ADAPTIVELY–REFINED OVERLAPPING GRIDS FOR THE NUMERICAL SOLUTION OF SYSTEMS OF HYPERBOLIC CONSERVATION LAWS *

Kristi D. Brislawn, David L. Brown, Geoffrey S. Chesshire and Jeffrey S. Saltzman
Los Alamos National Laboratory
Los Alamos, NM

SUMMARY

Adaptive mesh refinement (AMR) in conjunction with higher-order upwind finite-difference methods has been used effectively on a variety of problems in two and three dimensions. In this paper we introduce an approach for resolving problems that involve complex geometries in which resolution of boundary geometry is important. The complex geometry is represented by using the method of overlapping grids, while local resolution is obtained by refining each component grid with the AMR algorithm, appropriately generalized for this situation. The CMPGRD algorithm introduced by Chesshire and Henshaw is used to automatically generate the overlapping grid structure for the underlying mesh.

INTRODUCTION

Over the past decade, the Adaptive Mesh Refinement (AMR) algorithm pioneered by Berger and Oliger [1] has proven to be a successful, efficient strategy for obtaining high-resolution solutions to partial differential equations. Using AMR combined with high-order upwind finite-difference methods, one has the ability to simulate shock hydrodynamics problems, including those with multiple materials, in both 2-D and 3-D [2, 3, 4, 5, 6] not otherwise possible within the limitations of present computers. To date most of the developmental work done on the AMR method has concentrated on the perfection of the adaptive algorithm, and not on the development of the capability to represent complex geometry. A notable exception is the "Cartesian grid" method introduced by Berger and Leveque in [7] in which complex geometry is represented by cutting holes in an otherwise rectangular grid, and using special flux formulas in the resulting odd-shaped grid cells at the boundary.

More recent work on this method is reported in [8]. The overlapping grid approach introduced in the present paper uses a more accurate representation of boundary surfaces than the Cartesian

*This work performed under the auspices of the U.S. Department of Energy by Los Alamos National Laboratory under Contract W-7405-ENG-36.

grid method, although with correspondingly more work by the user required in order to construct the initial grid. A set of curvilinear component grids is used, the union of which completely covers the computational region. There are small regions of overlap between the individual component grids. Each component grid is logically rectangular with some of the cells possibly "blanked" out and unused. With this approach, a complex structure can be represented by combining many separate pieces, each represented by its own curvilinear grid. The potential of the overlapping grid method was first demonstrated by Starius [9, 10], Kreiss [11] and Steger et. al. [12]. Successful three-dimensional aerodynamic simulations involving configurations as complex as the space shuttle [13, 14], and with moving components [15], validated the usefulness of this technique as a practical engineering tool. A fully automatic grid overlapping procedure for two- and three-dimensional grids grids (CMPGRD) was developed by Chesshire and Henshaw which forms the basis for the current work [16, 17, 18]. The use of overlapping grids to represent a complex geometry was dubbed the "Chimera" method by the late Joe Steger.

The overlapping grid approach allows a great deal of flexibility in the placement of the component grids. Since the component grids may overlap, rather than being required to match exactly along an interface as with the block-structured grid method [19], they are relatively unconstrained. This additional freedom allows generation of smoother component grids. This is ideal for applying higher-order upwind finite-difference methods, since they perform best on grids whose transformation to the unit square or cube are smooth. The finite-difference method used in this paper is introduced in [20] and is an unsplit Godunov method based on the methods introduced by Colella [21].

Since CMPGRD produces sets of logically rectangular grids, the extension of the AMR method to this framework is natural. The AMR method developed in this paper follows closely the technique discussed by Berger and Colella in [3]. Each component grid of the overlapping grid structure is refined separately by the AMR algorithm. As in [3], the nested refinement grids are constrained to have boundaries coinciding with the underlying "parent" component grid, i.e. none of the refinements are allowed to be rotated with respect to that parent grid. The differences are in the treatment of the cutout and overlap regions of the underlying overlapping-grid.

THE OVERLAPPING GRID AMR ALGORITHM

The adaptive grid construction and solution procedures on an overlapping grid are straightforward extensions of the AMR procedures on a single grid. Modifications are made, as necessary, to accomodate the special requirements of the overlapping grid structure. In order to describe the grid construction and problem solution methods for overlapping grid AMR, we first briefly describe the relevant parts of the procedures for the non-overlapping AMR and the non-adaptive overlapping cases. For simplicity, in both cases we assume that the grid covers a two-dimensional region.

Solution Procedure Using AMR on a Single Grid

The basic AMR mesh construction and solution procedures are a recursive generalization of the basic two-level procedure that we will describe here. The reader is referred to [3] for a more detailed description. At each timestep in the two-level AMR procedure for a single underlying "base" grid, the grid hierarchy consists of the base grid and patches of *properly aligned* refinement grids that have been automatically placed by the AMR algorithm in regions where additional solution accuracy is required. Properly aligned grids have the property that a grid at some level n always has its boundary cells aligned with cell edges of the level $n-1$ grids. In the general n-level algorithm, the grids must have the additional property that they are *properly nested*, which means that a grid at some level n is always found embedded in some subset of the level $n-1$ grids. A level n grid is not allowed to be all or partially embedded in parts of the grid structure that contain only grids at lower levels $(1, 2, \ldots, n-2)$.

