, x_{j+k}] = \mathcal{O}\left(\Delta^{q-k} \lfloor U^q \rceil\right)$$

where $\lfloor U^q \rceil$ is the jump in the q^{th} derivative. Now consider two intervals, the first $[x_{j_1}, x_{j_1+k}]$ where the function $U(x)$ is smooth and the second $[x_{j_2}, x_{j_2+k}]$ where the function $U^{(q)}(x)$ has a discontinuity. Then for Δ sufficiently small $|U[x_{j_1}, x_{j_1+k}]| < |U[x_{j_2}, x_{j_2+k}]$.

Harten [3] shows that where $u(x)$ is smooth

$$(28) \qquad \frac{d^q}{dx^q} R_j(x; \bar{u}) = \frac{d^q}{dx^q} U(x) + \mathcal{O}(\Delta^{p+1-q})$$

for $q = 0, \dots, p$. Harten also showed that the reconstruction $R_j(x; \bar{u})$ is essentially non-oscillatory.

Note: Since all interpolation is of order p, reconstruction is of order $p+1$. The notation $l(j)$ will denote the quantity $l_{p+1}(j)$.

2.7 Reconstruction Accuracy

Let $R_j(x; \bar{u})$ be a function of order p which estimates the function $u(x)$ on the cell C_j given the cell averages $\bar{u}_j$. Then for $x \in C_j$ and $u(x)$ smooth for $x \in [x_{j-l(j)-1/2}, x_{j-l(j)+p+1/2}]$, then

$$(29) \quad R_j(x; \bar{u}) = u(x) + \mathcal{K} u^{(p)}(\xi_j) \prod_{k=j-l(j)-1/2}^{j-l(j)+p+1/2} (x - x_k)$$

where $\xi_j \in [x_{j-l(j)-1/2}, x_{j-l(j)+p+1/2}]$, and $\mathcal{K}$ is a constant independent of $u(x)$ and the cell sizes, $l(j) \in [0 \dots p]$.

However if $u^{(p)}(x)$ is discontinuous for $x \in [x_{j-l(j)-1/2}, x_{j-l(j)+p+1/2}]$, then the error will not be bounded so nicely. In this situation, the error can be larger around the discontinuity, and therefore the grid should be refined. The algorithm developed will refine around discontinuities.

2.8 Reconstruction for Systems

The reconstruction described so far is for scalar reconstruction. It may appear that the reconstruction for a system could be simply applied on each component of $u = (u_1, u_2, \dots, u_m)^T$. However many people have noted that applying reconstruction to the characteristics provides a better approximation. Therefore define the reconstruction $R_j(x, \bar{u})$ as

$$(30a) \qquad \bar{v}_k^j = \sum_i \{\bar{u}_k\}_i \, \mathbf{l}_i(\bar{u}_j)$$

$$(30b) \qquad R_j(x, \bar{u}) = \sum_i \{R_j(x, \bar{v}_k^j)\}_i \, \mathbf{r}_i(\bar{u}_j)$$

Harten [3, 4] shows the results of applying the ENO interpolation on the components and characteristics. Both give good results, but the characteristic method provides much better results.

3 Error Control

Most schemes are design to have a certain order of accuracy. The goal in designing this scheme is to create a method which has nearly uniform accuracy throughout the domain while using a minimal number of cells[1]. The nearly uniform accuracy is obtained by refining the grid in

[1] Schemes usually have a certain order of accuracy such that the error decays like h^p at points where the solution is smooth. The goal of this scheme is to have the error distributed throughout the domain.

areas of lower accuracy, and coarsening the grid in areas of higher accuracy.

First consider the truncation error when $u(x)$ is smooth. The error term in equation (29) is controlled by two factors, $u^{(p)}(x)$ and the sizes of cells $\{C_k\}_{k=j-l(j)}^{j-l(j)+p}$. This requires an estimate of $u^{(p)}(\xi)$. Define $Q_j(\bar{u})$ and $\sigma_k(j, C_{j-p/2}, \ldots, C_{j+p/2})$ such that

$$Q_j(\bar{u}) = \sum_{k=j-p/2}^{j+p/2} \bar{u}_k \; \sigma_k(j), \tag{31}$$

$Q_j\left(\overline{x^q}\right) = 0$, for $q = 0, \ldots, p-1$, and $Q_j\left(\overline{x^p}\right) = p!$. Then $Q_j(\bar{u}) \approx u^{(p)}(x_j)$, and $Q_j(\bar{u}) \approx u^{(p)}(\xi_j)$ for $\xi_j \in C_j$.

3.1 $\sigma_k(j)$ Specifics

$\sigma_k(j, C_{j-p/2}, \ldots, C_{j+p/2})$ is a whole class of filters. The coefficients are a function of the p neighboring cells. Because of the dependence on these cells there are many coefficients to compute (roughly $p^p + 1$.) Even so the computational costs are not great because all these coefficients can be pre-computed and stored.

3.1.1 Relationship to Wavelets

Lee Jameson [7] developed an algorithm which has a similar goal to this algorithm. The goal was to find a method to which would determine an optimal computational grid on which to compute a finite difference scheme. Jameson used wavelets to determine the computational grid. Wavelets (See [8, 9, 10, 11]) are compactly supported bases functions which also have multi-resolution analysis properties. One important characteristic of wavelets is that the wavelet basis function is orthogonal to the polynomials $1,\ x,\ x^2,\ \ldots,\ x^q$, for some specified q. Jameson used wavelet transforms to determine the local regularity of a function $v(x,t)$. With a knowledge of the local regularity of the function an optimal computational grid was computed. Because of the time evolution of the solution $v(x,t)$ a re-computation of the optimal grid was required periodically. This re-computation involved interpolating the function from the optimal grid onto a uniform fine grid.

Example 3.1 *Let the computational grid be the points* $\{0, 1/16, 1/8, 1/4, 1/2, 1\}$. *For the re-computation of the grid, the discrete function* v_k, *defined on the computation grid, needed to be interpolated onto the finest uniform grid,* $x = j/16$ *for* $j = 0, 16$. *Then a new computational non-uniform grid was computed.*

Jameson's work showed that this computational grid would provide a solution equally accurate as the solution computed on the uniform fine grid. Jameson's algorithm used far less computations. But the interpolation onto the uniform fine grid for the re-computation of the optimal grid required more computations marginally slows the algorithm down.

The filters $\sigma_k(j)$ designed here are similar to those of wavelets. The discrete filters $\sigma_k(j)$ are orthogonal to the cell averages of the polynomials $1,\ x,\ x^2,\ \ldots,\ x^{p-1}$. But because they are designed to be applied to all the possible non-uniform grids in $\mathcal{G}_0$, the need to interpolate onto the uniform fine grid has been eliminated.

3.2 Accuracy of $Q_j(\bar{u})$

This section computes estimates on the accuracy of $Q_j(x)$ to estimate $u^{(p)}(x)$ for $x \in C_j$. It is easy to find such an estimate for when $u(x) \in \mathcal{C}^{p+1}$. The estimate is not as nice for $u(x) \notin \mathcal{C}^{p+1}$, but this does not hurt the algorithm, rather it helps the algorithm.

