OVERTURE: OBJECT-ORIENTED TOOLS FOR APPLICATIONS WITH COMPLEX GEOMETRY

David L. Brown, William D. Henshaw, and Dan Quinlan[1]

Lawrence Livermore National Laboratory, Livermore, CA, USA,
dlb,henshaw,dquinlan@llnl.gov,
WWW home page: http://www.llnl.gov/CASC/people/dlb,henshaw,quinlan/

Abstract. The **Overture** framework is an object-oriented environment for solving partial differential equations in two and three space dimensions. It is a collection of C++ libraries that enables the use of finite difference and finite volume methods at a level that hides the details of the associated data structures. **Overture** can be used to solve problems in complicated, moving geometries using the method of overlapping grids. It has support for grid generation, difference operators, boundary conditions, data-base access and graphics. Short sample code segments are presented to show the power of this approach.

1 Introduction

The **Overture** framework is a collection of C++ libraries that provide tools for solving partial differential equations. **Overture** can be used to solve problems in complicated, moving geometries using the method of overlapping grids (also known as overset or Chimera grids). **Overture** includes support for geometry, grid generation, difference operators, boundary conditions, data-base access and graphics.

An overlapping grid consists of a set of logically rectangular grids that cover a domain and overlap where they meet. This method has been used successfully over the last decade and a half, primarily to solve problems involving fluid flow in complex, often dynamically moving, geometries[2, 3, 11, 12, 23]. Solution values at the overlap are determined by interpolation. The overlapping grid approach is particularly efficient for rapidly generating high-quality grids for moving geometries. As the component grids move only the boundary points to be interpolated change, the grid points do not have to be regenerated. The component grids are structured so that efficient and fast finite-difference algorithms can be utilized. We use adaptive mesh refinement to resolve fine features of the solution. [1, 2, 20]. The design of **Overture** has evolved over the past 15 years or so from the Fortran based CMPGRD [8] environment to the current C++ version [5, 6]. Although the Fortran implementation was used for complicated three-dimensional adaptive and moving grid computations, the programs were difficult to write and maintain. **Overture** was designed to have at least all the functionality of the

Fortran code but to be as easy as possible to use; indeed, an entire PDE solver on an overlapping grid can be written on a single page (see section 6).

Overture is an object-oriented framework. In the past a typical Fortran code would use a procedural model where subroutines and functions are the fundamental building blocks and data is passed to and from these procedures. In **Overture** the fundamental building blocks are objects such as grids and grid functions. These objects can be manipulated at a high level. Details of the implementation, such as how a grid is stored, are hidden from the user. In the object-oriented world this is known as *data encapsulation*. One major benefit of encapsulation is that changes can be made to the implementation of an object without forcing changes to be made to the code that uses the object. An object such as a grid function contains not only data (such as the values of density at each point on a grid) but also implements functions that operate on the data (these functions are often called *methods* in object-oriented terminology). Thus a grid function (which may live on a collection of grids on a overlapping grid) will know what it means to add itself to another grid function, which would be expressed in a C++ code as $u = v + w$. The '+' operator and the '=' operator are defined by the grid function class, a process known as operator overloading.

A common complaint of object-oriented languages such as C++ is that it is easy to write code that is elegant but runs many times slower than fortran. It is certainly true that since C++ is a much richer language than fortran that it is easy to write inefficient code. However, it is also true that by writing a code in C++ with basically a fortran style that it is possible to achieve the exact same performance of fortran (while still benefiting from some of the nice features of C++). **Overture** has been designed to be used at different levels to allow users to obtain full performance at the cost of writing at a lower level. Thus one can either operate at a high level to write an entire code (with a decrease in performance) or one may only use C++ to manage the complex data structures while calling Fortran or C routines to perform computationally intensive tasks. In the future it is likely that even the high level code can be made to run as fast as the low level approach. We are currently working on a preprocessor that will automatically convert high level C++ code into efficient C code. Initial results show that full fortran performance can be obtained using the preprocessor.

There are a number of other very interesting projects developing scientific object-oriented frameworks. These include the SAMRAI framework for structured adaptive mesh refinement[22] , PETSc (the Portable Extensible Toolkit for Scientific Computation)[18] , POOMA (Parallel Object Oriented Methods and Applications)[19] and Diffpack[10].

2 The OVERTURE FRAMEWORK

The main class categories that make up **Overture** are as follows:

- **Arrays** [21]: describe multidimensional arrays using A++/P++. A++ provides the serial array objects, and P++ provides the distribution and interpretation of communication required for their data parallel execution.

