Software Engineering Plan

P. Colella
B. Van Straalen

Applied Numerical Algorithms Group
NERSC Division
Lawrence Berkeley National Laboratory
Berkeley, CA

July 28, 2003

Contents

1 Introduction

1.1 Purpose

Under this proposal, we will develop a software framework for implementing block-structured AMR algorithms, based on the Chombo C++ framework for AMR applications developed at LBNL. There are two components to this activity. The first is the development of algorithm and software enhancements to the Chombo library to enable the support of two applications areas in the HPCC ESS project: microgravity and star formation. The second is the development of a data visualization and analysis package, called ChomboVis, that will meet the needs of a broad range of ESS users of block-structured AMR.

1.2 Background

The starting point for this work is the Chombo framework for AMR applications [CGM+]. The design approach in Chombo is based on two ideas. The first is that the mathematical structure of the algorithms maps naturally into a combination of data structures and operations on those data structures which are embodied in C++ classes. The second is that the mathematical structure of the algorithms can be naturally factored into a hierarchy of abstractions, leading to an analogous factorization of the framework into reusable components, or *layers*. Functionality within a layer results from a combination of generic programming and sub-classing. Most of the design of Chombo follows the work of Gamma *et al.* [GHJ+95]. Base classes are defined as pure virtual *Protocol* classes. Applications interact with the Chombo framework through *Template Method* and *AbstractFactory* design patterns. Most of the process of turning a one-off application into a reusable package involves factoring out these design patterns from the application.

In the work described here, we use the goal of developing applications codes to solve specific problems in microgravity and star formation as a focus for extending Chombo.

These applications codes will define the requirements for extending the various layers in the Chombo framework. These requirements include: support for volume-of-fluid discretizations of problems with moving interfaces; methods for representing the dynamics of particles suspended in a fluid; and integration with high-performance parallel iterative solvers for variable-coefficient elliptic problems arising in radiation diffusion and heat conduction. These will all be developed in the context of the existing Chombo architecture that supports AMR for time-dependent problems involving coupled elliptic / parabolic / hyperbolic partial differential equations, including refinement in time.

We will also deliver tools for performing data analysis and visualization for structured AMR data, called ChomboVis. ChomboVis is a tool that operates with structured AMR data in its native form (i.e. it does not flatten the data to a uniform or unstructured grid representation). This enables a much truer representation than commercial software and provides a performance gain. Since this approach to data analysis and visualization is closely tied to the underlying data representation, we will provide a standard API to I/O for block-structured AMR data, based on the HDF5 library developed at NCSA. This interface will be based on a framework-neutral method for passing aliases for AMR data between libraries and languages using opaque handles that is being developed in the AMR Common Component Architecture Forum provided using the HDF5 library developed at NCSA. The visualization and analysis package itself will be implemented using public domain tools: the C++ graphics library VTK and the scripting language Python. The powerful scripting capabilities in Python will allow us to provide a more general and robust tool that combines analysis and visualization capabilities.

1.3 Organization and Responsibilities

1.3.1 Project Personnel

Dr. Phillip Colella (PI) Dr. Colella will be providing the overall technical leadership on the the project. This includes being the lead on the algorithm and software design, as well as overall project management. He will also serve in the role as algorithm quality assurance officer for the project.
Brian Van Straalen will be taking the software engineering lead on this project. He will also serve in the role as software quality assurance officer for the project.
Dr. Daniel Martin has been developing incompressible Navier-Stokes AMR solvers using the Chombo framework. He will develop the coupling of such solvers to a blob-projection method for representing particles, and the extension to multi-fluid incompressible flows.
Noel Keen will be primarily responsible for performance and interoperability issues for the codes being developed here.
Theodore Sternberg will be primarily responsible for the development of the ChomboVis visualization and analysis tools.