$\mathcal{C}^q$ is the space of continuous functions which have q continuous derivatives.

$\mathcal{C}^q[\Omega]$ is the space of continuous functions which have q continuous derivatives for $x \in \Omega$.

3.2.1 $u(x) \in \mathcal{C}^{p+1}$

In order to find the accuracy of $Q_j(\bar{u})$, define

$$\Omega_j = \bigcup_{k=j-p/2}^{j+p/2} C_k. \tag{32}$$

From the definition of $\sigma_k(j)$, it is obvious that

$$Q_j(\bar{u}) \le u^{(p)}(x_j) + \mathcal{K}_1 \; u^{(p+1)}(\zeta_j)|\Omega_j|^{p+1} \tag{33a}$$
$$\zeta_j \in \Omega_j \tag{33b}$$

where $\mathcal{K}_1$ is independent of $u(x)$ and p. Also

$$u^{(p)}(\xi_j) \le u^{(p)}(x_j) + \mathcal{K}_2 \, u^{(p+1)}(\hat{\zeta}_j) \; |\Omega_j|^{p+1} \tag{34a}$$
$$\xi_j \in C_j \tag{34b}$$
$$\hat{\zeta}_j \in \Omega_j \tag{34c}$$

where $\mathcal{K}_2$ also is independent of $u(x)$ and p. Therefore

$$Q_j(\bar{u}) = u^{(p)}(\xi_j) + \mathcal{O}(|\Omega_j|^{p+1}) \tag{35a}$$
$$\forall \xi_j \in C_j \tag{35b}$$

3.2.2 Non-smooth $u(x)$

In the previous section, it was shown that $Q_j(\bar{u}) \approx u^{(p)}(x_j)$ when $u(x)$ is smooth. Now consider the case where $u^{(q)}(x)$ is discontinuous for $x \in C_j$. Define the sequence of functions $H_q(x)$ such that

$$H(x) \quad \text{is Heavyside function} \tag{36a}$$

$$H_0(x) = H(x) \tag{36b}$$

$$H_{q+1}(x) = \int H_q(y)dy \tag{36c}$$

Then rewrite $u(x)$ as

$$u(x) = u_s(x) + A\, H_q(x) \tag{37}$$

where $u_s(x) \in \mathcal{C}^\infty$. Then

$$Q_j(\bar{u}) = Q_j(\bar{u}_s) + A\, Q_j(\bar{H}_q) \tag{38}$$

From [3, 4],

$$Q_j(\bar{H}_q) = \mathcal{O}(\Delta^{q-p}). \tag{39}$$

For sufficiently small Δ, equation (38) can be rewritten as

$$Q_j(\bar{u}) = \mathcal{O}(A\, \Delta^{q-p}). \tag{40}$$

Therefore $Q_j(\bar{u})$ is a good estimate of $u^{(p)}(x)$ if $u(x) \in \mathcal{C}^{p+1}[\Omega_j]$, and otherwise $Q_j(\bar{u})$ is large. It is easy to verify that $Q_k(\bar{u})$ will also be large for $k \in [j - p/2, \leq j + p/2]$. Also if there exists a discontinuity in $u^{(p)}(x)$ for $x \in \Omega_j$ then Q_j will be large.

3.3 Error Indicators

The error term in equation (29) is a combination of $u^{(p)}(\xi_j)$ and a function of the cell sizes, therefore define

$$\delta_j = \max_{x \in C_j} \left\{ \left| \prod_{k=j-p/2}^{j+p/2} (x - x_k) \right| \right\}. \tag{41}$$

δ_j is a function of the cell sizes which, when combined with an estimate of $u^{(p)}(x)$, gives an upper bound on the error. δ_j can either be pre-computed and stored. In practice this is not done. To save the small amount of memory needed a quick estimate is used.

$$\epsilon = \sum_{k=-p/2}^{p/2} w_i/p \tag{42a}$$

$$\delta_j \leq 2^{-\epsilon} \Delta \left(\frac{p}{2}\right)! \tag{42b}$$

Then define

$$\overline{Q}_j = \delta_j \max_{k \in [j-p/2, j+p/2]} \{|Q_k|\} \tag{43}$$

$\overline{Q}_j$ is an bound on the error resulting from a centered stencil to reconstruct $u(x)$ on cell C_j. The magnitude of $\overline{Q}_j$ will determine whether cell C_j is refined or coarsened.

3.4 Centered Stencil Use

All estimates of the error are computed using a centered stencil. But why use a centered stencil to compute Q_j and $\overline{Q}_j$ when the reconstruction in section 2.4 is done using ENO interpolation?

$\overline{Q}_j$ is an estimate for interpolation over cell j. The ENO reconstruction error will be bounded by $\overline{Q}_j$ if the centered stencil is chosen from the p possible stencils. There are two possible ways that ENO could choose a centered stencil.

The first is if there is a discontinuity in $u^{(p)}(x)$ for $x \in C_j$, $[0 \leq q \leq p]$. The algorithm would like to divide any cell which has a discontinuity within it. Fortunately both Q_j and $\overline{Q}_j$ will be large, and cell C_j will be divided. Moreover, Q_k will be large for $j - p/2 \leq k \leq j + p/2$. Therefore the cells $\{C_{j-p/2}, \ldots, C_{j+p/2}\}$ will also be divided.

The cells which are refined around cell C_j above are a buffer zone of higher resolution. A buffer zone is a region of non-required higher resolution next to a region judged to need higher resolution.

Examples 3.1 (Buffer Zone)

i. *If for $x \in [.3, .4]$ the domain needs to be refined, it is better to refine an area larger than $[.3, .4]$. The actual buffer zone is not described in terms of a distance, but is only defined vaguely as a zone of non-required higher resolution.*

ii. *The need for a buffer zone can be seen by examining the following problem.*

$$u_x = u_t$$
$$u(x, t = 0) = \begin{cases} 0 & \text{if } x < .5 \\ 1 & \text{if } x \geq .5 \end{cases}$$

If there is no buffer zone present, this computational grid will only have a very few small cells right at the shock. But the advection of the shock will move the shock into the neighboring larger cells. However with a buffer zone, the shock will move into the small cells of the buffer zone.

Another way the ENO reconstruction could use a centered stencil is if the function $u(x) \in \mathcal{C}^{p+1}[\Omega_j]$. If this is the case then $\overline{Q}_j$ provides a good bound on the reconstruction error.

However what if ENO chooses a non-centered stencil? Then the centered stencil reconstruction error bound, Q_j, is a rough estimate of the ENO reconstruction error. Both the smooth and non-smooth cases are examined.

First consider the case where there is a discontinuity in $u^{(p)}(x)$ $[0 \leq q \leq p]$ in the cell $C_{k \neq j}$ and $C_k \subset \Omega_j$. Because of equation (40), Q_j will be large. The stencil for cell C_j will be non-centered, because the ENO interpolation will avoid using cell C_k. ENO will avoid cell C_k because an interpolating polynomial using cell C_k would be oscillatory and violate property 2.1.iii. Therefore Q_j will be larger than the actual reconstruction error using ENO reconstruction. Therefore the estimate $\overline{Q}_j$ will be also larger, causing cell C_j to be refined. The refinement of C_j creates a buffer zone around cell C_k. This buffer zone is the same buffer zone described several paragraphs ago.