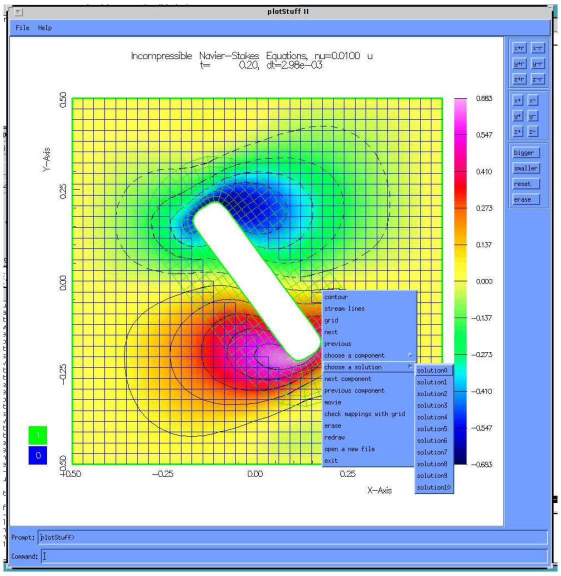

Fig. 1. Displaying results from a moving grid computation using the **Overture** framework.

- **Mappings** [15]: define transformations such as curves, surfaces, areas, and volumes. These are used to represent the geometry of the computational domain.
- **Grids** [9, 14]: define a discrete representation of a mapping or mappings. These include single grids, and collections of grids; in particular composite overlapping grids.
- **Grid functions** [14]: storage of solution values, such as density, velocity, pressure, defined at each point on the grid(s).
- **Operators** [4, 13]: provide discrete representations of differential operators and boundary conditions
- **Grid generation** [16]: the Ogen overlapping grid generator automatically constructs an overlapping grid given the component grids.
- **Plotting** [17]: a high-level interface based on OpenGL allows for plotting **Overture** objects.
- **Adaptive mesh refinement:** The AMR++ library for patch based refinement is described in section 4.

Solvers for partial differential equations, such as the **OverBlown** solver described in section (7) are written using the above classes.

3 Array Operations

A++ and P++ [21] are array class libraries for performing array operations in C++ in serial and parallel environments, respectively.

A++ is a *serial* array class library similar to FORTRAN 90 in syntax, but not requiring any modification to the C++ compiler or language. A++ provides an object-oriented array abstraction specifically well suited to large scale numerical computation. It provides efficient use of multidimensional array objects which serves to both simplify the development of numerical software and provide a basis for the development of parallel array abstractions. P++ is the *parallel* array class library and shares an identical interface to A++, effectively allowing A++ serial applications to be recompiled using P++ and thus run in parallel. This provides a simple and elegant mechanism that allows serial code to be reused in the parallel environment.

P++ provides a data parallel implementation of the array syntax represented by the A++ array class library. To this extent it shares a lot of commonality with FORTRAN 90 array syntax and the HPF programming model. However, in contrast to HPF, P++ provides a more general mechanism for the distribution of arrays and greater control as required for the multiple grid applications represented by both the overlapping grid model and the adaptive mesh refinement (AMR) model. Additionally, current work is addressing the addition of task parallelism as required for parallel adaptive mesh refinement.

Here is a simple example code segment that solves Poisson's equation in either a serial or parallel environment using the A++/P++ classes. Notice how the Jacobi iteration for the entire array can be written in one statement.

```
  // Solve u_xx + u_yy = f by a Jacobi Iteration
Range R(0,n)                           // ... define a range of indices: 0,1,2,...,n
floatArray u(R,R), f(R,R)              // ... declare two two-dimensional arrays
f = 1.; u = 0.; h = 1./n;              // ... initialize arrays and parameters
Range I(1,n-1), J(1,n-1);              // ... define ranges for the interior

for( int iteration=0; iteration<100; iteration++ )
  u(I,J) = .25*(u(I+1,J)+u(I-1,J)+u(I,J+1)+u(I,J-1)-f(I,J)*(h*h)); // ... data parallel
```

4 Adaptive Mesh Refinement

Adaptive mesh refinement is the process of permitting local grids to be added to the computational domain and thus adaptively tailoring the resolution of the computational grid. The block-structured AMR algorithm implemented in **Overture** provides such support for both simple problems with a single underlying grid, and problems that use the composite overlapping grid method. The AMR algorithm itself uses the multiple grid functionality provided by the basic **Overture** classes in an essential way. AMR results is greater computational efficiency but is difficult to support. AMR++ is a library within the **Overture** framework which builds on top of the previously mentioned components and provides support for **Overture** applications requiring adaptive mesh refinement. AMR++ is current work being developed and supports the adaptive regridding, transfer of data between adaptive refinement levels, parent/child/sibling operations between local refinement levels, and includes parallel AMR support.

