LA-UR- 10-00017

Approved for public release;
distribution is unlimited.

Title: Adaptive Mesh Refinement for Shocks and Material Interfaces

Author(s): William W. Dai

Intended for: High Performance Distributed Computing 2010

Los Alamos
NATIONAL LABORATORY
EST.1943

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Form 836 (7/06)

Adaptive Mesh Refinement for Shocks and Material Interfaces

William W. Dai
Los Alamos National Laboratory
Mail Stop T080, Los Alamos, NM 87545
Phone: (505)665-1967; email: dai@lanl.gov

ABSTRACT

There are three kinds of adaptive mesh refinement (AMR) in structured meshes. Block-based AMR sometimes over refines meshes. Cell-based AMR treats cells cell by cell and thus loses the advantage of the nature of structured meshes. Patch-based AMR is intended to combine advantages of block- and cell-based AMR, i.e., the nature of structured meshes and sharp regions of refinement. But, patch-based AMR has its own difficulties. For example, patch-based AMR typically cannot preserve symmetries of physics problems. In this paper, we will present an approach for a patch-based AMR for hydrodynamics simulations. The approach consists of clustering, symmetry preserving, mesh continuity, flux correction, communications, management of patches, and load balance. The special features of this patch-based AMR include symmetry preserving, efficiency of refinement across shock fronts and material interfaces, special implementation of flux correction, and patch management in parallel computing environments. To demonstrate the capability of the AMR framework, we will show both two- and three-dimensional hydrodynamics simulations with many levels of refinement.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical Sciences and Engineering – Physics.

General Terms

Algorithms.

Keywords

adaptive mesh refinement, shock, Euler Equation, material interface.

1. INTRODUCTION

There are three kinds of adaptive mesh refinement (AMR) [1-22]. Block-based AMR, for example [2,20,21], refines a pre-defined block when any cell within the block is marked to be refined. The advantages of the block-based AMR include the nature of structured meshes of each block and relatively simple data structures for blocks. But it sometimes over refines meshes. Cell-based AMR [22] refines only those cells that are supposed to be refined, and therefore refined regions are well focused. But in cell-based AMR, cells, including the cells that are not refined, are typically treated cell by cell even for structured meshes. The connectivity of meshes is often described by connectivity arrays, not by i-, j-, and k-indices of structured meshes. Therefore cell-based AMR often loses the advantage of the nature of structured meshes. Patch-based AMR, for example [2], dynamically group those cells, which are supposed to be refined, into several groups through clustering algorithms, and these groups then form rectangular patches. Therefore, patch-based AMR combines the advantages of block- and cell-based AMR, i.e., the nature of structured meshes and the sharp regions of refinement. But, patch-based AMR has its own disadvantages. For example, existing patch-based AMR typically cannot preserve symmetries of physics problems. Since patches in patch-based AMR are dynamically generated and have different sizes, efficiently managing the patches in parallel environments presents a challenge when many levels of refinement are involved.

In this paper, we will present patch-based AMR through which refined meshes are able to preserve symmetries of physics problems. Through the AMR, the grid cells near shock fronts and material interfaces are refined. We will also implement a strategy to efficiently manage patches and load balance. The plan of this paper is as follows. The second section describes the patch-based AMR that includes clustering, symmetry preserving, management of patches, the "smooth" nature of AMR, the requirement for conservation laws, communications involved in AMR, and load balance. The third section is for hydrodynamics solvers within the AMR framework, which include the basic Euler equations, linear interpolation for internal structure within cells, Riemann solvers on fixed Eulerian grids, and the dimension-split technique. The section after that is for numerical examples for mesh refinement and hydrodynamics simulations within the framework. The last section is for conclusions of this paper and a brief discussion about symmetry and remaining issues we will work on.

2. PATCH-BASED AMR

In this section, we will present the algorithms and procedures we will use to form patches for a given set of prescribed cells, flagged cells, which normally are shock fronts and material interfaces

2.1 Clustering

K-means is one of simple learning algorithms developed by MacQueen [23] that solve the well known clustering problem. The procedure follows a simple way to classify a given data set through a certain number, k, clusters fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids should be initially placed in a cunning way because of different locations cause different results of clustering. Therefore, a good choice is to place the initial centroids as much as possible far away from each other. The next step is to take each cell belonging to a given data set and associate it to the nearest centroid. When no cell is pending, the first step is completed. At this point we need to re-calculate k new centroids of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid, and so on. As a result of this loop, the k centroids changes their location step by step until no more changes are done or centroids do not move any more.

One of important and desired requirements for a clustering algorithm to be used in AMR is the symmetry preserving. If flagged cells are symmetrical with respect to x- or y- or z-axis, the refined cells determined through clustering algorithms should preserve the symmetry. If this symmetry cannot be preserved in the mesh, physics solutions obtained from AMR will immediately loss the symmetry. Unfortunately, the most clustering algorithms used in AMR are unable to preserve the symmetry.

We are trying to find a simple and practical approach to make sure that the resulting mesh after refinement preserves the symmetry of original data. We should point out that it is the area covered by patches, not the patches themselves that are important for symmetry preserving.

For a given set of n cells that are to be refined on a part of mesh for which one computer processor is responsible, we pick a set of k points on the part of mesh to be used as initial centroids. The number k is on the scale of n. After the initial centroids are determined, we proceed with the k-means algorithm. The result of this procedure is the generation of k rectangles in space. But, in general, some rectangles can be merged to form a larger rectangle. Also, some rectangles may partially overlap with others, which are split and redundant parts are removed.

As we stated before, generally the patches formed through this procedure do not have symmetries, but the area covered by these patches preserves symmetries, which is what we are actually looking for. Figure 1 show the patches we generated (the left image) and the mesh after one level of refinement (the right image). The mesh in Fig.1 clearly shows the symmetries with respect to the axes of x- and y-axes and the diagonal directions of the simulation domain.

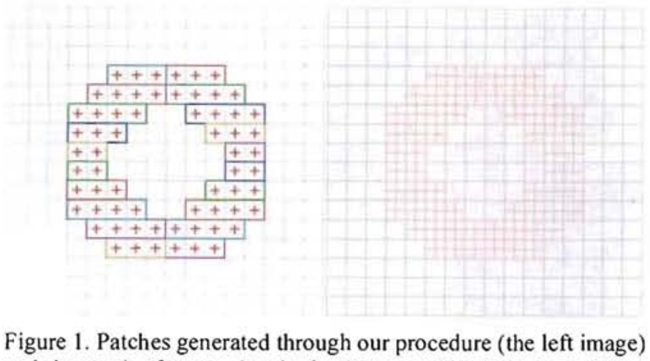
Figure 1. Patches generated through our procedure (the left image) and the mesh after one level of refinement (the right image) that shows the symmetries with respect to the x- and y-axes and the diagonal directions.