Also consider the case where $u(x) \in \mathcal{C}^{p+1}[\Omega_j]$. Then there are many possible stencils for which the ENO interpolation might use for reconstruction. Instead of trying to determine how close $\overline{Q}_j$ might be to an actual error bound, consider that $\overline{Q}_j$ acts as a warning light for when refinement *might* be necessary. Without knowing more information about $u(x)$ for $x \in \Omega_j$ or which stencil the ENO interpolation would choose, no bound can be obtained for the ENO interpolation Instead the estimate $\overline{Q}_j$ is used as a rough estimate. To determine whether this estimate is useful or not consider that $\overline{Q}_j$ probably is a good estimate to the reconstruction error from the ENO interpolation. If anything $\overline{Q}_j$ is larger than the actual error, because ENO interpolation seeks the smoothest polynomial. To get a better bound, significantly more computations will be required to compute the ENO interpolation and then find the error bound. Therefore $\overline{Q}_j$ will be used as the indicator for refinement and coarsening of the grid.

4 Grid Adaption

4.1 Actual Computational Grid

Initially $\mathcal{G}_0$ may seem to be the best set to use, but $\mathcal{G}_0$ admits many undesirable grids. Some of these undesirable grids are polarized grids which can be created created by refinement and coarsening of even grids. An example of a polarized un-even grid is where $w_{2i} = 0$ and $w_{2i+1} = 2$. This grid is less desirable than a uniform grid with $w_i = 1$. Beyond existence of polarized grids, initial numerical computations often created these grids. To eliminate these polarized grids from the set of admissible grids, define $\mathcal{G}_1 \subset \mathcal{G}_0$.

Definition: $\mathcal{G}_1$ is a subset the set grids of $\mathcal{G}_0$ such that given a grid $g \in \mathcal{G}_1$ and C_i is a cell of g with $w_i < S$, then the grid which is the result of dividing cell C_i in half is also a member of $\mathcal{G}_1$. The grid with $w_i = 0$ for $i = 1 \ldots R$ is in $\mathcal{G}_1$.

Figure 1 provides a detailed example of the set $\mathcal{G}_1$. This should provide a good example of the grids contained in $\mathcal{G}_1$. Figure 2 shows some examples of grids which are in $\mathcal{G}_0$ but not in $\mathcal{G}_1$, plus some more examples of grids in $\mathcal{G}_1$. Both of these figures should help provide a clearer understanding of the set $\mathcal{G}_1$, and why it is necessary to use this subset of $\mathcal{G}_0$. $\mathcal{G}_1$ does not eliminate all polarized grids nor does it eliminate all undesirable grids, but it does provide a better computational grid than $\mathcal{G}_0$. Enforcement of this requirement is difficult and requires additional *if-then* statements. Therefore application of the coarsening algorithm is restricted.

4.2 Adaptive Procedures

4.2.1 Refinement

Now with a measure of the error, the cell sizes can be adjusted to control the error of the scheme. Given a *user-defined* tolerance, **TOL**($\approx 10^{-5}$), for refining a cell if $|\overline{Q}_j(u)| >$**TOL**, then refine cell C_j. Refinement consists of creating two cells out of cell j. Refinement requires interpolating $\bar{u}_j$ to find the cell averages of the two new cells. This should be done using an ENO interpolation.

Denote the refinement process as a mapping $\mathcal{R} : \mathcal{G} \overset{u(x)}{\rightarrow} \mathcal{G}$. ie. $\mathcal{R}$ maps one grid into another, and which new grid it maps to is dependent on the function $u(x)$. By inspection, it is obvious that $\mathcal{R} : \mathcal{G}_0 \rightarrow \mathcal{G}_0$ and $\mathcal{R} : \mathcal{G}_1 \rightarrow \mathcal{G}_1$.

4.2.2 Coarsening

While it is important to keep the error within bounds, minimizing of the number of cells is just as important. Therefore combining cells, or coarsening of the grid, is necessary. If $|\overline{Q}_j(u)| <$**tol** and $|\overline{Q}_{j+1}(u)| <$ **tol**, then combine the cells C_j and C_{j+1}. (**tol** is a *user supplied* cut-off for combining cells $\approx 10^{-7}$) This coarsening of the grid will be denoted, $\mathcal{C} : \mathcal{G} \overset{u(x)}{\rightarrow} \mathcal{G}$.

Other than checking $|\overline{Q}_j(u)|$, there are a few other things which need to be checked before combining two cells. For two adjacent cells to be combined, they must have the same size. Without this requirement, newly formed cells