AMR++ is a parallel adaptive mesh refinement library because it is uses classes which derive their parallel support from the A++/P++ array class library.

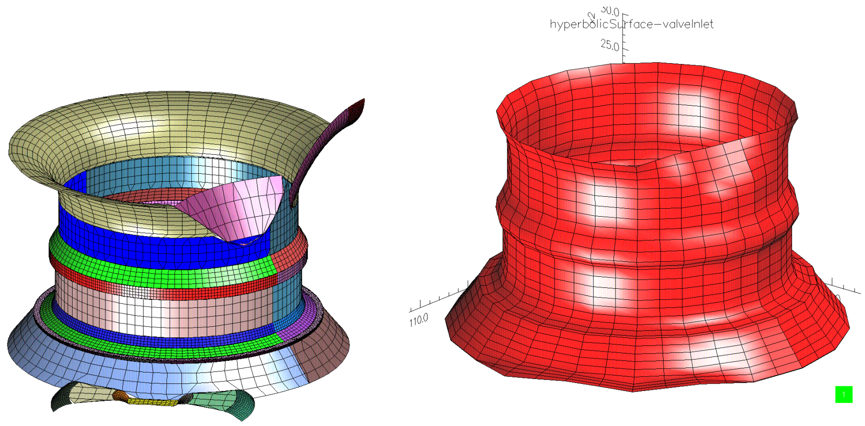

Fig. 2. Hyperbolic surface grid generation is used to generate a smooth surface grid over a surface coming from a CAD package.

5 Grid Generation

Overture has support for the creation of overlapping grids for complicated geometries. The process of generating an overlapping grid consists of two basic steps. In the first step a number of component grids are generated. Each component grid represents a portion of the geometry. The component grids must overlap but otherwise can be created locally. **Overture** provides a collection of `Mapping` classes that can be used to generate component grids including splines, NURBS, bodies of revolution, hyperbolic grid generation, elliptic grid generation, trans-finite interpolation and so on. In addition we are working on methods for reading files generated by CAD programs and generating grids. Figure (2) shows how hyperbolic grid surface grid generation can be used to generate a single smooth grid over a CAD surface described by a collection of trimmed NURBS. This is accomplished with the aid of the SURGRD hyperbolic surface grid generator[7].

Given the component grids, the overlapping grid then is constructed using the `Ogen` grid generator. This latter step consists of determining how the different component grids interpolate from each other, and in removing grid points from holes in the domain, and removing unnecessary grid points in regions of excess overlap. `Ogen` requires a minimal amount of user input. The grids in figure (3) were all created with `Ogen`.

