Interactive and Dynamic Control of Adaptive Mesh Refinement with Nested Hierarchical Grids

James. M. Hyman[1] and Shengtai Li[1,2]
Theoretical Division, Mail Stop, B284
Los Alamos National Laboratory
Los Alamos, NM 87545

[2] Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

We develop a new approach that combines interactive steering and solution adaptive control over adaptive mesh refinement based on nested refined grids. The adaptive grid can be generated dynamically during the solution process or controlled by directives in a grid refinement file. Instead of generating a new adaptive grid every time the equation is solved, this approach allows a user to use previous numerical simulations to directly control refinement in future simulations. Thus, a user can use knowledge from previous runs of similar problems to specify the regions where extra refinement is needed or eliminate refinement regions that don't affect the accuracy of the quantities he is interested in. We describe the advantages of converting the hierarchical AMR data structure to a conventional linear data structure for one dimensional problems to facilitate using implicit or multistep temporal integration. We present numerical examples and demonstrate the proposed algorithm can be easily incorporated into both new applications and legacy codes.

1 Introduction

The adaptive mesh refinement (AMR) method [1, 3, 14, 15] refines the numerical approximation of partial differential equations (PDEs) in space and time. The hierarchical data

structure requires fewer space-time elements than the adaptive methods on a single grid. Surprisingly, relatively few PDE codes use AMR methods because the traditional AMR approach is subtle to control and cumbersome to code for general problems. One of our goals is to seek new approaches so the AMR method can be incorporate in existing codes in a simple, consistent manner.

AMR uses block-structure and each level of grid can contain several logical rectangular grids (called *patch*) which can be integrated independently. It refines the time as well as space. More but smaller time steps are taken on fine grids than on coarse grids. The hierarchical nature of the grid system allows the different sized time steps to be interleaved such that the simulation remains time accurate.

The numerical solution of evolutionary differential equations is advanced in time in discrete steps that vary, depending on the local behavior of the solution; that is, the length of the time steps on each patch depends on accuracy and stability requirement of the solutions on the local grid, which is usually proportional to the local spacing.

Even though AMR has been designed for 2-D and 3-D problems (and the gain in efficiency is much greater than in 1-D), it is insightful to investigate the adaptation strategies and control of the AMR in 1-D. We simplify the hierarchical data structure for 1-D problems by adapting the data structure used by Berger [3] to work with an indexed linear array. This 1-D simplification improves the efficiency of finite difference methods (FDMs) and is easier to use with multistep and implicit temporal integration. The linear array data structure representation of the grid also simplifies the analysis and visualization of the solution.

Numerical simulations that use an adaptive single non-hierarchical global grid have a simpler data structure than the AMR hierarchical data structure we are using. Also as the non-hierarchical mesh adapts to the solution, except when the number of mesh points change, there is no data movement. However, even small changes in the number of mesh points can force large movements of data. A major disadvantage of an adaptive non-hierarchical grids is that the accuracy and stability constraints of numerical integration methods limit the global time step by the smallest grid spacing. Thus if a local highly refined region can force an extremely small global time step

Another advantage of using a hierarchical data structure to locally refine the mesh is that the grid spacing can be uniform on each of the refinement levels. This makes it easier to use higher order numerical methods such as fourth or sixth finite difference methods (FDMs) or pseudospectral methods. Because higher order FDMs require far fewer points than lower order methods this is a major advantage of the uniform adaptive hierarchical grid over a non-hierarchical nonuniform adaptive global grid.

One of the perplexing questions is why so very few large-scale production codes use modern adaptive mesh methods. A well worn phrase in large scale modeling is "if it is not in the design code, it will not affect the design." Because reliable robust adaptive mesh software for a production code can take years to develop there is huge threshold cost in time and funding for a code teams to do even a simple test case to see if it performs better than

the existing methods. Also, the majority of existing adaptive mesh software is so complex that if the adaptive mesh package breaks, the user (designer) cannot fix it without some significant help from the experts who wrote the package. The designer must have control over the design software, especially when there are tight deadlines. (It is not unusual to hear statements that "the product is going out the door next week and we have to have the best possible simulation now.") The adaptive mesh methods and software have not been at a stage where code developers are willing to trust their job/career on them.

We implement the AMR algorithms so they are easily used by existing codes with an independent module (subroutine) that can advance the solution on a single mesh. We have attempted to make the interface simple so that a user need only modify about a dozen lines of code for our AMR system to automate the refinement process.

Typically AMR packages try and do everything for the user, even when the user may know where the adaptation should be placed. Also, many time consuming computer runs solve similar problems many times. We have designed the AMR software so a user make use of information from previous runs to help guide and control the adaptive mesh of future runs. This allows a user to run carefully controlled convergence studies and overcome some of the limitations on the monitor functions that guide the AMR refinement algorithms.

Adaptive mesh refinement is guided by a function that monitors the local truncation error. Usually these monitor functions do well for adapting for problems with small errors, but can be erratic for problems with moderate size errors ($> 5\%$). Sometimes these moderate errors are unavoidable because of CPU or storage limits on the number of mesh points that a computer system can support. Also, there often are instances when the solution in certain regions of a complex simulation is no longer relevant and no longer needs to be refined. By giving the user a simple way to explicitly control the mesh refinement it can be easily turned off in these regions. We have included simple directives in our AMR software so a user can impose full control over where and when the refinement will take place.

One problem with many of the existing AMR software systems is that the user's code must conform to the computer language and the data structures of the AMR code. This sets an extremely high threshold for potential users, especially if the code is to convert a large-scale production code to use AMR. Adapting a production code just to determine if AMR is an effective approach is a major investment. We have taken the view point that the AMR code should be designed so that it can adapt to work in the same environment (framework) user's code.

The outline of the paper is as follows. After describing the basics of the 1-D AMR algorithm and specifying the details of our hierarchical data structure for the grid and solution in Section 2, we describe the regridding, injection, projection and clustering algorithms (Section 3). In Section 4 we describe the mechanism for users to control the refinement and integration. In Section 5, we use numerical examples to show the effectiveness of our approach.

2 Hierarchical Grid Structure

The basics of the 1-D AMR algorithm is illustrated in Fig. 1. The grid has three refinement levels (G_0, G_1, G_2) at time t_m. The integration starts from the coarsest level G_0, which advances one time step to t_{m+1}. Suppose the solution requires that the time stepsize for G_1 is half of that for G_0, and for G_2 is one-fourth of that for G_0. Then in the second step G_1 advances one time step to $t_{m+\frac{1}{2}}$, where the boundary conditions can be obtained from G_0. In the third step, G_2 advances two time step to $t_{m+\frac{1}{2}}$. After G_1 and G_2 reaches the same time, we update the solution of G_1 with the solution of G_2, because G_2 contains more accurate solution than G_1. In the fourth step, G_1 advances another step to t_{m+1}. In the fifth step, G_2 advances another two steps to t_{m+1}. Now G_0, G_1, G_2 reaches the same time level. We first update the solution of G_1 with that of G_2, and then update the solution of G_0 with that of G_1. The updating occurs only when the finest level grid reaches the same time level with the coarser grid. In step 6, we readapt the mesh and generate a new hierarchical grid. This readaptation can also be taken after each updating (e.g., at step 3).

To efficiently manage the data on each level, in the AMR algorithm the points are grouped (clustered) into disconnected segments, called patches. These patches are the building blocks for the hierarchical grid structure and are the basic data unit for refining the grid in both space and time. A patch is treated as a single grid with all the attributes of a single grid. Because there is only one parent and no siblings for patches in a 1-D AMR grid, the algorithms for clustering, refining, projection and injection, and integration are simpler than in higher dimensions [13].

We use an indexed linear array hierarchical data structure and store the grid descriptor information, node positions, and solutions in separate arrays. The hierarchical grid data structure $G = |n|G_1|G_2|...|G_n|$ contains the number of levels of the grid, n and pointers to the grid on each of the lower levels. The data structure for the grid on the i-th level $G_i = |m_i|p_1|G_{i,1}|p_2|G_{i,2}|...|p_{m_i}|G_{i,m_i}|$ contains information on the patches, where m_i denotes the number of patches. The data structure $G_{i,j}$ contains information for the j-th patch on the i-th level and contains information related to the patch. The pointer p_j is the index of the parent of the patch $G_{i,j}$ in the coarse grid G_{i-1} and facilitates the regridding and projection operations between the coarse and fine grids.

