LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JRNL-236681

Parallel Computation of Three-Dimensional Flows using Overlapping Grids with Adaptive Mesh Refinement

W.D. Henshaw, D.W. Schwendeman

November 20, 2007

Journal of Computational Physics

Disclaimer

Parallel Computation of Three-Dimensional Flows using Overlapping Grids with Adaptive Mesh Refinement

William D. Henshaw [1]

Centre for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94551, henshaw1@llnl.gov

Donald W. Schwendeman [2]

Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180, schwed@rpi.edu

Abstract

This paper describes an approach for the numerical solution of time-dependent partial differential equations in complex three-dimensional domains. The domains are represented by overlapping structured grids, and block-structured adaptive mesh refinement (AMR) is employed to locally increase the grid resolution. In addition, the numerical method is implemented on parallel distributed-memory computers using a domain-decomposition approach. The implementation is flexible so that each base grid within the overlapping grid structure and its associated refinement grids can be independently partitioned over a chosen set of processors. A modified bin-packing algorithm is used to specify the partition for each grid so that the computational work is evenly distributed amongst the processors. All components of the AMR algorithm such as error estimation, regridding, and interpolation are performed in parallel.

The parallel time-stepping algorithm is illustrated for initial-boundary-value problems involving a linear advection-diffusion equation and the (nonlinear) reactive Euler equations. Numerical results are presented for both equations to demonstrate the accuracy and correctness of the parallel approach. Exact solutions of the advection-diffusion equation are constructed, and these are used to check the corresponding numerical solutions for a variety of tests involving different overlapping grids, different numbers of refinement levels and refinement ratios, and different numbers of processors. The problem of planar shock diffraction by a sphere is considered as an illustration of the numerical approach for the Euler equations, and a problem involving the initiation of a detonation from a hot spot in a T-shaped pipe is considered to demonstrate the numerical approach for the reactive case. For both problems, the solutions are shown to be well resolved on the finest grid. The parallel performance of the approach is examined in detail for the shock diffraction problem.

1. Introduction

We describe an approach for the numerical solution of time-dependent partial differential equations (PDEs) defined on complex three-dimensional domains. The approach is based on the use of overlapping structured

[1] This work was performed under the auspices of the U.S. Department of Energy (DOE) by Lawrence Livermore National Laboratory in part under Contract W-7405-Eng-48 and in part under Contract DE-AC52-07NA27344 and by DOE contracts from the ASCR Applied Math Program and the ITAPS SciDAC Center.

[2] This research was supported by Lawrence Livermore National Laboratory under subcontract B548468, and by the National Science Foundation under grants DMS-0532160 and DMS-0609874.

grids to represent the three-dimensional domains [1]. In a typical grid construction, boundary-fitted curvilinear grids are overlapped with background Cartesian grids. The use of structured grids and Cartesian grids results in computationally efficient discretizations of the PDEs. The use of smooth boundary-fitted grids also leads to accurate approximations of the equations and boundary conditions. Block-structured adaptive mesh refinement (AMR) is incorporated into the overlapping grid framework, and used to locally increase the grid resolution of the simulation. Refinement grids are added in a hierarchical fashion to each base grid according to an estimate of the numerical error, and the positions of the refinement grids are recomputed every few time steps as the solution evolves. In this way, the numerical approach is especially amenable to PDEs whose solutions exhibit localized fine-scale features such as contact discontinuities, shocks, and detonations.

The numerical approach for solving PDEs on overlapping grids with AMR is implemented on parallel distributed-memory computers using a domain-decomposition approach. The implementation is flexible so that each base grid within the overlapping grid structure and its associated refinement grids can be independently partitioned over a chosen set of processors. The partitioning is specified to balance the computational work across the processors, and this is done using a modified bin-packing algorithm. All elements of the AMR algorithm, such as error estimation, regridding, and interpolation, are performed in parallel. The approach we describe may be used to compute the numerical solution of a wide range of initial-boundary-value problems (IBVPs) for PDEs. For the purposes of this paper, we consider IBVPs for two specific equations. The first is a linear advection-diffusion equation, and the second is the (nonlinear) reactive Euler equations of gas dynamics. The first equation provides a useful test case to study the behavior of the numerical approach and to verify its accuracy quantitatively, while the second builds on our earlier work in [2,3] and illustrates the numerical approach for a more difficult set of equations. While a brief description of the discretization of these two PDEs is given, the emphasis of the discussion is on the extension of the numerical approach for parallel computations in three-dimensional geometries. In addition, significant attention is given to establish the accuracy of the numerical calculations for both equations, and to assess the performance of the parallel implementation.

It is now well established that overlapping grids can be used to solve a wide class of problems efficiently and accurately. The technique is especially attractive for handling problems with complex geometry, and problems with moving or deforming boundaries. The first use of overlapping grids (called *composite grids* at the time) appeared in papers by Volkov [4,5], who considered approximations to Poisson's equation in regions with corners. Other pioneering work includes that of Starius [6–8], Kreiss [9], and Steger et al. [10] who referred to the approach as *Chimera* grids. Since this early work, the overlapping grid technique has been used successfully to solve a wide variety of problems in high-speed reactive flow [2,3,11], reactive and non-reactive multi-material flow [12,13], combustion [14], aerodynamics [15–21], blood flow [22], flows around ships [23], visco-elastic flows [24], and flows with deforming boundaries [25–27], among others. The use of AMR in combination with overlapping grids was considered by Brislawn et al. [28], Boden and Toro [29], Meakin [21], Henshaw and Schwendeman [2,3], and Banks et al. [12,13]. The approach described in this paper is related to the work of Meakin [17] who considered overlapping grids and AMR in parallel, although his refinement grids were restricted to the Cartesian background grids and thus his approach is not as general as the one presented here.

Implementing a parallel-AMR approach in software represents a considerable undertaking. To overcome this hurdle, there have been a number of AMR software infrastructures developed that support the parallel solution of PDEs. These include AmrLib/BoxLib [30], Chombo [31], DAGH [32], GrACE [33], PARAMESH [34], and SAMRAI [35]. A distinguishing feature of the work presented here is our support for AMR on curvilinear overlapping grids. As mentioned earlier, curvilinear grids are particularly useful to accurately represent boundaries. As the grids are refined, the original definition of the boundary-surface is evaluated to define the grid points at a high resolution. In our work we are able to represent complex three-dimensional geometries in a wide variety of ways, including, for example, analytically defined surfaces (such as cylinders and spheres) as well as splines and NURBS [36].

The software used to generate the results presented in this paper is freely available [3]. The parallel-AMR

[3] www.llnl.gov/casc/Overture

components that support regridding, interpolation, error estimation, etc. are part of a software toolkit within *Overture*, a object-oriented software package for the numerical solution of PDEs on overlapping grids. These component may be downloaded and used to solve many different types of equations. The AMR toolkit capabilities are accessed through a high-level C++ interface which we describe later. The **amrh** program within the Overture distribution solves advection-diffusion equations, and this program provides a demonstration of the use of this high-level interface. The PDE solver for the reactive Euler equation is called **cgcns**, and is also available from the Overture web site. It is part of the *CG* suite of solvers, which includes programs for solving a variety of PDEs such as the incompressible Navier-Stokes equations and Maxwell's equations, among others. (These latter two PDE solvers may be run in parallel but do not yet support AMR.)

The remaining sections of the paper are organized as follows. In Section 2, we introduce the general form of the initial-boundary-value problem for which the numerical approach may be applied, but then consider specific cases involving an advection-diffusion equation and the reactive Euler equations. This is followed in Section 3 by an overview of the overlapping grid approach with AMR. The details of the extension of the numerical approach for parallel computing are described in Section 4. In this section, we discuss parallel distributed arrays and their use in defining grids and grid functions. We also discuss parallel-AMR operations and our load-balancing algorithm in this section. Section 5 provides a brief description of the discretizations of the advection-diffusion equation and the reactive Euler equations on mapped grids. Numerical results are presented in Section 6. Here, we solve the advection-diffusion equations with forcing functions chosen so that exact solutions may be constructed a priori. We show that the error in the numerical solutions is independent of the number of processors. We also show that the error is independent of the number of refinement levels and refinement ratios provided the effective resolution on the finest grids are commensurate. In this section, we also solve the reactive Euler equations and validate the parallel-AMR approach for the problem of a planar shock diffracted by a sphere. For this problem, we also examine in detail the behavior and scalability of the parallel approach. As a final calculation presented in Section 6, we consider a reactive flow problem involving the initiation of a detonation from a hot spot in a complex three-dimensional domain which takes the form of a T-shaped pipe. Finally, conclusions drawn from the our work are discussed in Section 7.

2. Model Equations

We are interested in computing numerical solutions to well-posed initial-boundary-value problems (IBVPs) for time-dependent partial differential equations (PDEs) of the general form

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} = \mathcal{L}(\mathbf{u}, \mathbf{x}, t), & t > 0, \quad \mathbf{x} \in \Omega, \\ \mathbf{u} = \mathbf{u}_0(\mathbf{x}), & t = 0, \quad \mathbf{x} \in \Omega, \\ \mathcal{B}(\mathbf{u}, \mathbf{x}, t) = 0, & t > 0, \quad \mathbf{x} \in \partial\Omega, \end{cases} \tag{1}$$

where $\mathcal{L}$ is a differential operator involving spatial derivatives. The equations are to be solved for $t > 0$ on a domain Ω in three space dimensions with boundary $\partial\Omega$. The solution $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ is a vector with m components, and is a function of the independent variables $\mathbf{x} \in \mathbb{R}^3$ and $t \in \mathbb{R}$. The initial conditions for $\mathbf{u}$ are given by $\mathbf{u}_0(\mathbf{x})$ and the boundary conditions are $\mathcal{B}(\mathbf{u}, \mathbf{x}, t) = 0$, where $\mathcal{B}$ is a boundary operator involving, possibly, derivatives of $\mathbf{u}$ is the direction normal to $\partial\Omega$.

We consider two special cases of the general IBVP given in (1). The first involves a relatively simple advection-diffusion equation. The IBVP for this first case has the form

$$\begin{cases} \dfrac{\partial u}{\partial t} + \mathbf{a} \cdot \nabla u = \nu \Delta u + f(\mathbf{x}, t), & t > 0, \quad \mathbf{x} \in \Omega, \\ u = u_0(\mathbf{x}), & t = 0, \quad \mathbf{x} \in \Omega, \\ u = g(\mathbf{x}, t), & t > 0, \quad \mathbf{x} \in \partial\Omega, \end{cases} \tag{2}$$

where $u = u(\mathbf{x}, t)$ is a scalar function, $\mathbf{a} = \mathbf{a}(\mathbf{x}, t) \in \mathbb{R}^3$ is a given velocity, $\nu > 0$ is a constant diffusivity and $f(\mathbf{x}, t)$ is a given forcing function. The boundary conditions are taken to be of Dirichlet type but this is not essential. This linear IBVP with $m = 1$ components is particularly useful as a simple test case to study the behavior of our parallel approach involving adaptive mesh refinement on overlapping grids, and to verify the accuracy of the numerical results.

The second special case involves a more difficult set of equations given by the reactive Euler equations. This set of equations was considered in our previous papers [2] and [3] for two-dimensional flow in stationary and moving domains. Here, our focus is on reactive and non-reactive flow in three dimensions for which we consider the nonlinear conservation equations given by

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial}{\partial x_1}\mathbf{f}_1(\mathbf{u}) + \frac{\partial}{\partial x_2}\mathbf{f}_2(\mathbf{u}) + \frac{\partial}{\partial x_3}\mathbf{f}_3(\mathbf{u}) = \mathbf{h}(\mathbf{u}), \tag{3}$$

where

$$\mathbf{u} = \begin{bmatrix} \rho \\ \rho\mathbf{v} \\ E \\ \rho\mathbf{Y} \end{bmatrix}, \qquad \mathbf{f}_n = \begin{bmatrix} \rho v_n \\ \rho v_n \mathbf{v} + p\mathbf{e}_n \\ v_n(E+p) \\ \rho v_n \mathbf{Y} \end{bmatrix}, \quad n = 1, 2, 3, \qquad \mathbf{h} = \begin{bmatrix} 0 \\ \mathbf{0} \\ 0 \\ \rho\mathbf{R} \end{bmatrix}. \tag{4}$$

The symbols here have their usual meaning, namely, ρ is density, $\mathbf{v} = (v_1, v_2, v_3)$ is velocity, p is pressure and E is the total energy. For the reactive case, the flow is a mixture of m_r species whose mass fractions are given by $\mathbf{Y}$. The source term models the chemical reactions and is described by a set of m_r rates of species production given by $\mathbf{R} = \mathbf{R}(\mathbf{u})$. The total energy is taken to be

$$E = \frac{p}{\gamma - 1} + \frac{1}{2}\rho|\mathbf{v}|^2 + \rho q, \tag{5}$$

where γ is the ratio of specific heats and $q = q(\mathbf{Y})$ represents the heat energy due to chemical reaction. The number of components in the equations is $m = 5 + m_r$, but this reduces to $m = 5$ for the non-reactive case where we omit the m_r species equations in (4). The initial conditions for (3) are $\mathbf{u}(\mathbf{x}, 0) = \mathbf{u}_0(\mathbf{x})$ and the boundary conditions consist of inflow, outflow or solid-wall conditions as needed for each specific problem considered (see Sections 6.2 and 6.3).

3. Overlapping Grids with Adaptive Mesh Refinement

In this section, we give an overview of our general approach for solving initial-boundary-value problems of the type given in (1) on overlapping grids with adaptive mesh refinement (AMR). The description here builds primarily on the previous discussion given in [2] for solving IBVPs in two space dimensions. Since the numerical framework for solving problems in three space dimensions is similar, we will only outline the main elements of the approach here and refer the reader to our previous paper for further details. Once the general framework is established, we proceed in the next section to describe the extension of the numerical approach for parallel computations. It is worth noting that while the present discussion focuses on solving the IBVP given in (1), and the equations in (2) and (3) in particular, the numerical approach outlined here is more general and may be used to handle a wide range of time-dependent problems, such as those for the second-order Maxwell equations [37] and problems involving moving boundaries [3].

3.1. *Overlapping grids*

An overlapping grid $\mathcal{G}$ for Ω consists of a set of $\mathcal{N}_{\text{grid}}$ component grids G_g, i.e.,

$$\mathcal{G} = \{G_g\}, \qquad g = 1, 2, \ldots, \mathcal{N}_{\text{grid}} .$$

The component grids overlap and cover Ω. Each component grid is a logically rectangular, curvilinear grid defined by a smooth mapping $\mathbf{C}_g$ from parameter space $\mathbf{r}$ (e.g. the unit-cube in three dimensions) to physical space $\mathbf{x}$:

$$\mathbf{x} = \mathbf{C}_g(\mathbf{r}), \quad \mathbf{r} \in [0,1]^3, \quad \mathbf{x} \in \mathbb{R}^3 \ .$$

The mapping is used to define grid points at any desired resolution as required when a grid is refined. Variables defined on a component grid, such as the coordinates of the grid points, are stored in rectangular arrays. For example, grid vertices are represented as the array

$$\mathbf{x}^g_{\mathbf{i}} \ : \ \text{grid vertices}, \quad \mathbf{i} = (i_1, i_2, i_3), \quad i_\alpha = 0, \ldots, N_\alpha, \quad \alpha = 1, 2, 3 \ ,$$

where N_α is the number of grid cells in α-coordinate direction. We note that grid vertex information and other mapping information are not stored for Cartesian grids. This usually results in a considerable savings in memory use since most of the grid points belong to Cartesian grids for a typical overlapping grid.

Figure 1 shows a simple overlapping grid consisting of two component grids, a cylindrical grid and a background Cartesian grid. The top view shows the overlapping grid in physical space while the bottom views show each component grid in its parameter space. In this example, the cylindrical grid cuts a hole in the Cartesian grid so that the latter grid has a number of unused points. The other points on the component grids are classified as either discretization points (where the PDE or boundary conditions are discretized) or interpolation points. This information is supplied by the overlapping grid generator Ogen [38] and is held in an integer mask array. (In fact the bit representation of each element of the mask holds additional grid information including, for example, which points are hidden by refinement grids.) In addition, each boundary face of each component grid is classified as either a physical boundary (where boundary conditions are to be implemented), a periodic boundary or an interpolation boundary, and this information is held in the array $\mathtt{bc}(\beta, \alpha)$, where $\beta = 1, 2$ denotes the boundary side. Typically, one or more layers of ghost points are created for each component grid to aid in the application of boundary conditions.

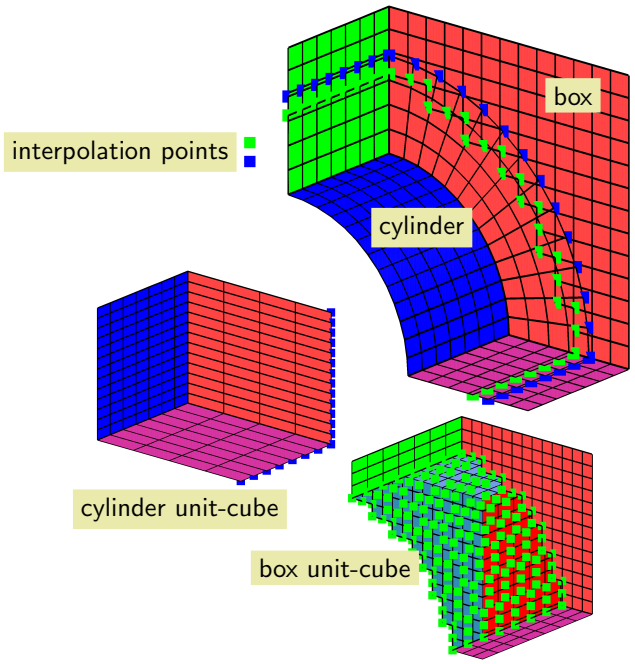

Fig. 1. Three-dimensional overlapping grid for a quarter-cylinder in a box: overlapping grid in physical space (top view) and the corresponding component grids on the unit cube in parameter space (bottom views). Interpolation points at the grid overlap are marked and color-coded for each component grid.