2.2 "Continuous" AMR

After the first level of refinement, we have two levels of cells, the base cells or cells of level zero, and the cells of level one. After getting the cells of level one, we can check cells against the refinement criterion to be used and apply clustering algorithms again to implement another level of refinement. Figure 2 shows the flagged cells that are to be further refined, the second level of patches, and the mesh after two levels of refinement. This procedure can continue until no more cells satisfy the refinement criterion. In this way, a mesh with a number of levels of refinement will be generated.

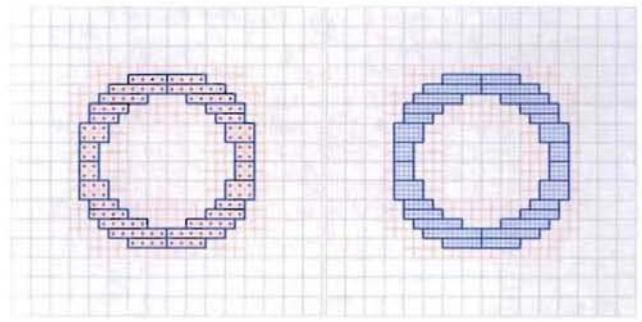
Figure 2. The flagged cells that are to be further refined and the patches resulted from the second level of refinement (the left image) and the mesh after two level of refinement (the right image).

It is often desired that mesh resolution smoothly change from locations to locations. If a cell of level m is to be further refined, any of its neighboring cells will be refined too if the neighboring cell is at (m-1) level. For example, if the cells marked with "+" in Fig.3 are to be further refined, their neighboring cells that are coarser are to be refined too. This requirement of continuous AMR will result in additional communications in parallel computer environments.

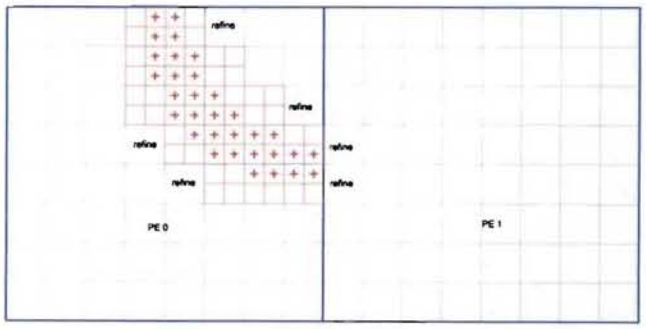

Figure 3. If the cells marked with "+" are to be further refined, their neighboring cells that are coarser have to be refined too. If the neighboring cells belong to other processors, communications are needed.

2.3 Ghost Cells and Their Communication

Each patch formed from clustering algorithms is a structured mesh on which physics solvers will be implemented. Therefore, the values of physics variables on the ghost cells have to be obtained before physics solvers can be implemented. The thickness of the layer of ghost cells depends on physics solvers to be used. Typically, higher order a solver is, thicker the layer is. In our AMR framework, the thickness of the layer of ghost cells is a parameter, which can be set to anything appropriate for the solvers to be used.

The values of physics variables on the ghost cells of a patch come from physics boundary conditions and the cells on the surrounding patches. The surrounding patches may be at the same level, or at a finer or coarser level. We require that the values of physics variables on any ghost cell come from the values of finest neighboring cells.

In our code, before each time step, each computer processor find all the neighboring patches of each of its own patches, pack all the variables on the areas that will be the ghost cells of the neighboring patches, and then send each of these datum buffers containing the variables to each of neighboring processors. Each neighboring processor receives one buffer from each of the neighboring processors, unpack the buffer, and put the values onto the ghost cells of each patch. In this way, for each time step, each processor communicates to each of its neighboring processors only once. We should point out that the list of neighboring processors doesn't change until the mesh is changed.

2.4 Management of Patches

The number of patches in patch-based AMR may dramatically increase with the sizes of physics problems and the number of levels of refinement. The way to manage these patches may significantly influence the performance of AMR in parallel environments.

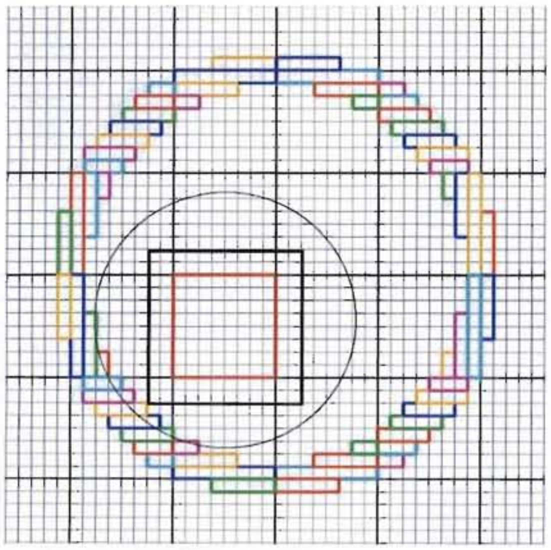

Figure 4. The base patch marked with the red square has eight neighboring base patches. Assume one processor owns only this patch. In addition to the patch owned by the processor, in Method 1 this processor knows all patches with all levels, in Method 2 this processor knows only patches reside in these eight neighboring patches, and in the approach we developed the processor knows only the patches overlapped by the bold black square.

A primitive approach to manage these patches, which will be labeled "Method 1" in subsequent figures, assumes the following. Each computer processor knows the metadata of all patches, no matter whether these patches are owned by the processor. Here "metadata" is used to stand for the locations, sizes, and level of patches. Metadata do not include the values of variables on patches. This approach is relatively simple in programming, but its performance is slower compared to other methods. More significantly, this approach will fail when the number of computer processors gets very large.

The second approach to manage patches, which will be labeled "Method 2" in our figures, is able to scale up with the processors and is faster in performance than the approach described above. In the second approach, in addition to the patches owned by the processor, each computer processor knows only those patches that reside in the neighboring base patches of each of the base patches owned by the processor. For example, as shown in Fig.4, if one processor owns only one base patch, for example, the base patch marked with the red square, the processor has eight (in two-dimensions) or twenty-six (in three-dimensions) neighboring base patches. In general, each processor and each base patch may have more neighboring base patches. In the method 2, this processor knows the metadata of only the patches reside in the neighboring base patches.