Returning to the two-level case, all refinement grids at the fine level are automatically replaced with new refinement grids after every m timesteps, where m is a user–specified interval. The solution data is interpolated onto the new grid hierarchy before the calculation continues. For the purposes of this discussion, assume that solution values are available in all cells of all grids in the current grid hierarchy. The grid refinement regeneration is done automatically by estimating the error in the calculation at the current timestep on all grids, and then defining new refinement grids in regions where the error is estimated to be higher than some user-specified tolerance. In practice, the error estimation is done either by a Richardson-extrapolation procedure that compares solutions on grids of different overall resolution [3, 4], or by measuring the size of local solution gradients and refining in regions where the gradients have become unacceptably large relative to the grid [6]. The latter approach was employed for the computations presented in the present paper. Using one of these procedures, the error is estimated in each cell on the grid, and cells with unacceptably high estimated error are "flagged." Since the grid will not be refined again for m timesteps on the base grid level, it is necessary to expand the refinement region somewhat before a refinement grid is constructed. A simple domain-of-dependence argument requires that an additional row of cells be added around each group of flagged cells for each timestep that the computation will proceed without re-refinement of the grid. This is referred to as the "cell-diffusion" step of the AMR grid construction procedure. Once this is done, the flagged cells are grouped into "boxes", or rectangular regions using a procedure described in [3]. Refined grids with a user-specified refinement factor n_{ref} relative to the base grid are then constructed in each of the boxes. The solution on the previous adaptive grid hierarchy is then interpolated onto the new grid hierarchy. In regions where the solution is defined on more than one grid refinement level, the solution values on the finest grid available are used.

Once the data is available on all of the grids in the new grid hierarchy, the solution procedure can continue. First the coarse grid solution is advanced by one timestep in all interior cells of the coarse grid. This includes coarse grid cells that are covered by a refined grid. Boundary conditions for this step are assumed to be provided as part of the original problem specification. Once the coarse grid solution has been advanced, the solution on the refinement grid patches can be advanced. Since the solution method is explicit, the refinement grid solutions are advanced using the same CFL timestep restriction as on the coarse grid. This means that n_{nref} timesteps must be taken on the fine grids for each single timestep on the coarse grid. Boundary conditions for each of the refinement patches are obtained again following the principle that the "best" values available should be taken. At fine grid boundaries where there is a refinement grid at the same refinement

level n immediately adjacent, values from the adjacent grid are used to provide the boundary conditions. At fine grid boundaries where the adjacent grid is at a coarser level, boundary values are obtained by interpolating the coarse-grid solution in space and time. When advancing the solution of a system of hyperbolic conservation laws with a conservative method, an additional "conservative update" step is taken in which values in coarse grid cells at the coarse-fine grid interface are recomputed using fluxes available from the fine grid calculations. Details of this procedure are found, for example, in [3]. Finally, once fine-level values are available at the new timestep of the coarse grid, the coarse grid solution values in cells covered by refinement patches are replaced by transferring or interpolating fine grid values to the coarse grid cells. This assures that the best possible values are always used in the solution procedure for the succeeding timestep.

Solution Procedure Using Overlapping Grids Without AMR

An overlapping grid in two space dimensions consists of a set of logically rectangular curvilinear component grids that overlap where they meet and whose union completely covers the computational domain for a system of partial differential equations. The cells on each component grid are classified according to their function during the PDE solution procedure. "Interior" or "discretization" cells are cells on a component grid that can be updated using an interior discretization formula for the PDE. This means that each discretization cell has a buffer zone of cells around it of sufficient width that the interior discretization formula can be applied. Near physical boundaries of the domain, "fictitious" or "ghost" cells are added to the component grid outside the physical boundary. Boundary conditions derived from the physical boundary conditions for the problem, or using considerations based on numerical analysis, are used to update the solution values in these cells. In regions of overlap between the component grids, "interpolation" cells are included in the grid. Solution values in these cells are updated using an interpolation formula applied to a stencil of cells on an adjacent component grid. As with the ghost cells, interpolation cells are included in the grid to provide the necessary buffer zone around every discretization cell so that the interior discretization formula may be applied.

The overlapping grids used in this paper are constructed using the CMPGRD overlapping grid software developed by Chesshire and Henshaw [17], [22]. Overset grids constructed using this software have the property that they overlap the minimum amount necessary in order that essentially centered interpolation formulas can be used to transfer values between adjacent grids during a PDE solution procedure. CMPGRD automatically generates an overlapping grid along with all the data necessary for communicating data between the component grids given a set of user-specified "component" grids, each of which is a logically-rectangular grid in general curvilinear coordinates. If the original user-specified component grids overlap more than this minimum required amount, the un-needed cells are marked as "inactive" and are not used in the PDE solution procedure. In the computations presented in this paper, CMPGRD was used both as an interactive package for the construction of the initial underlying overlapping grid, and also as part of the AMR PDE solver, where it is called as a subroutine and used for embedding AMR refinement grids within the underlying overlapping grid. For this project, we modified CMPGRD to be able to insert refinement grids adaptively during a PDE solution process. The basic principles and algorithm details for overlapping grid construction are described in more detail in [17] and [22].

Because of space considerations, the detailed modifications needed for the overlap algorithm for adaptive grids will be discussed in an archival paper.

Adaptively-refined Overlapping Grids

We discuss here the procedure for advancing a PDE solution by one coarse-level timestep on an adaptively-refined overlapping grid. An adaptively-refined overlapping grid structure consists of a set of underlying curvilinear component grids $\mathcal{A}, \mathcal{B}, \mathcal{C}, \ldots$ that make up the "base grid" for the problem, each of which may contain embedded refinement grids.

The embedded refinement grids have the property that they are both properly nested and properly aligned with their parent component grid just as in the single grid case discussed above in the section entitled "Solution procedure using AMR on a single grid". If a region of refinement is needed that extends beyond the boundary of active cells of one of the parent component grids and into a region covered by the active cells of another parent component grid, each of the parent component grids is refined separately rather than attempting to construct a single refinement patch that covers the entire refinement region. This is an important point in our adaptive mesh procedure, since it greatly simplifies the grid construction algorithm compared to what would be required if general adaptive refinement grids were allowed. The interior cells of a refinement grid patch must lie completely within the interior cell region of its parent grid(s) at the next coarser level. If a refinement patch is created adjacent to an overlap boundary of the parent grid, its overlap interpolation cells will lie completely within the set of overlap interpolation cells for the parent grid(s) as well.

The overlap interpolation rules for overlap regions on an adaptively-refined overlapping grid specify that values are interpolated from adjacent component grids preferentially from grids at the same level, followed in preference by grids at the next coarsest level. The implicit assumption here is that cell size and aspect ratio of grids at the same refinement level on adjacent component grids is roughly the same. Thus it is most appropriate to interpolate values from adjacent grids at the same refinement level unless such a grid is not available, in which case the best possible values should be used. The proper-nesting assumption implies that in the latter case, the values will come from coarser refinement levels on the adjacent grid. While it could occur that the adjacent component grid would have values available at *finer* levels, interpolation from the finer level adjacent grids is not necessary since the values would be approximately equally degraded by interpolating directly from the fine grids as they would be by first transferring values to the adjacent coarse grid and then interpolating.