1.3.2 Interfacing Groups

Microgravity. Dr. Emily Nelson of the Glenn Research Center and Dr. Mohammed Kassemi of the National Center for Microgravity Research will be collaborating with the with the LBNL development team on the design, testing and evaluation of the framework for the microgravity applications.
Star Formation. Dr. Richard Klein and Professor Christopher McKee of the UC Berkeley Astronomy Department, will collaborate in the design, evaluation, and validation of the framework for the star formation applications.
Visualization and Data Analysis. our points of contact for visualization and data analysis users in the NASA community are given as follows: Dr. Kevin Olson (NASA GSFC - Paramesh code) and Prof. Tamas Gombosi (U. Michigan).
Interactions between the development and the interfacing groups. As a baseline, all of the collaborators listed above will be on the Chombo and/or ChomboVis developers' mailing list, as appropriate; will have secure access to the CVS repositories at LBNL; and will be able to submit bug reports using the TTPro bug-tracking software. CVS access will provide a mechanism for disseminating both code updates and design documents. The star formation group will have students co-located with the members of the Chombo development team in office space at LBNL, and will meet formally with the developers no less than a quarterly basis. Representatives of the Chombo microgravity development team will also visit NASA Glenn Research Center on a quarterly basis. Finally, members of the ChomboVis development team will visit the principal stakeholder representatives listed above as required to assist in installation and training, as well as assisting Paramesh users at various sites as coordinated by Dr. Olson.

2 Statement of Problem

In the area of multi-phase flows in microgravity environments, we will address the problems arising in two different regimes: free-surface flows, and flows with a dispersed particulate phase. In free-surface flows, there are typically two different media separated by an interface, across which mass and energy may be transferred; in addition, the surface itself may exert forces on the fluid (surface tension). For both of these classes of systems, the flows that must be modeled are typically incompressible or low-Mach-number, with strong viscous effects and time-varying body forces (g-jitter). In the case of a free surface between two fluids, it is necessary to represent the geometry and dynamics of the free surface itself, as well as large discontinuous changes in the fluid properties (pressure and density) at the surface. For dispersed particle systems, large-scale motions of fluid-particle systems are represented by a collection of point particles moving with the fluid, with the drag coupling between the fluid and the particles represented as forcing terms in the fluid equations. In both of the applications discussed here, adaptivity is an essential requirement, with the maximum resolution required in the neighborhood of the free surface or particles.

In the area of astrophysics, the principal target application is the formation of low-

mass stars from magnetized molecular clouds. The models for low mass star formation include the equations of magnetohydrodynamics in 3-D with ambi-polar diffusion. These equations must be solved self-consistently with the equations of radiation transport (in the flux-limited diffusion approximation) and self-gravity. There are also extreme variations in length scale inherent in collapse (10^5 in spatial scale and 10^9 in density) which make adaptive mesh refinement a critical requirement of the calculation.

3 Technical Approach

3.1 Assumptions and Constraints

The technical assumptions and constraints are best understood in terms of the algorithmic components out of which we will assemble the complete applications.

Discretization Approach. For all of these problems, the underlying discretization approach will be based on the predictor-corrector methods in [BCG89, Col90] for solving coupled hyperbolic / parabolic / elliptic problems. The method is second-order accurate in space and time, and is based on the use of a higher-order Godunov method for approximating the hyperbolic terms, although other methods can be used.

Adaptive Mesh Refinement. We will use the extension of this approach to AMR applications described in [ABC+98, MC00a, PC00]. In this approach, refinement in time is applied to all of the equations. We perform multilevel synchronization solutions for the elliptic and parabolic subproblems at times at which the solutions at multiple levels of refinement coincide, and use a lagged correction from the multilevel solution at the other intermediate times. This is necessary to maintain uniform localization of the solution error in the presence of elliptic and parabolic terms, without having to solve equations on the full AMR hierarchy at every time step at the finest level.