Solution values at interpolation points of a grid G_g, for example, are determined by interpolation from *donor* points on another grid G_{g_d}. The donor points are required to be either discretization points or interpolation points. An interpolation formula is said to be *explicit* if the donor points are all discretization

points. If some donor points are themselves interpolation points then the interpolation is said to be *implicit*. Explicit interpolation is simpler, but the width of the overlap required is wider than that for implicit interpolation. Either choice is available in Ogen, but for the present work we use explicit interpolation for efficiency in our parallel algorithm (see Section 4). For each interpolation point $\mathbf{x_i}$ on grid G_g, its corresponding parameter space coordinates, $\mathbf{r_j} = \mathbf{C}_{g_d}^{-1}(\mathbf{x_i})$, on donor grid G_{g_d} may be found using the inverse mapping. In the parameter space of the donor grid, standard tensor-product polynomial interpolation is used about the point $\mathbf{r_j}$. For first-order hyperbolic systems, such as the reactive Euler equations in (3), linear interpolation is sufficient for second-order accuracy. For second-order parabolic systems, such as the advection-diffusion equation in (2), quadratic interpolation is needed for second-order accuracy; see the discussion in Chesshire and Henshaw [1] for further details.

3.2. *Adaptive mesh refinement*

The adaptive mesh refinement (AMR) approach is designed to locally increase the grid resolution where an estimate of the error is large. This is done by adding refined grid patches to the existing base-level component grids. The refinement grids are aligned with the underlying base grid (i.e. the refinement is done in parameter space) and are arranged in a hierarchy with the base grids belonging to level $\ell = 0$, the next finer grids being added to level $\ell = 1$ and so on. Grids on level ℓ are refined by a refinement ratio n_r from the grids on level $\ell - 1$. The grids are properly nested so that a grid on level ℓ is completely contained in the set of grids on the coarser level $\ell - 1$. This requirement is relaxed at physical boundaries to allow refinement grids to align with the boundary.

For simplicity the numerical solution on all grids is advanced in time using the same time step. After every n_{regrid} time steps, the whole refined-grid hierarchy is rebuilt to accommodate the evolution of sharp features of the solution. This is done by first re-computing an estimate of the error given by

$$e_{\mathbf{i}} = \sum_{k=1}^{m} e_{k,\mathbf{i}} + \tau_{\mathbf{i}} \ , \tag{6}$$

where $\tau_{\mathbf{i}}$ is an estimate of the truncation error in the integration of the reactive source term for the PDEs in (3), see Section 5.2, and

$$e_{k,\mathbf{i}} = \frac{1}{3} \sum_{\alpha=1}^{3} \left(\frac{c_1}{s_k} |\Delta_{0\alpha} U_{k,\mathbf{i}}| + \frac{c_2}{s_k} |\Delta_{+\alpha}\Delta_{-\alpha} U_{k,\mathbf{i}}| \right) . \tag{7}$$

In (7), $\Delta_{0\alpha}$, $\Delta_{+\alpha}$ and $\Delta_{-\alpha}$ are the centered, forward and backward undivided difference operators in the α index direction, respectively, $U_{k,\mathbf{i}}$ is the kth component of the numerical solution for $\mathbf{u}$ at grid index $\mathbf{i}$, s_k is a scale factor for component k, and c_1 and c_2 are weights. The error estimate used here follows that introduced in [2] and has been found to be an effective choice, although other methods are possible. Once the error estimate is computed, it is smoothed and then grid points are tagged for refinement where $e_{\mathbf{i}}$ is greater than a chosen tolerance. Buffer points are added to increase the region of tagged points slightly (so that fewer regrids are needed), and a new overlapping grid hierarchy is build to cover the buffered region of tagged points. (Typically, the width of the buffer is taken to be 2 so that $n_{\text{regrid}} = 2n_r$, see [2].) The numerical solution is then transferred from the old grid hierarchy to the new one, and the time-stepping proceeds for the solution on the new grid hierarchy until the next gridding step.

3.3. *Time-stepping algorithm*

For later reference, we show in Figure 2 the basic time-stepping algorithm for the numerical solution of the IBVP in (1) on a overlapping grid $\mathcal{G}$. The algorithm begins by specifying the initial conditions given by $\mathbf{u}_0(\mathbf{x})$ for the numerical solution on $\mathcal{G}$. The time-stepping loop performs a numerical integration of the equations from $t = 0$ to a specified time t_{final}. Within this loop there are a number of steps that perform an AMR regrid once every n_{regrid} time steps as outlined above. A value for the global time-step increment, Δt, is computed

every time step based on a suitable stability constraint for the equations, e.g. a CFL stability constraint for the reactive Euler equations. The numerical integration of the governing equations from t to $t + \Delta t$, based on a chosen discretization of the equations on $\mathcal{G}$, is performed by the function **advancePDE**. For the advection-diffusion equations in (2) this discretization consists of centered, second-order accurate differences in space, and a Runge-Kutta integration, either second or fourth order, in time. In the case of the reactive Euler equations in (3), a fractional-step method is used which involves a second-order accurate extension of Godunov's method for the convective part of the equations in one step and a Runge-Kutta scheme for the reactive source terms in the other step. Further details of these schemes are given in Section 5. Once the solution at all discretization points on $\mathcal{G}$ is found at the new time, the solution is communicated to other component grids via interpolation at the grid overlap. Finally, a numerical approximation of the boundary conditions, $\mathcal{B}(\mathbf{u}, \mathbf{x}, t) = 0$, is applied on the boundary of the overlapping grid $\mathcal{G}$, and this is done in the function **applyBoundaryConditions**.

```
PDEsolve(G, t_final)
{
  t := 0; n := 0;
  u_i^n := applyInitialCondition(G);
  while t < t_final
    if (n mod n_regrid == 0)
      e_i := estimateError(G, u_i^n);
      G* := regrid(G, e_i);
      u_i* := interpolateToNewGrid(u_i^n, G, G*);
      G := G*; u_i^n := u_i*;
    end

    Δt := computeTimeStep(G, u_i^n);
    u_i^{n+1} := advancePDE(G, u_i^n, t, Δt);

    t := t + Δt; n := n + 1;
    interpolate(G, u_i^n);
    applyBoundaryConditions(G, u_i^n, t);
  end
}
```

Fig. 2. The basic time-stepping algorithm for the numerical solution of the IBVP in (1) on a overlapping grid $\mathcal{G}$.

The basic time-stepping algorithm in Figure 2 is the same whether the problem is run in parallel or not. However, for the parallel case there are a number of issues for many of the steps in the algorithm that require further discussion. We address these issues in the next section.

4. Parallel Approach for Adaptive Overlapping Grids.

With the basic framework in hand for solving an initial-boundary-value problem for a PDE on an overlapping grid with AMR, we are now in a position to describe the extension of this framework for parallel computations. The extension is a domain-decomposition approach in which each component grid belonging to the overlapping grid is partitioned across different processors of a distributed-memory parallel computer as illustrated in Figure 3. The sample overlapping grid in the figure, shown in two dimensions for simplicity, consists of component grids labeled G_1 and G_2 at the base level, with G_2 cutting a hole in G_1, and refinement grids labeled G_3 and G_4. Each component grid is partitioned over a contiguous range of processors, e.g. $p = \{3, 4, 5\}$ for G_3. Grid functions defined on each component grid are partitioned in the same way as their component grid. Field data in a grid function, which may be distributed across several processors, is stored in a multidimensional P++ array. The P++ array, as described below, handles the updating of field data at ghost points associated with internal boundaries between processors. In addition, communication

is needed between processors to update interpolation points at the overlap between grids (the grid points marked with small squares in Figure 3) and to handle AMR interpolation involving values on refinement grids. These issues, as well as a method of load balancing, are described in detail in the sections below.

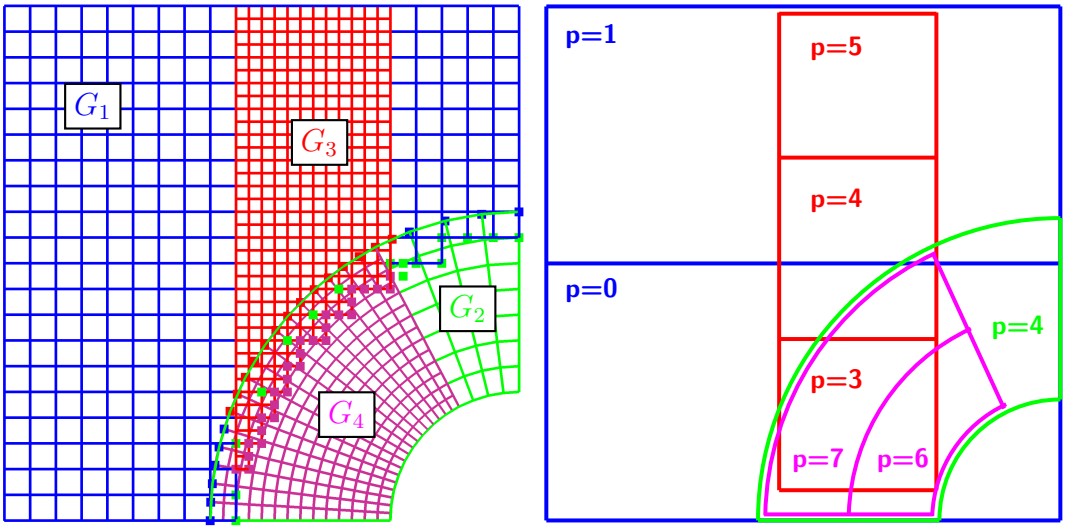

Fig. 3. Each base grid or refinement grid can be distributed over a contiguous range of processors. In this example the base grid G_1 is distributed over processors $[0,1]$, the base grid G_2 over processor $[4]$, the refinement grid G_3 over processors $[3,4,5]$ and the refinement grid G_4 over processors $[6,7]$.

4.1. *P++ distributed arrays*

A basic tool for our parallel approach is the P++ array class [39], a C++ class that can be used to represent distributed multi-dimensional arrays. Each P++ array can be independently partitioned across a set of processors. A distributed P++ array consists of a set of serial arrays, one serial array for each processor. Each serial array is a multi-dimensional array that can be manipulated using various array operations. The data from a serial array can also be passed to Fortran subroutines, which is useful to define optimized computational kernels, such as the discretization of a PDE representing an integration over a time step. When running in parallel, the serial arrays contain extra ghost values that hold copies of the data from the serial arrays on neighboring processors. The P++ array class is built on top of the Multiblock PARTI parallel communication library [40], which is used for updating the values on ghost boundaries from neighboring processors. All parallel communication is performed using the Message Passing Interface, MPI [41].

A P++ array can have up to six dimensions. A typical vector solution field, for example, resides in a four-dimensional array $\mathbf{u}(i_1,i_2,i_3,k)$, where (i_1,i_2,i_3) are the *coordinate dimensions* in index space and k is the *component dimension* that specifies the different components, e.g. ρ, ρv_1, ρv_2, etc. for the reactive Euler equations. Each array dimension can be a distributed dimension or not. If an array dimension is distributed, then it may be split across processors. For the solution vector field, $\mathbf{u}(i_1,i_2,i_3,k)$, the three coordinate dimensions are distributed while the component dimension is not. For each distributed dimension, we specify the width of the parallel ghost boundary according to the width of the stencil in the discretization of the PDE. For example, a stencil width of $2q+1$ generally requires q layers of ghost points.

For the version [4] of Multiblock PARTI we use, a P++ array must be distributed over a contiguous range of process numbers (as in Figure 3). In addition, the array is partitioned in a regular tensor-product fashion with each distributed dimension being split into n_α processors, where the total number of processors is the product $\prod_{\alpha=1}^{d} n_\alpha$ for d distributed dimensions. Thus, for example, if we have an array $\mathbf{u}(0:35,0:35,0:35,0:5)$ partitioned over $n_1 \times n_2 \times n_3 = 4 \times 3 \times 2 = 24$ processors, then the first processor would have a corresponding

[4] The latest version of PARTI can partition arrays in a more general fashion

serial array with dimensions $\mathbf{u}(0:8, 0:11, 0:17, 0:5)$, excluding ghost points. (The arrays on other processors would have the same size but with offset values for the coordinate bases and bounds.) If one layer of ghost points had been requested in each coordinate dimension, then the actual serial array would be dimensioned as $\mathbf{u}(-1:9, -1:12, -1:18, 0:5)$.

The properties of the parallel distribution of a P++ array are contained in a separate partitioning object defined by the `Partitioning_Type` class. The partitioning object contains information on the ghost boundary widths, which dimensions are distributed, and the set of processors used. Generally one partitioning object can be shared by many arrays. The partitioning object hides the fact that we are using Multiblock PARTI and thus would allow the use of other partitioning libraries without changes to the higher-level code.

Figure 4 presents some C++ code that illustrates the use of P++ arrays. In this code, a partitioning object is defined and its parameters are specified. A multidimensional P++ array $\mathbf{u}$ is built based on the partitioning object and its values are assigned (all to be 6 in the example). Next, the array values are changed using a high-level operation that automatically performs parallel updates of ghost values. We also show how to access the local serial array and its array bounds. We then illustrate an array operation involving the serial array and show how its data may be passed to a Fortran subroutine. After the serial data has been manipulated by the Fortran routine, the parallel ghost boundaries are updated explicitly in the sample code.

For efficiency, we usually avoid using high-level P++ array operations since these operations generally use message passing for each array statement. Instead, we typically operate on the local serial arrays directly. These are passed to optimized Fortran kernels, as mentioned earlier and illustrated in Figure 4, which are followed by an explicit call to the ghost-boundary-update function that synchronizes the data at the parallel ghost boundaries.

4.2. *Distribution of Grids and Grid Functions*

Grid functions hold field data such as the density, momenta or energy. A grid function defined on the entire overlapping grid (including all refinement grids) is represented by a C++ object defined by the `realCompositeGridFunction` class. A `realCompositeGridFunction` consists of a collection of objects defined by the `realMappedGridFunction` class, and these latter objects contain field data on the component grids, either on the base level or on refinement levels. The data in a `realMappedGridFunction` is stored in a P++ array. An overlapping grid, such as the one in Figure 3, is represented by an object defined by the `CompositeGrid` class and consists of a collection of `MappedGrid` objects. A `MappedGrid` represents a component grid and contains grid functions that hold geometric information, such as the grid vertices and the metric terms $\partial\mathbf{x}/\partial\mathbf{r}$ of the mapping $\mathbf{x} = \mathbf{C}_g(\mathbf{r})$. Each grid function is associated with a corresponding grid, so, for example, a `realCompositeGridFunction` has a pointer to a `CompositeGrid` while a `realMappedGridFunction` has a pointer to a `MappedGrid`. In a parallel setting, we associate a single parallel partition with each `MappedGrid`, and all grid functions belonging to this object are partitioned in the same way. Thus, the arrays local to each processor have the same size for all grid functions belonging to a given grid. This allows efficient operations, with reduced communication costs, between grid functions associated with the same grid. Different grids, however, are allowed to have different partitions. We note that other parallel-AMR frameworks may require each refinement grid to be partitioned over a single processor (SAMRAI [35], Chombo [31], Paramesh [34]), while some may require refinement grids to be on the same processor as the parent grid (DAGH [32]). Our approach employs a more general strategy where each grid can be independently partitioned over multiple contiguous processors. It is then left to the load balancer to efficiently distribute the grids over the processors.

4.3. *Interpolation*

There are a number of forms of interpolation and transfers of solution values between grids that occur during the time-stepping algorithm in Figure 2. These include interpolation at overlapping-grid boundaries, interpolation of refinement-grid boundaries and interpolation between coarse and fine grids, as mentioned in

```
// Define a parallel distribution object
Partitioning_Type partition;

// Set parameters for the distribution
partition.SpecifyProcessorRange(Range(7,14));        // partition over processors 7 to 14
partition.SpecifyDecompositionAxes(2);               // first 2 dimensions are distributed
partition.SpecifyInternalGhostBoundaryWidths(1,1);   // specify 1 parallel ghost boundary

// Create a distributed P++ array and define index range objects
realDistributedArray u(100,100,3,partition);    Range I(1,98), J(1,98);

// Assign all values of the array
u=6.;

// Sample parallel array operation with automatic communication
u(I,J,0)=.25*( u(I+1,J,1) + u(I-1,J,1) + u(I,J+1,1) + u(I,J-1,1) ) + sin(u(I,J,2))/3.;

// Access the serial array local to this processor and define index range objects
realSerialArray & uLocal = u.getLocalArray();
I=uLocal.dimension(0); J=uLocal.dimension(1);

// Sample array operation on the serial array
uLocal(I,J,0)=.5*( uLocal(I+1,J,1) + uLocal(I,J+1,1) ) + sin(uLocal(I,J,2))/3.;

// Sample call to a Fortran routine with the local serial array data
myFortranRoutine( *uLocal.getDataPointer() );

// Explicit update of the ghost boundaries on the distributed array
u.updateGhostBoundaries();
```

Fig. 4. Sample C++ code showing the use of the P++ distributed arrays.

Section 3. These interpolation steps generally require communication between processors. Here, we discuss how these interpolation steps are performed in parallel.

4.3.1. *Overlapping grid interpolation*

On the boundaries where grids overlap, the solution value at an interpolation point $\mathbf{x_i}$ of grid G_g is interpolated from values on a donor grid G_{g_d} using the interpolation formula

$$U_{\mathbf{i}}^{(g)} = \sum_{\mathbf{m}} c_{\mathbf{i},\mathbf{m}} U_{\mathbf{j}+\mathbf{m}}^{(g_d)}, \qquad \mathbf{m} = (m_1, m_2, m_3), \quad 0 \le m_\alpha \le w-1 \ . \tag{8}$$

Here, w is the width of the interpolation stencil, $U_{\mathbf{i}}^{(g)}$ represents the solution value at an interpolation point on G_g, and $U_{\mathbf{j}+\mathbf{m}}^{(g_d)}$ represents solution values on G_{g_d}. The weights $c_{\mathbf{i},\mathbf{m}}$ are determined from a tensor product Lagrange interpolant [1]. In a distributed-memory parallel computation, the value $U_{\mathbf{i}}^{(g)}$ of an interpolation point is usually located on a different processor from the donor values $U_{\mathbf{j}+\mathbf{m}}^{(g_d)}$. We choose the width of the parallel ghost boundaries to be at least $\lfloor w/2 \rfloor$ so that all solution information needed by the right-hand side of (8) can be evaluated on a single processor without the cost of communication with other processors. Thus, we can first evaluate the right-hand-side sums of (8) on each processor p_s (*source processor*) that owns the relevant donor values. The resulting sums for the source processors are then sorted into sets based on the *destination processor* p_d that owns the associated interpolation value $U_{\mathbf{i}}^{(g)}$. In this scheme, each processor p_s sends at most one message (containing the sums) to any other processor, p_d. The destination processor unpacks the messages it receives from other processors and then assigns the values to its interpolation points.