We now discuss the procedure for constructing the new refinement grid patches in the two-level refinement case where the adaptive grid hierarchy consists of the base overlapping grid together with refinement grid patches one level finer than the base grid. The solution data is assumed to be available in all cells on all grids in the adaptive overlapping grid hierarchy at the beginning of the coarse-grid timestep. As in the single-grid case, all the refinement grids at the fine level are replaced at user-specified timestep intervals using the adaptive procedure. The procedure for constructing the new refinement grid patches is essentially the same as for the one-grid case. An

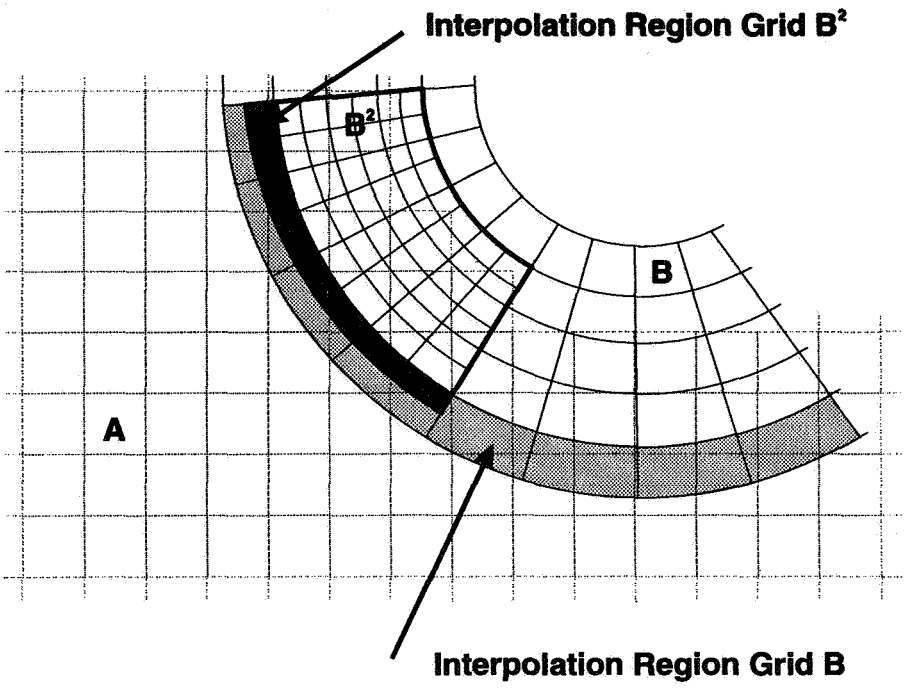

Figure 1: The interpolation cells for refinement grid B^2 lie within the interpolation region for grid B. Interior cells for grid B^2 may not lie inside the interpolation region for the coarser grid.

error estimation procedure is used on each parent grid in all interior cells, and cells of high estimated error are flagged. The set of flagged cells is then "diffused" as in the single-grid case. A difference in the overlapping grid case, however, is that we do not permit interpolation cells on the coarse grid to be flagged by either the error estimation or diffusion procedure. This is disallowed to simplify the AMR solution procedure on an overlapping grid. If the diffusion procedure indicates that an interpolation cell should be flagged, the "interpolee" cells [23] on the adjacent grid are flagged instead (If a cell on grid $\mathcal{A}$ interpolates from cells on grid $\mathcal{B}$, the interpolee cells for an interpolation cell on grid $\mathcal{A}$ are defined to be those cells on grid $\mathcal{B}$ that are included in the interpolation stencil used for determining the solution value in the interpolation cell on grid $\mathcal{A}$). This procedure of flagging interpolee cells is a logical extension to the overlapping grid case of the basic domain-of-dependence argument for the cell-diffusion process. It also provides for the expansion of a refinement region across overlap boundaries during the course of a time-dependent PDE solution, which otherwise would not take place. Figures 2–4 illustrate the procedure with the curved grid representing grid $\mathcal{A}$ and the rectilinear grid represents grid $\mathcal{B}$.

The solution on an adaptively-refined overlapping grid is updated as follows. Values are first transferred from the old adaptively-refined grid hierarchy. For interior discretization cells, this involves either copying values from grids at the same level with the same parent grid, or transferring values from grids at the next coarser level on the same parent grid. Values in interpolation cells are transferred in a different way. These values must either be copied from an old grid at the same level with the same parent component grid *or* they must be interpolated from data on an adjacent component grid according to the interpolation rules given above. It is important never to transfer coarse grid interpolation values to fine interpolation cells since a degradation of

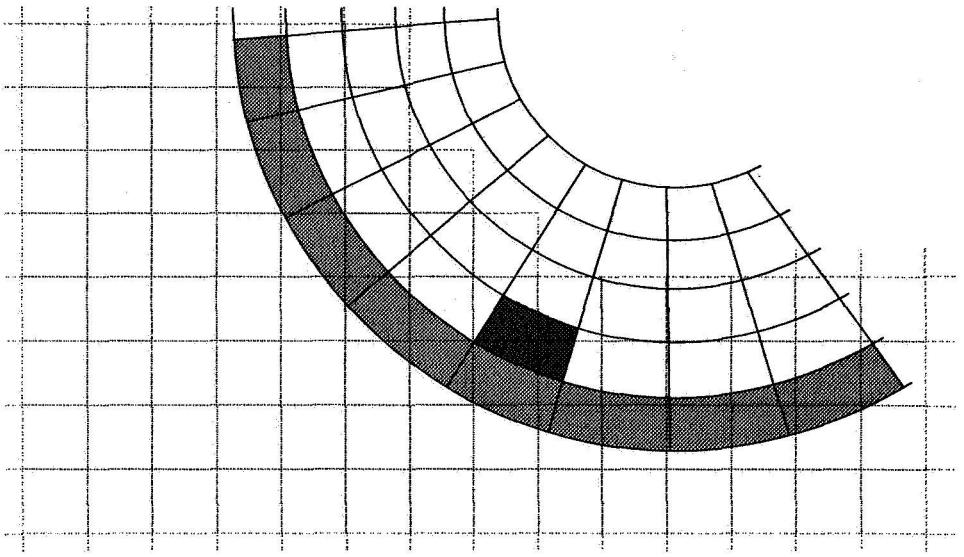

Figure 2: The cell on the curved grid shaded dark dark gray is flagged as a high error cell and will be refined.