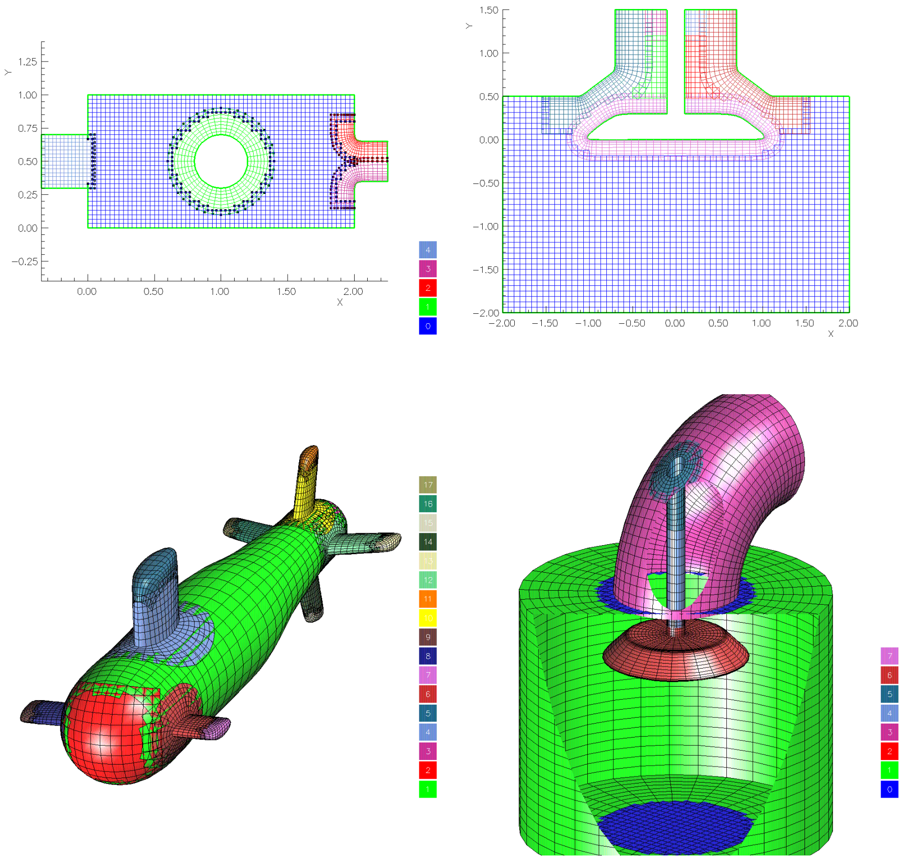

Fig. 3. Sample 2D and 3D overlapping grids generated with the Ogen grid generator.

6 Writing PDE solvers

This example demonstrates the power of the **Overture** framework by showing a basically complete code that solves the partial differential equation (PDE)

$$u_t + au_x + bu_y = \nu(u_{xx} + u_{yy})$$

on an overlapping grid.

The `PlotStuff` object is used to interactively plot contours of the solution at each time step[17].

```
int main()
\{
  CompositeGrid cg;                          // create a composite grid
  getFromADataBaseFile(cg,"myGrid.hdf");    // read the grid in
  floatCompositeGridFunction u(cg);          // create a grid function
  u=1.;                                      // assign initial conditions
  CompositeGridOperators op(cg);             // create operators
  u.setOperators(cg);
  PlotStuff ps;                              // make an object for plotting
  // --- solve a PDE ----
  float t=0, dt=.005, a=1., b=1., nu=.1;
  for( int step=0; step<100; step++ )
  \{
    u+=dt*( -a*u.x()-b*u.y()+nu*(u.xx()+u.yy()) );
    t+=dt;
    u.interpolate();                        // interpolate overlapping boundaries
    // apply the BC u=0 on all boundaries
    u.applyBoundaryCondition(0,dirichlet,allBoundaries,0.);
    u.finishBoundaryConditions();
    ps.contour(u);                          // plot contours of the solution
  \}
  return 0;
\}
% \end{verbatimcmd}
```

7 OverBlown

OverBlown is a fluid flow solver for overlapping grids built upon the **Overture** framework. **OverBlown** is being designed to solve the Navier-Stokes equations at all flow speeds, using different algorithms at different Mach numbers. Although the currently distributed version only solves the incompressible Navier-Stokes equations, we are developing a low-Mach number and high Mach number algorithms as well as adding support for chemically reacting flows. Ref 4th order.

Figure (4) shows streamlines of the solution to the incompressible Navier-Stokes equations around a rotating stirring stick and the script file that was used to run OverBlown. Figure (5) shows a result from an incompressible flow computation around a NACA 0012 airfoil. The grid was generated using the elliptic grid generator in **Overture** .

```
* Choose the
overlapping grid: stir.hdf
stir.show
incompressibleNavierStol
turn off twilight zone
project initial conditions
turn on moving grids
specify grids to move
  stir
  rotate
  0. 0. 0.
  .5
done
choose grids for implicit
  all=explicit
  stir=implicit
done
pde parameters
  nu
  .01
done
boundary conditions
  all=noSlipWall
done
initial conditions
  uniform flow
  p=1.
final time (tf=)
  .5
times to plot (tp=)
  .025
```