An array stores the starting position of each level and other attributes such as the current integration time, time step size for the current level, number of ghost boundaries, number of buffer zones in the current level and refinement ratio to the parent. For each patch, we use the notation I_{bx} and I_{ex} for the indexed of the first and last point where the solution is defined in the patch. The solution is being integrated for the indices in $i\epsilon[I_b, I_e]$ and the ghost boundary points are defined for indices between $i\epsilon[I_{bx}, I_b - 1]$ and $i\epsilon[I_e - 1, I_{ex}]$.

We have adapted the grid structure proposed by Berger([2, 3]). Her flexible approach is amenable to different dynamic memory allocation approaches. A patch is a basic computational unit and it has the following attributes:

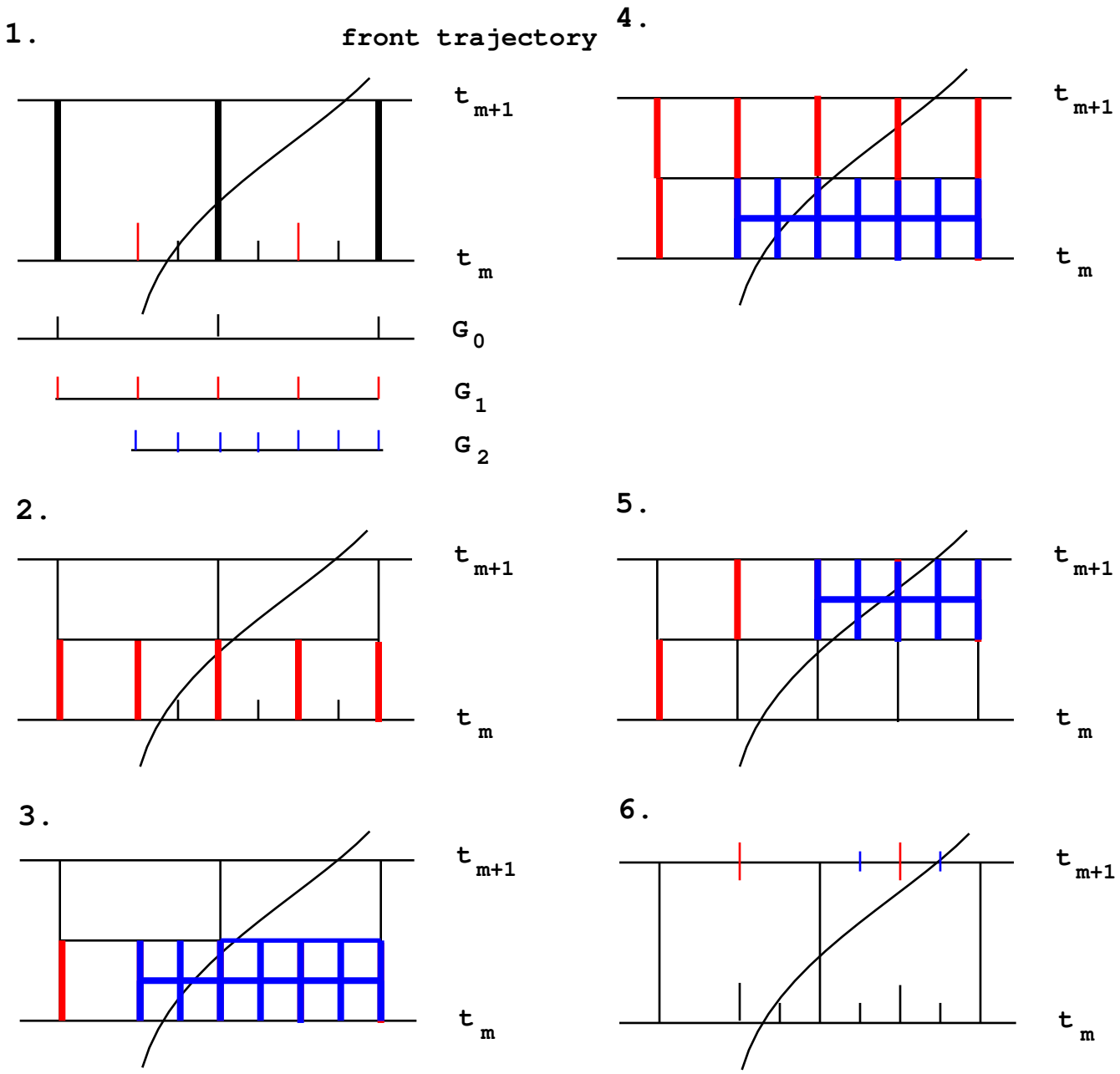

Figure 1: AMR processing: 1) Advance G_0 one step to t_{m+1}; 2) Advance G_1 one step to $t_{m+\frac{1}{2}}$; 3) Advance G_2 two steps to $t_{m+\frac{1}{2}}$, update the solution of G_1 at $t_{m+\frac{1}{2}}$ with that of G_2; 4) Advance G_1 one step to t_{m+1}; 5) Advance G_2 two steps to t_{m+1}, update the solution of G_1 (G_0) with that of G_2 (G_1); 6) Readapt the mesh to generate a new G_1 and G_2, goto 1).

- level of the current patch
- integration time of the current patch
- time step size Δt for this patch
- number of grid points
- logical grid index for each point
- physical grid location for each point in the patch
- number of ghost boundary points
- spatial step-length (Δx)
- solution values
- pointers to parent patches (only one parent in 1-D)
- refinement ratio between the parent and current patch
- multiple pointers to the siblings (in 2-D or 3-D).

3 The AMR Integration algorithm

The solution is advanced in time by an AMR method recursively (see Fig. 1), similar to a W-cycle in the multigrid method [4]. Figure 2 provides a top level pseudo-code description of the AMR algorithm.

3.1 The Remesh Algorithm

The *Remesh* stage to cover subdomains with grids of higher resolution is the most algorithmically complex AMR operation in the refinement process. The whole stage can be split into two processes: readapt (including refine and coarsen) the available grid and refine to generate new finer grid. In the first process, there may be features which appear in the finer levels that would not be captured if we start with the solution on the coarsest grid and then adapting the grid to the finer levels. To prevent this from happening, unlike the local uniform grid refinement (LUGR) method [19], we initialize the mesh refinement on the finest level possible. This grid is then coarsened or refined based on the selection algorithm. This process continues until all of the indicated levels have been readapted. The regridding step (defining the solution values for the readapted grid) is done reversely. It starts from the coarsest level possible. After the first process, if the finest level available does not reaches the

```
INTEGRATE(level)
begin
    Let maxlevel = the maximum level allowable, flevel = the finest level existing;
    Remesh Stage(level): If it is time to readapt the grid then
      flevel = min(flevel + 1, maxlevel);
      while (flevel-1 needs no refining) decrease flevel by 1;
      for slevel = flevel-1 down to level do
          Refine(slevel)
              Select (slevel): flag the inaccurate points which need refining;
              Expand (slevel): add buffer zones around the flagged region;
              Cluster (slevel): group the flagged points into clusters;
      for slevel = level up to flevel-1 do
          Regrid (slevel +1): define the solution values for the readapted grid;
      while (flevel < maxlevel and flevel needs refining) do
          Refine (flevel);
          Regrid (flevel + 1);
          increase flevel by 1;
    Advance Stage(level):
      Boundary Collection (level): obtain the boundary values from level-1;
      Advance (level): advance the current level one time step;
    Recursive Stage (level):
      If level ≠ flevel then for r = 1 to Δt_level/Δt_level+1 do
          INTEGRATE (level+1);
          Recursively advance the solution on level + 1 one time step;
    Project Stage (level, level+1);
      update the solution on level with the solution on level+1.
end
```

Figure 2: Pseudo-code for 1-D AMR recursive integration algorithm.

maximum level allowable and it needs further refinement, then we start the second process to refine and generate finer level.

One of the great advantages of the AMR method is that the refinement is in both space and time. Because of this, when there are significant fine scales in the problem, the majority of computing time is spent on the finest level. For nonlinear systems of PDEs the refinement factor in time for an explicit method must be estimated for each level based on the requirements of stability and accuracy. To simplify the projection and regridding algorithm we require the ratio of the time step sizes between two adjacent levels to be an integer, *i.e.*,

$\Delta t_{level}/\Delta t_{level+1}$ in Fig. 2 should be precisely an integer.