All the overlapping grid interpolation points can thus be evaluated with a minimal number of messages, and these messages are small since we only pass the interpolation sums.

The location of overlapping grid interpolation points on base-level grids is determined initially when the overlapping grid is generated. This information remains static during the computation and thus the communication schedules (i.e. information on the size and type of messages that will be sent/received) for transferring the interpolation sums described above can be determined once. On the other hand, the locations of overlapping grid interpolation points on refinement grids and the classification of points on refinement grids as discretization/interpolation/unused (as in the base-level grids in Figure 1) is not static, and must be determined at each AMR regrid step. This determination is done in parallel and requires the communication of information between neighboring base grids. The locations of some of the new interpolation points (those not co-incident with base-grid interpolation points) are computed by inversion of the mapping, $\mathbf{C}_g$, as discussed in Section 3, and this inversion may require communication depending on the representation of the mapping. A mapping for an annulus, for example, can be inverted directly from an analytic formula and thus requires no communication. Conversely, a mapping defined by interpolating a distributed set of data points may require communication to invert. Once the new overlapping grid interpolation points are found on refinement levels, the communication schedules can be determined to efficiently evaluate these interpolation points.

4.3.2. *AMR interpolation*

At each step in the time-stepping algorithm, values on refinement grid boundaries are interpolated from neighboring refinement grids on the same refinement level (requiring just a copy of values) or from grids at the coarser level (requiring interpolation). This *AMR-boundary-interpolation* may require communication between processors since each grid can have its own parallel distribution. For each refinement grid we determine how to interpolate its ghost boundary values from other grids. This is done by intersecting the box (in index space) that covers the ghost values on a given boundary with the index box for neighboring refinement grids. (Here, we only consider interpolation from grids belonging to the same base grid, and thus these grids will use the same index space.) Ghost points that cannot be copied from grids at the same refinement level are instead interpolated from grids at the next coarser level. Since this interpolation process is somewhat complicated, we have first implemented it by a straightforward extension of the algorithm we use for serial computations, which has been tested extensively. The computations presented in later sections of this paper use this extension, and the correctness and accuracy of the method is checked carefully. We recognize, however, that this initial parallel implementation is not especially efficient since separate messages are passed when interpolating each refinement grid, and this inefficiency plays a role in our study of the scalability of parallel computations with AMR (see Table 8 in Section 6.2). As a future optimization, we will determine a composite communication schedule for AMR-boundary-interpolation so that these smaller messages are merged into a fewer number of larger messages.

When the locations of the AMR grids are recomputed (every n_{regrid} time steps) it is necessary to transfer solution values from the old AMR grid hierarchy to the new grid hierarchy. We call this process the *refinement-grid-transfer* step. For each refinement grid on the new hierarchy, we determine the intersection of grids (from the same base grid) on the old hierarchy, and then copy or interpolate the best available solution data. The parallel distribution of the old grids can, in general, be completely different from the distribution of the new grids, and thus there can be significant communication required to update the solution values of the new grid hierarchy. As in the above case for AMR-boundary-interpolation, we have implemented the refinement-grid-transfer step in parallel by first extending our well-tested serial algorithm. In the future this step also needs to be optimized to reduce the number of messages.

4.4. *AMR regridding*

The basic procedure for AMR regridding for overlapping grids was described briefly in Section 3.2. The first step in the procedure involves computing an estimate of the error, and this step is done in parallel and requires communication across processors. The basic error estimate given in (7) can be computed with

no communication, but smoothing the error requires some communication as it may be viewed as taking a few time steps in the integration of a heat equation on an adaptively-refined overlapping grid. The error-smoothing step thus applies a version of the **interpolate** and **applyBoundaryConditions** functions in the time-stepping algorithm given in Figure 2. The error estimate is a scalar grid function which does not need to be computed on the finest refinement level. Thus, the error-estimation step is usually less expensive than taking a time step in solving the PDE in 1.

The AMR regridding algorithm uses the smoothed error estimate to determine the location of refinement grids. The clustering algorithm, an extension of the algorithm of Berger and Rigoutsos [42], determines a set of non-overlapping boxes that cover the grid cells tagged by the error estimator [2]. The boxes are computed in the index space of the grid so that in physical space the refinement grids align with the curvilinear grid lines. The algorithm begins by enclosing all tagged cells in a single box. This box is split into two parts which are then enclosed in two new (smaller) boxes. The algorithm continues to recursively divide the boxes until the boxes are sufficiently filled with tagged cells. The algorithm for serial computers was extended in a straight-forward manner to parallel computers. The basic operation of finding the bounding box for a set of tagged cells can be accomplished by first computing a bounding box for the portion of the grid on each processor and then merging these results. Our current implementation of the clustering algorithm may need to be improved when running on thousands of processors. Gunney et al. [43], for example, discuss some of the issues involved in getting clustering algorithms to work efficiently on $\mathcal{O}(10^5)$ or more processors by reducing global communication and increasing task parallelism.

4.5. *Load balancing*

There are a variety of approaches that can be used to balance the workload for AMR computations. These approaches include the use of space-filling curves, bin-packing algorithms, spatial-bisection algorithms, or a combination thereof [44,45]. It is also possible to use an adaptive technique that chooses the best algorithm from amongst a collection of different algorithms [46]. In general, a load-balancing algorithm must take into account the situation when the workloads on some grid points are much larger than other points, as might occur in chemically reactive flows with many species [47]. We have developed a modified bin-packing algorithm for load balancing that divides each grid into a contiguous range of processors (as described below). In the future we expect to enhance this algorithm and make use of other available methods. Our software is designed so that other load-balancing algorithms can be added easily.

For our purposes here, we consider the load-balancing problem to be a *bin-packing* problem, where each processor represents a bin and the problem is to fill the bins so that the workload is evenly distributed amongst the bins. Let W_g, $g = 1, 2, \ldots, \mathcal{N}_{\text{grid}}$, denote the given workload for grid g. This value is determined by the PDE application and could, for example, be a relative measure of the number of floating-point operations required per time step for all points on the grid. For the applications considered in this paper, we assume that the computational work per grid point is uniform so that W_g is simply the number of discretization points for grid g. Let $\overline{W} = \sum_g W_g / N_{\text{proc}}$ denote the average workload per processor. If the sum of workloads assigned to processor p is w_p, then the *maximum imbalance*, $\mathcal{I}$, is defined as

$$\mathcal{I} = \max_{0 \le p \le N_{\text{proc}}-1} |w_p/\overline{W} - 1|, \tag{9}$$

so that $\mathcal{I} = 0$ corresponds to a perfectly balanced problem. Thus, the basic goal of the load-balancing algorithm is to fill the bins so that $\mathcal{I}$ is as small as possible.

If we assign each grid to just one processor, then the load-balancing problem may be defined as follows: Given a set of $\mathcal{N}_{\text{grid}}$ workloads $\{W_g\}$ and a set of N_{proc} bins (processors), assign workloads to the bins to minimize $\mathcal{I}$. However, since we have the flexibility to distribute any grid over multiple processors, we consider the following generalized load-balancing problem: allow each workload to be split into M_g equal parts, $1 \le M_g \le N_{\text{proc}}$. Find a bin-packing distribution such that $\mathcal{I} \le \mathcal{I}_T$, where $\mathcal{I}_T > 0$ is a tolerance on the maximum load imbalance, and such that the total number of blocks, $\sum_g M_g$, is minimized. Bin-packing algorithms are NP hard and are often solved approximately with heuristic algorithms. We use a generalized

loadBalance(W_g, $N_{\rm proc}$, $\mathcal{I}_T$)
W_g (input) : workload for grid g, $g = 1, 2, \ldots, \mathcal{N}_{\rm grid}$
$N_{\rm proc}$ (input) : number of processors
$\mathcal{I}_T$ (input) : tolerance on the maximum load imbalance
$P_g = [a_g, b_g]$ (output) : processor range, $p \in [a_g, b_g]$, for grid g
{
Find π_i so that $W_{\pi_i} \geq W_{\pi_{i+1}},\ i = 1, \ldots, \mathcal{N}_{\rm grid} - 1$ (*sort by decreasing workload*)
$\overline{W} \leftarrow \sum_g W_g / N_{\rm proc}$ *(average work per processor)*
$\mathcal{I} \leftarrow \infty$ *(holds imbalance)*
$\eta \leftarrow \frac{1}{2}$ *(initial split fraction)*
while $\mathcal{I} > \mathcal{I}_T$
$w_p \leftarrow 0, \quad p = 0, .., N_{\rm proc} - 1$ (*initialize work per processor)*
for $i = 1, ..., \mathcal{N}_{\rm grid}$ (*loop from largest to smallest workload)*
$g \leftarrow \pi_i$ (*grid with next smaller workload)*
$M_g \leftarrow \max(1, \min(N_{\rm proc}, \lfloor W_g/(\eta\overline{W}) + \frac{1}{2} \rfloor))$ *(split grid g over M_g processors)*
Find $p_0 \in [0, N_{\rm proc} - M_g]$ to minimize
$K_q = \sum_{p=q}^{q+M_g-1} w_p,\ q = 0, \ldots, N_{\rm proc} - M_g + 1$
$P_g \leftarrow [p_0, p_0 + M_g - 1]$ (*assign processors for grid g)*
$w_p \leftarrow w_p + W_g/M_g,\ p = p_0, ..., p_0 + M_g - 1$ *(increment work)*
end
$\mathcal{I} \leftarrow \max_p |w_p/\overline{W} - 1|$ (*compute maximum imbalance)*
$\eta \leftarrow \eta/2$ *(decrease the split factor)*
end
}

Fig. 5. Algorithm to load balance a collection of grids with workloads given by W_g, $g = 1, \ldots, \mathcal{N}_{\rm grid}$, over $N_{\rm proc}$ processors when each grid can be split into M_g blocks and distributed over a contiguous range of processors.

bin-packing algorithm with adjustments made to take into account that grids can be split across a contiguous range of processors. We do not explicitly take communication costs into account.

The load-balancing algorithm we use is given in Figure 5. The basic *best fit decreasing* bin-packing algorithm would start by sorting the workloads from largest to smallest. The grids are assigned to processors (each processor being a bin) starting from the grid with the largest workload and successively adding each workload to the processor that currently has the least total workload. In the generalized situation, each grid can be split into M_g blocks and distributed over a contiguous range of M_g processors. A problem is to decide how to choose M_g for each grid. We want to avoid splitting a grid into too many small blocks, since this would likely increase communication costs. We define a *split factor* η, $0 \leq \eta < 1$, that determines the degree to which a grid is split with $M_g \rightarrow N_{\rm proc}$ as $\eta \rightarrow 0$. We base our splitting decision on the average workload per processor, $\overline{W}$. The ratio $W_g/\overline{W}$ indicates what fraction of an optimal-size bin that grid g would fill, if grid g were not split. We begin by guessing that a grid should be split into blocks that fill at most half of a bin, $\eta = \frac{1}{2}$, giving $M_g \approx W_g/(\eta\overline{W})$. (In practice we actually choose M_g more carefully as a product of integers, $M_g = m_1 m_2 m_3$, so that the resulting grid partitions have a more equal number of grid points in each coordinate direction.) After splitting a grid into M_g blocks we then find a contiguous set of M_g bins to fill by choosing the set of contiguous bins that currently has the minimal sum of workloads. After we have filled the bins we check how well the work has been balanced. If the maximum imbalance is larger than the tolerance, $\mathcal{I}_T$, we decrease η and try again. We note that as $\eta \rightarrow 0$, eventually all grids would be split across all processors resulting in a *perfect* balance. Of course the communication costs are likely to be higher in this case so that we prefer having fewer blocks with more work per block.

5. Discretization of the governing equations

We now turn to a discussion of the discretization of the governing equations in (1) for an overlapping grid $\mathcal{G}$. In general, the overlapping grid consists of a set of component grids, G_g, $g = 1, 2, \ldots, \mathcal{N}_{\text{grid}}$, which includes grids on the base level and possibly grids on refinement levels. As mentioned previously, each component grid is defined by a smooth mapping $\mathbf{C}_g$ from parameter space $\mathbf{r}$ to physical space $\mathbf{x}$. The basic strategy is to consider a generic grid G_g with its known mapping $\mathbf{x} = \mathbf{C}_g(\mathbf{r})$, and first make an exact change of variables from $(\mathbf{x}, t)$ to $(\mathbf{r}, t)$ in the governing equations. Once this is done, the mapped equations are approximated using a suitable method of discretization as we discuss below for the case of the linear advection-diffusion equation in Section 5.1 and for the nonlinear reactive Euler equations in Section 5.2.

5.1. *Discretization of the advection-diffusion equation*

The advection-diffusion equation in (2) is discretized in a straightforward manner. The first and second derivatives with respect to $\mathbf{x}$ are changed to derivatives with respect to $\mathbf{r}$ using the chain-rule formulas

$$\frac{\partial u}{\partial x_j} = \sum_{\alpha=1}^{3} \frac{\partial r_\alpha}{\partial x_j} \frac{\partial u}{\partial r_\alpha},$$

and

$$\frac{\partial^2 u}{\partial x_j^2} = \sum_{\alpha_1=1}^{3} \sum_{\alpha_2=1}^{3} \frac{\partial r_{\alpha_1}}{\partial x_j} \frac{\partial r_{\alpha_2}}{\partial x_j} \frac{\partial^2 u}{\partial r_{\alpha_1} \partial r_{\alpha_2}} + \sum_{\alpha=1}^{3} \frac{\partial^2 r_\alpha}{\partial x_j^2} \frac{\partial u}{\partial r_\alpha},$$

for $j = 1, 2, 3$. The mapped equations are then discretized in space using standard second-order, centered differences. The resulting ODEs have the form

$$\frac{d}{dt} U_{\mathbf{i}}(t) + \mathbf{a} \cdot \nabla_h U_{\mathbf{i}}(t) = \nu \Delta_h U_{\mathbf{i}}(t) + f_{\mathbf{i}}(t), \tag{10}$$

where ∇_h and Δ_h denote discrete approximations for the gradient and Laplacian operators on the mapped grid for a representative mesh spacing h, $U_{\mathbf{i}}(t)$ is an approximation for $u(\mathbf{x}_{\mathbf{i}}, t)$, and $f_{\mathbf{i}}(t)$ is the forcing function evaluated on the grid. Equation (10) is applied at all valid interior points. The values of the solution at valid points on physical boundaries are specified by Dirichlet boundary conditions. The values of $U_{\mathbf{i}}(t)$ at interpolation points are obtained by interpolation from another component grid. For the latter case, quadratic interpolation is used to maintain second-order accuracy since the equations involve second-order derivatives (see [1]). Finally, the ODEs in (10) are advanced in time using either a second-order or a fourth-order accurate Runge-Kutta scheme.

5.2. *Discretization of the reactive Euler equations*

The discretization of the reactive Euler equations in (3) follows the approach discussed in [2] for two-dimensional flow on stationary domains, and in [3] for moving domains. Here, we describe briefly an extension of the approach to handle three-dimensional flow. As in the previous case for the advection-diffusion equations, the governing equations are mapped to parameter space assuming a known mapping given by $\mathbf{x} = \mathbf{C}_g(\mathbf{r})$, but for this case special care is used to maintain a conservation form. These mapped equations are

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{J} \frac{\partial}{\partial r_1} \hat{\mathbf{f}}_1(\mathbf{u}) + \frac{1}{J} \frac{\partial}{\partial r_2} \hat{\mathbf{f}}_2(\mathbf{u}) + \frac{1}{J} \frac{\partial}{\partial r_3} \hat{\mathbf{f}}_3(\mathbf{u}) = \mathbf{h}(\mathbf{u}), \tag{11}$$

where J is the Jacobian of the transformation matrix $[\mathbf{x}_{\mathbf{r}}]$ and the mapped fluxes $(\hat{\mathbf{f}}_1, \hat{\mathbf{f}}_2, \hat{\mathbf{f}}_3)$ are given in terms of the original flux functions in (4) by

$$\hat{\mathbf{f}}_\alpha(\mathbf{u}) = s_{\alpha,1} \mathbf{f}_1(\mathbf{u}) + s_{\alpha,2} \mathbf{f}_2(\mathbf{u}) + s_{\alpha,3} \mathbf{f}_3(\mathbf{u}), \qquad \alpha = 1, 2, 3. \tag{12}$$

The metrics $s_{i,j}$ in (12) are components of matrix $S = J[\mathbf{r}_\mathbf{x}] = J[\mathbf{x}_\mathbf{r}]^{-1}$, i.e.

$$s_{1,1} = \det \begin{bmatrix} \dfrac{\partial x_2}{\partial r_2} & \dfrac{\partial x_2}{\partial r_3} \\ \dfrac{\partial x_3}{\partial r_2} & \dfrac{\partial x_3}{\partial r_3} \end{bmatrix}, \quad s_{1,2} = \det \begin{bmatrix} \dfrac{\partial x_3}{\partial r_2} & \dfrac{\partial x_3}{\partial r_3} \\ \dfrac{\partial x_1}{\partial r_2} & \dfrac{\partial x_1}{\partial r_3} \end{bmatrix}, \quad s_{1,3} = \det \begin{bmatrix} \dfrac{\partial x_1}{\partial r_2} & \dfrac{\partial x_1}{\partial r_3} \\ \dfrac{\partial x_2}{\partial r_2} & \dfrac{\partial x_2}{\partial r_3} \end{bmatrix}, \quad \text{etc.}$$

The mapped flux $\hat{\mathbf{f}}_\alpha(\mathbf{u})$ represents the flux of $\mathbf{u}$ across the surface r_α = constant. Finally, the source term $\mathbf{h}(\mathbf{u})$ in (11) is determined by the reaction rate and its form is given in (4).