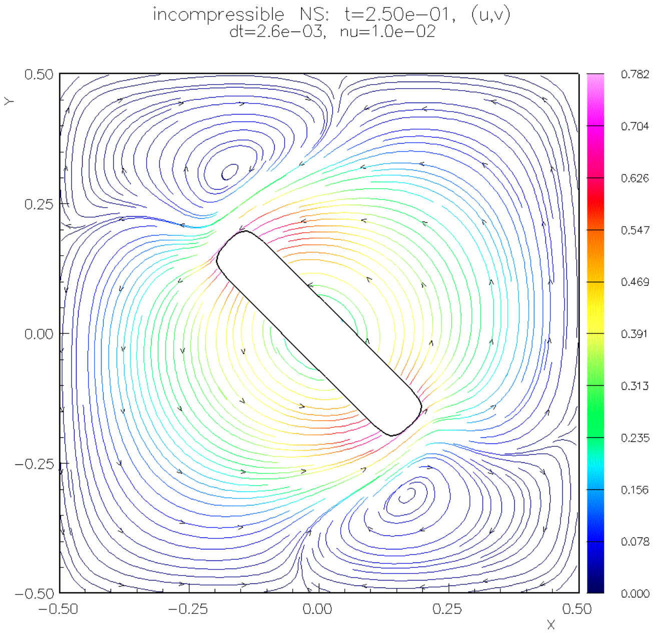

Fig. 4. On the left is a sample script command file for running the flow solver OverBlown. On the right is the result from the computation of incompressible flow around a rotating stirring stick.

8 Software Availability

The **Overture** framework and documentation is available for public distribution from the web site, `http://www.llnl.gov/casc/Overture`. The **OverBlown** flow solver is available for limited distribution, please contact Bill Henshaw for further information.

References

1. M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comp. Phys., 82 (1989), pp. 64–84.
2. K. D. Brislawn, D. L. Brown, G. Chesshire, and J. S. Saltzman, *Adaptive composite overlapping grids for hyperbolic conservation laws*, LANL Unclassified Report 95-257, Los Alamos National Laboratory, 1995.
3. D. L. Brown, *An unsplit Godunov method for systems of conservation laws on curvilinear overlapping grids*, Math. Comput. Modelling, 20 (1994), pp. 29–48.
4. ———, *Classes for finite volume operators and projection operators*, LANL unclassified report 96-3470, Los Alamos National Laboratory, 1996.
5. D. L. Brown, Geoffrey S. Chesshire, William D. Henshaw and Daniel J. Quinlan, **Overture** *: An Object Oriented Software System for Solving Partial Differential Equations in Serial and Parallel Environments*, Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing, 1997.

```
*
* OverBlown command file for
*  flow past a naca0012 airfoil
*
* grid name:
naca0012
* show file name
ob.show
  incompressibleNavierStokes
  turn off twilight zone
  final time (tf=)
    5.
  times to plot (tp=)
    .5
  plot and always wait
  * no plotting
  pde parameters
    * this next value for nu is too sı
    * have any effect on this grid.
    nu
      .00001
    * choose 2nd-order artificial vis
    ad21
      2.
    ad22
      2.
    done
  boundary conditions
    all=noSlipWall
    backGround(0,0)=inflowWithVelocit
                    uniform(p=1.,u=1.
    backGround(1,0)=outflow
    backGround(0,1)=slipWall
    backGround(1,1)=slipWall
    done
  initial conditions
  uniform flow
    p=1., u=1.
  project initial conditions
  exit
```