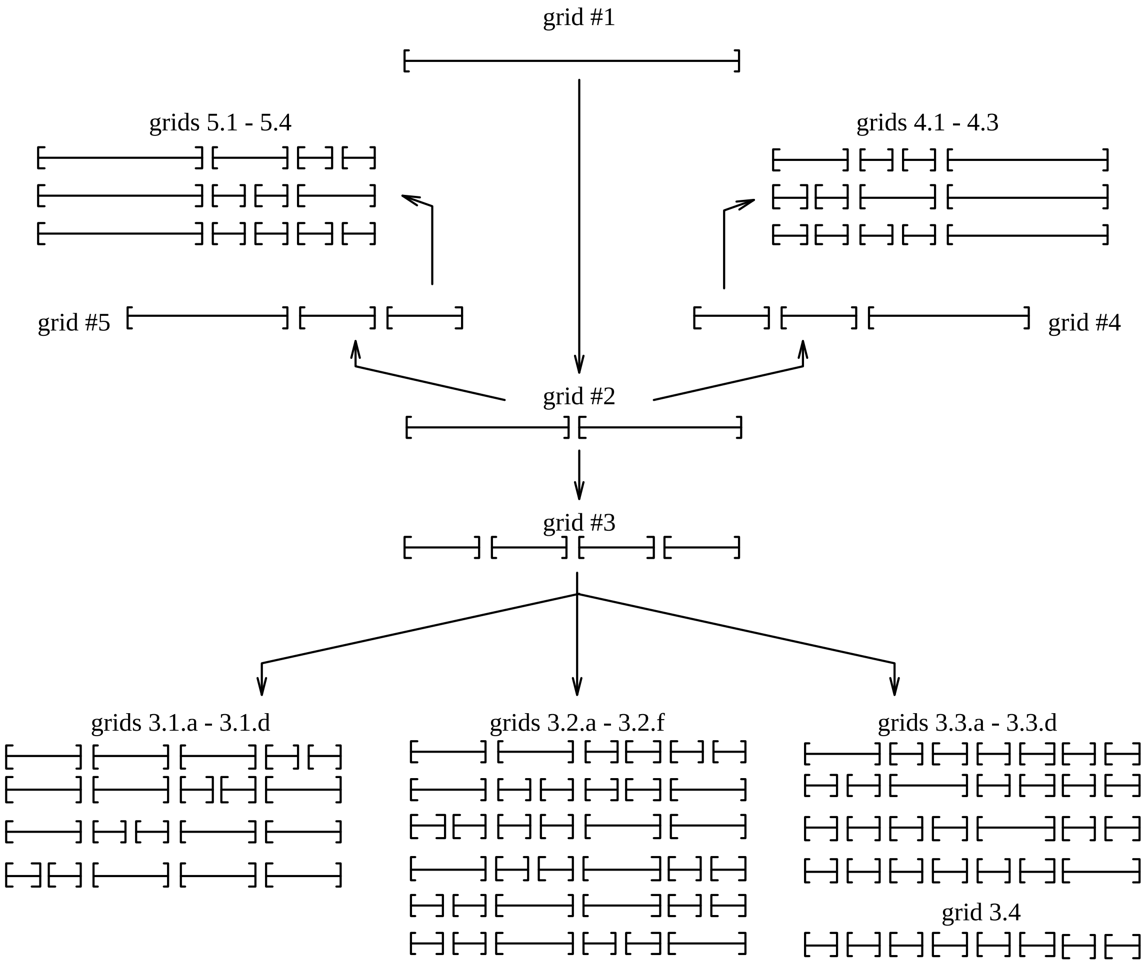

Figure 1: Example of $\mathcal{G}_1$ for $R = 1$ and $S = 3$. The first element in $\mathcal{G}_1$ is the grid with only one cell (*grid #1*). The second element results from splitting grid #1 into two equal sized cells (*grid #2*). The next three grids in $\mathcal{G}_1$ result from splitting both cells of grid #2 (*grid #3*), the left cell of grid #2 (*grid #4*), or the right cell of grid #2 (*grid #5*). Working on grid #3, there are many different combinations of cells to split. Grids 3.1.a – 3.1.d result from splitting only one of the cells in grid #3. Grids 3.2.a – 3.2.f result from splitting any two of the cells in grid #3. Grids 3.3.a – 3.3.d result from splitting any three of the cells in grid #3. Grid 3.4 results from splitting all four of the cells in grid #3. Now consider the different ways to split the smaller two left cells of grid #4. The three different possibilities for this splitting of cells from grid #4 results in grids 4.1 – 4.3. (Note: Splitting the right cell of grid #4 would result in grid #3.) A similar splitting of the two smaller right most cells of grid #5 produces grids 5.1 – 5.3. $|\mathcal{G}_1| = 26$ for $R = 1$ and $S = 3$. Also $|\mathcal{G}_1| = 677$ for $R = 1$ and $S = 4$.

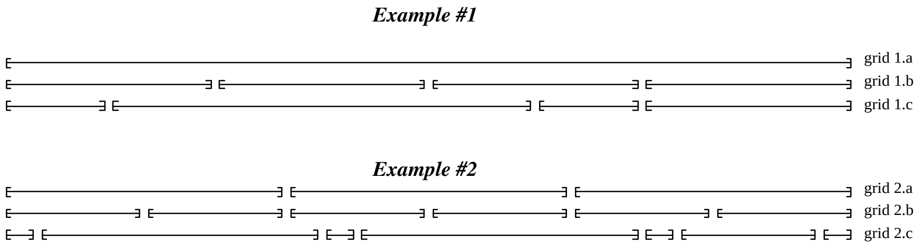

Figure 2: Examples of grids in $\mathcal{G}_0$ which are excluded from $\mathcal{G}_1$. **Example 1** ($R = 1$ and $S = 3$): Grid 1.a and grid 1.b are elements of $\mathcal{G}_1$, grid 1.c is not. Grid 1.c is undesirable because the rapid change from a very small grid cell to a larger grid cell back to a very small grid cell. Grid 1.c cannot be created by refining grid 1.a. Grid 1.b is the equivalent of Grid #3 in previous figure and is a more desirable grid. One reason that grid 1.b is prefered over grid 1.c is that there is less change of cell sizes, while giving equivalent approximation errors. **Example 2** ($R = 3$ and $S = 3$): Grids 2.a and 2.b are both elements of $\mathcal{G}_1$. Grid 2.c is not in $\mathcal{G}_1$ and is undesirable because to the rapid change from cells of size 1/24 to cells of size 1/3. Grid 2.b is a better choice of grids over grid 2.c because grid 2.b lacks any change in cell size over the entire domain, while grid 2.c rapidly changes. Grids such as 1.c and 2.c only occur if the user does not monitor the coarsening subroutine to exclude these undesirable grids.

would not have widths consistent with equation (9). The next requirement on joining two cells is that the resulting grid be in $\mathcal{G}_1$.

4.3 Procedures Applications

4.3.1 Resolution Boundaries

There are three different procedures which are used to change the grid. Generic refinement and coarsening have just been discussed. These procedures are applied to the entire grid and are the basic methods for updating the grid. While using only these two procedures to change the grid will work, it is inefficient. The PDE's advective forces move structures throughout the domain and require many checks of the grid. Using a knowledge of the PDE, a more efficient algorithm can be designed.

Consider the set of cells which are on the interface between two different resolutions. Define the set $\mathcal{B}$ as

$$\mathcal{B} = \{C_i | w_i < w_{i+1} \quad or \quad w_i < w_{i-1}\} \tag{44}$$

Example 4.1
Consider the PDE $u_t = u_x$ with initial conditions such that the right part of the domain needs one level of refinement to maintain accuracy. Then $w_i = 0$ for $i = 1 \dots N_0$ and $w_i = 1$ for $i = (N_0 + 1) \dots N$. Then $\mathcal{B} = \{C_{N_0}\}$.

These cells will be called the boundary cells. If the the problem is purely advective, then the only cells which need to be refined are contained in $\mathcal{B}$. Checking only the cells in $\mathcal{B}$ eliminates a majority of the computations that would be needed to check the entire grid.

Example 4.2
Consider the PDE

$$u_t = u_x \tag{45a}$$

$$u(x,0) = \begin{cases} 0 & if\ x < .5 \\ 1 - \cos(20\pi x) & if\ .5 \le x < .6 \\ 0 & if\ .6 \le x \end{cases} \tag{45b}$$

Outside the region $[.5, .6]$ the cells will be large, and within the region $[.5, .6]$ the cells will be very small. The boundary cells will on the boundary of the region $[.5, .6]$. As the numerical method steps forward in time, the features in the region $[.5, .6]$ are going to propagate to the left. In order to maintain proper resolution, the first large cell to the left of the fine-scale features will need to be refined[2]. This cell will be in $\mathcal{B}$. As time progresses, the set $\mathcal{B}$ will move to the left, just in front of the propagating wave.

4.3.2 Complete Refinement

This procedure of checking the boundary between resolution will ensure proper resolution will maintained for the advective forces. But boundary refinement will not ensure

[2] The actual application to this problem will also have a buffer zone. So in fact the propagation of the features will move into the smaller cells of the buffer zone. Therefore the boundaries cells are actually used to adjust the buffer zone, keeping it ahead of the fine scale features.

proper resolution of fine scale features which are the result of the non-linear effects of the PDE. Consider Burger's equation with $u(x, t = 0) = \sin(2\pi x)$. The smooth initial conditions will lead to $\mathcal{B}$ be empty, but it is well known that a shock will form and refinement will be needed. Therefore, non-linear effects require complete refinement of the domain. The entire domain will be checked every T_R time steps. $T_R \approx [5, 50]$.