In most AMR methods and systems and for first order systems of hyperbolic PDEs, if the refinement factor between a level $l+1$ and the next coarser level l is r, then grids on the finer level $l+1$ will be advanced r time-steps for each coarser time-step [2, 1, 14, 15] to keep the time step stability restriction (CFL number) the same for all of the refinement levels. For second order parabolic PDEs, when the stability limitation for explicit time integration is proportional to $\Delta t/(\Delta x^2)$ instead of $\Delta t/\Delta x$, there should be r^2 time steps on the finer grid for every time step on the next level.

We now describe the two small steps of the remesh stage: refine and regrid.

3.1.1 Refine step

In the refine step, the monitor function is evaluated on the current level and the points with significant errors are flagged for refinement. After buffer points are added around these points, they clustered and organized into patches. Refinement step includes three small steps: select, expand, and cluster.

Select step

In the select step we identify (flag) the grid points that need to be refined. We adopt the monitor function proposed by Verwer et al. [19] to flag the over or under resolved mesh points and identify the regions to coarsen or refine. This monitor function

$$\mathrm{SPCMON}(i) := \max_j \mathrm{SPCTOL}(j)(|\Delta x^2 u^j_{xx}(i)|) \tag{1}$$

where

$$\mathrm{SPCTOL}(j) := \frac{\mathrm{SPCWGT(j)}}{\mathrm{UMAX}(j)\cdot \mathrm{TOLS}}.$$

is defined for each grid point x_I for the jth solution component u^j_I.. The weighting factor $0 < \mathrm{SPCWGT} \leq 1$ is a user specified parameter to indicate the relative importance of a PDE component. The parameters UMAX denotes the approximate maximum absolute value for each component, and TOLS is the spatial error tolerance. The second order derivative is approximated by a three point FDM. We initial a level refinement if there is a point where $\mathrm{SPCMON}(i) > 1.0$, and then all grid points with $\mathrm{SPCMON}(i) > 0.5$ are flagged. In order to ensure proper nesting, if the current level grid has a grandchild, those points are also flagged.

The mesh refinement can also be guided by a monitor function based on the Richardson truncation error estimation proposed by Berger et al. [1, 2]. However, this approach requires the solutions on a half resolution (of current fine grid) grid, which may not be easily implemented without much alteration of the time integration solver. To accomordate the need of different type of monitor functions, including the type of Rechardson truncation error

estimation, we do have an option in our implementation that allows the users to provide a new one to replace our default one.

Expand step

Because the regions needing refinement may move, to avoid the most interesting features of the solution escape from the refinement region and to reduce the amount of rezoning, we surround (expand) the flagged points with a buffer zone of flagged points where the mesh function is below the threshold value. Expansion serves to ensure coverage of the grids on the current level for the next k time steps. To reduce the overhead of the refinement, we perform the refinement every k time steps instead of every time step. We add buffer zones to the flagged points so that a region needing refinement will not escape onto the coarse grid. The size of the buffer zone depends on k and the current time stepsize.

To estimate the trade-off between the number of buffer zones and the size of k we compared two approaches. In one approach we performed refinement **Refine(1)**. only after integration on the whole grid was completed The number of buffer zones is different for different levels in this approach, because finer levels take more integration steps so that they need more buffer zones. Usually, we assume the discontinuity moves no more than one buffer zone within one time integration. So we choose the number of buffer zones to be equivalent to one base grid. (In our tests we observed that the the number of buffer zones can be much less than this). We found this approach is most efficient when the user is directly controlling the mesh refinement and the refinement regions change very slowly.

In the second approach, we perform refinement for the current level after it has advanced time steps and fix number of points in the buffer zones for all the grid levels. Therefor, there are fewer points in the buffer zones than in the first approach and the patches for finer levels are smaller. This approach has more overhead costs (about 8% of the cost of the AMR is in this step) but it reduced the total computation time because of the savings integrating the smaller fine grid levels.

Cluster step

Following expansion, the flagged points at level l are clustered. We search from left to right for each coarse patch, identify the adjacent flagged points and group them together. If two patches are within two buffer zones, we join them together. The clustering algorithm is far more complex in higher dimensions than in 1-D.

3.1.2 Regrid step

Regridding the $(l+1)$ level includes computing the physical locations for each fine grid and copying or injecting the solution from the old grid to the new grid. The physical positions are computed by linear interpolation. Although the solution can be obtained from the

coarse grid by injection or interpolation, usually a new refinement is partially or completely contained in an existing patch. It is far more accurate to use as many existing values as possible from the previous patch on the same level which partially overlays with the new grid. These overlap regions retain accuracy when there are discontinuities in the solution within overlapping regions between the new grid and old grid. This approach also guarantees that near discontinuities the numerical solution largely exhibits the properties of the numerical scheme used for the single mesh integration process. To preserve the monotonicity of the solution for hyperbolic conservative law on the new refined grid, a monotonicity preserved cubic interpolation [6] is used.

3.2 The Advance Stage

During the **Advance stage** in Fig. 2, the solver or *kernel* of a simulation advances the solution for all the patches in a level by advancing the solution to a specified time. Because the details of the method to advance the solution are not directly relevant to AMR, many existing codes written to advance a solution on a single mesh can be easily adapted to use our software. This is especially true if the existing code allows the user to specify the boundary conditions, the initial condition, a routine that defines the time derivatives (or residual for differential algebraic equations), and other related information such as the time step size, the spatial step length then it can probably be easily extended to an AMR code.

In this stage, one of the important issues is the **boundary collection** for the current level. Because the sub-grid (patch) may have only one or no external boundary whereas the solver input by the user always assumes two external boundaries, it is necessary to turn off the boundary collection inside the user solver and to do the boundary collection inside the AMR system. It is usually a minor change for the code to be structured so the boundary conditions are imposed outside the solver.

3.2.1 Boundary collections

Computing accurate boundary conditions for a system of partial differential equations can be as important as accurately approximating the spatial derivatives in the interior of the domain. The errors introduced into the calculation from improper boundary conditions persist even as the mesh spacing tends to zero. If the boundary conditions are not properly incorporated into the discrete approximation, a well-posed problem in differential equations can be changed into an ill-posed (unstable) discrete one.

There are two types of boundaries in an AMR system: External boundaries are given by the problem definition; internal boundaries are needed by the patches imbedded within the domain. The left and the right end (boundary) of each 1-D patch could be either an external or internal boundary point. It is especially important to apply the boundary conditions at the proper time when the equations have time dependent coefficients or when intermediate time

derivatives of the boundary cells are required. This is similar to the care that is most taken to keep the proper time for the boundary conditions when using multistage Runge-Kutta type methods.

For internal boundary conditions, we use linear or cubic Hermite interpolation between the values for the internal boundaries from the parent coarse grid at the forward time t_{n+1} and the boundary values at t_n on the finer grid. The external boundary conditions may also be included by defining the solution at ghost cells external to the problem domain.

Our AMR system is automated to impose external Dirichlet, periodic, symmetric, anti-symmetric or extrapolated boundary conditions. The periodic boundary conditions require information at both sides of the domain, while the other boundary conditions need only local information. We do not allow a periodic boundary to be inside a patch, therefore when a patch has an external boundary that is periodic, we check whether another patch is adjacent to it on the other external boundary. If so, the ghost cells are copied from one patch to the other. After clustering in a periodic problem, if there is an adjacent patch at the other side and if appropriate, we insert a patch at the other end and modify the grid structure appropriately.

Another simple method to treat periodic boundary conditions is to perform the copying from one end to the other only when both ends have patches. This method does not need to alter the grid structure to accommodate periodic boundary conditions. This approach works well when we perform the refinement during the integration of the finer grid. This is because for the second approach, the new patches of the finer level can be added during the integration. For a semi-discretized PDE solver and method of lines approach we combine and integrate the periodic boundary patches as a single patch.

Many AMR methods (e.g. [1, 2, 14, 15]) do not store the solutions for the ghost boundary, and obtain the initial values just before integration. This works when the solver does not collect boundary values from the parent grid or external boundaries during a one time step integration. Many fully-discretized PDE solvers with a predictor-corrector integration methods belong to this category and can be plugged into our AMR system directly.