The third approach is the one we developed and used in this paper. For a given set of physics solvers, the thickness of ghost cells of each patch is determined. In addition to the patches owned by the processor, the computer processor knows only those patches that are needed for values of physics variables on the ghost cells of the patches owned by the processor. For the base patch marked with the red color in Fig.4, the processor knows only the patches overlapped by the bold black square. Therefore, this is the list with the minimum number of patches that related to the processor.

The performances of the three approaches to manage patches in parallel environments may be dramatically different when many levels of refinements are involved. In performance, the first two approaches are close to each other if only a small number of processors are involved, and the second and third approaches are close to each other if only few levels of refinement are involved. Figure 5 shows the costs of three approaches excluding the costs of physics solvers in two-dimensional simulations. The third approach is much more superior in performance when many processors and many levels of refinement are involved. Figure 6 shows the performance when more levels of refinement are involved. Figure 7 shows a three-dimensional case when five levels of cells are involved.

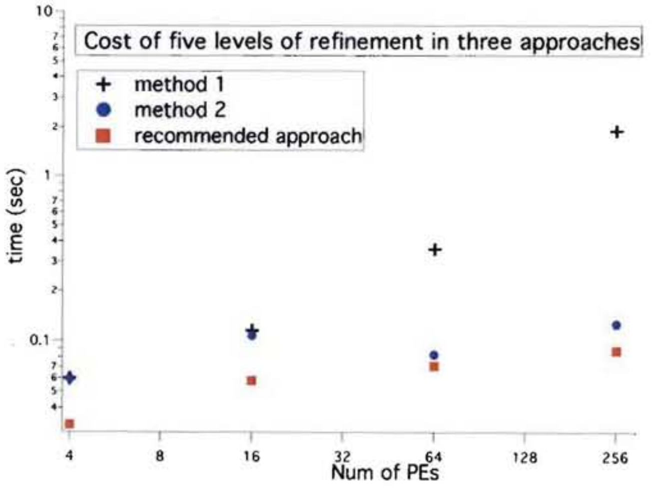

Figure 5. The costs to manage patches with five levels of cells in two-dimensional simulations.

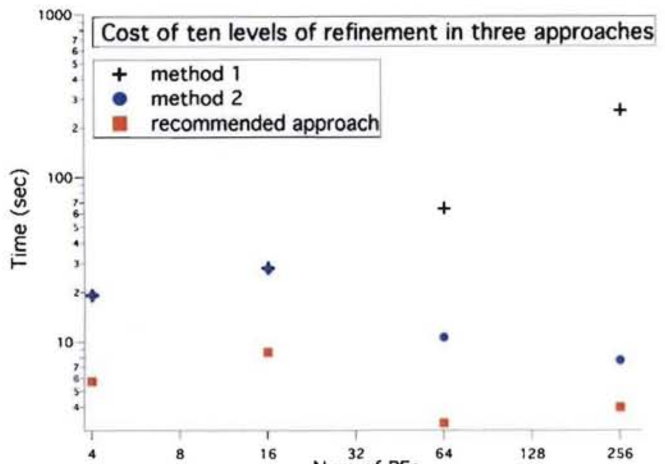

Figure 6. The costs to manage patches with eleven levels of cells in two-dimensional simulations.

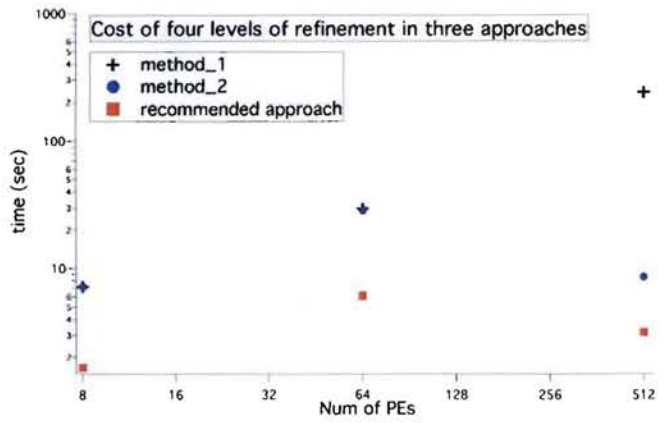

Figure 7. The costs to manage patches with five levels of cells in three-dimensional simulations.

2.5 Flux Correction for Conservation Laws

As block- and cell-based AMR, patch-based AMR may go through an additional step to enforce conservation laws in simulations for hydrodynamics shocks. As stated before, after patches are formed through clustering algorithms, a physics solver, for example, hydrodynamics solver, will be run on patches. Therefore, patch is a computational unit, and in principle, we may run physics in any order, from coarse to fine patches, or from fine to coarse patches. But, from the requirement of conservation laws, we will run hydrodynamics on finest patches first and then gradually to the coarsest patches.

As shown in Fig.8, there are two coarse patches and one fine patch. When we run hydrodynamics solver on the fine patch, we save the fluxes of mass, momentum, and energy at the four boundaries of the fine patch, which will be used to correct the fluxes on the coarse patches at the same physical locations when we run the solver on the coarse patches. If any coarse patch resides in other computer processors, communication is needed from the fine patch to the coarse patch.

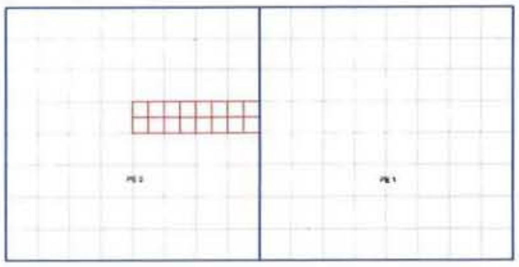

Figure 8. An illustration for flux correction. A hydrodynamics solver is run on the fine patch first, and the fluxes at the boundaries of the fine patch are saved and used to correct the fluxes on the same locations of the coarse patches. Communications are needed for the flux correction.

In our code, the flux correction is implemented as follows. When we run hydrodynamics solver on a patch, we first calculate the fluxes of mass, momentum, and energy at each cell interface. Then at the boundaries of its child patches, we replace the fluxes by those obtained during running hydrodynamics on the child patches. After this, we save and collect the fluxes at the boundaries of this patch for the flux correction of coarser patches. After we run all the patches with this level, we do one communication to send the fluxes at all the boundaries of the patches to other computer processors for the flux correction at coarser patches.

2.6 Load Balance

In a parallel computing environment, each computer processor is responsible for certain amount of workload. Because of the dynamics nature of shock fronts and material interfaces, the workload of each processor dynamically changes. Even if each processor has roughly same amount of workload initially, the workload may be out of balance during a simulation if load balance is not implemented.