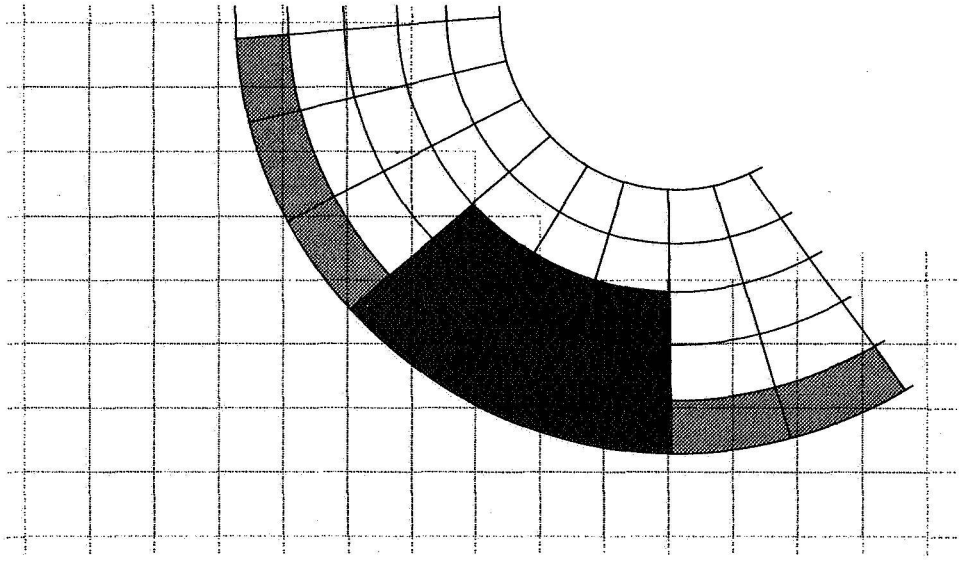

Figure 3: A single "diffusion" step of flagging surrounding cells is done. Note that interpolation cells (lighter gray) are touched by the diffusion. These cells are only tagged for the purpose of flagging the underlying "interpolee" cells and will not be refined.

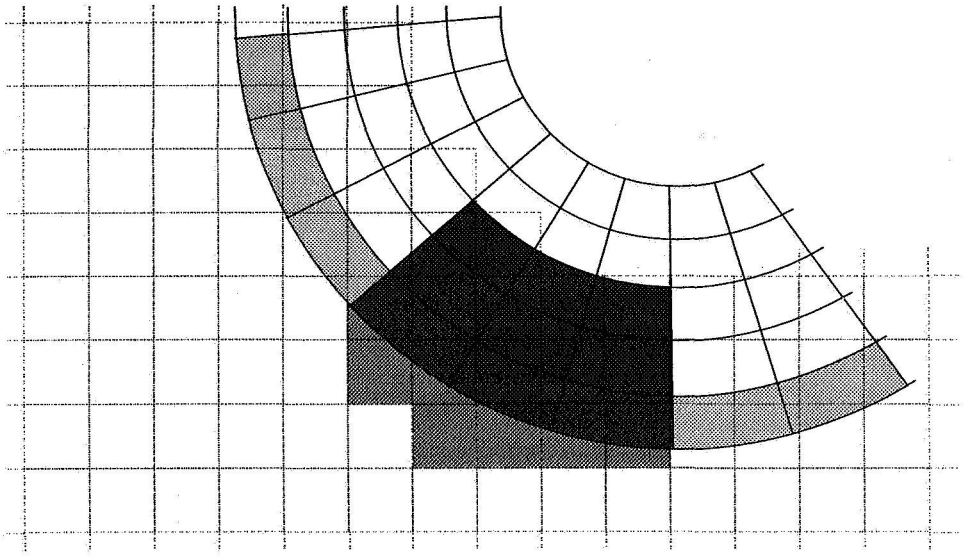

Figure 4: The cells are on the rectilinear grid are flagged (shaded an intermediate gray) because they are used for interpolation data by the flagged interpolation points on the curvilinear grid.

the computed solution can result.

Away from interpolation boundaries, the solution advance procedure is identical to the single–grid AMR procedure. The only difference in the solution advance procedure for interpolation cells is that no conservative update procedure is used. This is largely due to the fact that we feel that satisfactorily efficient methods for conservative update of overlap boundary values on general overlapping grids have not been developed. Some research has been done in this area, however, cf. [23, 24].

SOME IMPLEMENTATION DETAILS

AMR codes have greater code complexity than single or even block structured logically rectangular grid methods. The primary reason for the programming complexity is the demands made on the programming environment for dynamic allocation and deallocation of data structures. Even communication between grids at the same or different levels is nontrivial. To handle the programming complexity, we have begun to move to a programming language more capable in the manipulation of complex data structures. We use C++ [25] to handle the dynamic memory management of the data structures and FORTRAN for the numerical parts of the algorithm such as the integration. This is the first step in moving towards a C++ based programming environment that not only abstracts out the data structures but hides details of a parallel implementation as well.

C++

C++ is a superset (with some very minor exceptions) of the programming language C. For those who appreciate C, the postfix operator "++" is used to increment by one the variable it follows. Therefore C++ is literally an addon to C. C++ adds new capabilities that bring it into the realm of what is called Object Oriented Programming (OOP). OOP is a technique, discipline or style of writing programs. Here algorithms are organized around data structures called objects that both hold data and supply the functions needed to manipulate the data in safe ways. Encapsulation is often used to describe this process. The goal of OOP is to generate reusable program modules with few side effects. The idea seems a good and simple concept from a common sense viewpoint but the implementation of flexible and reusable object libraries requires a great deal of forethought and design. Practice shows us that several design iterations are often required to "get it right".