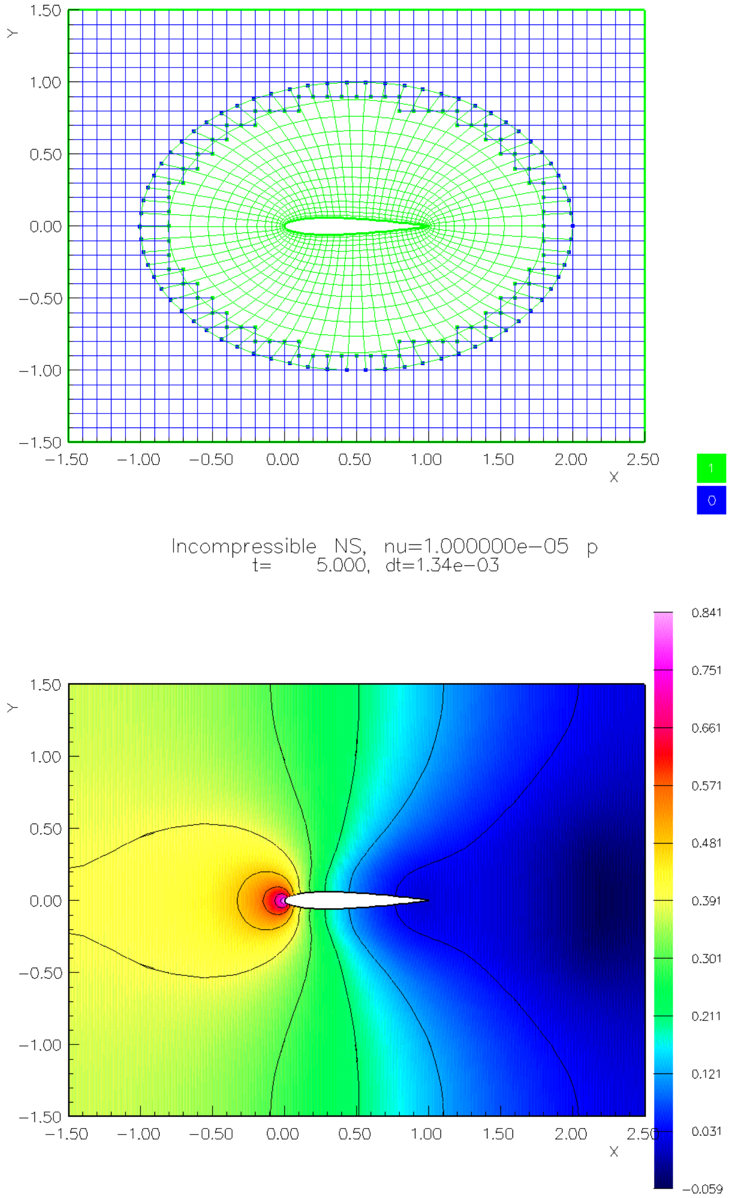

Fig. 5. On the left is a sample script command file for running the flow solver OverBlown. On the right is the result from the computation of incompressible flow around a NACA 0012 airfoil.

6. , D. L. Brown, William D. Henshaw and Daniel J. Quinlan, **Overture** *: An Object Oriented Framework for Solving Partial Differential Equations*, Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, 1343, 1997.
7. W.M. Chan and P.G. Buning, *A Hyperbolic Surface Grid Generation Scheme and Its Applications*, AIAA paper 94-2208, 1994.
8. G. Chesshire and W. D. Henshaw, *Composite overlapping meshes for the solution of partial differential equations*, J. Comp. Phys., 90 (1990), pp. 1–64.
9. G. S. Chesshire, **Overture** *: the grid classes*, LANL unclassified report 96-3708, Los Alamos National Laboratory, 1996.
10. Diffpack homepage, `http://www.nobjects.com/diffpack`.
11. F. C. Dougherty and J. Kuan, *Transonic store separation using a three-dimensional Chimera grid scheme*, AIAA paper 89-0637, AIAA, 1989.
12. W. D. Henshaw, *A fourth-order accurate method for the incompressible Navier-Stokes equations on overlapping grids*, J. Comp. Phys., 113 (1994), pp. 13–25.
13. ———, *Finite difference operators and boundary conditions for Overture, user guide, version 1.00*, LANL unclassified report 96-3467, Los Alamos National Laboratory, 1996.
14. ———, *Grid, GridFunction and Interpolant classes for* **Overture** *, AMR++ and CMPGRD, user guide, version 1.00*, LANL unclassified report 96-3464, Los Alamos National Laboratory, 1996.
15. ———, *Mappings for* **Overture** *: A description of the mapping class and documentation for many useful mappings*, LANL unclassified report 96-3469, Los Alamos National Laboratory, 1996.
16. ———, *Ogen: an overlapping grid generator for Overture*, LANL unclassified report 96-3466, Los Alamos National Laboratory, 1996.
17. ———, *PlotStuff: a class for plotting stuff from* **Overture** , LANL unclassified report 96-3893, Los Alamos National Laboratory, 1996.
18. Satish Balay, William Gropp, Lois Curfman McInnes and Barry Smith, *The Portable Extensible Toolkit for Scientific Computation*, `http://www.mcs.anl.gov/petsc/petsc.html`.
19. Steve Karmesin et.al, *Parallel Object Oriented Methods and Applications*, `http://www.acl.lanl.gov/PoomaFramework`.
20. D. Quinlan, *Adaptive Mesh Refinement for Distributed Parallel Processors*, PhD thesis, University of Colorado, Denver, June 1993.
21. ———, *A++/P++ manual*, LANL Unclassified Report 95-3273, Los Alamos National Laboratory, 1995.
22. Xabier Garaizar, Richard Hornung and Scott Kohn, Structured Adaptive Mesh Refinement Applications Infracture, `http://www.llnl.gov/casc/SAMRAI`.
23. J. L. Steger and J. A. Benek, *On the use of composite grid schemes in computational aerodynamics*, Computer Methods in Applied Mechanics and Engineering, 64 (1987), pp. 301–320.