For the load balance, we have to move data, variables, and metadata, from processors to processors. But, we would like to move as minimum data as possible. Therefore, the load balance is not a redistribution of workload. Our purpose is to move minimum data based on the current distribution of workload. Also, during the load balance, we would like to keep any patch and its child patches on the same processor.

Our procedure for the load balance is as follows. We first calculate the total amount of workload of each base patch and its all generations, for example, the total number of cells at all the levels. Base on this information and the number of processors in the simulation, we then estimate the optimized workload (e.g., the number of cells) for which each processor will be responsible. After this, we start to move base patches together with their generations of child patches toward this optimized goal with constraints. An example of the constraints is the connectivity of a moved patch with the target region. We would like to have a moved base patch physically connected to the target region although our code will work even if the regions for which each processor is responsible are physically disconnected.

We would like to point out that in our algorithm, the workload (e.g. the number of cells) of each processor will not be exactly the same after the load balance, since a parent and its child patches will be in the same processor. Figure 9 shows the distributions of cells in a mesh with ten levels among four processors before and after the load balance, in which each color is the region for which one processor is responsible. Before the load balance, two middle processors each have about 0.6 million cells while other two processors each have about 6000 cells. After the load balance, the middle two processors have about 305,000 cells each and other two processors have about 294,000 cells each.

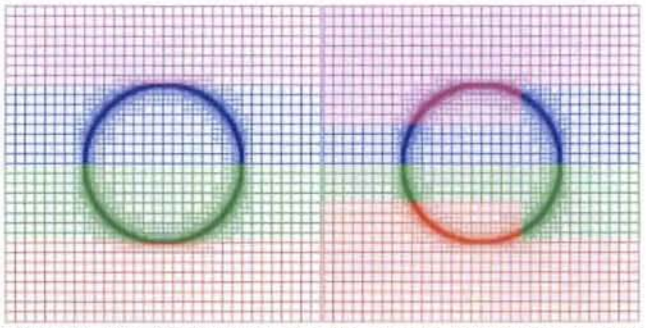
Figure 9. The distributions of cells among four PEs in a mesh with ten levels before (the left image) and after (the right image) the load balance. Each color represents the cells for which one PE is responsible. Before the balance, two middle processors each have about 0.6 million cells while other two processors each have about 6000 cells. After the balance, the middle two processors have about 305,000 cells each and other two processors have about 294,000 cells each.

3. NUMERICAL SCHEME FOR EULER EUATIONS

In this section we will describe the numerical scheme we use for multi-dimensional Euler equations in the framework of AMR described in the section above. In this particular solver, the multi-dimensionality is treated through the dimension split technique [27]. In each dimensional pass, linear interpolation is used for internal structure within cell. The nonlinear Riemann solver used in this paper is an extension of the Riemann solver for Lagrangian coordinate [28,29] to fixed Eulerian coordinate, which is based on shock jump conditions.

3.1 Basic Equations

The basic equations we will solve are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0,$$

$$\rho(\frac{\partial u}{\partial t} + u \cdot \nabla u) = -\nabla p,$$

$$\rho(\frac{\partial E}{\partial t} + u \cdot \nabla E) = -\nabla \cdot (pu).$$

Here ρ is density, u is the vector of flow velocity, p is thermal pressure, and E is specific total energy, $E = e + u^2/2$ with e being the specific internal energy. The variables, ρ, p, and e, are related to each other through the equation of state. We write the Euler equations in the form

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

Here

$$U \equiv \begin{pmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho u_z \\ \rho E \end{pmatrix}, \qquad F \equiv \begin{pmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_x u_y \\ \rho u_x u_z \\ u_x(\rho E + p) \end{pmatrix}.$$

Here u_x, u_y, and u_z are three component of vector u. G and H have similar expressions.

3.2 One-dimensional Pass

We will use the dimension split technique [27]. One-dimensional form of the basic equations becomes

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0.$$

Through any point in (x,t)-space, there exist three characteristic curves defined by $dx/dt = u_x(x,t)$ and $dx/dt = u_x \pm c_s$. Here c_s is the sound speed. Along the curve $dx/dt = u_x(x,t)$, $dR_0 = 0$, $du_y = 0$, and $du_z = 0$. Along the curves $dx/dt = u_x \pm c_s$, $dR_\pm = 0$ respectively. Here $C_s \equiv \rho c_s$, R_0 and $R_\pm$ are defined as

$$dR_0 \equiv d(p/\rho^\gamma),$$

$$dR_\pm \equiv dp \pm C_s du_x.$$

Considering a numerical grid $\{x_i\}$ in a one-dimensional domain, we integrate the basic equation within a grid cell $x_i \le x \le x_{i+1}$ and obtain

$$U_i(\Delta t) = U_i(0) + \frac{\Delta t}{\Delta x_i}[\overline{F}(x_i) - \overline{F}(x_{i+1})].$$

Here Δt and Δx are the size of time step and the width of the cell. $U_i(t)$ is the cell-average value of U within the cell at time t, $\overline{F}(x_i)$ is the time-averaged flux at cell interface x_i, and they are defined as

$$U_i(t) \equiv \frac{1}{\Delta x}\int_{x_i}^{x_{i+1}} U(t,x)dx,$$

$$\overline{F}(x_i) \equiv \frac{1}{\Delta t}\int_0^{\Delta t} F[U(t,x_i)]dt.$$

Therefore, one of the key points in the solver is the calculation of the time-averaged flux at cell interfaces. The time-averaged flux is approximately calculated through the time-averaged values of U at cell interfaces,

$$\overline{F}(x_i) \approx F[\overline{U}(x_i)].$$

The time-averaged value of U is calculated through an approximate Riemann solver.

A Riemann problem is an initial value problem with the initial condition:

$$U_0(x) = \begin{cases} U_L & if \quad x < 0, \\ U_R & if \quad x > 0. \end{cases}$$

Here U_L and U_R are any two constant states. We will modify the approximate Riemann solver developed in PPM [28,29] to fit to fixed Eulerian grids, since we prefer thinner layers of ghost cells. The solver in PPM was originally developed for Lagrangian hydrodynamics.

The solution of a Riemann problem generally includes either a shock or a rarefaction wave in each direction and a contact discontinuity. The waves in two directions and the contact discontinuity divide the whole (x-t)-space into four regions, L, R2, R3, and R, as shown in Fig.10. In the approximate solver, shock jump conditions are used for rarefaction waves. For a given left state U_L, the post-wave state for the wave propagating in the negative x-direction should be on the Rankin-Hugoniot (R-H) curve L (see Fig.11), and for a given right state U_R, the post-wave state for the wave propagating in the x-direction should be on another R-H curve R. The pressure p and the component of flow velocity u_x at the state U^* are what we are looking for, which are the interaction of the two R-H curves in Fig.11.