The primary way C is augmented to become a language that supports OOP is through the introduction of the data structure called a *class*. A *class*, in its most simple form, is a structure which in turn is much like a common block in FORTRAN. However, classes are used to instantiate objects by associating member functions with the class. Member functions are simply functions that operate on the data within the *class/structure*. Once classes are introduced C++ can handle two basic OOP programming paradigms. The first is called *inheritance* and the second *polymorphism*.

Inheritance is a mechanism of reuse of objects. New objects can be created from currently available objects by inheritance. The inheriting objects will have all the properties of the inherited objects plus whatever is added (data or more member functions). Inheritance facilitates a rich structure of objects through multiple inheritance (inheriting an inherited class which may in turn inherit other classes) yet allows the developer to encapsulate data at all levels. However, no programming paradigm will prevent people from writing sloppy code.

Inheritance is further enhanced by Polymorphism. Polymorphism literally means many shapes. In C++ the same function name or even operators such as "+", "*", ... can be used for many purposes. A simple example is to consider the type double, a double precision number. Doubles can be added, subtracted, multiplied, along with a host of arithmetic operations. A simple example of polymorphism is to define a new class that would represent complex numbers as a pair of real numbers. Many of the same arithmetic operators that are used to manipulate doubles can be "overloaded" to manipulate complex numbers as well. The array class library we will describe below is another example of polymorphism.

Data Structures

The implementation of the adaptive overlapping code heavily uses classes to manipulate and manage user data. In addition, inheritance is also incorporated to make the code much more ammenable to modification and/or reuse. Polymorphism is used very little at this point. However, our current direction is to use a C++ array class called A++/P++ [26] with syntax similar to F90 to develop newer versions of this and other overlapping grid codes. Here polymorphism will come into play as we are overloading the arithmetic operations found in C or C++ to manipulate multidimensional arrays.

The design of the adaptive overlapping code splits the overall algorithm into two sets of pieces. The first set is a collection of objects and functions that implement an "abstract" adaptive overlapping code. The second piece is supplied by a code developer who wants to make his or her own adaptive overlapping mesh code. Within the first collection are objects that describe the logical layout of the overlapping grids. These objects don't perform any approximation of the solution of PDEs but can be modified so that they do. Here inheritance is used. The abstract objects are inherited by developer defined objects that contain the necessary data to perform useful computations. Other objects within the abstract set develop a skeleton set of functions that give a roadmap that can be used by code developers to modify or rebuild a new algorithm. These functions are called *virtual*. When objects are inherited, the inherited object functions that are labeled virtual can be replaced by the inheriting object. This defines a clean interface between the developer and the abstract code. The developer has the flexibility to design the right functions for his or her needs and the abstract interface does not have to be changed.

The primary data structure within the abstract code is a list of objects that represent individual grids. All logical information about a component grid or any refinement is stored in what is called a *Patch* class. Logical information includes logical coordinates of a grid, interpolation information and boundary condition information. A *patchNode* is derived from a Patch so that it can be put in a linked list class called a *patchList*. This linked list contains only patches at the same refinement level on a single overlapping grid component. A *levNode* is derived from patchList to be added to a list of refinement levels on a single grid. This list class is called a *levList*. In addition, there are pointers within a levNode to point to list of patches on other components at the same level. Finally a class called *compNode* is derived from levList to be contained in a list of refinements at all levels on all components called a *compList*.

Although complex, implementation of this list structure is greatly simplified by using C++ inheritance. This allows the developer to concentrate on numerical algorithms rather than the data management aspects. Figure 5 illustrates the complete hierarchy.

The actual grid generation is performed by calling as a subroutine the CMPGRD mesh generation code [17] modified to generate mesh refinements as discussed in the previous section. Finally, many smaller objects and functions are used to help in regridding and managing refinements within components. These objects came from a very useful and reusable C++ library from the Center for Computational Sciences at Lawrence Livermore National Laboratory called BoxLib [27].

NUMERICAL RESULTS

The computational examples presented in this paper are two dimensional simulations of compressible fluid flow as described by a numerical approximation to the compressible Euler equations. The code runs both two- and three dimensional problems but three-dimensional results will be described in a separate journal article because of space limitations. In general curvlinear coordinates, these are given by

$$\partial_t \mathbf{u} + J_\xi \sum_{\ell=1}^{n_d} \partial_{\xi^\ell} \mathbf{F}^\ell(\mathbf{u}) = 0. \tag{1}$$