Multiphase Effects. In the microgravity area, we need to be able to represent free boundaries, for which PDE's are being solved on both sides of the boundary, whose location and shape is changing as a function of time and is computed as part of the solution. We will use a volume-of-fluid representation of the discretized solution near the boundary. In this approach, the surface is represented by its intersection with an underlying rectangular grid. This leads to a natural, finite-volume discretization of the PDE on irregular control volumes adjacent to the boundary. Finally, we will provide both level-set and volume-of-fluid options for representing the surface itself on the finite difference grid.

To represent particles in an incompressible fluid, we will use a particle-in-cell approach. In this approach, the particles impose a localized force on the fluid. If the particles are of sufficiently low mass, one can use a version of this method based on the assumption that the particles are moving with the fluid [Pes82]. Otherwise, the particles have their own velocities, and are coupled to the fluid via drag terms. This method is a powerful and flexible method for representing a variety of interactions of discrete mechanical components with a fluid, including forces coming from the interaction of the particles with one another. We will use a method for interpolating the force on the fluid from the

particles developed by Cortez and Minion [CM00], which is based on the method of local corrections algorithm of Anderson [And86, ABC94]. In this approach, the effective force on the fluid can be specified independent of the mesh spacing, making it much easier to apply in an AMR setting.

The algorithmic approach described here leads to a software environment assembled from the following components:

- Explicit finite difference operators defined on unions of rectangles.
- Operators that implement the coupling between different unions of rectangles in a nested hierarchy.
- Implicit solvers for well-behaved discretizations of elliptic PDEs defined on unions of rectangles.

This collection of algorithmic components maps naturally into the following layered architecture for Chombo.

Layer 1: Classes for representing data and computations on unions of rectangles, including a mechanism for managing the distribution of rectangular patches across processors in an SPMD distributed-memory execution model, and an interface to Fortran for obtaining acceptable uniprocessor performance.

Layer 2: Classes for representing inter-level interactions, such as averaging and interpolation, interpolation of coarse-fine boundary conditions, and managing conservation at coarse-fine boundaries.

Layer 3: Classes that implement algorithms on AMR data, such as the Berger-Oliger time-stepping algorithm and AMR multigrid.

Layer 4: Implementations of specific applications or classes of applications using these tools, such as a Berger-Oliger algorithm for hyperbolic conservation laws or for incompressible flow, and AMR multigrid for Poisson's equation.

Utility Layer: Support for problem setup, I/O, and visualization that leverages off of existing de-facto standards, such as Chombo I/O, which is built on top of HDF5, and the ChomboVis visualization tool, built on top of VTK.

These algorithmic and software components will be available for both single phase problems, as well as in extensions to the numerical representations arising in multiphase flow problems, i.e. particles and volume-of-fluid methods for free boundaries.

3.2 Development Environment

3.2.1 Hardware Platforms

Day to day development work is performed on 80*86 microprocessor based workstations running Linux. Parallel development is performed on Halem: Halem is the NCCS Compaq

AlphaServer SC45 System and it currently consists of 416 user-available processors. The halem processors are clustered into 104 symmetric multiprocessor nodes (4 processors per node). An additional set of 12 nodes is allocated as system and spare nodes. Halem is a hybrid system in the sense that memory is distributed among nodes, but within a node memory is shared.

3.2.2 Software Platform

Chombo and ChomboVis are built using the *GNU make* system and utilize *perl* for some code transformation and wrapping functionality.

Chombo consists of a mixture of C++ and Fortran 77 code. We have been regularly using the GNU C++ and fortran compilers (gcc 2.95.3 and 3.1.1) as well as KAI and Intel compilers. Our code is ISO C++ standard compliant and is compiled and run regularly on IBM SP3 with xlC/xlf and Sun CC/f77 platforms.

We utilize *MPI* for message passing based parallelism. I/O is handled using *HDF5* (including advanced parallel I/O capabilities and out-of-core functions).

ChomboVis utilizes *Python* for it's main development, with C++ compiled code for CPU intensive operations. ChomboVis runs independently from the Chombo applications. Initially, it will run as serial code on workstations. As part of Milestone O4.3, we will develop a SMP parallel version based on threads.