This approach is not sufficient for a semi-discretized PDE being solved with method of lines (MOL) when the values at the ghost boundaries are not calculated by the solver but collected from the parent coarse grid or according to the external boundary conditions during the integration. If an implicit or high order explicit temporal integration method (such as a third order TVD Runge-Kutta method [16]) is used, the solutions at intermediate times are required at the ghost boundaries. Therefore, we store both the ghost boundary values and the internal node values. For a semi-discretized PDE and a single-step high order explicit method, we compute the time derivatives at the ghost boundaries using the boundary values at the forward time (collected before the time integration of the current patch) and backward time, and keep them as constant during a one time-step integration. The external boundary values are collected if the current patch reaches the external boundaries. After the time derivatives for the internal nodes are computed by the user routine. We advance the whole

grid, including the ghost boundaries, one intermediate step. The process continues until the one time-step integration is completed. This approach also works for implicit methods.

3.2.2 Advance one time step

Given the user solver for a single grid, we can reuse two kinds of routines. First, if the solver collects the boundary values only once and at the beginning of each time integration, we can directly plug the whole solver (for one time step) into our AMR system without modifications. Apart from the external boundaries and parent coarse grids, the ghost boundary values are also collected from the internal boundaries. Therefore, we cannot integrate each patch separately if boundary collections are done more than once for a single time step. If the solver for a single grid treats each boundary as an external boundary, and imposes the boundary conditions several times while advancing the solution during a time step, then the routine must be modified to account for the internal patch boundary conditions.

The 1-D algorithm is much simpler than the higher dimensional AMR methods because of the lack of siblings and overlayed internal boundaries.

3.3 Recursive Stage

In this stage, the integration forms a W-cycle.

3.4 The Project Stage

After integrating the finest grid T time steps, the finest grid reaches the same time level as the coarser grid, and the coarser grid starts integration again. Before the coarser grid starts integration, the solution needs updating. That is, the more accurate values of the solution on the finer level replace the values of the solution on the coarser level where they coincide. When the refinement ration r is even, then the solution variables located at the nodes are copied and the cell centered are interpolated. When r is odd, then the opposite is true.

Whnen a conservation law is being solved, a flux correction procedure, proposed by Berger and Colella [1], must be implemented for the coarse cells at the coarse-fine interface to preserve global conservation. The flux correction needs the flux information of both the fine and coarse grids. Therefore, the solver must store and return the flux during the discretization and integration. Note that the flux correction must be performed before the projection to ensure that the coarse grid has the optimal solution values.

4 User Control over the AMR process

For some applications, the monitor function may fail to identify all the regions that need to be refined. For example, when we solve the Euler equation by the artificial viscosity

method, the region near the contact discontinuity cannot be resolved as well as the region near the shock wave. After some time, the refinement level at the region near the contact discontinuity is reduced. We need to add finer levels in that region to ensure that the discontinuity is properly resolved. Also, there are situations such as the early evolution of the solution where we are only interested in the final steady state solution where the efficiency of a calculation can be improved if a user has control over the AMR process. In the extreme situation, a user may want complete control to guide the refinement process at any time and any place.

The user has this control of our AMR system through a grid file. (see Figure 3). While

$$amrGF
  next refinement time = t,
  number of refinement levels = $numlev$,
  number of patches of level 1 = $patch0$,
    patch(1, 1) = $startx$, $endx$,
    ⋮
  number of patches of level i = $patchi$,
    ⋮
    patch($ipatch$, $ilevel$) = $startx$, $endx$,
    ⋮
$$end

Figure 3: Data format for grid file.

the equations are being integrated, the patch and the base grid information in the grid file are used to compute the logical coordinates for the refined grid. The algorithm is: first find the logical cell in the base grid that contains the end point; then divide it $l-1$ times if the patch is at the l level; finally, locate the nearest point and assign its logical coordinate to the end point.

Most error estimators are good for estimating small errors, but there are situations where there are insufficient computer resources to drive the truncation errors to very small values. In these situations, the grid must be coarser and the errors may be of moderate size and the error estimators fail. The grid file can be used to explicitly control the refinement based on limitations in the computer resources rather than just minimizing the truncation error.

We have designed a friendly graphical user interface to modify and optimize the grid file. The above routine can also communicate through calls or piped data channels (you can forget about the files and pipe the data between these routines on every time step during

a run and the package will work like a traditional adaptive mesh refinement). We've lost nothing but gained in the flexibility and equally important lowered the buy in threshold

The input grid file can also be used to prevent chattering where a region is first refined because of, say, oscillations near a steep front. When the oscillations disappear then the error estimate monitor the smooth solution and signal that the local refinement may be coarsened. On the coarsened grid, the oscillations reappear and trigger a new AMR refinement and the cycle starts over again.

For 1-D AMR grids it facilitates output and display of the solutions that transforming the hierarchical data to a conventional linear array data, *i.e.*, instead of outputting the data as many patches, the data can be analyzed and plotted as a single nonuniform grid. We provide a software tool to convert the data using the W-cycle algorithm illustrated in Figure 4 for four refinement levels.

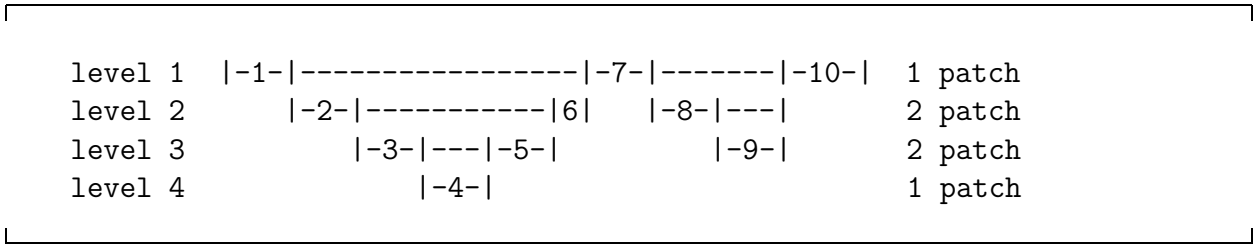

Figure 4: Order for hierarchical data structure to output as one patch

The user is given additional control over the AMR process by

- Allowing the user to specify the time-step size at any time.
- Allowing the user to modify the refinement parameter, monitor function, maximum refinement level, etc.
- Allowing the user to save all the solution and grid at any time for debugging, modification or re-starting the simulation.

5 Numerical Experiments

5.1 Software design

Our goal port is to improve the effectiveness of applications programmers solving systems on nonlinear PDEs, but are not necessarily experts in AMR. In designing the AMR software we have emphasized maximizing the flexibility, modularity, reliability, efficiency, and restricted data flow and documentation. Because these are global properties, we have built them in from the initial design phase so the AMR software fits together in a logical structure.

Types of computers and operating systems to run the applications codes will proliferate in the next few years. We must design the interface between the module and applications code to be as stable as possible. In this way the applications code can be easily moved from machine to machine, always using the modules that are best suited to and optimized for the current computer hardware. It is a major task to design support software with a stable interface and still fully exploit the available machine hardware.

We also recognize that programmers considering using the software must plan for possible situations where the software may fail. Although most of the development time in a AMR software system goes into debugging and maintenance, programmers make errors. Even though the designers of AMR software packages make every effort to protect their users, exhaustive testing of all the options on large problems is impractical. To evaluate the AMR software on large scale problems we have tried to make it compatible with existing techniques and simple enough so that potential users can observe tangibly better results in a trial run. Software packages that can reproduce the capabilities of the existing ones, but work better, are the most readily accepted.

Also, the changes needed to evaluate the software should be chosen so that the modified code would have increased capabilities independent of the success of the AMR.

For these new models to affect a design, they must be in the design code. It is unrealistic to expect the physics model, the numerical methods, or the computer architecture to be stable during the lifetime of the code. Any up-to-date large production program will eventually exist in a multitude of different versions. It is common sense to leave options open so we can anticipate and profit from advances whenever possible. Thus, flexibility was a major design criterion. A code that is more flexible than necessary is far better than one that is inflexible and unable to handle future problems. Also, we realize that it is human nature to underestimate the size of a software package, such the AMR, necessary to achieve the robust efficiency in production problems. Our approach has been to develop the code around a few simple structures and reduce the number of interconnections that the programmer must be aware of by dividing the software into a set of easy-to-understand code modules.

Applications programmers can understand what a module does and then use it without knowing what goes on inside. But before they do, they must have confidence in the reliability and efficiency of the numerical methods and the allowed data flow. Not only must the module produce correct results in a timely fashion, but it must do so without influencing the rest of the code by severely restricting the communication links between the routines.