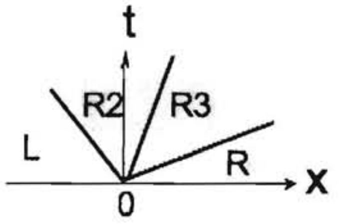

Figure 10. For a given pair of left and right states, the solution of a Riemann problem divides the whole x-t space into four possibly different states.

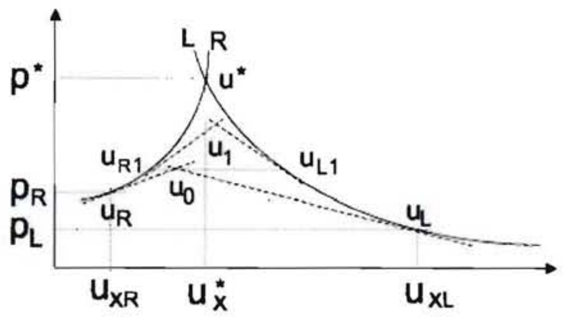

Figure 11. An illustration for iteration to find the state at the intersection of two Rankine-Hugoniot curves.

The values of p and u_x at the intersection may be iteratively obtained. Referring to Fig.11, as an initial step, we draw a pair of tangential lines U_LU_0 and U_RU_0 of the R-H curves, which pass through points U_L and U_R respectively. These two lines intersect with each other at the point $U_0(u_{x0}, p_0)$, which is approximately the initial solution of U^*. Then we draw a horizontal line $p = p_0$, which intersects with two R-H curves at points $U_{L1}(u_{xL1}, p_0)$ and $U_{R1}(u_{xR1}, p_0)$, respectively. After this, we draw another pair of tangential lines $U_{L1}U_1$ and $U_{R1}U_1$ of the R-H curves, which pass through points U_{L1} and U_{R1} respectively. The intersection U_1 is an approximate solution after the first iteration. We may repeat the steps stated above to obtain a more accurate solution.

The values of p and u_x at the intersection U^* are the time-averaged values in a Lagrangian coordinate. We have to approximately obtain the time-averaged values of p and u_x at the cell interface of fixed Eulerian grids. To do that, we first calculate the shock speeds in the Eulerian coordinate, s_r and s_l, for the shocks emerging from the cell interface, which propagate to the positive and negative x-directions respectively, for example,

$$s_r = w_r + u_{xR},$$
$$s_l = w_l - u_{xL}.$$

Here w_r and w_l are the speed of the shocks in a Lagrangian coordinate calculated from shock jump conditions. After we find the shock speeds, the time-averaged-values at the interface of the fixed Eulerian grid are obtained as the following

$$\overline{U} = \begin{cases} U_L & if \quad s_l > 0, \\ U_R & if \quad s_r < 0, \\ \frac{1}{2}(U_L + U_R) & if \quad s_l = 0 \quad and \quad s_r = 0, \\ U^* & otherwise. \end{cases}$$

The left and right states in the Riemann problem at the interface x_i are obtained through the values of U within domains of dependence with linear interpolations and a monotone condition for the internal structure within cells, which in fact is the MUSCL scheme [30].

3.3 Multi-dimensional Flows

The framework of the AMR described in Section 2 is suitable to dimensionally both split and unsplit solvers, but multi-dimensional flow to be presented in this paper is simulated through the dimensionally split technique [27]. For the two-dimensional case, the solution of Euler equation is obtained through applying a one-dimensional operator to each row and column of data in a two-dimensional domain. The solution of the equations after two time steps may be written as

$$U(2\Delta t) = L^x_{\Delta t} L^y_{\Delta t} L^y_{\Delta t} L^x_{\Delta t} U(0).$$

Here $L^x_{\Delta t}$ is the operator in the x-direction with time step Δt, which was described in the last subsection, and $L^y_{\Delta t}$ is the same operator but acting on the y-direction. It is worth to mention that the solution from this dimension split method will not preserve the symmetry along the diagonal direction of a two-dimensional square domain even for a uniform mesh without AMR involved.

For three-dimensional flows, the solution after two time steps may be obtained through

$$U(2\Delta t) = L^x_{\Delta t} L^y_{\Delta t} L^z_{\Delta t} L^z_{\Delta t} L^y_{\Delta t} L^x_{\Delta t} U(0).$$

Here $L^z_{\Delta t}$ is the one-dimensional operator acting on the z-direction. In a three-dimensional simulation with the dimension split technique described through the formula, y-dimension is special.

4. NUMERICAL EXAMPLES

In this section, we will present some examples to demonstrate the capability of the AMR framework. The first set of examples is for mesh refinement, in which there is no physics involved, and the second set of examples is for calculations of hydrodynamics that involve strong discontinuities.

4.1 Mesh Refinement

The first example is to show how the strategy of "continuous" AMR discussed in Section 2.3 influence meshes. Figure 12 shows meshes with five levels of cells without and with the continuous AMR requirement implemented. In the left image, the neighboring cells of a twice-refined cell may be not refined. In the right image, the cells are gradually refined to the finest level from location to location.

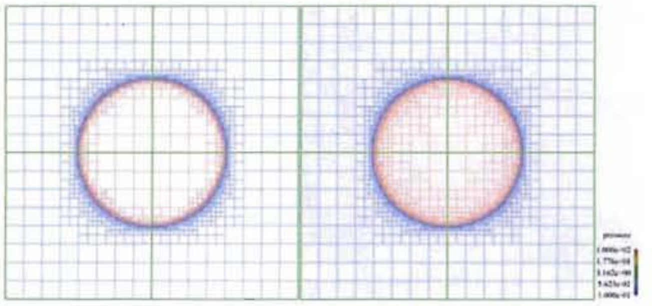

Figure 12. The left image is generated without and the right one is generated with the "continuous" AMR requirement implemented.

The second example is to check the meshes obtained from the different numbers of computer processor elements (PE) and the different numbers of base patches. The mesh shown in the left image of Fig.13 has ten levels of cells, obtained from a single PE with four base patches. The mesh shown in the right image of the figure is the mesh obtained from four PEs and sixteen base patches. These two meshes are identical as they are expected.