4.3.3 Complete Coarsening

While it is important to keep the number of cells to a minimum, performing the coarsening algorithm is a costly computation. The additional computations needed to perform $\mathcal{C}$ arise from the requirement that $\mathcal{C} : G_1 \to G_1$. Therefore complete refinement[3] is done less often. The entire grid is coarsened every T_C time steps. $T_C \approx [10, 50]$.

4.4 Number of computations

Let N_e be the number of cells used for a uniform grid ENO scheme, and N_a be the average number of cells used by an AGENO scheme. Let T be the number of intermediary steps for the Runge-Kutta time stepping algorithm. I_0 is ratio of the time to perform 1 *if-then* to the time to perform 1 *multiplication*. I_0 is estimated to be 5. [See table 1] The number of computations per time step on an uniform grid will be $N_e T(2p + I_0)$. [See table 2] The total number of computations for only the time stepping for the AGENO scheme will be $2pN_aT$. Let the average number of cell which are refined and coarsened as $\{R_0 \cdot N_a\}$ and $\{C_0 \cdot N_a\}$, respectively. The number of computations for the total refinement will be $N_a [2p + 2I_0 + R_0(2p + 5 + I_0)]$ and will be only be performed every T_R time steps. The number of computations for the coarsening will be $N_a [2p + I_0(2 + C_0(5p + 4))]$ and will be performed every T_C time steps. Define $\{B_0 \cdot N_a\}$ and $\{B_1 \cdot N_a\}$ as the average size of $\mathcal{B}$ and the average number of boundary cells refined, respectively. The number of computations to check the boundary will be $N_a \cdot B_0(2p+I_0)$ and to refine the boundary will be $N_a \cdot B_1(2p + 5 + I_0)$. The boundary checking is performed every T_B time steps. Therefore for an adaptive grid with N_a cells the number of computations will be

$$\begin{aligned} & N_a\{2pT + [2p + 2I_0 + R_0(2p + 5 + 2I_0)]/T_R \\ & \qquad + [2p + I_0(2 + C_0(5p + 4))]/T_C \\ (46) \qquad & \qquad + B_0 (2p + I_0)/T_B + B_1 (2p + 5 + I_0)/T_B\} \end{aligned}$$

[3] Complete coarsening works on the entire grid the same as complete refinement

Values used for Computations	
T	3
p	4
R_0	.01
C_0	.01
B_0	.1
B_1	.01
T_R	10
T_C	50
T_B	1

Table 3: Values used for comparison of number of computations needed. T, p, are the actual values used form computations. R_0, C_0, B_0 and B_1 are conservative averages of actual values recorded solving Burger's equation with initial value a square wave.

Using the values from table 3, if $N_e < 1.12N_a$, then the number of computations to be reduced. This is a very reasonable requirement, and is easily satisfied.

All these computations assume that the number of time steps for the adaptive grid and the uniform grid are the same. This is not necessarily going to be true. The adaptive grid will allow the largest time step be taken each time. Consider Burger's equation with initial conditions $u(x, t = 0) = \sin(2\pi x)$. Initially the adaptive grid program will use a coarse grid, which will allow a larger time step. As the shock develops, and the grid is refined, the time step will be reduced. The uniform grid computation will need to use the same smaller time step for all steps.

5 Applications to the Inviscid Burgers' Equation

The AGENO scheme was applied to Burger's equation with periodic boundary conditions.

$$u_t(x, t) + (u^2(x, t))_x = 0 \tag{47a}$$

$$u(x, t = 0) = \frac{1}{3} + \frac{2}{3}\sin(2\pi x) \tag{47b}$$

Three different sets of run-parameters were used to demonstrate possible settings. (See Table 4). The final results of number of cells used, number of cells added and joined, the size of $\mathcal{B}$ and the number of time steps needed are shown in table 5.

The figures on page 14 are the results from run–2. Figure 3 show the solutions at $t = 1.5$, $t = 2.5$, and $t = 5.0$. Fig-

Table of constants		
Constant	Definition	Typical Values
R	the number of cells on the coarsest resolution	32–128
S	the number of level of refinement	3–6
T	the number of steps to perform 1 full time step	1–4
p	the order	3–5
I_0	ratio of the time to perform 1 *if-then* to the time to perform 1 *multiplication*	≈ 5
N_e	the number of cells for a uniformly fine grid	$R\ 2^S$
N_a	the average number of cells used per time step	——
$R_0 \cdot N_a$	the average number of cells refined each full refinement time	$0 - .1$
$C_0 \cdot N_a$	the average number of cells combined each coarsening time	$0 - .1$
$B_0 \cdot N_a$	the average size of $\mathcal{B}$ over all time steps	$0 - .125$
$B_1 \cdot N_a$	the average number of boundary cells to be refined each time	$0 - .02$
T_R	the number of time steps between full refinements	$\approx 5 - 30$
T_C	the number of time steps between coarsening	$\approx 15 - 50$
T_B	the number of time steps boundary refinements	1–2

Table 1: List of constants which are used in computations

Number of computations					
Procedure	*Multiplies*	*Additions*	*if-then*	Points	Regularity [a]
ENO [b]	p	p	p	N	T
Computation of $\overline{Q}_j(u)$	p	p	0	N	$1/T_R$ or $1/T_C$
$\lvert\overline{Q}_j(u)\rvert > TOL$	0	0	1	N	$1/T_R$
$\lvert\overline{Q}_j(u)\rvert < tol$	0	0	1	N	$1/T_C$
Refinement [c]	p	$p+5$	0 or p	R_0	$1/T_R$
Coarsening [d]	0	5	$5p$	C_0	$1/T_C$
Find Boundary	0	0	1	N	$1/T_R$ or $1/T_C$
Check Boundary	p	p	1	B_0	$1/T_B$
Refine Boundary	p	$p+5$	0 or p	B_1	$1/T_B$

[a]Number of times per time-step this subroutine needs to be called.
[b]For computation of $F_x(u)$
[c]Depends on whether method is a fixed stencil or ENO
[d]Extra computations required to ensure that $\mathcal{C} : \mathcal{G}_1 \to \mathcal{G}_1$

Table 2: Table listing number of computations required to perform each of the different processes used in this adaptive grid method.

ure 4 show the pointwise errors for the solutions in figure 3. Figure 5 plots the cell sizes at the chosen times. Figure 5 shows that refinement not only took place around the shock, but that coarsening also took place behind the shock. At $t = 2.5$ and $t = 5.0$ the method uses all four levels of refinement to ensure accuracy around the shock. But at $t = 1.5$ (when the shock just formed and is small in magnitude,) figure 5 shows that only two levels of refinement are used there.