Grid Data Structure Hierarchy

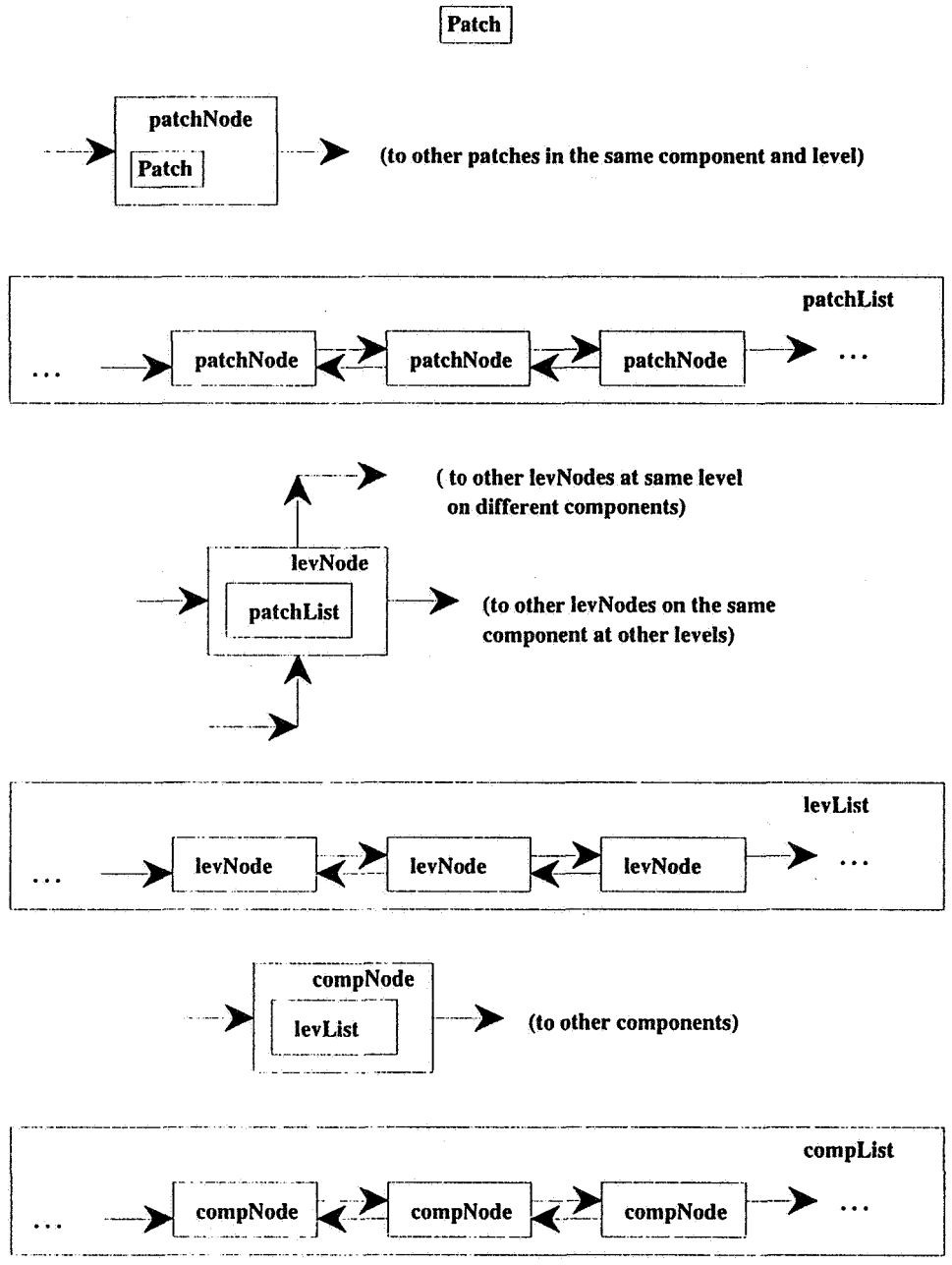

Figure 5: The list elements from the top are incorporated into more complex data structures further down.

Here

$$\mathbf{F}^{\ell}(\mathbf{u}) =: \begin{pmatrix} \rho U^{\ell} \\ u\rho U^{\ell} + \tilde{\xi}^{\ell}_{x} p \\ v\rho U^{\ell} + \tilde{\xi}^{\ell}_{y} p \\ w\rho U^{\ell} + \tilde{\xi}^{\ell}_{z} p \\ U^{\ell}(\rho E + p) \end{pmatrix}, \ell = 1,2,3, \tag{2}$$

$$U^{\ell} := \sum_{i=1}^{n_d} \tilde{\xi}^{\ell}_{x^i} u^i$$

are the contravariant velocities, and $J_{\xi} := det|\frac{\partial \xi}{\partial x}|$ is the determinant of the coordinate transformation from Cartesian coordinates $\mathbf{x} := (x^1, x^2, x^3)$ to curvilinear coordinates $\xi := (\xi^1, \xi^2, \xi^3)$. The dependent varibles are the density ρ, the three components of velocity $u^i, i = 1,2,3$, and the energy E. The pressure, p is related to the other variables through the equation of state for an ideal gas: $p = (\gamma - 1)(\rho E - \frac{1}{2}\rho \sum_i (u^i)^2)$. The finite difference method used for the computations is based on the conservative cell-centered upwind-centered Godunov method in [20]. It has been modified to use a linearized approximate Riemann solver described in unpublished work by Colella, Glaz and Ferguson. (The original method in[20] used a computationally more expensive approximate flux function based on [28].)

Several test problems were outlined, in advance, by the conference organizers to stimulate discussion at the workshop. We chose to compute the double wedge geometry where an oncoming Mach 2.16 shock hits a wedge at an angle of 20 degrees followed by a wedge at an angle of 50 degrees three horizontal units later. Quiescent preshock values are unity in the pressure and density. The ratio of the specific heats (γ) is 1.4. Because the geometry was simple enough, two computations were preformed. The first computation used a single grid that was deformed to fit the double wedge geometry. The second computation uses two component grids — the first grid being a cartesian grid and the second grid conforms to the wedge boundary cutting away the cartesian grid.

Figures 6 and 7 show the grids and density for times near 2.5 time units. The solutions are very nearly the same in structure. However, single grid computation ran in twice the number of cycles that the two grid computation did. This is primarily caused by the timestep in the single grid case being unnecessarily CFL limited in the upper right hand corner of the computation. The two grid case has uniform cell sizes throughout the computational region. Another issue brought out by these computations is conservation. The two grid computation is not conservative yet still matches the single conservative grid case. This reflects our experience that if care is taken in making sure that cells sizes don't vary greatly from component to component then nonconservative interpolation is sufficient for computational purposes.