We discretize (11) on a uniform grid with grid spacings Δr_α, $\alpha = 1, 2, 3$. Let

$$\mathbf{U}_\mathbf{i}(t) = \frac{1}{\Delta r_1 \Delta r_2 \Delta r_3} \iiint_{V_\mathbf{i}} \mathbf{u}(\mathbf{r}, t)\, d\mathbf{r}$$

denote the average of $\mathbf{u}$ over a grid cell $V_\mathbf{i}$ of width $(\Delta r_1, \Delta r_2, \Delta r_3)$ about the point $\mathbf{r}_\mathbf{i}$. The cell average of $\mathbf{u}$ is advanced from a time t to $t + \Delta t$ using the second-order fractional-step method

$$\mathbf{U}_\mathbf{i}(t + \Delta t) = S_h(\Delta t/2) S_f(\Delta t) S_h(\Delta t/2) \mathbf{U}_\mathbf{i}(t), \tag{13}$$

where S_h and S_f are operators representing discretizations of the source term and the hydrodynamic terms of (11), respectively, and where Δt is a global time step determined for all component grids by a CFL condition as discussed below.

The two reaction steps in (13) are performed by solving the ordinary differential equations

$$\frac{\partial}{\partial t}\mathbf{u} = \mathbf{h}(\mathbf{u}), \tag{14}$$

over a time interval $\Delta t/2$. These equations reduce to the system of m_r ODEs

$$\frac{\partial}{\partial t}\mathbf{Y} = \mathbf{R}, \quad \text{with } (\rho, \rho\mathbf{v}, E) \text{ held fixed}, \tag{15}$$

which is solved numerically using an adaptive Runge-Kutta error-control scheme as described in [2]. This second-order scheme allows sub-CFL time steps at grid cells where the reaction is active, and delivers an estimate for the truncation error $\tau_\mathbf{i}$ which is used in (6). We use this estimate to tag cells for refinement which, in turn, results in a reduced CFL time step, Δt, so that generally at most 2 or 3 sub-CFL steps are taken for any grid cell.

The hydrodynamic step in (13), $\mathbf{U}^*_\mathbf{i} = S_f(\Delta t)\tilde{\mathbf{U}}_\mathbf{i}$ say, involves the convective terms in (11). This step is performed using the conservative scheme

$$\begin{aligned} \mathbf{U}^*_\mathbf{i} = \tilde{\mathbf{U}}_\mathbf{i} &- \frac{\Delta t}{J_\mathbf{i} \Delta r_1} \left(\hat{\mathbf{F}}_{1, i_1+1/2, i_2, i_3} - \hat{\mathbf{F}}_{1, i_1-1/2, i_2, i_3} \right) \\ &- \frac{\Delta t}{J_\mathbf{i} \Delta r_2} \left(\hat{\mathbf{F}}_{2, i_1, i_2+1/2, i_3} - \hat{\mathbf{F}}_{2, i_1, i_2-1/2, i_3} \right) - \frac{\Delta t}{J_\mathbf{i} \Delta r_3} \left(\hat{\mathbf{F}}_{3, i_1, i_2, i_3+1/2} - \hat{\mathbf{F}}_{3, i_1, i_2, i_3-1/2} \right). \end{aligned} \tag{16}$$

The numerical flux functions in (16) are calculated using a second-order, slope-limited, Godunov method with an approximate Roe Riemann solver. Full details of the flux calculations are given in [2]. It is worth noting that for the case of a Cartesian grid, the metrics of the mapping simplify and we exploit this in the various formulas for the Godunov scheme to reduce computational cost and memory usage.

A global time step Δt is used for all component grids, including refinement grids, and is determined by

$$\Delta t = \sigma_{\text{CFL}} \min_{1 \le g \le \mathcal{N}_{\text{grid}}} \Delta t_g, \tag{17}$$

where σ_{CFL} is a constant taken to be 0.8 in our calculations and Δt_g is the time step suitable for grid g. This time step is determined from an analysis of the real and imaginary parts of the time-stepping eigenvalue (see [2]). For the Euler equations the dominant term comes from the imaginary part of the eigenvalue so that Δt_g is essentially governed by a CFL stability constraint for the numerical solution on grid g.

6. Numerical Results

We now present numerical results for various initial-boundary-value problems (IBVPs) of the type given in (1). We begin in Section 6.1 with results for the advection-diffusion equation in (2). For this relatively simple equation, we perform a careful validation study of the parallel numerical approach on overlapping grids with AMR. This is done by first constructing exact solutions for a number of test problems. We then check the accuracy of the numerical solutions for these test problems for various choices of grids, refinement parameters, and number of processors. Once the accuracy of the numerical approach is established for the equations in (2), we then consider results for the more difficult reactive Euler equations in (3). In Section 6.2, we consider a non-reactive problem involving planar shock diffraction by a rigid sphere in a rectangular channel. For this problem, we compute the solution for a sequence of overlapping grids with increasing grid resolution to verify convergence. Since the problem is axisymmetric in the neighborhood of the sphere, we also check the accuracy of the solution by comparing the results of fully three-dimensional calculations with corresponding results given by a highly resolved axisymmetric calculation. It is also of interest to examine the parallel scalability of the numerical implementation, and this is done for the shock-diffraction problem. Finally, in Section 6.3, we illustrate the numerical approach for a complex reactive flow problem involving detonation initiation in a T-shaped pipe.

6.1. *Test problems for an advection-diffusion equation*

We first consider our numerical approach for various test problems involving the advection-diffusion equation in (2). For these problems, we construct exact solutions of the equation using the method of analytic solutions so that we may later check our numerical solutions with these exact solutions. For example, consider the IBVP given in (2) and a choice for a smooth function $\bar{u}(\mathbf{x}, t)$. For a chosen domain Ω, the function $\bar{u}(\mathbf{x}, t)$ is an exact solution of the IBVP if we set

$$f(\mathbf{x}, t) = \bar{u}_t + \mathbf{a} \cdot \nabla \bar{u} - \nu \Delta \bar{u}, \qquad u_0(\mathbf{x}) = \bar{u}(\mathbf{x}, 0),$$

and

$$g(\mathbf{x}, t) = \bar{u}(\mathbf{x}, t), \qquad \text{for } \mathbf{x} \in \partial\Omega.$$

In our numerical implementation, we have a number of choices available for $\bar{u}$, including polynomials, trigonometric functions, and exponential functions, among others. For the purposes of this paper, we consider two choices. The first choice is a polynomial of degree 2 in space and degree 1 in time given by

$$\bar{u}(\mathbf{x}, t) = \sum_{i=0}^{2} \sum_{j=0}^{2} \sum_{k=0}^{2} \sum_{l=0}^{1} b_{i,j,k,l} x^i y^j z^k t^l, \tag{18}$$

where b_{ijkl} are the coefficients of the polynomial and $\mathbf{x} = (x, y, z)$. The second choice is a translating pulse given by

$$\bar{u}(\mathbf{x}, t) = c_0 \exp\left\{ - \left(|\mathbf{x} - \mathbf{x}_c(t)| / c_1 \right)^2 \right\}, \tag{19}$$

where c_0 and c_1 are parameters, and $\mathbf{x}_c(t) = \mathbf{x}_0 + \mathbf{v}_0 t$ gives the position of the center of the pulse at a time t. Here, $\mathbf{x}_0$ is the position of the center of the pulse at $t = 0$ and $\mathbf{v}_0$ is its constant velocity.

We note that the spatial discretization of the advection-diffusion equation is exact (to within round-off error) on Cartesian grids (or rotated Cartesian grids) for the case when $\bar{u}(\mathbf{x}, t)$ is given by the polynomial in (18). The Runge-Kutta time-stepping algorithm, either second or fourth order, is exact for polynomials of degree 1 in time at most. This is due to the fact that the boundary conditions on the intermediate Runge-Kutta stages are given by $\bar{u}(\mathbf{x}, t)$ exactly, but the numerical intermediate stage solutions may be only first-order accurate [48]. While it is very useful to consider a test function whose numerical solution should be exact for Cartesian grids, the polynomial does not give an interesting distribution of refinement grids for the purpose of testing the AMR implementation. We address this issue by considering an auxiliary function given by

$$\mathcal{H}(\mathbf{x},t) = \begin{cases} 1 & \text{if } |\mathbf{x}-\mathbf{x}_c(t)| \le 1, \\ 0 & \text{if } |\mathbf{x}-\mathbf{x}_c(t)| > 1, \end{cases}$$

where $\mathbf{x}_c(t)$ has the same form as that used in (19), and use this sharp hat function only for the error estimate in (6) and (7) to generate the refinement grids, instead of using the smooth polynomial. We refer to this combined test function as the *poly-hat* solution. Finally, we note that the discretization error for the translating pulse solution in (19) is not zero, but should converge with second-order accuracy.

We now consider a series of numerical tests for the advection-diffusion equation in (2) with $\mathbf{a} = (1,1,1)$ and $\nu = .01$, and for two choices for the domain Ω represented by two basic overlapping grids. All of the numerical calculations for these tests are performed in parallel using the modified bin-packing load balancer as described in Section 4.5.

6.1.1. *Tests using a box-in-a-box grid*

For the first series of calculations, we introduce a box grid $\mathcal{B}$ defined by

$$\begin{aligned} \mathcal{B}\left([x_a,x_b]\times[y_a,y_b]\times[z_a,z_b],N_1,N_2,N_3\right) = \big\{ &(x_a + i_1\Delta x,\ y_a + i_2\Delta y,\ z_a + i_3\Delta z) \ \big| \\ &\Delta x = (x_b-x_a)/N_1,\ \Delta y = (y_b-y_a)/N_2,\ \Delta z = (z_b-z_a)/N_3,\ i_\alpha = 0,1,\ldots,N_\alpha,\ \alpha = 1,2,3\big\}\ . \end{aligned} \tag{20}$$

We also consider a rotated-box grid, $\mathcal{R}([x_a,x_b]\times[y_a,y_b]\times[z_a,z_b],N_1,N_2,N_3,\theta_x,\theta_y,\theta_z)$, which is obtained from $\mathcal{B}$ by rotating it through an angle θ_x about the x-axis, followed by a rotation through an angle θ_y about the y-axis, and finally by a rotation through an angle θ_z about the z-axis. We now consider an overlapping grid, $\mathcal{G}_{\rm b}^{(j,\ell)}$, for a domain Ω given by $\mathbf{x}\in[-1,1]^3$, where the integer $j \ge 1$ determines the mesh spacing on the base level and ℓ gives the maximum number of refinement levels allowed (with $\ell = 0$ being the base level). For $\ell = 0$, this overlapping grid consists of a rotated-box grid embedded in a box grid and is defined by

$$\mathcal{G}_{\rm b}^{(j,0)} = \mathcal{B}([-1,1]^3,20j,20j,20j)\ \cup\ \mathcal{R}([-.4,.4]^3,8j,8j,8j,\pi/4,\pi/4,\pi/4). \tag{21}$$

The mesh spacings on the base level are equal in all directions and given by $h_{j,0} = 1/(10j)$. The grid is generated by the overlapping-grid generator [38] which determines the portion of the box grid $\mathcal{B}$ that is cut away by the rotated-box grid $\mathcal{R}$, and determines the corresponding interpolation points that connect the numerical solution on each component grid across the overlap. We note that the overlapping grid given by (21) is a useful choice since it involves a simple (non-rotated) Cartesian grid whose numerical treatment is handled by our computational kernels that have been optimized for Cartesian grids and a rotated-box grid which is handled by our computational kernels for general curvilinear grids. Thus, both the optimized and general kernels are tested for this choice.

Table 1 shows results for the poly-hat solution computed on the box-in-a-box grid, $\mathcal{G}_{\rm b}^{(j,\ell)}$, defined in (21) for the base level. The coefficients of the polynomial in the poly-hat solution are given by

$$b_{i,0,0,l} = b_{0,j,0,l} = b_{0,0,k,l} = 1, \qquad \text{for } i,j,k = 0,1,2 \text{ and } l = 0,1,$$

and $b_{i,j,k,l} = 0$ otherwise, and the initial position and the constant velocity which specify $\mathbf{x}_c(t)$ in the hat function are given by $\mathbf{x}_0 = (-.25,-.25,-.25)$ and $\mathbf{v}_0 = (1,1,1)$, respectively. For each case, the equations are integrated numerically to $t = .5$ using a second-order Runge-Kutta time-stepper (RK2), and the numerical solution is compared to the exact poly-hat solution. Adaptive mesh refinement is used for each run, and the results are shown for AMR with refinement grids up to level ℓ so that the mesh spacings on the finest level is given by $h_{j,\ell} = h_{j,0}/n_r^\ell$, where n_r is the refinement ratio, taken to be 2 or 4 for all calculations. The table gives the number of processors used for each calculation, $N_{\rm proc}$, the number of time steps taken, $N_{\rm step}$, and the number of times the AMR grid was recomputed $N_{\rm regrid}$. There is also information provided concerning the (min,max) number of grids, $\mathcal{N}_{\rm grid}$, used during the calculations, which includes grids on the base level and all refinement grids, and the average number of discretization points, $\mathcal{N}_{\rm point}$. Finally, the maximum error, $\mathcal{E}_{j,\ell}$, at $t = .5$ is computed and displayed in the table for each calculation. For this first test, the aim is to check the error in the numerical solution for a variety of choices for (j,ℓ), n_r and $N_{\rm proc}$. We observe

Grid	n_r	N_{proc}	N_{step}	N_{regrid}	$\mathcal{N}_{\text{grid}}$	$\mathcal{N}_{\text{point}}$	$\mathcal{E}_{j,\ell}$
$\mathcal{G}_{\text{b}}^{(2,1)}$	2	4	150	38	(2, 12)	1.5e+5	9.24e−14
$\mathcal{G}_{\text{b}}^{(2,2)}$	2	2	476	120	(3, 34)	2.9e+5	2.47e−13
$\mathcal{G}_{\text{b}}^{(2,1)}$	4	8	477	60	(2, 10)	3.7e+5	2.36e−13
$\mathcal{G}_{\text{b}}^{(4,1)}$	2	2	33	9	(2, 29)	7.0e+5	2.40e−14
$\mathcal{G}_{\text{b}}^{(4,1)}$	2	4	164	41	(2, 36)	7.8e+5	4.62e−14
$\mathcal{G}_{\text{b}}^{(4,1)}$	2	8	164	41	(2, 36)	7.8e+5	4.62e−14

Table 1
Parallel AMR results for a variety of runs involving a box-in-a-box overlapping grid with the poly-hat solution. $\mathcal{G}_{\text{b}}^{(j,l)}$ denotes a grid with a resolution factor j and l refinement levels. The numerical errors, $\mathcal{E}_{j,\ell}$, are of the order of the round-off error for each run.

that the errors are all very small, of the order of the round-off error for 64-bit double-precision arithmetic, which agrees with the expected result for the poly-hat solution.

For our next test, we consider the same domain and overlapping grid as before, but now use a solution given by the moving pulse function defined in (19). For this function we take $c_0 = 1$, $c_1 = .15$, $\mathbf{x}_0 = (-.25, -.25, -.25)$ and $\mathbf{v}_0 = (1, 1, 1)$. As before, we integrate the advection-diffusion equation numerically using RK2 and measure the error in the numerical solution at $t = .5$. The results are displayed in Table 2. For this test, a range of values for N_{proc} are used, but the values for (j, ℓ) and n_r are chosen so that the effective resolution given by $h_{j,\ell}$ is the same for each run. Since the effective resolution is the same for each run, the numerical errors given by $\mathcal{E}_{j,\ell}$ should be approximately equal. The maximum errors shown in the table verify this expectation.

Grid	n_r	N_{proc}	N_{step}	N_{regrid}	$\mathcal{N}_{\text{grid}}$	$\mathcal{N}_{\text{point}}$	$\mathcal{E}_{j,\ell}$
$\mathcal{G}_{\text{b}}^{(2,1)}$	4	8	245	31	(2, 37)	2.2e+6	3.22e−3
$\mathcal{G}_{\text{b}}^{(2,2)}$	2	2	245	63	(9, 100)	2.1e+6	3.22e−3
$\mathcal{G}_{\text{b}}^{(2,2)}$	2	4	245	63	(9, 100)	2.1e+6	3.22e−3
$\mathcal{G}_{\text{b}}^{(4,1)}$	2	8	245	62	(2, 129)	2.3e+6	3.21e−3
$\mathcal{G}_{\text{b}}^{(8,0)}$	–	16	245	–	(2, 2)	4.8e+6	3.21e−3

Table 2
Parallel AMR results for a variety of runs involving the box-in-a-box overlapping grid with the moving pulse solution. The effective resolution is the same for each run and the numerical errors, $\mathcal{E}_{j,\ell}$, are found to be approximately equal.