Tables 6, 7, and 8 show the errors and *cpu* times for these runs compared to uniform grid computations. All times are from computations on a `Sparc 10`. Two different errors were computed. The first error is the L_2 norm between the exact solution and the computed solution on 768 evenly spaced points throughout the domain. The second error, L_2^*, is the L_2 norm between the exact solution and the computed solution on the set of evenly spaced points at a distance greater than .03 away from a shock.

Burger's Equation – Time = 1.5

Non-Adaptive (ENO)

Cells	L_2	L_2^*	Time	Steps
48	4.1×10^{-2}	1.0×10^{-2}	1 sec	38
96	2.3×10^{-2}	3.0×10^{-3}	3 sec	77
192	1.4×10^{-2}	8.9×10^{-4}	9 sec	153
384	7.4×10^{-3}	2.8×10^{-4}	34 sec	306
768	5.4×10^{-3}	8.8×10^{-5}	129 sec	612

Adaptive (AGENO)

Cells	L_2	L_2^*	Time	Steps
run–1	1.5×10^{-2}	1.1×10^{-3}	1 sec	50
run–2	1.5×10^{-2}	1.1×10^{-3}	1 sec	50
run–3	4.0×10^{-3}	1.4×10^{-4}	8 sec	215

Table 6: Errors and *cpu* times for Adaptive and Non-Adaptive computations using high-pass filter based adaptive grid ENO scheme applied to Burger's equation at $t = 1.5$

5.1 Euler Gas Dynamics

The AGENO scheme was applied to the one dimensional Euler gas equations for conservation of density, momentum and energy

$$\mathbf{w}_t + \mathbf{F}(\mathbf{w})_x = 0$$

with

$$\mathbf{w} = \begin{pmatrix} \rho \\ m \\ E \end{pmatrix}$$

where ρ is the density, m is the momentum, and E is the energy. Also

$$\mathbf{F}(\mathbf{w}) = u\mathbf{w} + \begin{pmatrix} 0 \\ P \\ Pu \end{pmatrix}$$

with $u = m/\rho$ (velocity) and $P = (\gamma - 1)\left(E - \frac{1}{2}\rho u^2\right)$ (pressure) and $\gamma = 1.4$ is the polytropic gas constant. The initial conditions are

$$\rho(x, t = 0) = \begin{cases} 1.0 & \text{if } x \leq 1/2 \\ 0.1 & \text{if } x > 1/2 \end{cases} \tag{48a}$$

$$m(x, t = 0) = 0.0 \tag{48b}$$

$$E(x, t = 0) = \begin{cases} 2.5 & \text{if } x \leq 1/2 \\ .25 & \text{if } x > 1/2 \end{cases} \tag{48c}$$

Both third order AGENO scheme and a uniform grid third order ENO scheme with equivalent resolutions were applied. The AGENO scheme used 37% the number of cells as the ENO scheme and 36% of the *cpu* time. Again all times are from computations on a `Sparc 10`. Figure 6 plots the AGENO solution versus the ENO solution. The solutions are indistinguishable from each other and provide a good estimate of the exact solution.

Table 9 provides the parameters which where used for the AGENO computations and table 10 provides numerical results of number of cells used, *cpu* time, and average number of cells added, number of cells joined, and size of boundary. Unfortunately AGENO fails to combine enough cells. In figure 7 the error indicator $\overline{Q}_j$ is plotted at $t = .4$. While the solution looks constant between the shocks, the computed solution has small oscillations on the order of Δ^3 which cause the grid to remain over-resolved. To overcome this problem a modification of the error indicator will be required. Even so, AGENO produces a solution in one third the time of typical ENO scheme.

6 Conclusion

The AGENO scheme described here provides a quick algorithm for adapting the computational grid. By using a quick centered difference approximation to the error, the grid is easily changed to always be optimal.

This adaptive grid algorithm has been applied to test its robustness on other type schemes. Flux limiter schemes, finite difference and finite element schemes all proved successful tests for the adaptive grid algorithm [12]. However, because of the nature of the non-uniform grid generated, it is not possible to apply this adaptive grid technique to *pointwise* ENO schemes.

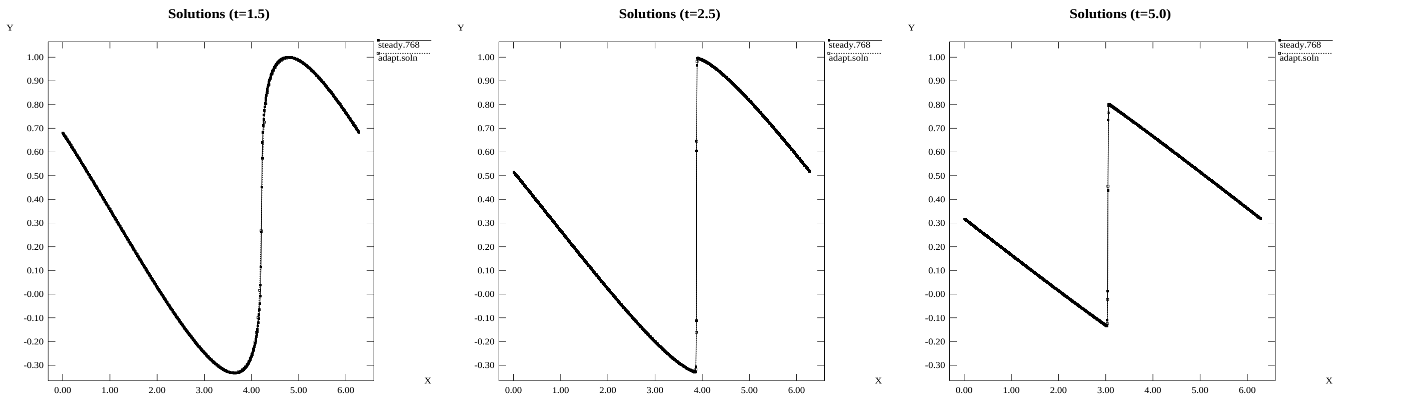

Figure 3: Solutions to Burger's equations at times 1.5, 2.5 and 5.0 with initial conditions $u(x, t = 0) = 1/3 + 2/3\sin(x)$.