To make efficient use of the hardware without inefficiently using the application programmer's time we have provided interfaces in C, FORTRAN and C++ that will allow the applications programmers to program in the language they prefer as long as it is compatible with the interfaces provided by the AMR routines.

The user can try different combinations of numerical and physical models in the library, tailor the code and optimize it to a specific problem. Also, because the source codes of the library routines are available, users can easily modify these codes to develop and test new

ideas on the AMR code validation test problems, as well as on own full-scale production problems. This approach also allows specialists in physics, numerical analysis, or computer science to explore new models, methods, and systems software in production codes without being required to know the details of the AMR software.

Fundamental advances in the capability and efficiency of future production code software will require research and innovations in numerical methods and software design before the software will perform as advertised over a broad range of input data, compilers, operating systems, and machine hardware. There must be checks in the software to detect and recover from (or report) anomalous situations.

The AMR algorithms in large-scale scientific codes can be highly complicated and problem dependent. That is, a method developed for a particular test problem may not work for similar but larger problems. Methods that work well in one space dimension may not be easily extended to two or three dimensions. This fact puts a huge burden on AMR code developers, who must small-scale test problems solved by simple methods, to choose algorithms that will scale properly to model a far more complex system. After the choice is made, they must redevelop the software necessary to test it-usually duplicating the efforts of the original author.

To avoid repetition of expensive, error-prone, and time-consuming coding of AMR we have developed a library of high-level software to make the common elements of the code are available as modules, the applications programmers can plug into these routines and eliminate much of their effort.

There is no guarantee that the AM package will be stable, well supported in the future, be available on MY next computer system. I need a fall back (safety net) position.

In parameter studies where a series of computer simulations are made with only slight changes in the problem parameters or levels of refinement the grid file can be used to explicitly define the mesh in a current simulation based on what it was in past simulations. The mesh function for this programmed mesh can still be evaluated to ensure that the solution is being adequately resolved, but it need not be evaluated as often as when there is no a prior information on the mesh adaptation. This can significantly reduce the costs of monitoring the error and the other overhead for the AMR.

In the following numerical examples experiments, the refinement ratio $r = 2$, and there are a maximum of six refinement levels.

5.2 Burgers' Equation

Burgers' equation,

$$u_t + uu_x = 0, \quad x \in [0, 1], \quad t > 0 \tag{2}$$

with smooth initial conditions

$$u(x, 0) = 0.5 \sin(\pi x) + \sin(2\pi x), \quad x \in [0, 1],$$

and homogeneous boundary conditions, $u(0,t) = 0$ and $u(1,t) = 0$ serves as a standard test for adaptive methods ([13]).

We integrated the equation using 50 uniform base grid points and $TOLS = 0.075$ with the internal monitor. The finest (6th) level has a resolution of $2^6 \times 50 = 1600$ grid points. The equation is integrated with the third-order Runge-Kutta method and third-order WENO scheme ([16]) (Fig. 5.5-a) with 50 base grid points. The CFL number $||u||_{max}\frac{\delta t}{\Delta x} = 0.8$. For comparison, we plot (the dotted line in the figures) a reference solution computed with an adaptive moving mesh method (see [12]) with 601 grid points. We also integrated the equation the forward Euler method (instead of the third-order Runge-Kutta method) and observed that the decreased time integration errors caused the numerical solution to significantly behind the reference solution.

To illustrate the flexibility of our AMR system and the re-usability of the user's code, we plugged in the hyperbolic PDE solver routine *claw*1 from CLAWPACK [9]. Since *claw*1 collects the boundary values only once at the beginning of the each time step, it was easy to block the boundary condition routine by providing a dummy subroutine and write a simple interface routine to define the CLAWPACK arguments and parameters. We did not to modify any of the original CLAWPACK software. The results, shown in Figure 5.5-b for 21 base grid points, demonstrate how effective the combination of the CLAWPACK solver and AMR is for solving these types of PDEs.

5.3 Linear Wave Propagation

The second example is a wave equation

$$u_t + u_x = 0, \quad x \in [0,1], \ t > 0 \tag{3}$$

with periodic boundary conditions and the initial conditions

$$u_{t=0} = \begin{cases} 1, & \text{if } x \in [0.4, 0.6], \\ 0, & \text{otherwise.} \end{cases}$$

The solution shown in Fig. 5.6 is a square wave propagating to the right side at a speed of 1. The leading edge of the square wave reaches the right boundary at $t = 0.4$, and the trailing edge reaches the right boundary at $t = 0.6$. The wave emerges from the left side because of the periodic boundary conditions. The base grid has 40 nodes, $TOLS = 0.1$, CFL number is $\frac{\Delta t}{\Delta x} = 0.5$ and the equations were integrated with a third order WENO scheme [10].

5.4 Scalar Combustion Model

The single-step, reaction-diffusion PDE,

$$\begin{aligned} &u_t = u_{xx} + D(2-u)\exp(-d/u), \quad 0 < x < 1, \quad 0 < t, \\ &u_x(0,t) = 0, \quad u(1,t) = 1, \quad u(x,0) = 1, \end{aligned} \tag{4}$$

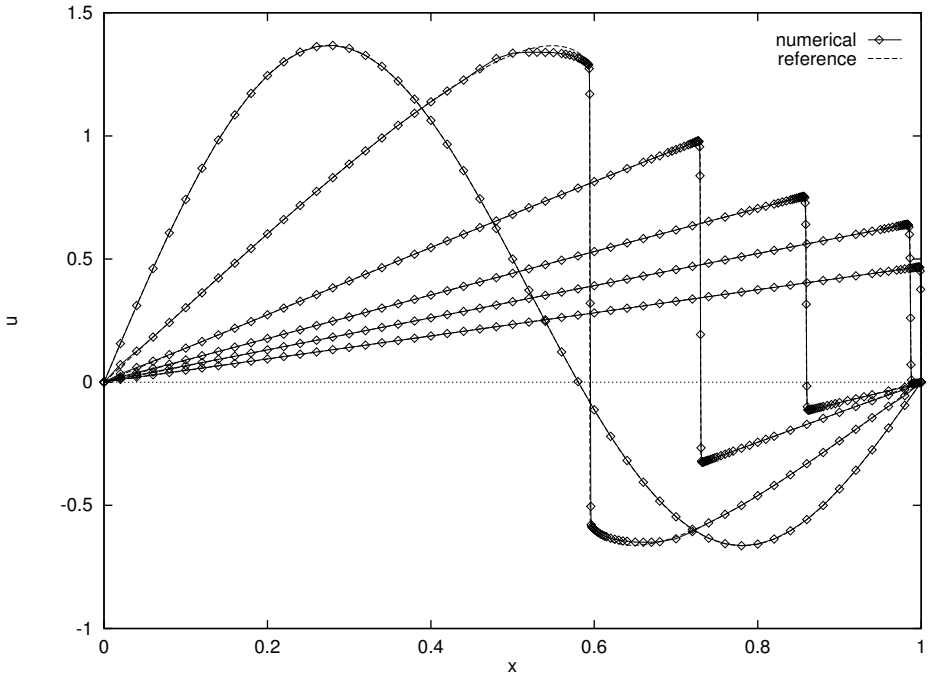

Figure 5.5-a: The AMR solution of Burgers' equation (2) is shown at times $t = 0, 0.2, 0.6, 1.0, 1.4, 2.0$ was calculated using a third-order Runge-Kutta method and a third-order WENO scheme for the spatial derivatives.