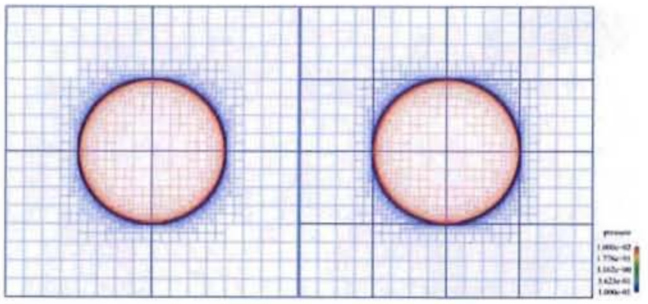

Figure 13. Two meshes generated through different numbers of PEs and different numbers of base patches. The mesh to the left is obtained through a single PE and four base patches, and the mesh to the right is obtained through four PEs and sixteen base patches. The two meshes are identical, as expected.

The third example is to show the capability to generate meshes with many levels of refinement. Figure 14 shows an example mesh that has twenty levels of cells near a circular shock front. Each color in the upper left image represents one level of cells. Only the first few levels of coarse cells are recognizable. The upper right image in the figure shows the coarsest and finest cells. As expected, the finest cells seem to be a thin line. The bottom image in the figure shows the finest seven levels of cells near one location. The numbers of the twenty levels of cells in this mesh, from the coarsest to finest, are 256, 304, 592, 1166, 2320, 4624, 9232, 18448, 36880, 73744, 147440, 294864, 589520, 1178128, 2352432, 4690128, 9321936, 18406656, 35890816, 68047520 respectively. If one cell of a base patch were fully refined 19 times, the number of the finest cells in the cell would be 274,877,906,944.

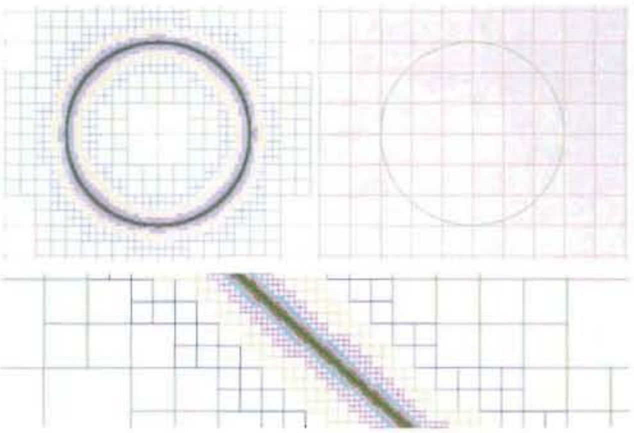

Figure 14. A mesh with 20 levels of cell near a circular shock front (the upper left image), the coarsest and finest levels of cell (the upper right image), and the finest seven levels of cell near one location (the bottom image).

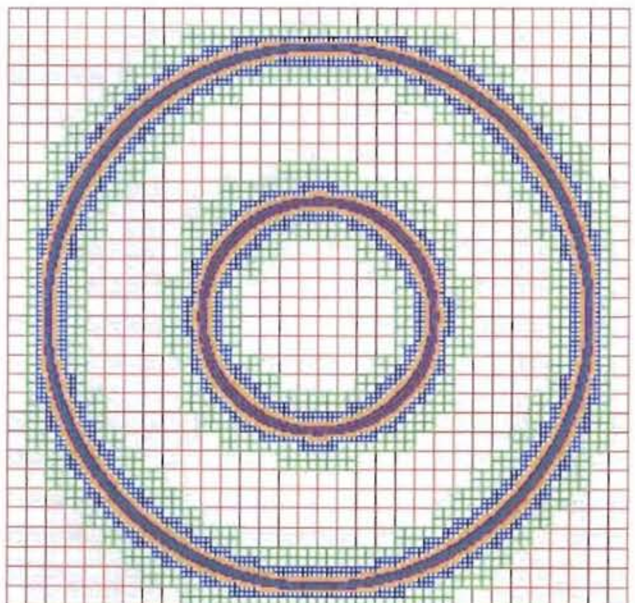

Figure 15. Eighteen levels of refinement near a circular material interface (the inner circle) and a circular shock front (the outer circle).

The fourth example is about the refinement near material interfaces as well as shock fronts. Figure 15 displays eighteen levels of refinement near a circular material interface (the inner circle) and a circular shock front (the outer circle). As stated before, we refine any cell that contain more than one material or whose neighboring cells have different materials. The cells at the same level are painted with one color in the figure.

As stated before, the mesh refinement will be applied to shock fronts as well as material interfaces. For material interfaces, we will refine any cell with more than one material or mixed cell, and any pure cell whose neighboring cells contain other materials. Figure 15 shows a refined mesh with nineteen levels of cells, a circular shock front, and a circular material interface

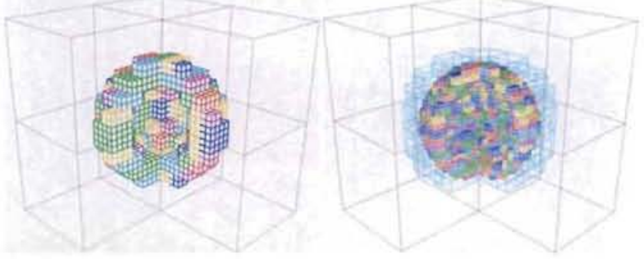
Figure 16. The first level of refinement near a spherical shock front (the left image) and the patches after the second level of refinement.

Although we have not specifically mentioned AMR for the three-dimensional case in Section 2, the statements in that section are applicable to the three-dimensional meshes. The next two examples involve three-dimensional meshes. The left image in Fig.16 shows the first level of patches near a spherical shock front, and the image to the right shows the patches resulting from the second level of refinement. It is possible to visually check the symmetries of the refinement with respect to the x- and y-axes.

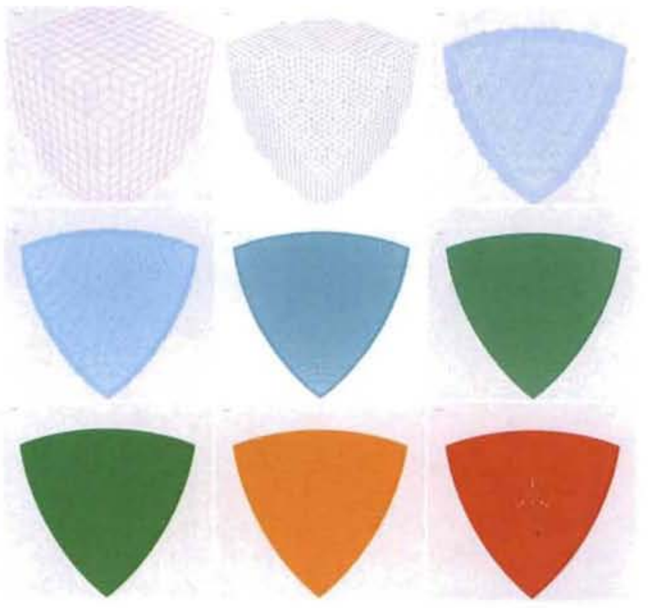
Figure 17. One eighth of ten levels of cell near a spherical shock front. The cells at each level are supposed to be buried in the cells of the previous level.

