Issues in Adaptive Mesh Refinement

William W. Dai
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, New Mexico, USA
E-mail: dai@lanl.gov

Abstract—In this paper, we present an approach for a patch-based adaptive mesh refinement (AMR) for multi-physics simulations. The approach consists of clustering, symmetry preserving, mesh continuity, flux correction, communications, management of patches, and dynamical load balance. Among the special features of this patch-based AMR are symmetry preserving, efficiency of refinement, special implementation of flux correction, and patch management in parallel computing environments. Here, higher efficiency of refinement means less unnecessarily refined cells for a given set of cells to be refined. To demonstrate the capability of the AMR framework, hydrodynamics simulations with many levels of refinement are shown in both two- and three-dimensions.

Keywords—adaptive mesh refinement; compressible fluid dynamics; shock waves; Euler equations.

I. INTRODUCTION

There are three kinds of AMR [1-12]. Block-based AMR, for example [2,8], 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. On the other hand, cell-based AMR [12] 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 that 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 sharp regions of refinement. But, patch-based AMR has its own disadvantages. 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 an approach for patch-based AMR through which refined meshes are able to preserve symmetries of physics problems. We will also implement a strategy to efficiently manage patches. We use the k-means clustering algorithm [13] for the generation of patches, with additional steps for the capability to preserve symmetries. Although the framework we are developing is for multi-physics simulations, we will present hydrodynamics simulations within the framework.

The plan of this paper is as follows. The second section describes the issues in patch-based AMR that includes clustering algorithms, symmetry preserving, management of patches, the “smooth” nature of AMR, the requirement for conservation laws, communications involved in AMR, and dynamical 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.

II. PATCH-BASED ADAPTIVE MESH REFINEMENT

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

A. Clustering

K-means is one of simple learning algorithms developed by MacQueen [13] that solve the well known clustering problem. 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, 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 a refinement preserves the symmetry of original data. We should point out that it is areas covered by patches, not 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 owned my one computer processor, we pick a set of k points on the part of mesh to be used as initial centroids. The number k is in the order of n. For the symmetry
preserving. It is extremely important for the set of k points to be symmetrical. 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 partially overlapped with others, which will be split and redundant parts will be removed.

Figure 1. Patches generated through our procedure. The cells on the patches will be refined and the resulting mesh shows the symmetries with respect to the axes of x and y and the diagonal directions.

As we stated before, in general 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 shows the flagged cells and the patches we generated. The cells on the patches will be refined and the resulting mesh shows the symmetries with respect to the x- and y-axes and the diagonal directions of the simulation domain.

B. “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 a refinement criterion against the cells and apply clustering algorithms again to implement another level of refinement. This procedure can continue until no refined cells satisfy the refinement criterion. In this way, a mesh with a number of levels of refinement will be generated.

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. This requirement of continuous AMR will result in additional communications in parallel computer environments.

C. Ghost Cells of Patches and Their Communications

Each patch formed from clustering algorithms is a structured mesh on which physics solvers will be implemented. Therefore, ghost cells surrounding the patch and 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 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 its neighboring processors, unpacks the buffer, and puts the values onto the ghost cells of each patch. In this way, for each time step, each processor sends to and receives from 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.

D. Management of Patches

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

Figure 2. The costs to manage patches with eleven levels of cells in two-dimensional simulations.

A primitive approach to manage these patches, which will be labeled “method 1” in the subsequent figure, 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 large.
E. 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 move to the coarsest patches.

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 any 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 of all the boundaries of the patches to neighboring processors for the flux correction at coarser patches.

F. Dynamical 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. Based on this information and the number of processors in the simulation, we then estimate the optimum workload (e.g., the number of cells) for 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.

Figure 3. 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 load balance, two middle processors each have about 0.6 million cells while other two processors each have about 6000 cells. After the load, the middle two processors have about 305,000 cells each and other two processors have about 294,000 cells each.

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 3 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.

III. NUMERICAL SCHEME FOR EULER EQUATIONS

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 [17]. In each dimensional pass, linear interpolation is used for the internal structure within a cell. The nonlinear Riemann solver used in this paper is an extension of the Riemann solver for Lagrangian coordinate [18,19] to fixed Eulerian coordinate, which is based on shock jump conditions.

### A. Basic Equations

The basic equations we will solve are

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

\[
\rho \frac{\partial u}{\partial t} + \rho u \cdot \nabla u = -\nabla p,
\]

\[
\rho \frac{\partial E}{\partial t} + \rho u \cdot \nabla E = -\nabla \cdot (p u).
\]

Here \( \rho \) 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, \( \rho, p, \) 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 \( U \) is a conserved quantity. \( F, G, \) and \( H \) are fluxes.

### B. One-dimensional Pass

We will use the dimension split technique [17]. One-dimensional form of the basic equations becomes

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

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

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

Here \( \Delta t \) and \( \Delta x \) 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 \( t \), \( \bar{F}(x) \) is the time-averaged flux at cell interface \( x \).