6.1.2. *Tests using a sphere-in-a-box grid*

The sphere-in-a-box grid is an overlapping grid for the region exterior to a sphere of radius $\frac{1}{2}$ and inside the cube $[-2, 2]^3$. For this grid, we use a box grid to describe the boundary of the cube as well as the bulk of the interior of the region, and then consider various curvilinear boundary-fitted grids to describe the boundary of the sphere. For the latter, a simple choice would be a single boundary-fitted grid based on a mapping using spherical-polar coordinates, i.e.

$$\begin{aligned}\mathcal{S}\left([\rho_a,\rho_b]\times[\theta_a,\theta_b]\times[\phi_a,\phi_b],N_1,N_2,N_3\right) = \big\{&(\rho_{i_1}\cos\theta_{i_2}\sin\phi_{i_3},\ \rho_{i_1}\sin\theta_{i_2}\sin\phi_{i_3},\ \rho_{i_1}\cos\phi_{i_3})\ \big|\\ &\rho_{i_1}=\rho_a+i_1(\rho_b-\rho_a)/N_1,\ \theta_{i_2}=\theta_a+i_2(\theta_b-\theta_a)/N_2,\ \phi_{i_3}=\phi_a+i_3(\phi_b-\phi_a)/N_3,\\ &i_\alpha=0,1,\ldots,N_\alpha,\ \alpha=1,2,3\big\}\ .\end{aligned} \tag{22}$$

with limits taken as $\rho_b > \rho_a = \frac{1}{2}$, $\theta_a = 0$, $\theta_b = 2\pi$, $\phi_a = 0$ and $\phi_b = \pi$. The difficulty with this simple choice is that it has coordinate singularities at $\phi = 0$ (north pole) and $\phi = \pi$ (south pole). To avoid these singularities, we employ boundary-fitted grids, one centered about each pole, based on orthographic projections of rectangular grid patches onto the sphere. These grids may be defined by first introducing an orthographic transform, $\mathbf{O}_p$, given by

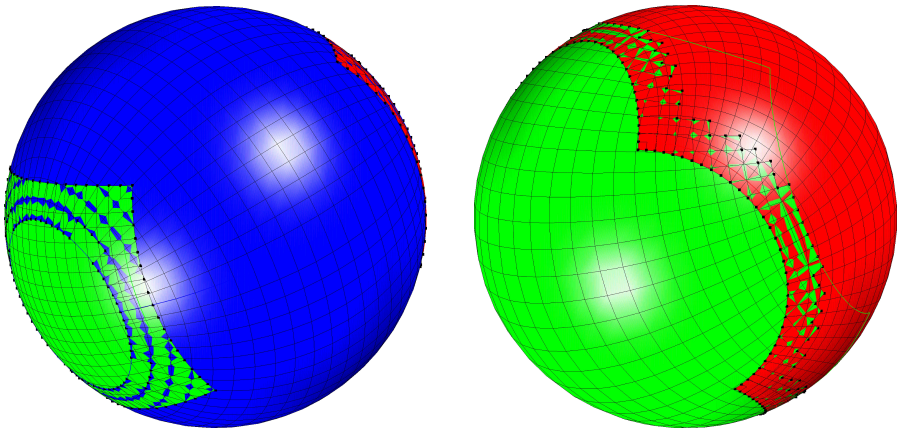

Fig. 6. Sample overlapping-grid constructions for a spherical shell. Left: an overlapping grid consisting of $\mathcal{S}([.5,1.]\times[0,2\pi]\times[.15\pi,.85\pi],5,64,22)$ and $\mathcal{O}_{\pm1}([.5,1.],.6,.6,5,16,16)$. Right: an overlapping grid consisting of $\mathcal{O}_{\pm1}([.5,1.],2.1,2.1,5,31,31)$.

$$\mathbf{x}=\mathbf{O}_p\left(\mathbf{r};\ [\rho_a,\rho_b],\hat{s}_2,\hat{s}_3\right)\equiv\left(p\frac{(1-\sigma^2)\rho}{1+\sigma^2},\ \frac{2\rho s_2}{1+\sigma^2},\ p\frac{2\rho s_3}{1+\sigma^2}\right),$$

where ρ, s_2, s_3 and σ are given in terms of $\mathbf{r}=(r_1,r_2,r_3)\in[0,1]^3$ by

$$\rho=\rho_a+r_1(\rho_b-\rho_a),\quad s_2=\left(r_2-\frac{1}{2}\right)\hat{s}_2,\quad s_3=\left(r_3-\frac{1}{2}\right)\hat{s}_3,\quad \sigma^2=s_2^2+s_3^2,$$

and $p=+1$ for the transformation near the north pole and $p=-1$ for the transformation near the south pole. The parameters $[\rho_a,\rho_b]$ specify the radial extent of the region, while $\hat{s}_2$ and $\hat{s}_3$ determine its lateral extent. The orthographic grid, $\mathcal{O}_p$ centered about pole p, is now defined as

$$\mathcal{O}_p\left([\rho_a,\rho_b],\hat{s}_2,\hat{s}_3,N_1,N_2,N_3\right)=\left\{\mathbf{x_i}\ \middle|\ \mathbf{x_i}=\mathbf{O}_p\left(\mathbf{r_i};\ [\rho_a,\rho_b],\hat{s}_2,\hat{s}_3\right),\ i_\alpha=0,1,\ldots,N_\alpha,\ \alpha=1,2,3\right\}\ . \quad (23)$$

Using the boundary-fitted grids defined in (22) and (23), we may represent a spherical shell for $.5\le\rho\le1$ as shown in Figure 6. Two methods of construction are shown in the figure. On the left, we use a spherical-polar grid given by (22) for the region near the equator ($\phi=\pi/2$) and two orthographic grids given by (23) with $p=\pm1$ for the regions near the poles. We note that the grid points on the spherical-polar grid near the singularities at the poles are removed by the orthographic grids in this overlapping-grid construction. If the extent of the two orthographic grids is increased, it is possible to represent the entire spherical shell without the spherical-polar grid as shown on the right in the figure. An advantage of this latter approach is that one less component grid is used for the overlapping grid, while a disadvantage is that the distortion of the grid cells near the equator is larger. The construction on the left is the basis for the overlapping grid used in the next section for the problem of shock diffraction by a sphere. For the present tests, we use the construction on the right, and define the sphere-in-a-box grid on the base level $\ell=0$ as

$$\mathcal{G}_{\rm s}^{(j,0)}=\mathcal{B}\left([-2,2]^3,40j,40j,40j\right)\ \cup\ \mathcal{O}_{\pm1}\left([.5,.9],2.1,2.1,4j,N(j),N(j)\right).$$

where $N(j)=\lfloor 22.4j+0.5\rfloor$. As before, the integer j specifies the grid resolution on the base level. The sphere-in-a-box grid includes two non-trivial curvilinear component grids and therefore provides an additional level of complexity to test the numerical implementation.

Table 3 presents results for a pulse function moving in a domain Ω represented by a sequence of sphere-in-a-box grids with increasing grid resolution. The pulse function is given by (19) with $c_0=1$, $c_1=.2$, $\mathbf{x}_0=(-.5,-1.25,-.5)$ and $\mathbf{v}_0=(1,1,1)$. For each case, one level of refinement grids is used with $n_r=2$, and the equations are integrated to $t=.25$ using a fourth-order Runge-Kutta time-stepper, RK4. (Nearly identical results are obtained using RK2.) The problem is solved using sphere-in-a-box grids, $\mathcal{G}_{\rm s}^{(j,1)}$, $j=1$, 2 and 3, with representative mesh spacings on the finest level given by $h_{j,1}=1/(20j)$. Thus, the effective resolution increases with j, and our aim in this test is to check whether the numerical solution converges

at the correct rate which should be second-order for our spatial discretization. For each j, we compute the maximum error at $t = .25$, and then perform a least squares fit to the formula $\mathcal{E}_{j,1} = C(h_{j,1})^{\mu}$, where C is a constant and μ is the rate of convergence. For the errors reported in the table, we find that $\mu = 2.0$ which shows that the approximation is second-order accurate.

Grid	n_r	N_{proc}	N_{step}	N_{regrid}	$\mathcal{N}_{\text{grid}}$	$\mathcal{N}_{\text{point}}$	$\mathcal{E}_{j,1}$
$\mathcal{G}_{\text{s}}^{(1,1)}$	2	32	48	24	(3, 23)	2.0e+5	2.84e−2
$\mathcal{G}_{\text{s}}^{(2,1)}$	2	32	120	60	(3, 49)	1.1e+6	6.91e−3
$\mathcal{G}_{\text{s}}^{(3,1)}$	2	32	376	188	(3, 128)	6.7e+6	1.70e−3

Table 3
Parallel AMR results for runs involving the sphere-in-a-box grid with the moving pulse solution. The mesh spacing on the finest level decreases by a factor of 2 while the maximum error, $\mathcal{E}_{j,\ell}$, decreases by a factor of approximately 4 thus demonstrating second-order accuracy.

As a final test, we compute numerical solutions to the advection-diffusion equation using the sphere-in-a-box grid for a range of values for (j, ℓ) and n_r such that the effective resolution is held fixed. The exact solution is given by the pulse function with the same choice of parameters used for the previous test case. For each run, we compute the numerical error, $\mathcal{E}_{j,\ell}$, and display the results in Table 4. Numerical solutions are computed for a range of values for N_{proc}, and, as expected, the maximum errors in the computed solution are approximately equal in all cases.

Grid	n_r	N_{proc}	N_{step}	N_{regrid}	$\mathcal{N}_{\text{grid}}$	$\mathcal{N}_{\text{point}}$	$\mathcal{E}_{j,\ell}$
$\mathcal{G}_{\text{s}}^{(1,2)}$	2	8	126	64	(13, 53)	6.0e+5	7.25e−3
$\mathcal{G}_{\text{s}}^{(1,2)}$	2	32	126	64	(13, 53)	6.0e+5	7.25e−3
$\mathcal{G}_{\text{s}}^{(1,1)}$	4	16	187	47	(3, 21)	6.6e+5	7.25e−3
$\mathcal{G}_{\text{s}}^{(1,1)}$	4	32	187	47	(3, 21)	6.6e+5	7.25e−3
$\mathcal{G}_{\text{s}}^{(2,1)}$	2	1	120	60	(3, 49)	1.1e+6	6.91e−3
$\mathcal{G}_{\text{s}}^{(2,1)}$	2	32	120	60	(3, 49)	1.1e+6	6.91e−3
$\mathcal{G}_{\text{s}}^{(4,0)}$	–	8	166	–	(3, 3)	4.9e+6	6.76e−3
$\mathcal{G}_{\text{s}}^{(4,0)}$	–	32	166	–	(3, 3)	4.9e+6	6.76e−3

Table 4
Parallel AMR results for runs involving the sphere-in-a-box grid with the moving pulse solution. The effective resolution is the same for all runs and we observe that the numerical errors, $\mathcal{E}_{j,\ell}$, are approximately equal.

6.2. *Shock diffraction by a sphere*

Having established the accuracy of the numerical implementation for the advection-diffusion equation, we now consider more complex problems involving the reactive Euler equations. For a first problem, we consider planar shock diffraction by a solid sphere. We assume that the state of the non-reactive flow ahead of the shock, initially at $x_1 = -1.5$, is at rest with ambient density, pressure and sound speed given by

$$\rho_0 = 1, \qquad p_0 = 0.7143, \qquad a_0 = 1,$$

respectively, assuming an ideal gas with $\gamma = 1.4$. The state of the flow behind the shock is given by

$$\rho = 2.667, \qquad v_1 = 1.25, \qquad v_2 = v_3 = 0, \qquad p = 3.214, \qquad a = 1.299,$$

so that the Mach number of the shock is $M_0 = U/a_0 = 2$, where U is the velocity of the shock. The radius of the sphere is taken to be 1 and its center is at the origin, $x_1 = x_2 = x_3 = 0$. The flow is considered in a channel with a square cross section for $|x_2| \le 2.5$ and $|x_3| \le 2.5$, but for numerical convenience we compute the flow in the portion of the channel with $x_2 \ge 0$ and $x_3 \ge 0$, and use symmetry boundary conditions on

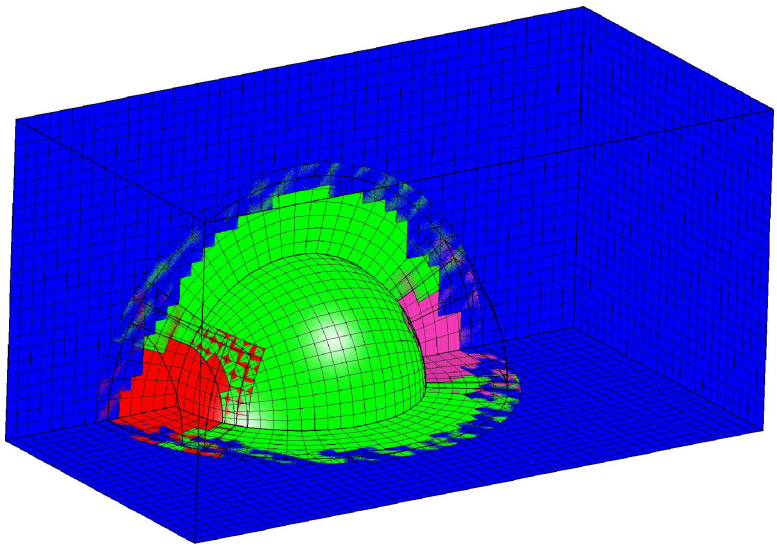

Fig. 7. Overlapping grid $\mathcal{G}_{\rm q}^{(1,0)}$ for the quarter-sphere problem. The Cartesian grid is blue, the spherical-polar grid is green, and the orthographic grids are red and magenta.

the planes $x_2 = 0$ and $x_3 = 0$. Slip-wall boundary conditions are used on the walls of the channel and on the surface of the sphere.

The base grid for the full channel with cross section $|x_2, x_3| \le 2.5$ is similar to the sphere-in-a-box grid used in the previous section, but the sphere is represented by a spherical-polar grid near the equator at $x_1 = 0$ and two orthographic grids at the poles as in the overlapping-grid construction on the left in Figure 6. This full-channel grid covering the rectangular channel for $-2.5 \le x_1 \le 2.5$ is defined as

$$\begin{aligned}\mathcal{G}_{\rm c}^{(j,0)} = \mathcal{B}\left([-2.5,2.5]^3, 50j, 50j, 50j\right) \;&\cup\; \mathcal{S}\left([1, 1+6h_{j,0}] \times [0,2\pi] \times [.15\pi, .85\pi], 6, N_\theta(j), N_\phi(j)\right) \\ &\cup\; \mathcal{O}_{\pm 1}\left([1, 1+6h_{j,0}], .6, .6, 6, N_0(j), N_0(j)\right),\end{aligned}$$

where $h_{j,0} = 1/(10j)$, $N_\theta(j) = \lfloor 20\pi j + 0.5 \rfloor$, $N_\phi(j) = \lfloor 7\pi j + 0.5 \rfloor$ and $N_0(j) = \lfloor 4\pi j + 0.5 \rfloor$. As mentioned earlier, all calculations are performed on a quarter of this grid in the region $-2.5 \le x_1 \le 2.5$ and $0 \le x_2, x_3 \le 2.5$. This quarter-sphere grid, which we denote by $\mathcal{G}_{\rm q}^{(j,\ell)}$, where ℓ specifies the number of refinement levels used, is shown in Figure 7 for the case $j = 1$ and $\ell = 0$.

6.2.1. *Solution behavior and accuracy*

As a fine-grid calculation, we compute the flow in the channel using the overlapping grid $\mathcal{G}_{\rm q}^{(4,2)}$ with $n_r = 2$ for times $t = 0$ to 1.8. At $t = 0$ there are 4 grids at the base level and 2 refinement grids covering the initial planar shock. Later in the calculation as many as 1827 refinement grids are used with a maximum of 55 million grid points. The calculation is performed in parallel using 32 processors. The bin-packing algorithm described in Section 4.5 is used to balance the workload. For this algorithm the maximum imbalance defined in (9) satisfies $\mathcal{I} \le \mathcal{I}_T$, where the target imbalance is taken to be $\mathcal{I}_T = 0.1$. The average and maximum values for $\mathcal{I}$ recorded during the calculation are found to be 0.005 and 0.099, respectively. These values are representative of all parallel calculations in this paper.

Figure 8 shows shaded contours of density on the symmetry planes $x_2 = 0$ and $x_3 = 0$ for the flow at $t = 0.6$, 1.0, 1.4 and 1.8. The initial impact of the shock at the nose of the sphere generates a reflected shock which travels back into the flow entrained by the incident shock. The reflection at the surface of the sphere is regular at first, but then transitions to Mach reflection as the shock travels around the spherical surface. By $t = 0.6$ (top-left view in the figure), the transition to Mach reflection has just occurred, and a Mach stem-like shock and associated triple-point is shown clearly in the two symmetry planes. As the incident shock continues down the channel, the reflected shock and Mach stem continue to grow. A region of high density forms near the nose of the sphere due to the reflection there, and a region of low density forms near

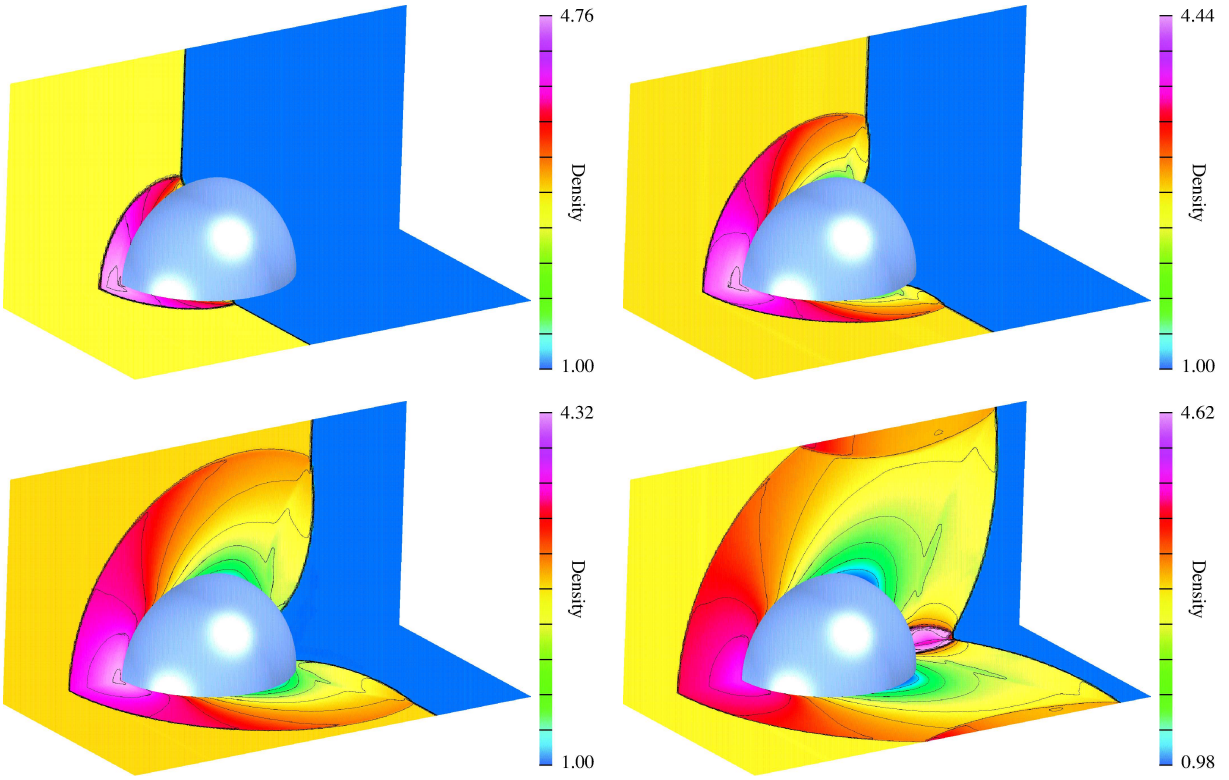

Fig. 8. Shaded contours of density for the quarter-sphere problem at $t = 0.6$ (top left), 1.0 (top right), 1.4 (bottom left) and 1.8 (bottom right) using $\mathcal{G}_{\rm q}^{(4,2)}$ with a refinement ratio $n_r = 2$.

the back of sphere (see the lower-left plot at $t = 1.4$) due to the diffraction of the Mach stem. By $t = 1.8$ (bottom-right view) the Mach stem has traveled around the sphere and has converged near the back creating a very high density region. The reflected shock created by the initial impact has grown and has reflected off the channel walls at $x_2 = 2.5$ and $x_3 = 2.5$. The four shaded contour plots show the expected symmetry of the solution even though no such symmetry is assumed in this fully three-dimensional treatment of the equations.