Figure 17 shows one eighth of refined cells near a spherical shock front with ten levels of cells. Cells at each level are actually buried in the cells of the previous level. The white dots in the last image indicates that the cells of the second finest do not undergo the refinement at these spots.

4.2 Hydrodynamics Calculations

In this subsection, we will present two hydrodynamics calculations. In these calculations, the periodic boundary condition is used, and initial conditions are $\rho = 1$, $u = (0,0,0)$, and

$$p = \begin{cases} p_{inner} & if \quad r < r_0 \\ p_{outer} & if \quad r > r_0 \end{cases}.$$

Here p_{inner}, p_{outer}, and r_0 are constants, and they are 0.1, 1000, and 0.5 respectively. The simulation domain is (8×8) for two dimensions or $(8 \times 8 \times 8)$ for three dimensions. The size of the base mesh is (16×16) or $(16 \times 16 \times 16)$. The jump in pressure is used as the criterion for the refinement of cells.

In the first calculation, Figure 18 shows the distribution of pressure at t = 0 and 0.68. In this example there are ten levels of cells, the location of refinement follows the shock propagating outward. Through careful examination of the images in the figure, it is found out that the mesh preserves symmetries with respect to x- and y-axes.

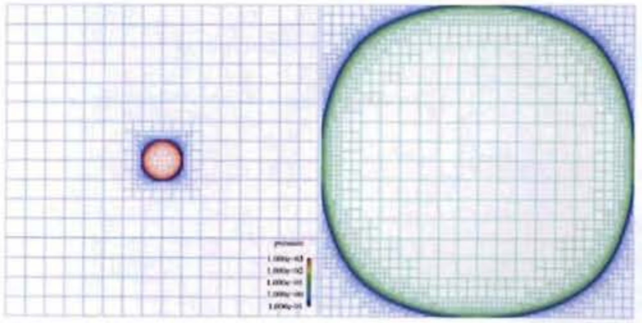
Figure 18. A calculation of two-dimensional flow. The simulation domain is 8x8x8. Initial density is constant, flow is at rest, and pressure is 1000 at the center r < 0.5, and 0.1 when r > 0.5. The periodic boundary condition is used. The image to the left is the initial pressure, and the image to the right is pressure at t = 0.68.

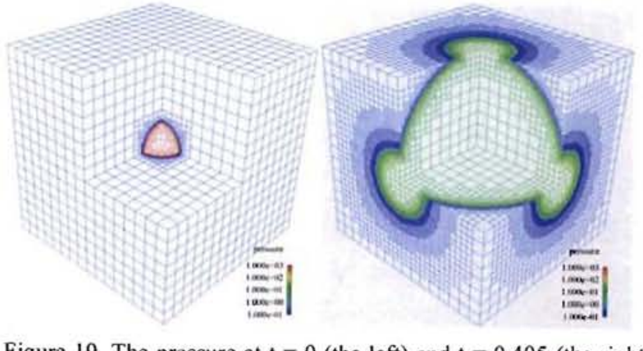
Figure 19. The pressure at t = 0 (the left) and t = 0.405 (the right image) in a 3D simulation. Periodic boundary condition is used.

The second example is a three-dimensional problem, which is the extension of problem described above in three dimensions. Here we limit the number of refinement to 4, and there are five levels of cells in the problem. Figure 19 shows the pressure and meshes at the initial time (the left image) and at $t = 0.405$ (the right image).

5. CONCLUSIONS and DISCUSSIONS

In this paper, we have developed and implemented a patch-based AMR that preserves symmetries of original physics problems. In the implementation, we have emphasized symmetry preserving, effective patch management, smoothness of refinement, efficient communication for the values on ghost cells of patches; we have compared three different approaches to manage patches. The approach, in which each computer processor holds the metadata of the minimum number of patches, performs much better than other two in both two- and three-dimensions. For hydrodynamics algorithms, we modified a nonlinear Riemann solver in Lagrangian coordinate for its use in fixed Eulerian grids. Preliminary numerical examples show that this patch-based AMR is able to keep shocks front very sharp.

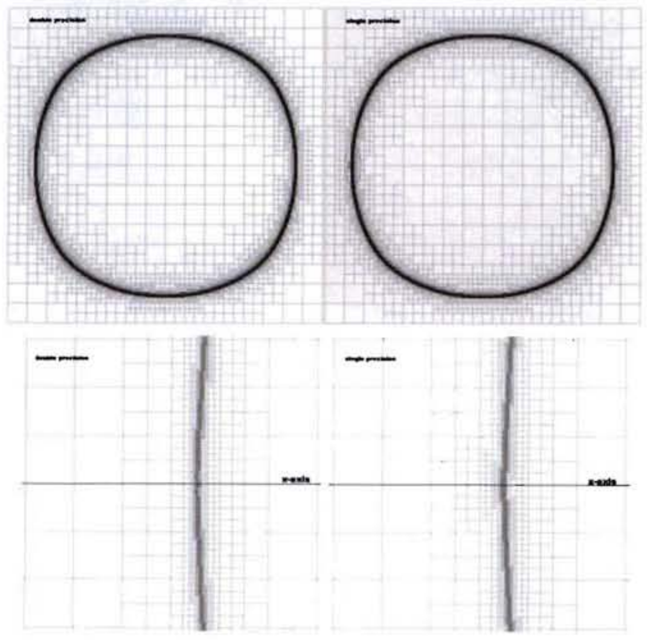

Figure 20. The resulting meshes of two simulations with the same code, one with double precision (the upper left image) and the other with single precision (the upper right image). The lower two images are detailed parts of the upper two. The symmetry with respect to x-axis is broken in the lower right image of single precision.

The capability for a computer code to preserve symmetries in a numerical simulation depends on many factors, including physics solvers and machine rounding-off errors. Rounding-off errors are much more visible in the simulations involving AMR than in those without AMR involved.

Whether a cell is to be refined may be determined by rounding-off errors. Also rounding-off errors may destroy the symmetries of problems. To demonstrate this point, we use the same computer code to simulate the same problem twice, one through double precision and the other through single precision. Figure 20 shows two meshes at one instant, one (the upper left image) from the simulation with double precision and the other (the upper right image) from single precision. Because of machine rounding-off errors, the two meshes are different. Furthermore, the simulation with single precision breaks the symmetries, for example, with respect to the x-axis. To see more details of the two meshes, the parts marked with ovals are enlarged, as shown in the lower two images of Fig.20.