The last subject to discuss is overall performance of the adaptive algorithm. At this point the time spent in the integrator is less than 50% of the entire run time. This is primarily due to the current implementation of the composite grid generation package. This package handles each point on each grid separately instead of processing a list of points in a vectorized manner. We are

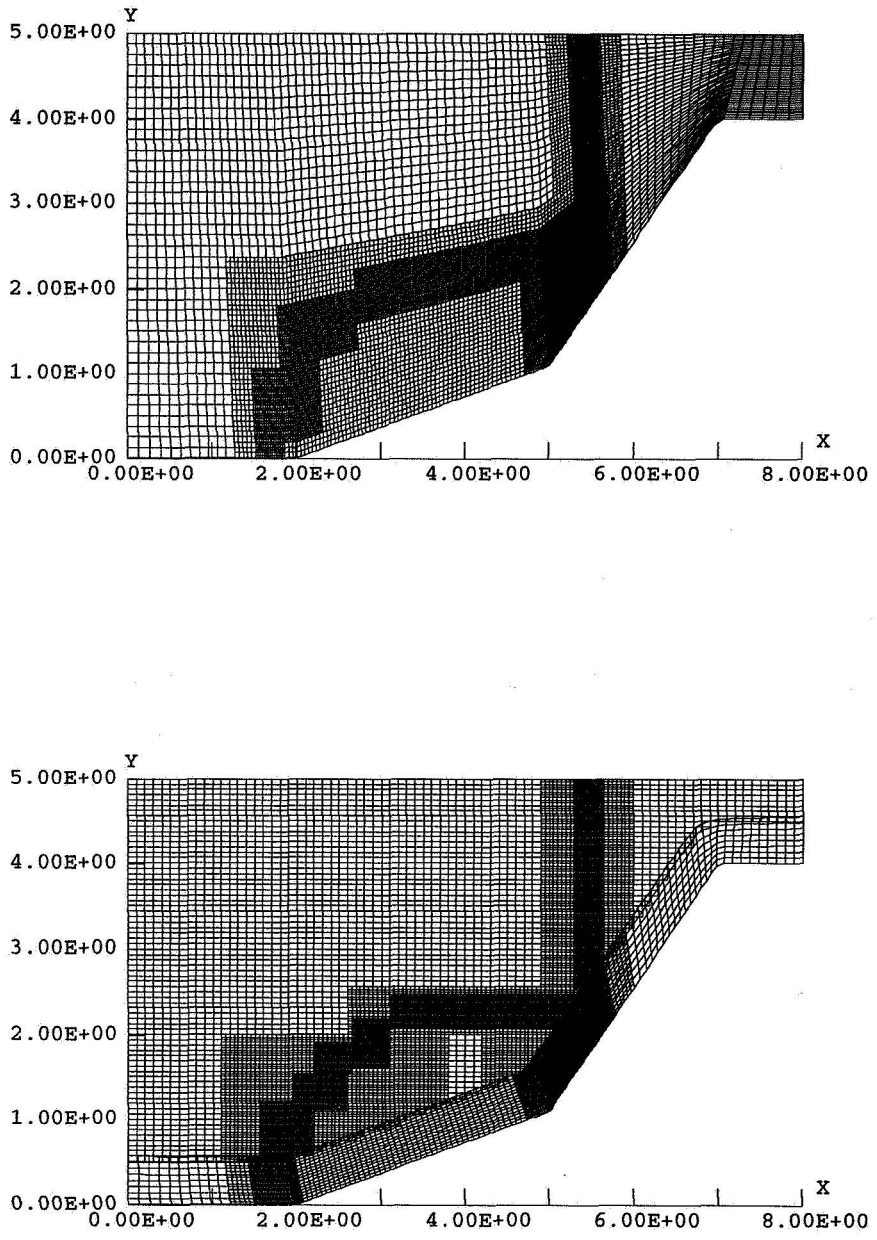

Figure 6: The adaptive grids for a single base grid (top) and two base grid (bottom) computation near time 2.5.

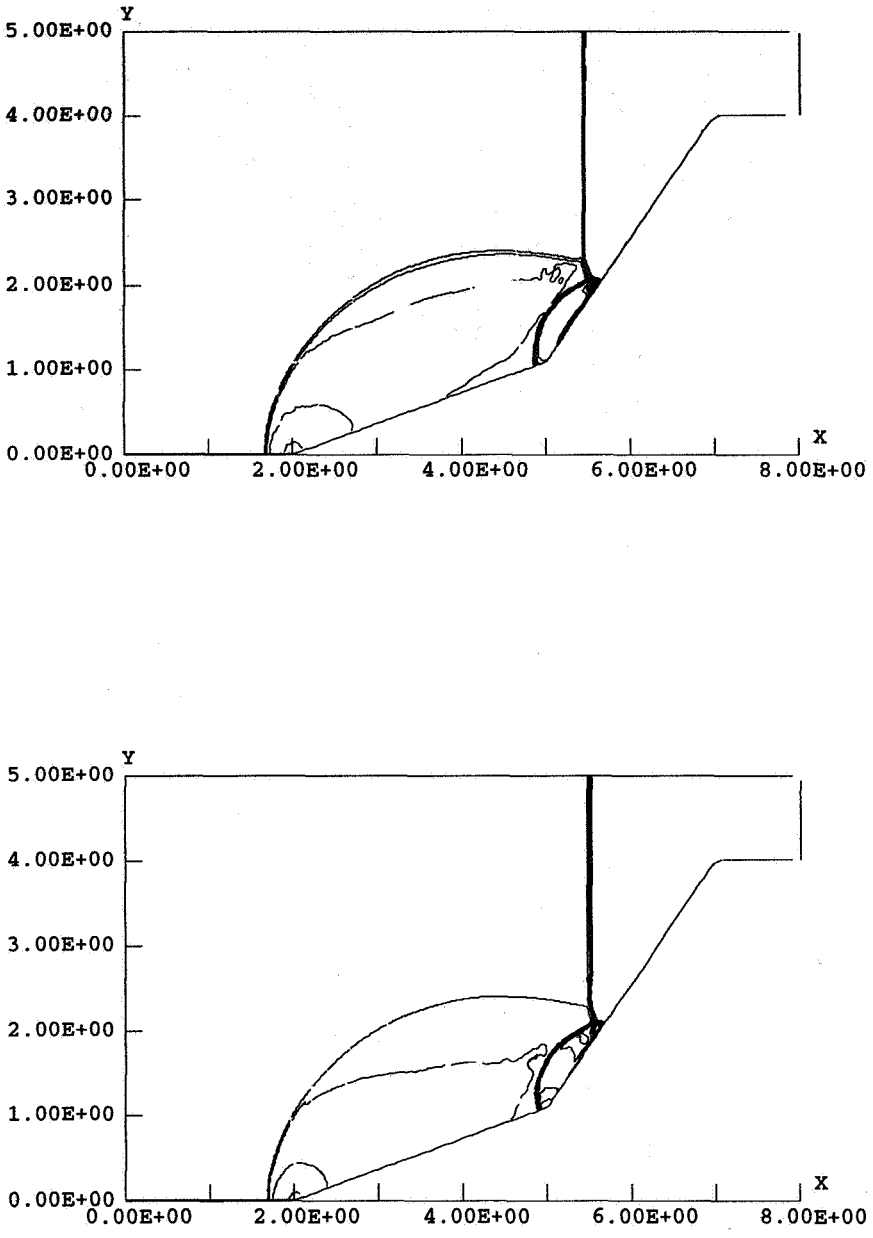

Figure 7: The density contours for a single base grid (top) and two base grid (bottom) computation near time 2.5.

currently rewriting the mesh generation package so that it can achieve vector performance.