As mentioned previously, the flow is computed using the overlapping grid $\mathcal{G}_{\rm q}^{(4,2)}$ with $n_r = 2$ so that up to 2 refinement-grid levels are used. Figure 9 provides representative views of the refinement-grid structure at $t = 0.6$ and $t = 1.4$. The plot on the left shows the grid structure at $t = 0.6$. Here, we note that the incident shock and the shock reflected from the sphere are represented by grids at the highest refinement level. There is a small planar disturbance in the flow (in the $v_1 - a$ characteristic field behind the incident shock) created by the discontinuous initial state, and this has triggered refinement at the first refinement-grid level that appears during early times of the calculation. At $t = 1.4$ (right plot), the reflected shock has propagated well away from the sphere, and the diffracted Mach stem is well developed. Both of these features, as well as the remains of the incident shock, are represented by grids at the highest refinement level. There is also a contact surface behind the Mach stem that emerges from the junction of the incident shock, reflected shock, and Mach stem, and this feature seen in the plot at $t = 1.4$ is represented by grids at the highest refinement level. The plots in Figure 9 indicate an effective use of AMR to accurately compute the flow.

In order to assess the accuracy of the numerical solution, we plot the solution at $t = 1.4$ computed using overlapping grids with increasing resolution at the finest level. The top two solutions in Figure 10 are computed using $\mathcal{G}_{\rm q}^{(2,\ell)}$, $\ell = 0$ and 1 (with $n_r = 2$ for $\ell = 1$) so that the effective mesh spacings are 1/20 and 1/40, respectively. The bottom two solutions in the figure are computed using $\mathcal{G}_{\rm q}^{(4,\ell)}$, $\ell = 1$ and 2 (both with $n_r = 2$) so that the effective mesh spacings for these two solutions are 1/80 and 1/160, respectively. The qualitative behavior of the solution agrees in all four plots, but sharp features of the solution, such as

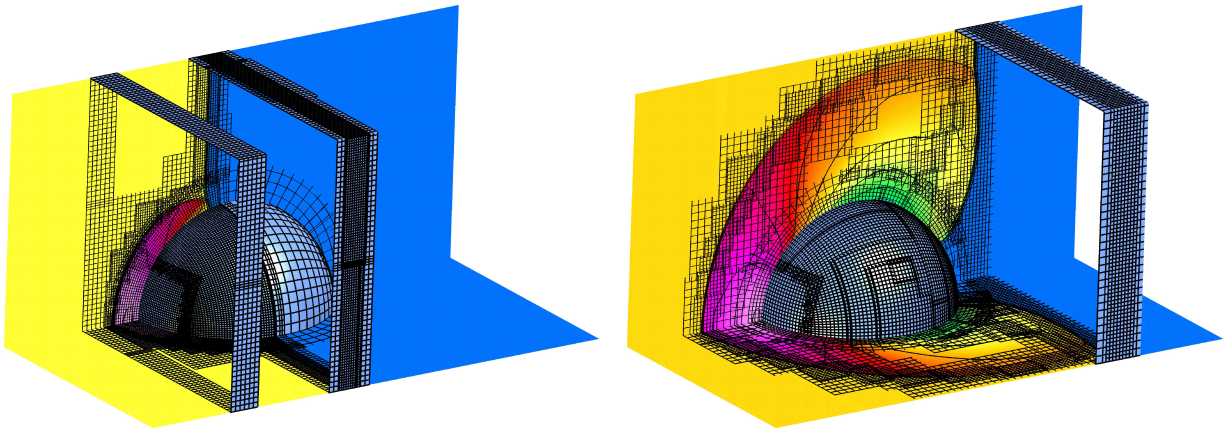

Fig. 9. Shaded contours of density for the quarter-sphere problem at $t = 0.6$ (left) and $t = 1.4$ (right) along with the corresponding refinement grid structure. (The grid is coarsened by a factor of 4 for illustrative purposes.)

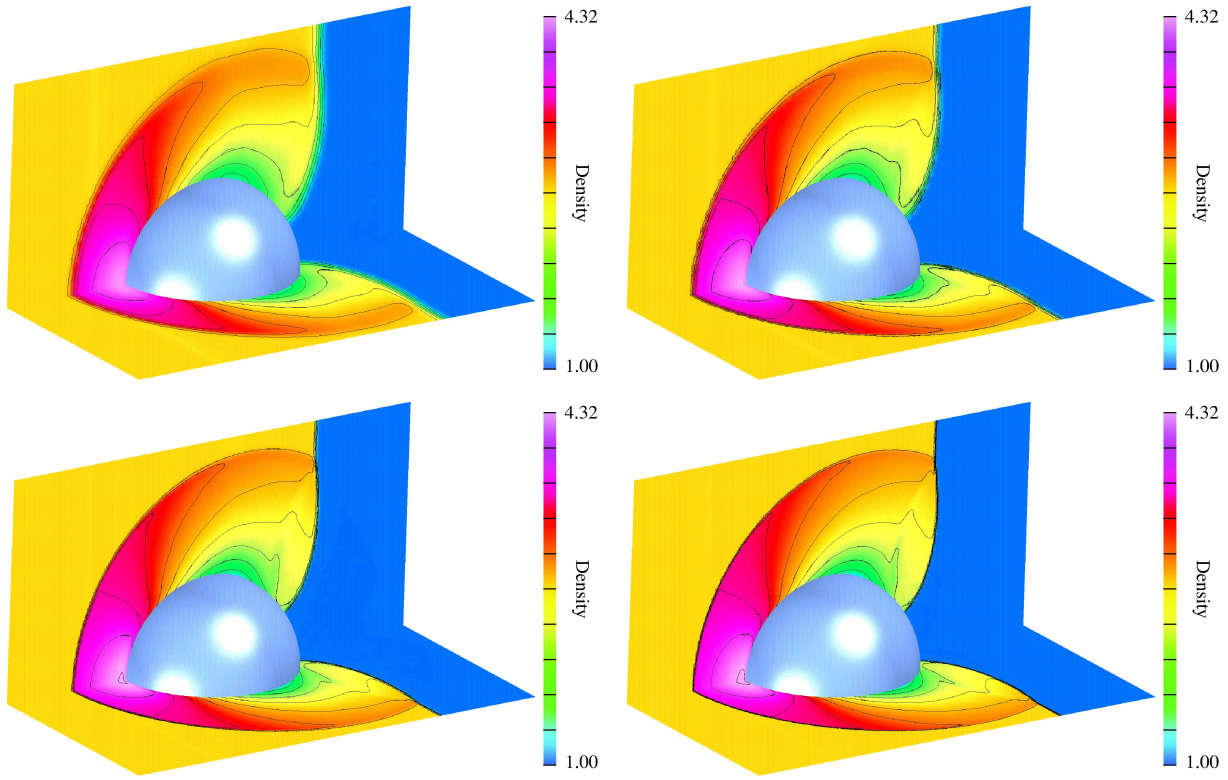

Fig. 10. Shaded contours of density at $t = 1.4$ using overlapping grids $\mathcal{G}_{\rm q}^{(2,0)}$ ($h = 1/20$, top left), $\mathcal{G}_{\rm q}^{(2,1)}$ ($h = 1/40$, top right), $\mathcal{G}_{\rm q}^{(4,1)}$ ($h = 1/80$, bottom left), and $\mathcal{G}_{\rm q}^{(4,2)}$ ($h = 1/160$, bottom right).

contacts and shocks, improve significantly with increasing grid resolution. Of particular note is the contact that appears in the flow behind the Mach stem. There is a hint of this contact in the solution on the coarsest grid, $\mathcal{G}_{\rm q}^{(2,0)}$, but it is not well resolved until the solution on the finest grid, $\mathcal{G}_{\rm q}^{(4,2)}$.

A further measure of grid convergence is shown in Figure 11. In this plot, we show the behavior of the density on the surface of the sphere in the plane $x_3 = 0$ at times $t = 0.3, 0.5, \ldots, 1.3$ from the numerical solution using overlapping grids $\mathcal{G}_{\rm q}^{(2,1)}$, $\mathcal{G}_{\rm q}^{(4,1)}$ and $\mathcal{G}_{\rm q}^{(4,2)}$, all with $n_r = 2$. The effective mesh spacings for these grids are 1/40, 1/80 and 1/160, respectively, as noted before. These solutions are compared with a

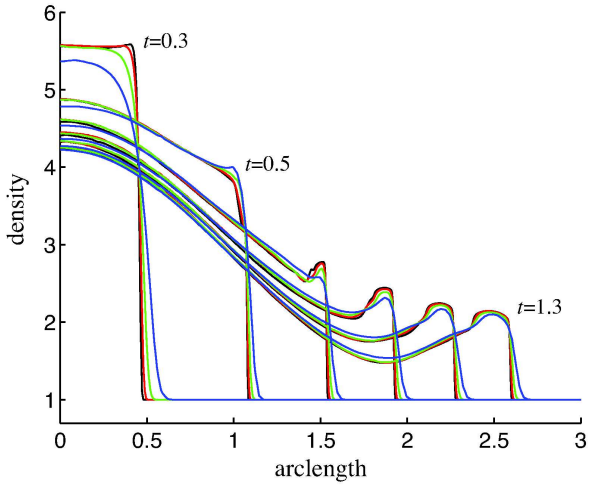

Fig. 11. Behavior of density along the surface of the sphere at $t = 0.3, 0.5, \ldots, 1.3$ using overlapping grids $\mathcal{G}_{\rm q}^{(2,1)}$ ($h = 1/40$, blue curves), $\mathcal{G}_{\rm q}^{(4,1)}$ ($h = 1/80$, green curve), and $\mathcal{G}_{\rm q}^{(4,2)}$ ($h = 1/160$, red curves), all with $n_r = 2$. The black curves are given by an axisymmetric calculation using a overlapping grid with effective mesh spacing equal to 1/320.

highly-resolved axisymmetric solution on a two-dimensional overlapping grid with effective mesh spacing equal to 1/320. The convergence of the density to the axisymmetric solution is seen clearly which provides a good test of the accuracy of the fully three-dimensional calculations. In addition, it is found that there is a negligible difference in the computed density between that shown in the plane $x_3 = 0$ and other cut planes through the axis of symmetry for $\mathcal{G}_{\rm q}^{(4,2)}$.

6.2.2. *Parallel performance*

The problem of shock diffraction by a sphere provides a good test problem to assess the scalability of the parallel numerical method. We first consider numerical solutions for the case when $\ell = 0$, i.e. no AMR. For each run, we integrate the equations from $t = 0$ to 1.8, as before, and record the number of time steps taken and the total CPU time used (wall clock time). From this information we compute $\mathcal{T}_k$, the average CPU time per step for run k. Ideally, $\mathcal{T}_k$ would be proportional to the number of active points per processor (assuming a perfectly balanced workload and no communication costs), so that the *scaled* CPU time per step given by

$$\mathcal{T}_k^* = \frac{\mathcal{T}_k}{\mathcal{N}_{\rm point}^{(k)}/N_{\rm proc}^{(k)}}$$

would be the same for each run. Here, $\mathcal{N}_{\rm point}^{(k)}$ is number of active points on the overlapping grid and $N_{\rm proc}^{(k)}$ is the number of processors used, both for run k. Generally, the scaled CPU time per step does not behave ideally, and increases as the size of the problem grows due primarily to an increased cost associated with communication between processors. To measure this behavior, we define a *parallel scale factor*

$$\mathcal{S}_k = \frac{\mathcal{T}_0^*}{\mathcal{T}_k^*},$$

which compares the scaled CPU times per step between runs 0 and k. For each set of runs, $k = 0$ is taken to be a reference computation with $N_{\rm proc}^{(0)} = 1$. All calculations in this section are performed on a 26-node Linux cluster with 4 CPU cores and 6 gigabytes of main memory per node, and with a Myrinet communication system.

For our first experiment, we consider the CPU times required to compute the solution on the overlapping grid $\mathcal{G}_{\rm q}^{(4,0)}$ with $N_{\rm proc}^{(k)} = 2^k$ for $k = 0, 1, \ldots, 6$. Table 5 presents the *strong scaling* results of this experiment in which the number of active grid points is held fixed as the number of processors increase. The number of

k	Grid	$\mathcal{N}_{\rm point}^{(k)}$	$N_{\rm proc}^{(k)}$	$\mathcal{N}_{\rm point}^{(k)}/N_{\rm proc}^{(k)}$	$N_{\rm step}^{(k)}$	$\mathcal{T}_k$	$\mathcal{S}_k$
0	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	1	2.01e+6	617	15.2	1.00
1	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	2	1.00e+6	617	7.77	0.98
2	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	4	5.02e+5	617	3.96	0.96
3	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	8	2.51e+5	617	2.09	0.91
4	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	16	1.26e+5	617	1.09	0.87
5	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	32	6.27e+4	617	0.587	0.81
6	$\mathcal{G}_{\rm q}^{(4,0)}$	2.01e+6	64	3.14e+4	617	0.341	0.70

Table 5
Strong scaling results for the calculation of shock diffraction by a sphere with no AMR. The is the CPU time in seconds per step is given by $\mathcal{T}_k$. The parallel scaling factor $\mathcal{S}_k$ should be 1 for perfect parallel scaling.

time steps taken, $N_{\rm step}^{(k)}$, is fixed for each k and the CPU time per step, $\mathcal{T}_k$, is given in seconds. The behavior of the scale factor, $\mathcal{S}_k$, shows the expected result. As the number of processors increase, $\mathcal{S}_k$ decreases. However, the decrease is not very large so that the code scales reasonably well with no AMR. The main reason for the decrease in the scale factors is the cost for communication. This may be seen in the breakdown of the timings for specific parts of the time-stepping algorithm listed in Table 6. As $N_{\rm proc}^{(k)}$ increases and thus the number of grid points per processor decreases, the percentage of time spent computing $\Delta \mathbf{U}_{i,j}^n$ in the Godunov step decreases while the percentage of time spent for interpolation and updating the parallel ghost boundaries, both requiring communication, increases. All of the calculations use the modified bin-packing algorithm for load balancing.

	$N_{\rm proc}^{(k)}=1$		$N_{\rm proc}^{(k)}=4$		$N_{\rm proc}^{(k)}=16$		$N_{\rm proc}^{(k)}=64$	
	$\mathcal{T}_k$	%	$\mathcal{T}_k$	%	$\mathcal{T}_k$	%	$\mathcal{T}_k$	%
compute $\Delta \mathbf{U}_{i,j}^n$	14.0	92.0	3.38	85.4	0.833	76.4	0.216	63.3
interpolation	0.0152	0.1	0.0396	1.0	0.0491	4.5	0.0351	10.3
boundary conditions	0.684	4.5	0.182	4.6	0.0534	4.9	0.0201	5.9
update ghost boundaries	0.0	0.0	0.218	5.5	0.112	10.3	0.0501	14.7
(other)	0.517	3.4	0.139	3.5	0.0425	3.9	0.0198	5.8
total	15.2	100.0	3.96	100.0	1.09	100.0	0.341	100.0

Table 6
Breakdown of the CPU time per step (in seconds) for various parts of the calculation of shock diffraction by a sphere with no AMR using $N_{\rm proc}^{(k)} = 1$, 4, 16 and 64. The time spent in "interpolation" and "update ghost boundaries" increases as the number of processors increases since these functions require parallel communication.

For the next experiment, we consider the *weak* scaling behavior for the calculation of shock diffraction by a sphere with no AMR. This is done by recording the CPU times per step for calculations on overlapping grids $\mathcal{G}_{\rm q}^{(j,0)}$ with $j = 1$, 2, 3 and 4. For each run k, the number of processors is chosen so that the number of active grid points per processor given by $\mathcal{N}_{\rm point}^{(k)}/N_{\rm proc}^{(k)}$ is fixed (approximately). If the cost of communication is approximately proportional to the number of points per processor, then the CPU time per step for each k should be similar and the scale factors should be approximately 1. The results of this experiment are given in Table 7 where we observe that the scaling factors, $\mathcal{S}_k$, are in fact close to 1 for all k. We note that the scaling results are good even though the number of grid points per processor, about 3.5×10^4, is not very large.

For our final experiment, we consider the parallel performance for a set of calculations using AMR. For this study, we consider the CPU times for calculations using the overlapping grid $\mathcal{G}_{\rm q}^{(2,1)}$ which employs one level of refinement grids. The refinement factor, n_r, is taken to be 2 for the AMR calculations in this study, and $n_{\rm regrid}$ is taken to be 8. Table 8 presents strong scaling results in which $N_{\rm proc}^{(k)} = 2^k$, $k = 0, 1, \ldots, 5$ while

k	Grid	$\mathcal{N}^{(k)}_{\text{point}}$	$N^{(k)}_{\text{proc}}$	$\mathcal{N}^{(k)}_{\text{point}}/N^{(k)}_{\text{proc}}$	$N^{(k)}_{\text{step}}$	$\mathcal{T}_k$	$\mathcal{S}_k$
0	$\mathcal{G}^{(1,0)}_{\text{q}}$	3.50e+4	1	3.50e+4	170	0.346	1.00
1	$\mathcal{G}^{(2,0)}_{\text{q}}$	2.61e+5	8	3.26e+4	330	0.373	0.92
2	$\mathcal{G}^{(3,0)}_{\text{q}}$	8.58e+5	24	3.57e+4	473	0.355	0.99
3	$\mathcal{G}^{(4,0)}_{\text{q}}$	2.01e+6	60	3.35e+4	617	0.373	0.89

Table 7
Weak scaling results for the calculation of shock diffraction by a sphere with no AMR. The time per step, $\mathcal{T}_k$, and the parallel scaling factor, $\mathcal{S}_k$, are nearly constant indicating good parallel scaling.

the *average* number of active grid points given by $\mathcal{N}^{(k)}_{\text{point}}$ is held fixed. The average CPU time (in seconds) per time step and the resulting scale factors given by $\mathcal{S}_k$ are listed in the table for each k. Here it is found the that scale factors decrease faster as the number of processors increase as compared to that for calculations without AMR. This is due primarily to the communication needed for the AMR interpolation. As mentioned earlier, our initial focus in the extension of our implementation of AMR on overlapping grids has been accuracy of the numerical method. We recognize that the parallel implementation of AMR interpolation could be more efficient (see Section 4.3.2), and the scaling results in Table 8 verify this. Future developments of the AMR implementation will focus on improved efficiency of the parallel AMR interpolation algorithm.

k	Grid	$\mathcal{N}^{(k)}_{\text{point}}$	$N^{(k)}_{\text{proc}}$	$\mathcal{N}^{(k)}_{\text{point}}/N^{(k)}_{\text{proc}}$	$N^{(k)}_{\text{step}}$	$\mathcal{T}_k$	$\mathcal{S}_k$
0	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	1	1.61e+6	645	11.8	1.00
1	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	2	8.05e+5	645	6.23	0.95
2	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	4	4.02e+5	645	3.23	0.91
3	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	8	2.01e+5	645	1.82	0.81
4	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	16	1.01e+5	645	1.02	0.72
5	$\mathcal{G}^{(2,1)}_{\text{q}}$	1.61e+6	32	5.03e+4	645	0.591	0.62

Table 8
Strong scaling results for the calculation of shock diffraction by a sphere with AMR.