There are several remaining issues to be addressed in our work. It is in our plan to add multi-materials physics solvers to the AMR package to demonstrate the advantages of compressed material data structures. We have noticed that in applications with many materials involved, sparse material data structures will eventually consume all the computer memories, which will limit the capability of computer codes to do large-scale simulations with many materials.

6. ACKNOWLEDGMENTS

The work presented here has been supported by the Department of Energy through its ASC program. The author is very grateful to Paul Woodward, Robert Lowrie, John Wohlbier, and Shangtai Li for discussions about AMR.

7. REFERENCES

[1] CHOMBO, https://seesar.lbl.gov/anag/chombo/index.html

[2] PARAMESH, http://www.physics.drexel.edu/~olson/paramesh-doc/Users_manual/amr.html

[3] SAMRAI, https://computation.llnl.gov/casc/SAMRAI/index.html

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

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

[6] M. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, J. of Comput. Phys. **157** (1989) 64-84.

[7] J. B. Bell, M. J. Berger, J. S. Saltzman, and M. Welcome. A three–dimensional adaptive mesh refinement for hyperbolic conservation laws. SIAM J. on Scientific Computing **15** (1994) 127–138.

[8] M. Combi, K. Kabin, T. Gombosi, D. De Zeeuw, K. Powell, Io's plasma environment during the galileo flyby: global three-dimensional MHD modeling with adaptive mesh refinement, J. of Geophysical Research-Space Physics **103** (1998) 9071-9081.

[9] M.N. Murty, A.K. Jain, P.J. Flynn, Data clustering: a review, ACM Compt. Surv. **31** (1999) 264-323.

[10] P. MacNeice, K. M. Olson, C. Mobarry, R. deFainchtein, C. Packer, PARAMESH : A parallel adaptive mesh refinement community toolkit, Computer Physics Communications **126** (2000) 330-354.

[11] Dawes, Parallel multi-dimensional and multi-material Eulerian staggered mesh schemes using localized patched based adaptive mesh refinement (AMR) for strong shock wave phenomena, Adaptive Mesh Refinement – Theory and Applications, T. Plewa, T. Linde, V.G. Weirs (Eds.), Vol. **41**, Lecture Notes in Computational Science and Engineering, Springer, (2003) 295-302.

[12] H. Jourdren, HERA: A hydrodynamic AMR platform for multi-physics simulations, Adaptive Mesh Refinement – Theory and Applications, T. Plewa, T. Linde, V.G. Weirs (Eds.), Vol. **41**, Lecture Notes in Computational Science and Engineering, Springer, (2003) 284-294.

[13] A. Theodorakakos, G. Bergeles, Simulation of sharp gas liquid interface using VOF method and adaptive grid local refinement around the interface, Int. J. Numer. Meth. Fluids **45** (2004) 421-439.

[14] T. Plewa, T. Linde, V.G. Weirs (Eds.), Adaptive Mesh Refinement – Theory and Applications, Vol. **41**, Lecture Notes in Computational Science and Engineering, Springer, 2003, Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, September 3-5, 2003.

[15] R. Anderson, N. Elliott, R. Pember, An arbitrary Lagrangian-Eulerian method with adaptive mesh refinement for the solution of Euler equations, J. of Comput. Phys. **199** (2004) 598-617.

[16] A. Kongies, R. Anderson, P. Wang, B. Gunney, R. Becker, D. Eder, B. MacGowan, M. Schneider, Modeling NIF experimental designs with adaptive mesh refinement and Lagrangian hydrodynamics, Inertial Fusion Sciences and Applications 2005, J. Phys. IV France 133.

[17] B. T. N. Gunney, A. M. Wissink, D. A. Hysom, Parallel Clustering Algorithms for Structured AMR, J. of Parallel and Distributed Computing **66** (2006) 1419-1430.

[18] M. Malik, E.S.-C. Fan, M. Bussmann, Adaptive VOF with curvature-based refinement, International J. for Numer. Methods in Fluids **55** (2007) 693-712.

[19] J. Morrell, P. Sweby, A. Barlow, A cell by cell anisotropic mesh ALE method, Int. J. Numer. Meth. Fluid, DOI:10:1002/fld.1599.

[20] K. Olson and P. MacNeice, 2005, An Over of the PARAMESH AMR Software and Some of Its Applications, in Adaptive Mesh Refinement-Theory and Applications, Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Series: Lecture Notes in Computational Science and Engineering, Vol. **41**, eds. T. Plewa, T. Linde, and G. Weirs (Berlin: Springer).

[21] K. Olson, 2006, PARAMESH: A Parallel Adaptive Grid Tool, in Parallel Computational Fluid Dynamics 2005: Theory and Applications: Proceedings of the Parallel CFD Conference, College Park, MD, U.S.A., eds. A. Deane, A. Ecer, G. Brenner, D. Emerson, J. McDonough, J. Periaux, N. Satofuka, and D. Tromeur-Dervout (Elsevier).

[22] M. Gittings, R. Weaver, M. Clover, T. Betlach, N. Byrne, R. Coker, E. Dendy, R. Hueckstaedt, K. New, W. R. Oakes, D. Rantal, R. Stefan, The RAGE radiation-hydrodynamic code, Comput. Sci. Disc. **1** (2008) 015005, doi:10.1088/1749-4699/1/1/015005.

[23] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, Proceedings of 5th Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 281-297 (1967).

[24] M. Berger, I. Rigoutsos, An algorithm for point clustering and grid generation, IEEE Transactions Systems, Man, and Cybernetics **21** (1991) 1278-1286.

[25] M.N. Murty, A.K. Jain, P.J. Flynn, Data clustering: a review, ACM Compt. Surv. **31** (1999) 264-323.

[26] A. Likas, N. Vlassis, J.J. Verbeek, The global k-means clustering algorithm, Pattern Recognition **36** (2003) 451-461.

[27] G. Strang, On construction and comparison of differential schemes, SIAM J. Numer. Anal. 5 (1968) 506-517.

[28] P. Colella and P. R. Woodward, The piecewise parabolic method (PPM) for gas-dynamical simulations, J. Comput. Phys. **54** (1984) 174-201.

[29] P. R. Woodward, Piecewise-parabolic methods for astrophysical fluid dynamics, in Astrophysical Radiation Hydrodynamics, K.-H.A. Winkler and M.L. Norman (Eds.), D. Reidel Publishing Company, 245-326 (1986).

[30] B. Van Leer, Toward the ultimate conservative difference scheme, V. a second-order sequel to Godunov's method, J. Comput. Phys. **32** (1979) 101-136.