*

References

[1] M. J. Berger and J. Oliger. Adaptive mesh refinement for hypberbolic partial differential equations. *J. Comp. Phys.*, 53:561-568, March 1984.

[2] M. J. Berger and A. Jameson. Automatic adaptive grid refinement for the Euler equations. *AIAA Journal*, 23:4:561-568, 1985.

[3] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comp. Phys.*, 82:64-84, 1989.

[4] J. Bell, M. Berger, J. Saltzman, and M. Welcome. Three dimensional adaptive mesh refinement for hyperbolic conservation laws. *SIAM J. Sci. Comput.*, 15:127-138, 1994.

[5] E. G. Puckett and J. S. Saltzman. A 3D adaptive mesh refinement algorithm for multimaterial gas dynamics. *Physica D*, 60:84-93, 1992.

[6] M. J. Berger and J. Saltzman. AMR on the CM-2. *Applied Numerical Mathematics*, 14:239-253, 1994.

[7] M. J. Berger and R. Leveque. Cartesian meshes and adaptive mesh refinement for hyperbolic partial differential equations. In B. Engquist and B. Gustafsson, editors, *Third International Conference on Hyperbolic Problems*, pages 67-73. Chartwell-Bratt, 1991.

[8] R. B. Pember, J. B. Bell, P. Colella, W. Y. Crutchfield, and M. L. Welcome. Adaptive Cartesian grid methods for representing geometry in inviscid compressible flow. In *Proceedings of the Eleventh AIAA Computational Fluid Dynamics Conference*, pages 530-539. AIAA, June 1993.

[9] G. Starius. Composite mesh difference methods for elliptic boundary value problems. *Num. Math.*, 28, 1977.

[10] G. Starius. On composite mesh difference methods for hyperbolic differential equations. *Numer. Math.*, 35:241-255, 1980.

[11] B. Kreiss. Construction of a curvilinear grid. *SIAM J. Sci. Stat. Comput.*, 4:270-279, 1983.

[12] J. A. Benek, J. L. Steger, and F. C. Dougherty. A flexible grid embedding technique with application to the Euler equations. *AIAA paper 831944*, 1983.

[13] J. A. Benek, P. G. Buning, and J. L. Steger. A 3-D Chimera grid embedding technique. In *Proceedings of the 7th AIAA Computational Fluid Dynamics Conference, Cincinnati*, pages 322-331. AIAA, 1985.

[14] P. G. Buning, I. T. Chiu, S. Obayashi, Y. M. Rizk, and J. L. Steger. Numerical simulation of the integrated space shuttle vehicle in ascent. AIAA paper 88-4359-CP, AIAA, 1988.

[15] F. C. Dougherty and J-H Kuan. Transonic store separation using a three-dimensional Chimera grid scheme. AIAA paper 89-0637, AIAA, 1989.

[16] G. S. Chesshire. *Composite Grid Construction and Applications.* PhD thesis, California Institute of Technology, 1986.

[17] G. Chesshire and W. D. Henshaw. Composite overlapping meshes for the solution of partial differential equations. *J. Comp. Phys.*, 90(1):1–64, 1990.

[18] D.L. Brown, G. Chesshire, W.D. Henshaw, and H.O. Kreiss. On composite overlapping grids. In *Proceedings of the Seventh International Conference on Finite Element Methods in Flow Problems*, 1989.

[19] J. F. Thompson. A composite grid generation code for general 3D regions - the Eagle code. *AIAA J.*, 26:271, 1988.

[20] D. L. Brown. An unsplit Godunov method for systems of conservation laws on curvilinear overlapping grids. *Mathl. Comput. Modelling*, 20(10):29–48, 1994.

[21] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comp. Phys.*, 87:171–200, 1990.

[22] D. L. Brown, G. Chesshire, and W. D. Henshaw. Getting started with CMPGRD, introductory user's guide and reference manual. LANL unclassified report LA-UR-90-3729, Los Alamos National Laboratory, 1990.

[23] G. Chesshire and W. D. Henshaw. A scheme for conservative interpolation on overlapping grids. *SIAM J. Sci. Comput.*, 15(4):819–845, July 1994.

[24] M. J. Berger. On conservation at grid interfaces. *SIAM J. of Numer. Anal.*, 24:967–984, 1987.

[25] Bjarne Stroustrup. *The C++ Programming Language.* Addison Wesley, second edition, 1992.

[26] R. Parsons and D. Quinlan. Run-time recognition of task parallelism within the p++ parallel array class library. Technical Report LA-UR-93-3856, Los Alamos National Laboratory, October 1993. proceedings of the IEEE Conference on Scalable Parallel Libraries at Mississippi State University, Oct. 6-8, 1993.

[27] M. Welcome, W. Crutchfield, C. Rendleman, J. Bell, L. Howell, V. Beckner, and D. Simkins. *BoxLib User's Guide and Manual.* Lawrence Livermore National Laboratory, Lawrence Livermore National Laboratory, Livermore, CA 94550, version 0.02 beta edition, September 1994. A Library for Managing Rectangular Domains.

[28] J. Bell, P. Colella, and J. Trangenstein. Higher order Godunov methods for general systems of hyperbolic conservations laws. *J. Comp. Phys.*, 82:362–397, 1989.