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 \( \bar{U} \) at cell interfaces,

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

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: \( U_L(x) = U_{L_i} \) if \( x < 0 \), and \( U_R(x) = U_{R_k} \) if \( x > 0 \). Here \( U_L \) and \( U_R \) are any two constant states. We will modify the approximate Riemann solver developed in PPM [18,19] to fit to fixed Eulerian grids, since we prefer thinner layers of ghost cells. The Riemann solver in PPM was originally developed for Lagrangian hydrodynamics, but we need the time-averaged values at grid points of an Eulerian grid. To do that, we first solve Riemann problem on Lagrangian coordinate, and the 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,

\[
s_r = w_r + u_{xL},
\]

\[
s_l = w_l - u_{xL}.
\]

Here \( w_r \) and \( w_l \) are the speed of the shocks in a Lagrangian coordinate calculated from Riemann solver on the Lagrangian grid. After we find the shock speeds, the time-averaged-values at the interface of the fixed Eulerian grid are obtained as the following. \( \bar{U} = U_L \) if \( s_r > 0 \); \( \bar{U} = U_R \) if \( s_l < 0 \); \( \bar{U} = (U_L + U_R)/2 \) if \( s_r = 0 \) and \( s_l = 0 \); and otherwise \( \bar{U} = U' \) that is the solution at the interface of the Lagrangian grid.

The left and right states in the Riemann problem at the interface \( x \) 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 [20].

### C. 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 [17]. For the two-dimensional case, the solution of Eulerian 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

\[
U(2\Delta t) = L_{xL}^\prime L_{yL}^\prime L_{yR}^\prime L_{xR}^\prime U(0).
\]

Here \( L_{xL}^\prime \) is the operator in the x-direction with time step \( \Delta t \), which was described in the last subsection, and \( L_{yL}^\prime \) 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

\[
U(2\Delta t) = L_{xL}^\prime L_{yL}^\prime L_{zL}^\prime L_{yR}^\prime L_{zR}^\prime L_{xR}^\prime U(0).
\]

Here \( L_{zL}^\prime \) 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.
Figure 4. The mesh at the top is obtained through a single computer processor and four base patches, and the mesh at the bottom is obtained through four processors and sixteen base patches. The two meshes are identical, as expected.

**IV. 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 about simple calculations of hydrodynamics that involve strong discontinuities.

**A. Mesh Refinement**

The first 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 at the top of Fig.4 has ten levels of cells, obtained from a single PE with four base patches. The mesh shown at the bottom is the mesh obtained from four PEs and sixteen base patches. These two meshes are identical as they are expected.

Figure 5. Twenty levels of cells near a circular shock front. Each color represents one level of cells.

The second example is to show the capability to generate meshes with many levels of refinement. Figure 5 shows an example mesh that has twenty levels of cells near a circular shock front. Each color in the image represents one level of cells. Only the first few levels of coarse cells are recognizable. The numbers of the twenty levels of cell in this mesh, from the coarsest to finest, are 256, 304, 592, 1166, 2320, 4624, 9232, 18448, 36880, 73744, 147484, 294864, 589520, 1178128, 2352432, 4690128, 9321936, 18406656, 35890816, 68047520. If one cell of a base patch were fully refined 19 times, the number of the finest cells in one base cell would be 274,877,906,944.

Figure 6. A mesh with fifteen levels of cells near a circular shock front and material interfaces of the five materials involved. The color represents pressure.
To resolve material interfaces, we will refine any mixed cell as well as any pure cell whose neighboring cell contains different materials. Figure 6 displays a mesh with fifteen levels of cells near a circular shock front and the circular interfaces of five materials. The color represents values of pressure on the mesh. Actually, there are many mixed cells near the interface between any two materials, although they are not recognizable in the figure.

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 upper image in Fig.7 shows the first level of patches near a spherical shock front, and the lower image 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.

B. Hydrodynamics Calculations

In this subsection, we will present two hydrodynamics calculations. In these calculations, the periodic boundary condition is used, and initial conditions are \( \rho = 1, \ u = 0 \). \( p = p_{\text{inner}} \) if \( r < r_0 \), and \( p = p_{\text{outer}} \) if \( r > r_0 \). Here \( p_{\text{inner}}, p_{\text{outer}} \), and \( r_0 \) are constants, and they are 0.1, 1000, and 0.5 respectively. The simulation domain is \((8 \times 8)\) for two dimensions or \((8 \times 8 \times 8)\) for three dimensions. The size of the base mesh is \((16 \times 16)\) 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 8 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 a 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 9. An example of three-dimensional simulations. The upper image is the initial pressure, and the lower image is the pressure at $t = 0.405$. Initially density is one everywhere, and flow is at rest. The 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 9 shows the pressure and mesh at the initial time (the upper image) and at $t = 0.405$ (the lower image).

V. 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, flux correction for conservation laws, and dynamical load balance; 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 shock fronts and material interfaces very sharp.

Figure 10. The resulting meshes of two simulations with the same code, one with double precision and the other with 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. Machine 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 machine 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 10 shows two meshes at one instant, one (the upper image) from the simulation with double precision and the other (the lower 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 the symmetry, the part of images in Fig. 10 for the left edge of the shock is enlarged in Fig. 11.

Figure 11. The enlarged parts of the resulting meshes of two simulations with the same code, one with double precision and the other with single precision. The symmetry with respect to x-axis is broken in the lower image.

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.

VI. 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.

REFERENCES