6.3. *Detonation initiation in a T-shaped pipe*

As a final illustration of our numerical approach, we consider a reactive flow problem involving the initiation of a detonation in a domain consisting of two cylinders that intersect to form a T-shaped pipe. The geometry of the pipe is shown in Figure 12. The main section of the pipe is a cylinder with radius equal to 1 and axis of symmetry given by the x_1-axis. At the base level, it is represented by the union of a box grid defined previously in (20) and a boundary-fitted cylindrical grid. The latter grid is defined by

$$\begin{aligned}\mathcal{C}\left([x_a,x_b]\times[r_a,r_b]\times[\theta_a,\theta_b],N_1,N_2,N_3\right) = \big\{&(x,r\cos\theta,r\sin\theta)\ \big|\\ &x = x_a + i_1(x_b-x_a)/N_1,\ r = r_a + i_2(r_b-r_a)/N_2,\ \theta = \theta_a + i_3(\theta_b-\theta_a)/N_3,\\ &i_\alpha = 0,1,\ldots,N_\alpha,\ \alpha = 1,2,3\big\}\ .\end{aligned} \tag{24}$$

Thus, for the main section of pipe, we use

$$\mathcal{B}\left([-2,2]\times[-1,1]\times[-1,1],240,120,120\right)\ \cup\ \mathcal{C}\left([-2,2]\times[1-6h_0,1]\times[0,2\pi],240,6,358\right)$$

where $h_0 = 1/60$ gives a representative mesh spacing for the grid on the base level. The top section of pipe is also given by a cylinder, but with radius equal to 0.7 and axis of symmetry given by the x_2-axis. For this section of pipe, we use

$$\mathcal{B}\left([0,2]\times[-.7,.7]\times[-.7,.7],120,84,48\right)\ \cup\ \mathcal{C}\left([0,2]\times[.7-6h_0,.7]\times[0,2\pi],120,6,245\right)$$

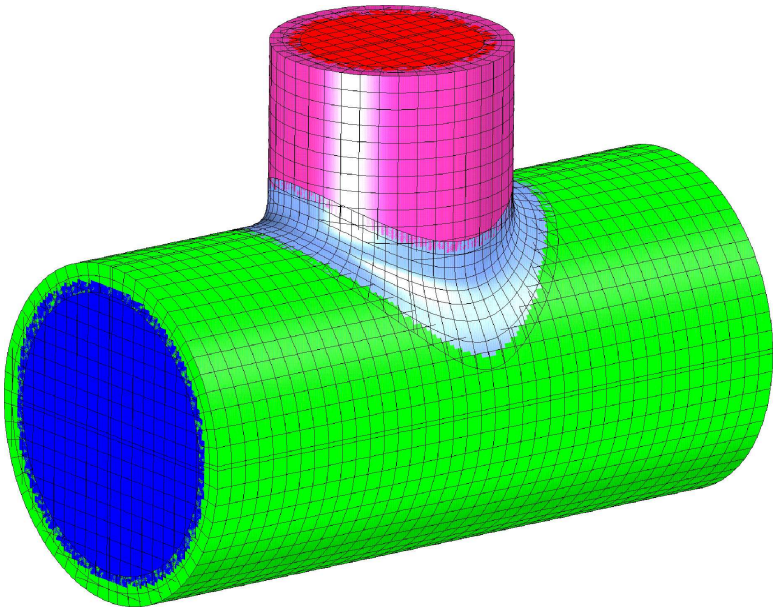

Fig. 12. Overlapping grid for the T-shaped pipe geometry. The Cartesian grids are blue and red, the cylindrical boundary-fitted grids are green and magenta, and the fillet grid is light blue. (The grid is coarsened by a factor of 6 for illustrative purposes.)

and then rotate this latter two-grid configuration by 90° about the x_3-axis. The last grid used to form the overlapping grid shown in Figure 12 is a *fillet* grid which smoothly connects the main section of pipe with the top section. This grid is constructed by first defining a mapping for a three-dimensional surface, $\mathbf{F}_s : \mathbb{R}^2 \to \mathbb{R}^3$, that smoothly transitions from the boundary of one cylinder to the boundary of the other. The fillet volume mapping, $\mathbf{F} : \mathbb{R}^3 \to \mathbb{R}^3$, is defined by extruding the surface mapping in the normal direction. The fillet volume mapping is evaluated on grid of points, $\mathbf{x_i} = \mathbf{F}(\mathbf{r_i})$, to define the fillet volume grid with mesh spacing approximately equal to h_0 and 7 grid points in the normal direction.

We assume that the state of the reactive flow in the T-shaped pipe at any time t is determined by its density ρ, velocity $\mathbf{v}$, pressure p and the mass fraction of the product of reaction given by the scalar reaction progress variable Y. The reaction rate needed in the governing equations in (3) and (4) is taken to be a one-step, Arrhenius rate with linear depletion of the form

$$\mathbf{R} = \sigma(1-Y)\exp\left[\frac{1}{\epsilon}\left(\frac{1}{T_c}-\frac{1}{T}\right)\right], \qquad m_r = 1, \tag{25}$$

where σ is a pre-exponential frequency factor, ϵ is a reciprocal activation energy, $T = p/\rho$ is a temperature (with gas constant normalized to 1) and T_c is a cross-over temperature. The contribution to the total energy in (5) is given by $q = YQ$, where $Q < 0$ is the heat release, taken to be negative for an exothermic reaction. The value for σ in this reaction model essentially picks the time scale. Following [2], we choose an induction time scale given by

$$\sigma = \frac{\epsilon}{(\gamma-1)|Q|}. \tag{26}$$

This choice implies that a spatially uniform sample with $T = T_c = 1$ initially would explode at $t = 1$ for the limiting case $\epsilon \to 0$. For the problem discussed here, we take $\epsilon = .08$ in (25), and use $Q = -4$ and $\gamma = 1.4$.

We are interested in solving an initial-boundary-value problem in which the initial state of the flow in the pipe is at rest with $p = 1$ and $Y = 0$, but at a critical stage in which the temperature T is near the cross-over value $T_c = 1$. Motivated by a reactive flow problem discussed in [2], which built upon the earlier work in [49], we consider the response of the flow to an initial temperature distribution given by

$$T(\mathbf{x}) = 1 - \delta\|\mathbf{x} - \mathbf{x}_0\|,$$

where $\mathbf{x}_0 = (-2, 1, 0)$ gives the location of a hot spot and $\delta = .03$ gives the rate of decrease in temperature away from the hot spot. Since $T = T_c = 1$ at $\mathbf{x} = \mathbf{x}_0$ and $T < T_c$ otherwise, the reaction is strongest near

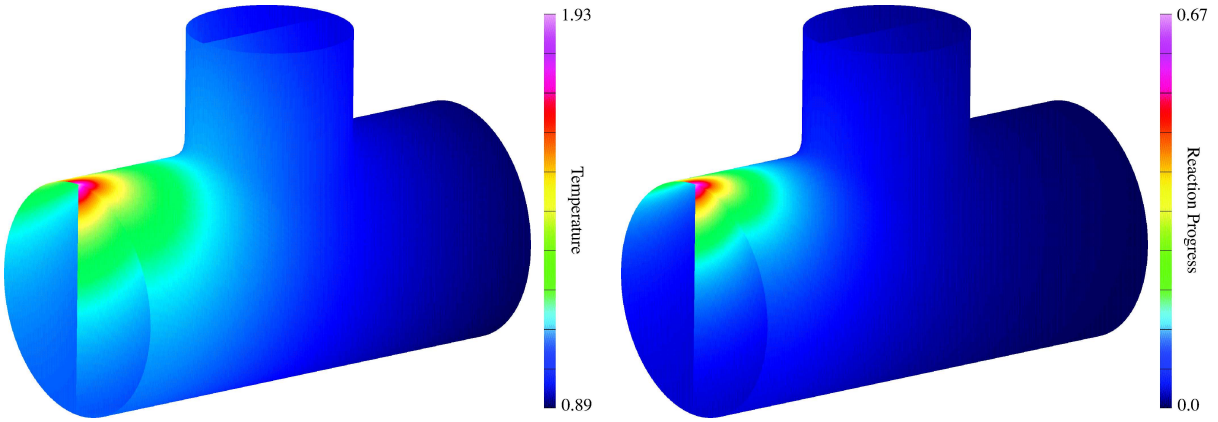

Fig. 13. Temperature (left) and reaction progress (right) at $t = 1.5$.

$\mathbf{x}_0$ leading to a local explosion there which occurs at a time roughly equal to 1 for the choice of σ given in (26). The main focus of the problem is the initiation of a detonation following the local explosion, and its subsequent propagation throughout the domain whose boundaries are assumed to be rigid slip walls.

The IBVP is solved numerically using the overlapping grid for the T-shaped pipe with 1 refinement level using a refinement ratio $n_r = 4$. The calculation requires 4930 time steps to integrate the equations from $t = 0$ to a final time $t = 2.8$, and uses 48 processors. The number of component grids, at the base level and refinement level, ranges from a minimum of 5 at $t = 0$ to a maximum of 682. The maximum number of grid points employed during the calculation is approximately 100 million. If the base grid were refined everywhere to achieve the finest resolution of the present AMR calculation, then the effective number of grid points would be approximately 400 million.

Figure 13 shows that behavior of the temperature and reaction progress at a time $t = 1.5$ just prior to the local explosion at $\mathbf{x} = \mathbf{x}_0$ (when Y first becomes 1), which is located in the upper left of the main section of the pipe. The view on the left shows shaded color contours of T on the surface of the domain and on the symmetry plane $x_3 = 0$, while the view on the right shows shaded color contours of Y. Here, we note that the reaction progress has achieved a maximum value $Y = 0.67$ near $\mathbf{x}_0$ and that the temperature there is $T = 1.93$, a value significantly greater than $T_c = 1$ indicating a rapid reaction there. Prior to $t = 1.46$, approximately, the solution is smooth enough so that no cells are tagged for refinement. For $t > 1.46$, one level of refinement grids with $n_r = 4$ is used to locally increase the grid resolution where the reaction rate is strong (as determined by $\tau_{\mathbf{i}}$ in (6)) or where spatial gradients are sharp (as measured by $e_{k,\mathbf{i}}$ in (7)).

A further increase in the reaction rate after $t = 1.5$, and its associated release of heat, creates a state of high temperature and pressure near the site of the initial hot spot. Acoustic signals from this high-pressure state travel outward from this site and raise the pressure and temperature in the neighborhood of the hot spot. The increased temperature there leads to an increased reaction rate which, in turn, leads to the formation of a wave of reaction (a fast flame) which propagates outward away from $\mathbf{x} = \mathbf{x}_0$. This wave is seen clearly in the top frames of Figure 14 at $t = 1.8$. In this figure, shaded contours of pressure are shown on the left while the corresponding shaded contours of Y are shown on the right. As the fast flame advances away from $\mathbf{x} = \mathbf{x}_0$, acoustic signals ahead of it steepen to form a detonation (by $t = 2.0$ in the figure). The detonation strengthens as it propagates towards the bottom of the main section of the pipe due to a lateral geometric compression, while it weakens as it turns the 90° corner into the smaller top section of the pipe (see the plots at $t = 2.2$ in the figure).

The sequence of plots in Figure 15 shows the complex wave structure of the solution at three times after the detonation has reached the bottom of the main section of the pipe. For these times, it is convenient to cut away the portion of the pipe for $x_3 > 0$ to reveal the behavior of the solution on the symmetry plane $x_3 = 0$. As before, the behavior of pressure is shown on the left and Y is shown on the right. The top view at $t = 2.4$ shows the detonation which is convex forward and propagating from left to right. The detonation is strongest near the bottom of the main section of the pipe where a regular reflection is observed at its

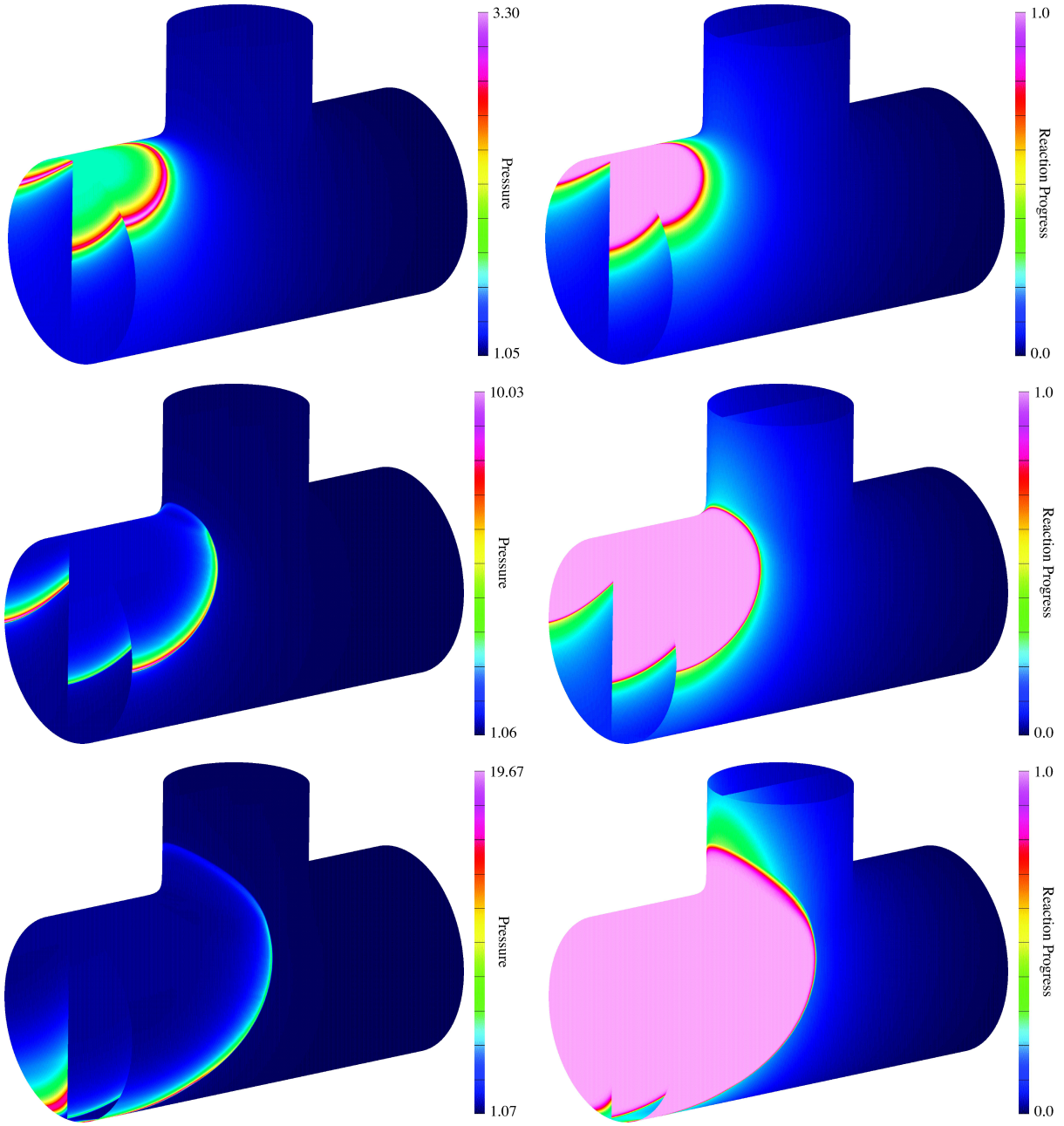

Fig. 14. Pressure (left) and reaction progress (right) at $t = 1.8$ (top), $t = 2.0$ (middle) and $t = 2.2$ (bottom).

bottom surface. By $t = 2.6$ (middle views), the regular reflection has transitioned to a Mach reflection and small a Mach stem-like detonation may be seen near the bottom surface. Meanwhile, the detonation in the top section of the pipe has reflected off the planar surface at the top, generating a reflected shock that is propagating back into the reaction products (at $Y = 1$). The detonation has also met the back edge of the T-shaped pipe where the top section joins to the main section of pipe generating a reflected shock from that collision. The final view at $t = 2.8$ shows the detonation, almost planar now, approaching the back face of the main section of the pipe, and a complex system of interacting reflected shocks traveling back and generally downward from the top of the T-shaped pipe, and forward and generally upward from the bottom portion of the pipe.