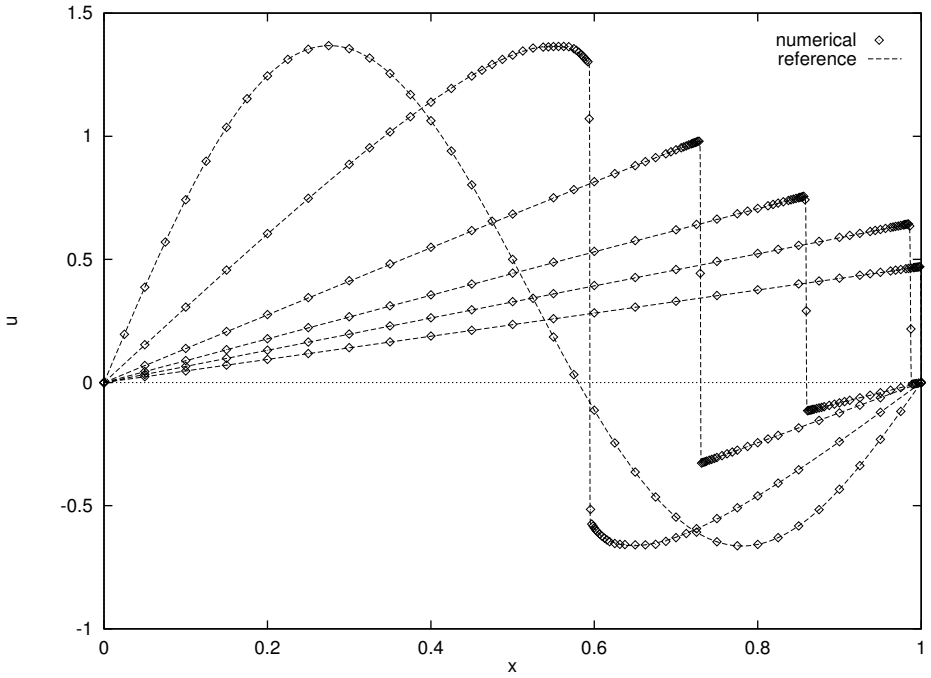

Figure 5.5-b: The AMR solution of Burgers' equation (2) is shown at times $t = 0, 0.2, 0.6, 1.0, 1.4, 2.0$ was calculated using the CLAWPACK solver.

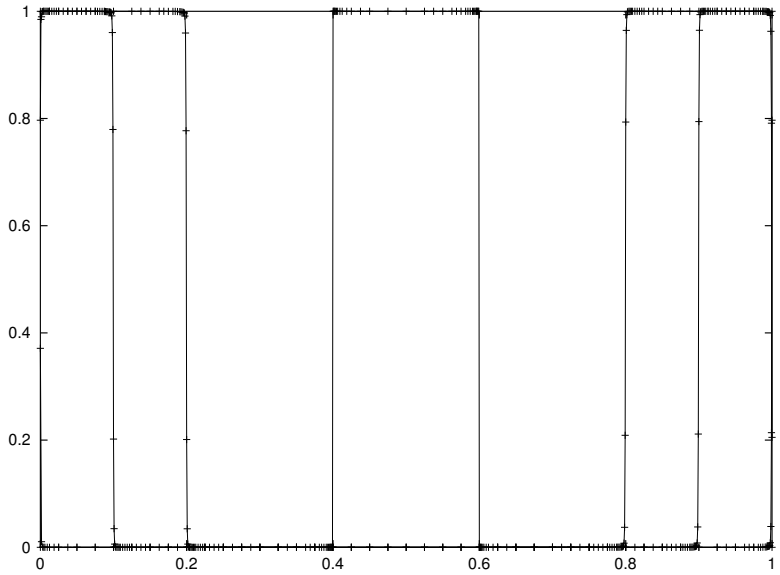

Figure 5.6: AMR with WENO scheme for wave propagation problem (3) with periodic boundary conditions. The results show that our AMR system has no problem in handling the periodic boundary conditions. Plotted at $t = 0.0, 0.4, 0.5, 0.6$.

where $D = 5e^d/d, d = 30$. The solution is the temperature of a reactant that gradually increases from unity until a "hot spot" forms at $x = 0$ causing the temperature to rapidly increase to 2. A front forms and quickly propagates towards $x = 1$ with speed proportional to d [12].

We used 40 nodes for the base grid and five refinement levels, which corresponds to the resolution of a 640 point uniform grid. Because the steep wave front develops extremely fast after $t = 0.24$, we choose the error tolerance for space to be 0.0005 (we had tried 0.001 and the integration failed). We set the initial time step for the base grid to be 0.0001 and integrated the solution with the explicit second order improved Euler method. The spatial derivatives are discretized by central difference scheme. For comparison, we plot the solution in Fig. 5.7 at the same time as published for the moving mesh method in [12]. Because small time steps are unnecessary before the wave front has been formed, the adaptive refinement is turned on at $t = 0.24$ and the time step is chosen according to the CFL condition before the wave forms to reduce the computation time

5.5 Fisher's Equation

In this example we demonstrate how, even with the AMR refinement and small time steps on the finer grids, the solution can be inaccurate unless there is sufficient accuracy in the

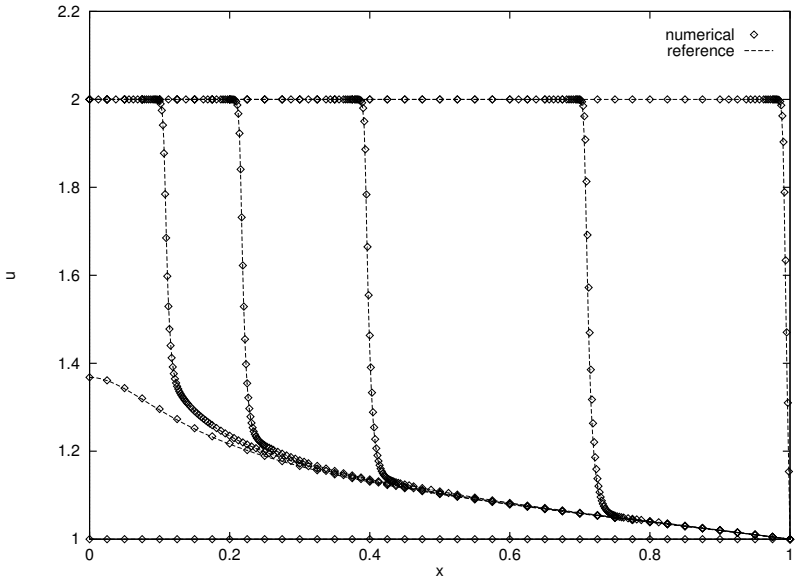

Figure 5.7: Results of explicit AMR method for scalar combustion model. We manipulate the refinement by using only 1 refinement level before $t = 0.24$, and 5 level after that time, which sharply reduces the computation time. Output at $t = 0.0$, 0.24, 0.2405, 0.241, 0.242, 0.244, 0.246.

time inegration. Fisher's equation

$$u_t = u_{xx} + \alpha u(1-u), \quad x \in [-1,1],\ t > 0 \tag{5}$$

is often used to illustrate this weakness in moving mesh methods [12]. We defined initial conditions consistent with the analytic solution

$$u(x,t) = \left(\frac{1}{1 + e^{\sqrt{\frac{\alpha}{6}} x - \frac{5}{6}\alpha t}} \right)^2, \tag{6}$$

with $\alpha = 10^4$. The initial and boundary conditions are derived from equation (6) except that we choose the left boundary to be fixed, *i.e.*, $u(1,t) = 0$. We choose the error tolerance for space to be 0.0001. We used 40 nodes for the base grid and five refinement levels, which corresponds to the resolution of a 640 point uniform grid. The allowable time stepsize by the stability restriction is $(\frac{1}{40 \cdot 2^{level-1}})^2$, where one time step at the coarse level corresponds to 4 time steps at the next finer level because the CFL number is proportional to $(\Delta x)^2$. We used 10^{-4} as the time step for the base grid, which is much smaller than 6.25×10^{-4}, which is required by the stability restriction. The accuracy is $O(10^{-4})$ for the first order forward Euler method and $O(10^{-8})$ for the second order Runge-Kutta method. That's why (see Fig. 5.8) the second order Runge-Kutta method is more accurate. Actually the time step for the base grid should be selected to maintain error tolerance for the temporal integration to be

less than 10^{-6} per unit time for this problem. If the time step on the coarser grid is much smaller than required by the CFL condition, then the same time step on the finer grids would also satisfy the temporal error estimate. Therefore, it is possible that the time step size on the coarse grid may be equal to the time step size on the fine grid. The time step is determined by both the accuracy and the stability at any level, not by the refinement ratio (as is typical in most implementations of the AMR method [1, 2, 14, 15]).