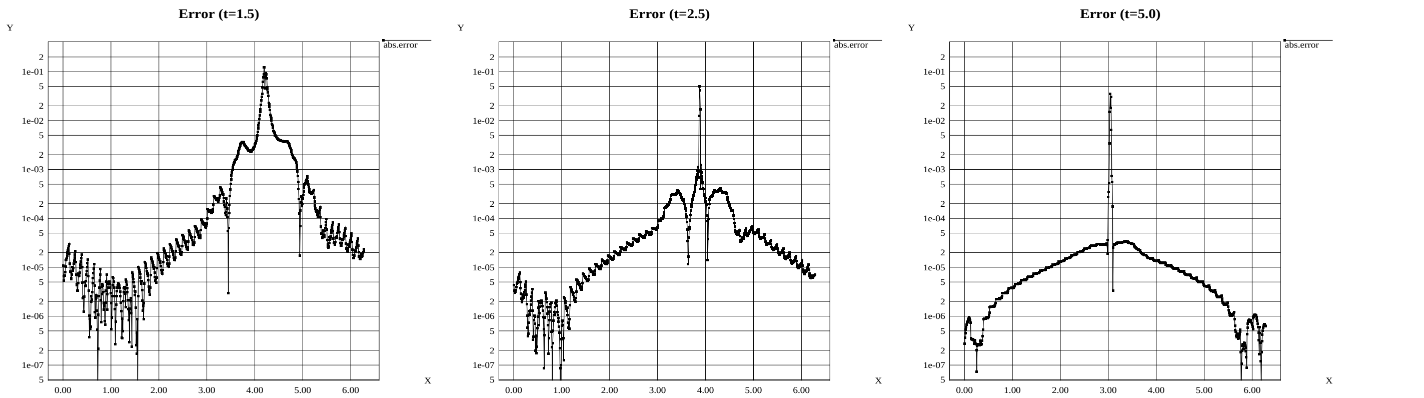

Figure 4: Computed errors

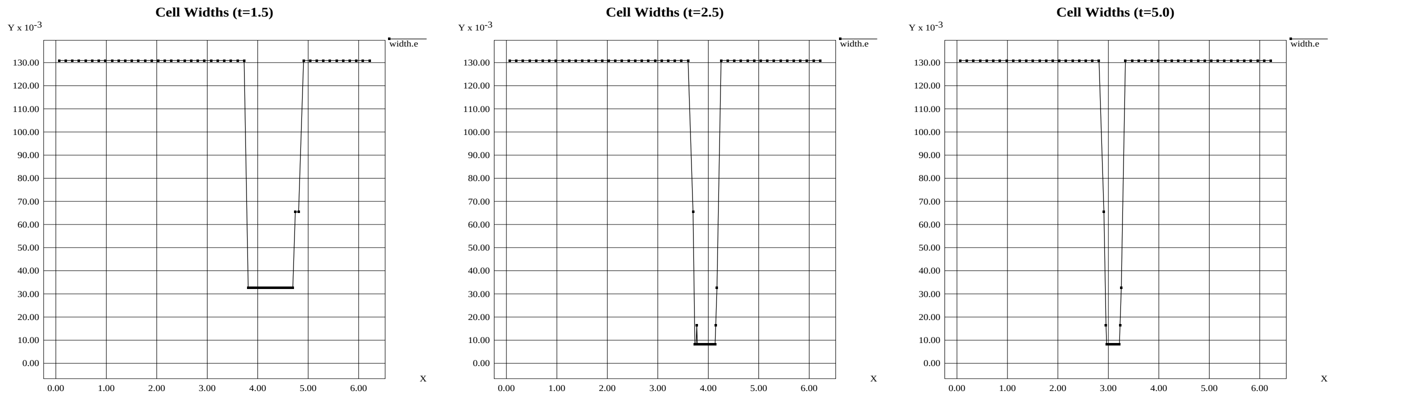

Figure 5: Cell Sizes

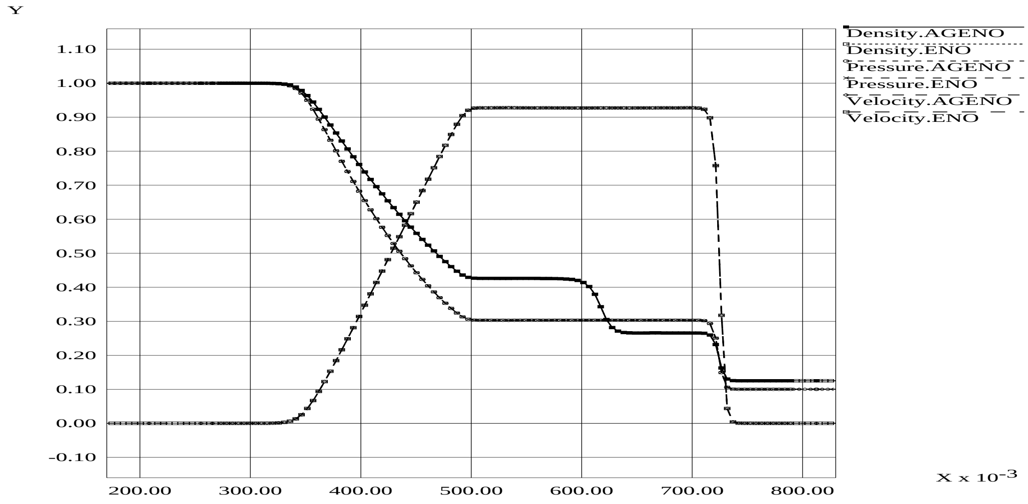

Figure 6: Solution to Euler gas equations using both AGENO and ENO.

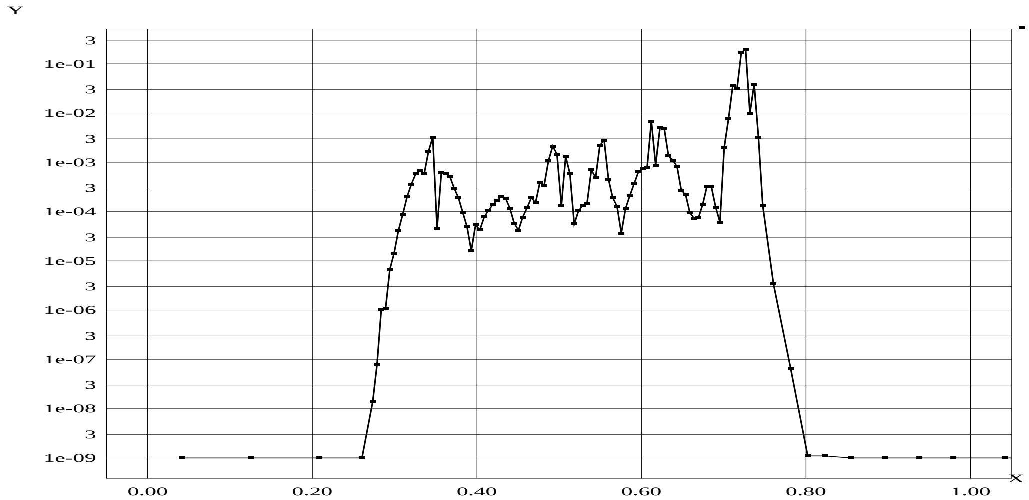

Figure 7: $\overline{Q}_j$ for solution of Euler gas equations at $t = .4$. Notice that in the active part of the solution, $\overline{Q}_j$ fails to decay enough to allow coarsening.

One possible improvement could be the rate at which points are added. Many times the grid is refined as many times in one area. This leads to areas areas of complete refinement and areas of no refinement. Because of this the method might be improved if, instead of the cell being divided into only two cells, cells are divided into possibly four cells. The only reason why dividing into two cells was the close relationship between this and wavelets. Dividing into four cells may decrease the number of computations used.