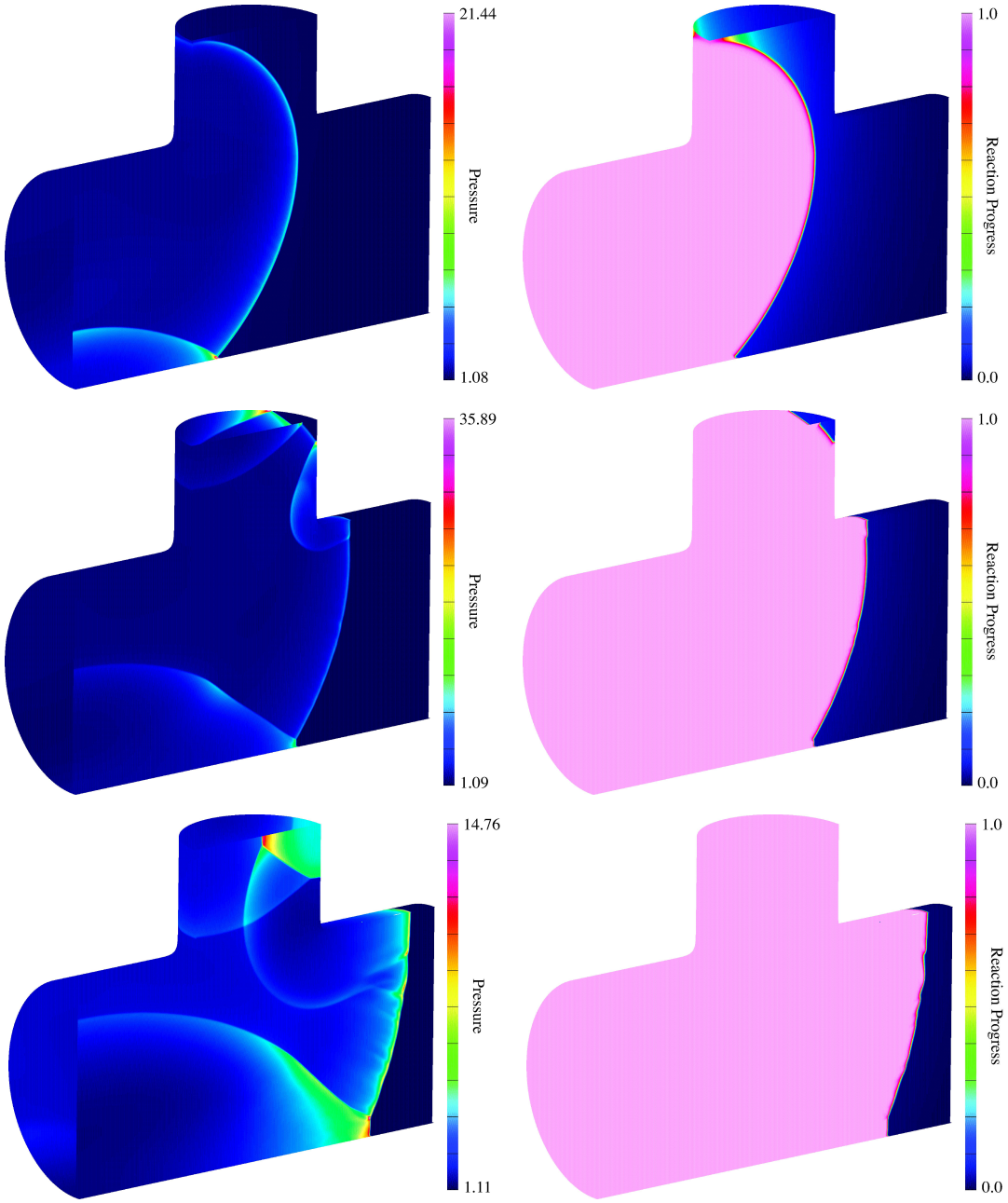

Fig. 15. Pressure (left) and reaction progress (right) at $t = 2.4$ (top), $t = 2.6$ (middle) and $t = 2.8$ (bottom).

As a final set of plots, we show in Figure 16 the behavior of the pressure at $t = 2.8$ for a range of grid resolutions. The top left and right plots in the figure show the pressure computed using base grids with $h_0 = 1/20$ and $h_0 = 1/40$, respectively, while the plot on the bottom shows the pressure computed using a base grid with $h_0 = 1/60$ (as used for the previous plots in Figures 13, 14, and 15). All three calculations use 1 refinement level with $n_r = 4$. We note that the qualitative behavior of the solutions are similar. The location of the detonation wave is in good agreement for all three grid resolutions, and the locations of shocks and contacts in the burnt flow behind the detonation are in good agreement as well. The main difference appears in the fine structure of the solution near the detonation and shock triple-points, in particular, as the grid is refined. Overall, the plots indicate good grid convergence of the solution.

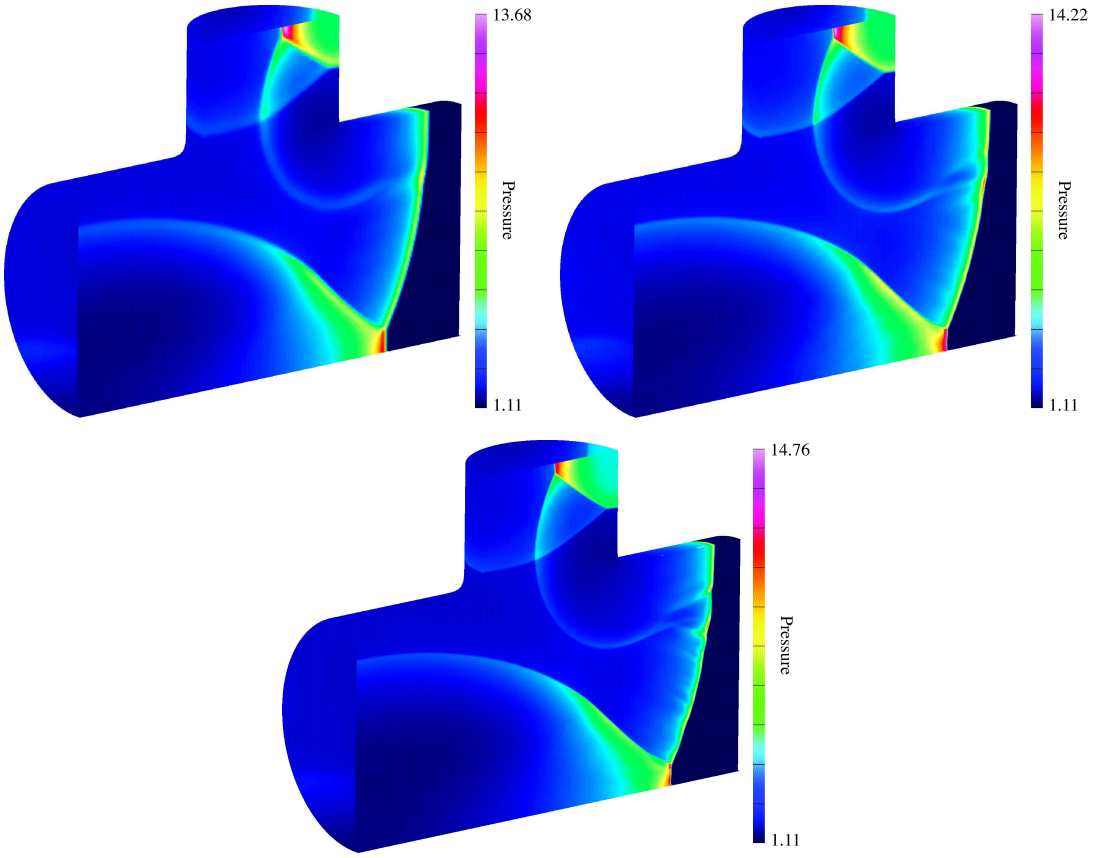

Fig. 16. Pressure at $t = 2.8$ using a base grid with $h_0 = 1/20$ (top left), $h_0 = 1/40$ (top right) and $h_0 = 1/60$ (bottom). All three calculations use 1 refinement level with $n_r = 4$.

7. Conclusions

We have described an approach for the numerical solution of initial-boundary-value problems for PDEs in complex three-dimensional domains using overlapping grids and adaptive mesh refinement. The technique is implemented for parallel distributed-memory computers using a domain-decomposition approach. We have discussed various aspects of the parallel algorithm such as the decomposition of grids and grid functions, and the operations of interpolation, error estimation, refinement-grid generation and load balancing.

We have considered two particular PDEs, an advection-diffusion equation and the reactive Euler equations, and have described how these equations are discretized and solved numerically. We have verified the accuracy of the parallel AMR approach by solving the advection-diffusion equation with forcing functions chosen so that exact solutions can be constructed a priori. We showed that the error in the numerical solution is of the order of the machine round-off error when the exact solution of the equations is chosen to be of a polynomial form and when the grids are rectangular. For more general curvilinear grids we showed that the errors were second-order accurate. The results were shown to be independent of the number of processors, and independent of the number of refinement levels and refinement ratios provided the effective resolution on the finest grids are commensurate.

The approach was further verified by solving the Euler equations for planar shock diffraction by a sphere. The solution was shown to converge as the grids were refined, and the results of the fully three-dimensional calculation were shown to agree well with the results of a corresponding highly-resolved axisymmetric calculation. Parallel scaling results were presented for this problem, and they showed good strong and weak-scaling for the non-AMR case using up to 64 processors. Strong parallel scaling results for the AMR case were reasonably good, although the scaling results degraded as the number of processors increased. This was ex-

pected due to our initial implementation of the AMR interpolation routines. As future work we will improve the parallel efficiency of these routines by combining the large number of small messages currently being sent.

As a final illustration of the approach, we simulated the initiation and propagation of a gaseous detonation in a T-shaped pipe. This parallel AMR computation was run on 48 processors, involved a maximum of approximately 100 million grid points, and showed the detailed structure of the formation and propagation of the detonation wave as it moved through the complex three-dimensional pipe geometry. A comparison of numerical solutions of the problem using base grids with different resolutions indicated that the solution on the finest grid was well resolved.

References

[1] G. Chesshire, W. Henshaw, Composite overlapping meshes for the solution of partial differential equations, J. Comput. Phys. 90 (1) (1990) 1–64.

[2] W. D. Henshaw, D. W. Schwendeman, An adaptive numerical scheme for high-speed reactive flow on overlapping grids, J. Comput. Phys. 191 (2003) 420–447.

[3] W. D. Henshaw, D. W. Schwendeman, Moving overlapping grids with adaptive mesh refinement for high-speed reactive and non-reactive flow, J. Comput. Phys. 216 (2) (2006) 744–779.

[4] E. A. Volkov, A finite difference method for finite and infinite regions with piecewise smooth boundaries, Doklady 168 (5) (1966) 744–747.

[5] E. A. Volkov, The method of composite meshes for finite and infinite regions with piecewise smooth boundaries, Proc. Steklov Inst. Math. 96 (1968) 145–185.

[6] G. Starius, Composite mesh difference methods for elliptic and boundary value problems, Numer. Math. 28 (1977) 243–258.

[7] G. Starius, On composite mesh difference methods for hyperbolic differential equations, Numer. Math. 35 (1980) 241–255.

[8] G. Starius, Constructing orthogonal curvilinear meshes by solving initial value problems, Numer. Math. 28 (1977) 25–48.

[9] B. Kreiss, Construction of a curvilinear grid, SIAM J. of Sci. Stat. Comput. 4 (2) (1983) 270–279.

[10] J. L. Steger, J. A. Benek, On the use of composite grid schemes in computational aerodynamics, Computer Methods in Applied Mechanics and Engineering 64 (1987) 301–320.

[11] A. Kapila, D. Schwendeman, J. Bdzil, W. Henshaw, A study of detonation diffraction in the ignition-and-growth model, Combust. Theory and Modeling 11 (5) (2007) 781–822.

[12] J. Banks, D. Schwendeman, A. Kapila, W. Henshaw, A high-resolution Godunov method for compressible multi-material flow on overlapping grids, J. Comput. Phys. 223 (2007) 262–297.

[13] J. Banks, D. Schwendeman, A. Kapila, W. Henshaw, A study of detonation propagation and diffraction with compliant confinement, Combust. Theory and Modeling (submitted).

[14] J. Y. Tu, L. Fuchs, Calculation of flows using three-dimensional overlapping grids and multigrid methods, International Journal for Numerical Methods in Engineering 38 (1995) 259–282.

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

[16] M. Hinatsu, J. Ferziger, Numerical computation of unsteady incompressible flow in complex geometry using a composite multigrid technique, International Journal for Numerical Methods in Fluids 13 (1991) 971–997.

[17] R. Meakin, Moving body overset grid methods for complete aircraft tiltrotor simulations, paper 93-3350, AIAA (1993).

[18] D. Pearce, S. Stanley, F. Martin, R. Gomez, G. L. Beau, P. Buning, W. Chan, T. Chui, A. Wulf, V. Akdag, Development of a large scale Chimera grid system for the space shuttle launch vehicle, paper 93-0533, AIAA (1993).

[19] R. Maple, D. Belk, A new approach to domain decomposition, the Beggar code, in: N. Weatherill (Ed.), Numerical Grid Genertation in Computational Fluid Dynamics and Related Fields, Pineridge Press Limited, 1994, pp. 305–314.

[20] D. Jespersen, T. Pulliam, P. Buning, Recent enhancements to OVERFLOW, paper 97-0644, AIAA (1997).

[21] R. L. Meakin, Composite overset structured grids, in: J. F. Thompson, B. K. Soni, N. P. Weatherill (Eds.), Handbook of Grid Generation, CRC Press, 1999, Ch. 11, pp. 1–20.

[22] C. Kiris, D. Kwak, S. Rogers, I. Chang, Computational approach for probing the flow through artificial heart devices, J. Biomech. Eng. 119 (4) (1997) 452–460.

[23] Y. Tahara, R. Wilson, P. Carrica, F. Stern, RANS simulation of a container ship using a single-phase level-set method with overset grids and the prognosis for extension to a self-propulsion simulator, Journal of Marine Science and Technology 11 (4) (2006) 209–228.

[24] F. Olsson, J. Yström, Some properties of the upper convected Maxwell model for viscoelastic fluid flow, J. Non-Newtonian Fluid Mech. 48 (1993) 125–145.

[25] N. A. Petersson, A numerical method to calculate the two-dimensional flow around an underwater obstacle, SIAM J. of Numer. Anal. 29 (1992) 20–31.

[26] P. Fast, Dynamics of interfaces in non-Newtonian Hele-Shaw flow, Ph.D. thesis, New York University, Courant Institute of Mathematical Sciences (1999).

[27] P. Fast, M. J. Shelley, A moving overset grid method for interface dynamics applied to non-Newtonian Hele-Shaw flow, J. Comput. Phys. 195 (2004) 117–142.
[28] K. Brislawn, D. L. Brown, G. Chesshire, J. Saltzman, Adaptively-refined overlapping grids for the numerical solution of hyperbolic systems of conservation laws, report LA-UR-95-257, Los Alamos National Laboratory (1995).
[29] E. P. Boden, E. F. Toro, A combined Chimera-AMR technique for computing hyperbolic PDEs, in: Djilali (Ed.), Proceedings of the Fifth Annual Conference of the CFD Society of Canada, 1997, pp. 5.13–5.18.
[30] C. A. Rendleman, V. E. Beckner, M. Lijewski, W. Y. Crutchfield, J. B. Bell, Parallelization of structured, hierarchical adaptive mesh refinement algorithms, Computing and Visualization in Science 3 (2000) 147–157.
[31] P. C. et.al., Chombo, software package for amr applications, Tech. Rep. http://seesar.lbl.gov/anag/chombo, Lawrence Berkely National Laboratory (2007).
[32] M. Parashar, J. C. Browne, Distributed adaptive grid hierarchy, http://www.caip.rutgers.edu/∼parashar/dagh/, Rutgers University (2007).
[33] M. Parashar, GrACE, grid adaptive computational engine, http://www.caip.rutgers.edu/tassl/projects/grace/, Rutgers University (2007).
[34] K. Olson, Paramesh: A parallel adaptive grid tool, in: A. D. et.al. (Ed.), Parallel Computational Fluid Dynamics, Elsevier, 2006, pp. 341–348.
[35] X. Garaizar, R. Hornung, S. Kohn, Structured adaptive mesh refinement applications infracture, Tech. Rep. http://www.llnl.gov/casc/SAMRAI, Lawrence Livermore National Laboratory (1999).
[36] W. Henshaw, Mappings for Overture, a description of the Mapping class and documentation for many useful Mappings, Research Report UCRL-MA-132239, Lawrence Livermore National Laboratory (1998).
[37] W. D. Henshaw, A high-order accurate parallel solver for Maxwell's equations on overlapping grids, SIAM Journal on Scientific Computing 28 (5) (2006) 1730–1765.
URL `http://link.aip.org/link/?SCE/28/1730/1`
[38] W. Henshaw, Ogen: An overlapping grid generator for Overture, Research Report UCRL-MA-132237, Lawrence Livermore National Laboratory (1998).
[39] D. Quinlan, A++/P++ class libraries, Research Report LA-UR-95-3273, Los Alamos National Laboratory (1995).
[40] A. Sussman, G. Agrawal, J. Saltz, A manual for the Multiblock PARTI runtime primitives, revision 4.1, Technical Report CS-TR-3070.1, University of Maryland, Department of Computer Science (1993).
[41] W. Gropp, E. Lusk, R. Thakur, Using MPI-2: Advanced Features of the Message-Passing Interface, The MIT Press, Cambridge, MA, 1999.
[42] M. Berger, I. Rigoutsos, An algorithm for point clustering and grid generation, IEEE Trans. Systems Man and Cybernet 21 (1991) 1278–1286.
[43] B. T. N. Gunney, A. M. Wissink, D. A. Hysom, Parallel clustering algorithms for structured AMR, Journal of Parallel and Distributed Computing 66 (11) (2006) 1419–1430.
[44] C. A. Rendleman, V. E. Beckner, M. Lijewski, W. Y. Crutchfield, J. B. Bell, Parallelization of structured, hierarchical adaptive mesh refinement algorithms, Computing and Visualiztion in Science 3 (2000) 137–147.
[45] B. Hendrickson, K. Devine, Dynamic load balancing in computational mechanics, Computer Methods in Applied Mechanics and Engineering 184 (2) (2000) 485–500.
[46] H. Johansson, J. Steensland, A performance characterization of load balancing algorithms for parallel SAMR applications, Technical report 2006-047 2006-047, Uppsala University, Information technology (2006).
[47] S. Chandra, M. Parashar, J. Ray, Dynamic structured partitioning for parallel scientific applications with pointwise varying workloads, in: Parallel and Distributed Processing Symposium, 2006, p. 10.
[48] M. H. Carpenter, D. Gottlieb, S. Abarbanel, W. S. Don, The theoretical accuracy of Runge-Kutta time discretizations for the initial boundary value problem: a study of the boundary error, SIAM J. Sci. Comput. 16 (1995) 1241–1252.
[49] A. K. Kapila, D. W. Schwendeman, J. J. Quirk, T. Hawa, Mechanisms of detonation formation due to a temperature gradient, Combust. Theory and Modeling 6 (2002) 553–594.