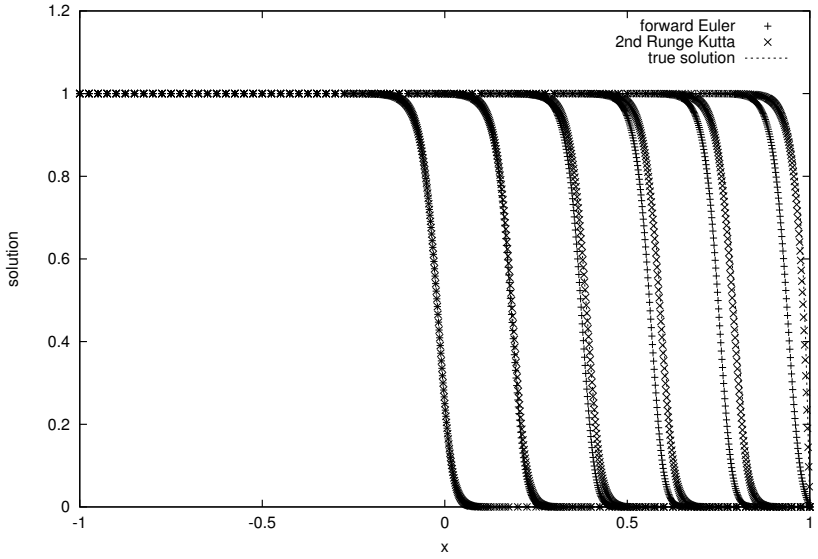

Figure 5.8: Result of Fisher's equation by AMR method. The second order Runge-Kutta method is more accurate. Our AMR sysmtem has no problem in plugging in this kind of the method of lines solvers. Plotted at $t = 0.0, 0.001, 0.002, 0.003, 0.004, 0.005$.

5.6 Shock-tube Problem

The example will illustrate how to flexibility of using previous simulations to fine tune the AMR grid to better track features in the solution.

The Euler equations for gas dynamics,

$$\begin{aligned} \rho_t + (\rho u)_x &= 0, \\ (\rho u)_t + (p + (\rho u)^2/\rho)_x &= 0, \\ e_t + ((e+p)(\rho u)/p)_x &= 0, \end{aligned}$$

are an effective test for both extremely steep gradients and moving fronts.

5.6.1 Sod Shock tube problem

The initial conditions from Sod's shock tube problem [17]

$$(\rho, (\rho u), e) = \begin{cases} (1.0, 0.0, 2.5), & \text{if } x \leq 0.5, \\ (0.125, 0.0, 0.25), & \text{if } x > 0.5. \end{cases}$$

result in a fast moving shock wave to the right ahead of a slower moving contact discontinuity and a rarefaction wave to the left of the initial discontinuity. We first solve this problem with the artificial viscosity and 40 points in the base grid. The tolerance for the monitor function in space is $TOLS = 0.1$. We plot density ρ at time $t = 0.1$ in Fig. 5.10. Except for the shock wave, the other three corners are not resolved well by automatic adaptivity in our AMR system because of the effect of the artificial viscosity. So we dump the grid file and modify it using our grid file editor, adding refinement at the three corners, and let the AMR system read our grid file instead of automatically refining. For comparison, an exact solution of 1000 points is also provided. The slope of the contact discontinuity is due to the

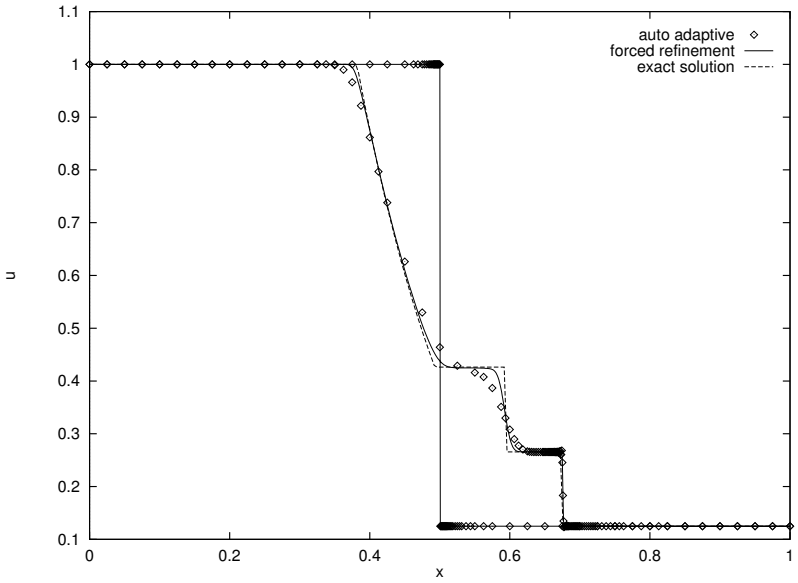

Figure 5.9: AMR with artificial viscosity method for Sod's shock-tube problem. $t = 0.1$. Manipulation of the refinement can greatly improve the results.

artificial viscosity.

We also plugged in the CLAWPACK solver to solve this problem with 50 points in the base grid and compared it with the performance of one single uniform grid. The CPU time to solve this problem with 401 mesh points is 28.2 seconds compared to 4.8 seconds for a 4 level AMR method with the same fine grid resolution. The CPU time for 801 mesh points (122.4 seconds) is almost 13 times longer than for the five level AMR method (9.5 seconds).

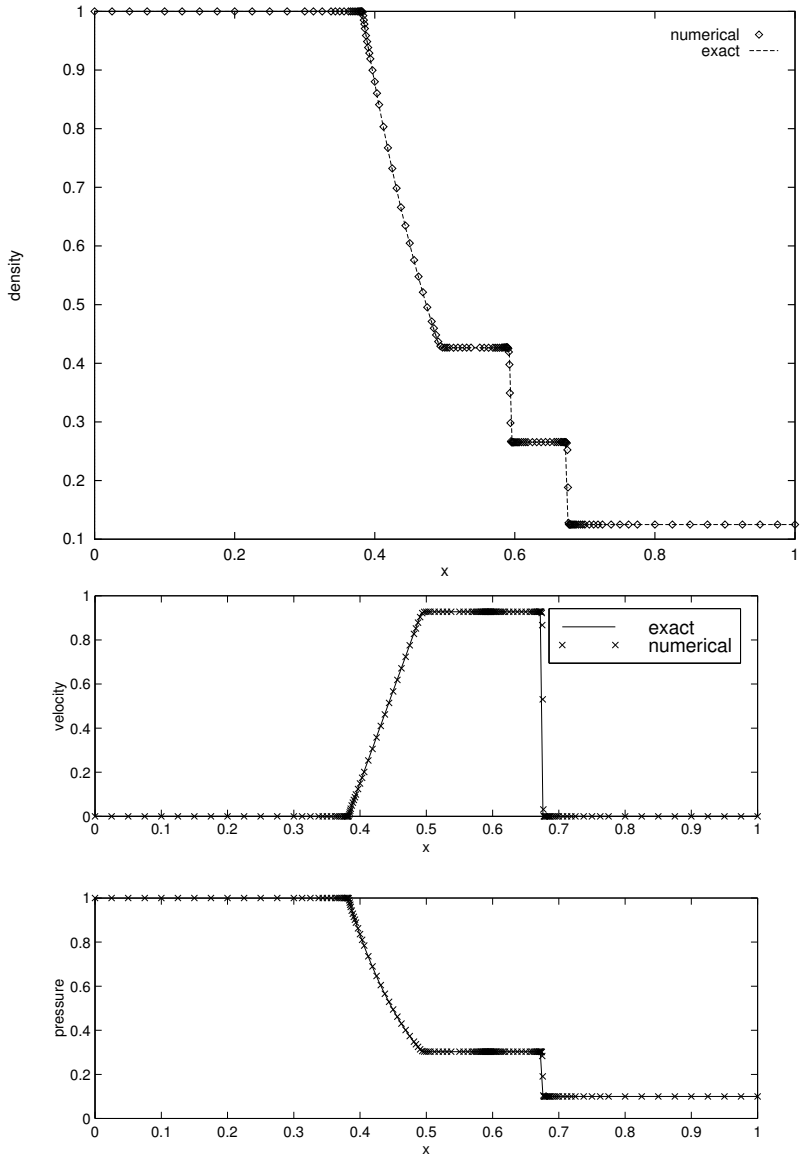

Figure 5.10: AMR with CLAWPACK solver for Sod's shock-tube problem. $t = 0.1$.

Note that on a uniform grid if the number of grid points double, the CPU time increases by more than a factor of 4, while it only doubled when increasing the number of levels for the AMR refinement. This is one of the advantages of choosing the time step locally for each patch.