Another area of possible improvement is the time stepping procedure. This procedure applies a Runge-Kutta ODE solver on the system of ODE's for the time evolution of the cell averages. The Δt for the Runge-Kutta is controlled by the minimal size of all the cells C_j. Therefore if one small area of the domain is completely refined, then the smallest Δt is used for the evolution of all cells. Instead of applying the Runge-Kutta the same to all cells, a more intelligent plan of applying the Runge-Kutta ODE solver differently for each size cell might be better. Consider just one level of refinement and a forward Euler ODE solver. Since the smaller cells require a time step of half that of the larger cells, apply the forward Euler twice on the smaller cells, and only once on the larger cells. This will require special treatment of the boundaries between resolutions to ensure proper application.

The big test is whether this algorithm can be developed to work in higher dimensions. To develop this algorithm to two dimensions, new methods for computing the flow in and out of cells will be required. Also a systematic method for applying ENO to the non-uniform two dimensional grid will also be needed. But once these obstacles are overcome, AGENO will be more efficient in two dimensions than it is in one.

7 Acknowledgments

I am grateful to Prof. David Gottlieb for his helpful comments and direction along the way. Without a lot of helpful conversations with and insight from Lee Jameson, none of this would have been possible. I would like to thank Professor Marsha Berger for her insight into adaptive grids and Prof. Chi-Wang Shu for many helpful discussions on ENO schemes. I would like to acknowledge the support of this research by AFOSR grant 93-1-0090, DARPA grant N00014-91-J-4016, and NSF grant DMS-92-11820.

References

[1] S. Osher and C. W. Shu. Efficient implementation of essentially non-oscillatory shock capturing schemes. *Journal of Computational Physics*, 77:439–471, 1988.

[2] Ami Harten, B. Engquist, Stanley Osher, and Sukumar R. Chakravarthy. Uniformly high order accurate essentially non–oscillatory schemes, II. *Journal of Computational Physics*, 83:231–303, August 1987.

[3] Ami Harten and Sukumar R. Chakravarthy. Uniformly high order accurate essentially non–oscillatory schemes, III. *Journal of Computational Physics*, 71(2):231–303, 1987.

[4] Ami Harten and James M. Hyman. Self adjusting grid methods for one–dimensional hyperbolic conservation laws. *Journal of Computational Physics*, 50(2):235–269, May 1983.

[5] P. L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[6] S. Osher and C. W. Shu. Efficient implementation of essentially non-oscillatory shock capturing schemes II. *Journal of Computational Physics*, 83(1):32–78, 1989.

[7] Lee Jameson. On the wavelet-optimized finite difference method. *ICASE Report*, (94-9, NASA CR-191601), 1994. Submitted to Journal of Computation Physics.

[8] Ingrid Daubechies. Orthonormal basis of compactly supported wavelets. *Comm. Pure Appl. Math.*, 41:909–996, 1988.

[9] S. Mallat. Multiresolution approximations and wavelet orthonormal bases of $l^2(r)$. *Trans. Amer. Math. Soc.*, 315(1):69–87, September 1989.

[10] S. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. and Machine Intel.*, 11:674–693, 1989.

[11] Ingrid Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, XLI:909–996, 1988.

[12] Robert Bruce Bauer. An efficient adaptive grid algorithm:II. To appear.

Burger's Equation – Time = 2.5

Non-Adaptive (ENO)

Cells	L_2	L_2^*	Time	Steps
48	8.4×10^{-2}	3.3×10^{-3}	1 sec	64
96	5.3×10^{-2}	8.5×10^{-4}	3 sec	128
192	4.6×10^{-2}	9.5×10^{-5}	14 sec	255
384	2.2×10^{-2}	2.8×10^{-5}	57 sec	510
768	1.6×10^{-2}	5.6×10^{-6}	215 sec	1020

Adaptive (AGENO)

Cells	L_2	L_2^*	Time	Steps
run–1	1.4×10^{-2}	1.2×10^{-4}	12 sec	444
run–2	1.4×10^{-2}	1.4×10^{-4}	4 sec	224
run–3	1.4×10^{-2}	1.4×10^{-5}	22 sec	622

Table 7: Errors and *cpu* times for Adaptive and Non-Adaptive computations using high-pass filter based adaptive grid ENO scheme applied to Burger's equation at $t = 2.5$

Burger's Equation – Time = 5.0

Non-Adaptive (ENO)

Cells	L_2	L_2^*	Time	Steps
48	5.8×10^{-2}	4.5×10^{-3}	2 sec	128
96	4.7×10^{-2}	5.2×10^{-4}	7 sec	255
192	2.6×10^{-2}	3.3×10^{-5}	28 sec	520
384	1.9×10^{-2}	7.8×10^{-6}	110 sec	1020
768	1.5×10^{-2}	2.0×10^{-6}	442 sec	2040

Adaptive (AGENO)

Cells	L_2	L_2^*	Time	Steps
run–1	1.3×10^{-2}	1.3×10^{-5}	39 sec	1463
run–2	1.3×10^{-2}	1.4×10^{-5}	29 sec	1243
run–3	1.3×10^{-2}	1.7×10^{-6}	53 sec	1641

Table 8: Errors and *cpu* times for Adaptive and Non-Adaptive computations using high-pass filter based adaptive grid ENO scheme applied to Burger's equation at $t = 5.0$

Burger's Equation Run-Parameters for Computations

Run Variables	Coarsest # of Cells	Levels of Refinement	Break Time	Join Time	Break Increment	Join Increment
run–1	48	4	10	50	10^{-5}	3×10^{-6}
run–2	48	4	30	75	10^{-5}	3×10^{-6}
run–3	96	3	10	50	10^{-5}	3×10^{-6}

Table 4: User supplied constants for application of AGENO scheme to Burger's equation

Burger's Equation Run Results

Run Variables	Final Time	Initial Cells	Final Cells	Average Cells	Time (sec)	Average # Added	Average # Joined	Average Boundary	# of Time Steps
run–1	1.5	48	70	53	1.0	4.0	n/a	.64	50
	2.5	48	96	136	13.0	1.15	3.36	8.83	144
	5.0	48	81	114	34.0	0.49	2.40	6.16	1463
run–2	1.5	48	70	53	1.0	4.0	n/a	.64	50
	2.5	48	91	71	1.0	6.1	12.0	3.82	224
	5.0	48	145	187	29.0	1.42	9.00	4.61	1243
run–3	1.5	96	239	197	8.0	4.14	3.75	3.33	215
	2.5	96	145	187	22.0	1.55	10.67	4.79	622
	5.0	96	129	162	53.0	0.70	7.97	4.19	1641

Table 5: Results of using AGENO scheme. The figures are various results of running AGENO on Burger's equation.

Euler Gas Equations Parameters

Coarsest # of Cells	Levels of Refinement	Break Time	Join Time	Break Increment	Join Increment
12	4	10	50	10^{-5}	3×10^{-6}

Table 9: User supplied constants for application of AGENO scheme to Euler gas equations

Euler Gas Equations Run Results

Run Type	Final Time	Initial Cells	Final Cells	Average Cells	Time (sec)	Average # Added	Average # Joined	Average Boundary	# of Time Steps
AGENO	.4	26	106	71	40.0	8.9	1.0	10.2	82
ENO	.4	192	192	192	118.0	n/a	n/a	n/a	82

Table 10: Run results from the application of both ENO and AGENO to the Euler gas equations. Notice the *cpu* time is reduced by a third.