5.6.2 Woodward-Colella blast wave problem

The interaction of two blast waves solutions of Eq. 7 has served a test example to compare different numerical schemes by Woodward and Colella [20]. The initial conditions are

$$(\rho, (\rho u), e) = \begin{cases} (1.0, 0.0, 2500.0), & \text{if } 0 \le x \le 0.1, \\ (1.0, 0.0, 0.025), & \text{if } 0.1 < x < 0.9, \\ (1.0, 0.0, 250.0), & \text{if } 0.9 \le x \le 1 \end{cases}$$

with the same reflecting boundary conditions as for Sod's shock-tube problem. The rapidly changing solution is difficult to resolve by moving mesh methods, but not by AMR. The spatial discretization is the second-order Roe's method with superbee limiter. We use a 50 points base grid and 6 refinement levels. The results in Figs. 5.11-a and 5.11-d compare very favorably with the high resolution solutions given in Ref. [20].

6 Conclusion

The numerical experiments, verify the effectiveness of the AMR method for 1-D problems. Our experience with other applications indicates that our AMR system easily accommodates new applications and requires very little modification of existing codes that work for a single grid. Moreover, the user can have full control over the refinement and integration. Our AMR method involves relatively little overhead in refinement and data management.

Our AMR system described here is for 1-D problems. However, most of the algorithms and the data structure have been extended to 2-D and are being extended for 3-D problems with minor modification. In higher dimensions, and the clustering algorithm is far more complicated, the existence of siblings and multiple parents for each patch introduce complications. We will describe some new features of our 2-D AMR system in Part II [13].

Acknowledgement The work was performed under the auspices of the U. S. Department of Energy (DOE) contract W-7405-ENG-36 and the DOE/MICS Program in Applied Mathematical Sciences.

References

[1] M. J. Berger and P. Colella, Local adaptive mesh refinement for shock hydrodynamics, *J. Comput. Phys.* 82 (1989), 64-84.

[2] M. J. Berger and J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, *J. Comput. Phys.* 53 (1984), 484-512.

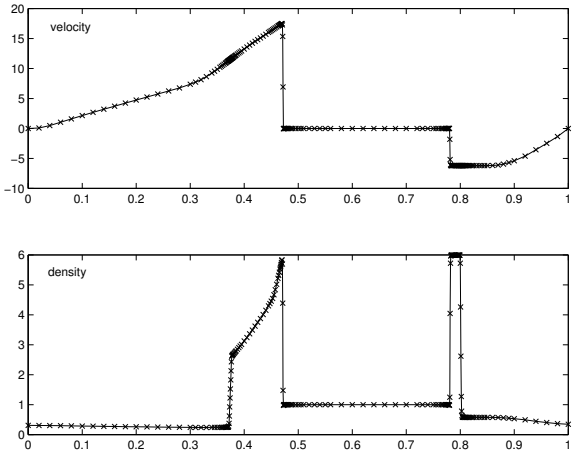

Figure 5.11-a: Two interacting blast waves problems $t = 0.016$.

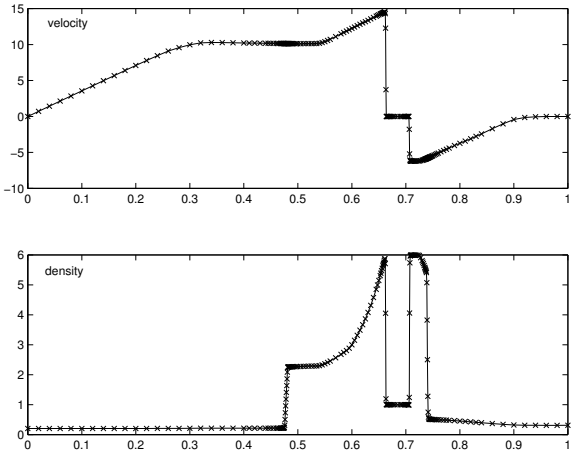

Figure 5.11-b: Two interacting blast waves problems $t = 0.026$.

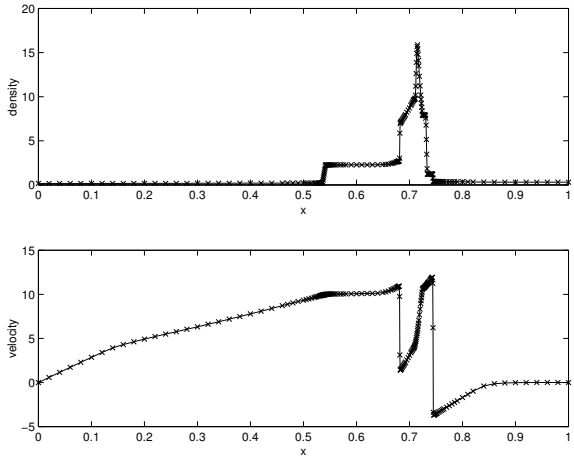

Figure 5.11-c: Two interacting blast waves problems $t = 0.032$.

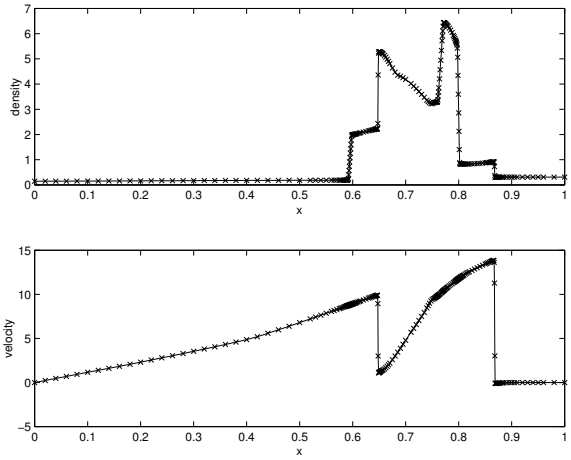

Figure 5.11-d: Two interacting blast waves problems $t = 0.038$.

[3] M. J. Berger, Data structures for adaptive grid generation, *SIAM J. Sci. Stat. Comput.* 3 (1986), 904-916.

[4] W. L. Briggs, A Multigrid Turorial, SIAM Pub., Philidelphai, PA (1987)

[5] S. K. Godunov, A finite-difference method for the numerical computation of discontinuous solutions of the equations of fluid dynamics, *Mat. Sb.*, 47 (1959), 271-290.

[6] J. M. Hyman, Accurate monotonicity preserving cubic interpolation, *SIAM J. Sci. Stat. Comp.*, 4 (1983), 645-654

[7] J. M. Hyman and M. Naughton, Static rezone methods for tensor-product grids in Lectures in Applied Mathematics, Vol.22, 321-343, 1985.

[8] B. Van Leer, Towards the ultimate conservative difference scheme II, monotonicity and conservation combined in a second order scheme, *J. Comput. Phys.* 14(1974), 361-470.

[9] R. LeVeque, CLAWPACK codes for hyperbolic conservative laws, in Netlib, 1995.

[10] X-D. Liu, S. Osher and T. Chan, Weighted essentially non-oscillatory schemes, *J. Comput. Phys.*, 115 (1994) 200-212.

[11] S. Li and L. Petzold, An efficient adaptive mesh method for one-dimensional time dependent PDEs, U. Minn. preprint, 1996

[12] S. Li, L. Petzold and Y. Ren, Stability of moving mesh systems of partial differential equations, *SIAM J. Sci. Comput.* 20 (1998), 719–739.

[13] J. M. Hyman, S. Li and L. R. Petzold, A simple AMR method for time-dependent PDEs, Part II, 2-D, Los Alamos National Lab. Report (1998)

[14] H. Neeman, Autonomous Hierarchical Adaptive Mesh Refinement for Multiscale Simulation, Ph.D thesis, (1996), Department of Computer Science, University of Illinois at Urbana-Champaign.

[15] J. Quirk, An Adaptive Grid Algorithm for Computational Shock Hydrodynamics, Ph.D thesis (1991), College of Aeronautics, Cranfield Institute of Tech.

[16] C. W. Shu and S. J. Osher, Efficient implementation of essentially non-oscillatory shock capturing schemes II, *J. Comput. Phys.*, 83(1989), 32-78.

[17] G. A. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, *J. Comput. Phys.* 43 (1978) 1-31.

[18] R. A. Trompert, Local uniform grid refinement for time-dependent partial differential equations, *CWI Tracts* 107, 1994.

[19] J. G. Verwer, J. G. Blom, VLUGR2: A vectorized local uniform grid refinement code for PDEs in 2D, *CWI Report NM-R9307*, (1993).

[20] P. R. Woodward and P. Colella, The numerical simulation of two-dimensional fluid with strong shocks, J. Comp. Phys. 54 (1984), 115-173.