Portability issues. Chombo applications will run on any Unix platform which for which there is available standards-compliant C++ and Fortran 77 compilers and ports of MPI and HDF5. ChomboVis will run on any workstation that supports VTK3.2 and Python. In particular, although the software development for this project is being carried out on the specific hardware platforms described above, we are using no platform-specific hardware or systems features, other that the availability of widely-accepted standard software.

3.3 Activities, Tools, and Products

3.3.1 Overview

We will develop the following code capabilities for the two applications being supported here.

• Microgravity research. We will extend the incompressible AMR code developed by ANAG to add two sets of simulation capabilities. One is an interface-tracking method for a vapor-liquid system with a sharp interface, with surface tension and mass transfer at the interface. The other is a code that simulates the interaction of the fluid dynamics with suspended particles. These will require the development of free-boundary elliptic and hyperbolic AMR solvers, based on the ideas in [BCW91, JC98, MC00b]. The particle methods will use the existing grid solvers for incompressible flow, coupled to new particle representations based on the ideas in [CM00].

• Star formation. The code for this area will be a self-gravitating MHD code. We will also provide interoperability with the existing AMR code base, developed jointly at UC Berkeley and LLNL. In particular, we will provide mechanisms for using the radiation diffusion solvers developed at LLNL within our framework, and support for calling the various Chombo solvers from the existing LLNL and UC Berkeley AMR codes.

In the area of visualization and analysis tools, we will develop the following capabilities.

• Visualization tools. ChomboVis will provide a set of tools for representing three-dimensional time-varying data visually in a workstation environment. These include isosurfaces, contour plots of slices, streamlines and other representations of vector and tensor data, volume rendering, and the ability to browse floating-point values in spreadsheets. We will also provide presentation/publication level graphics, with titles and arbitrary labeling, axis systems, cut away isosurfaces, color bars, and direct postscript output.

• Analysis tools. ChomboVis will be migrated to a Python interpretive system. Structured AMR data structures will then be mapped to new data structures. The Python interpreter will then be exposed in a console (much like the interpreter found in Matlab). This will allow users to define in an interactive session new derived data, which can then be displayed or saved.

• Support for Terascale data. ChomboVis will provide a variety of mechanisms for analyzing and visualizing data that exceeds the RAM capacity of a deskside workstation. It will support out-of-core data manipulation, through the use of HDF5 hyperslab access technology and the pipeline graphics model that will enable the manipulation and display of data sets that greatly exceed a processor's available RAM. ChomboVis will support for batch processing, providing a capability of non-interactive processing of graphics data, either for the purpose of generating visualization data, or for applying analysis tools for data mining. Finally, we will develop an SMP parallel version of ChomboVis based on multithreading to provide higher-performance within the VTK framework.

It will be possible to use ChomboVis in three distinct modes: interactively with a GUI for visualization; interactively with a combination of a GUI and the Python command-line interface, for visualization and analysis; and batch processing of Python scripts (e.g. for generating animations).

3.3.2 Design Methodology

The Spiral design approach [McC96] has a high degree of schedule visibility. On each run through a design cycle, a working, running executable program is produced and baseline performance measures are taken. At the next level of improved functionality, the previous spiral baseline code is optimized. Each cycle adds a higher level of modeling capabilities.

Initially, each subsystem will compile to its own application (space physics, microgravity, star formation). After the first design cycle of adding functionality, a sub-system goes

through the design work of becoming a *package* including integrating into the Chombo build architecture and following our coding standards. Packages have the added requirement of interoperability with other packages at the same abstraction level.

For each major increment in capability, a spiral design cycle in the present project proceeds in two phases.

• In collaboration with the applications stakeholders, we will develop the necessary software components and integrate them to provide an initial implementation of a new simulation capability. New applications are benchmarked for physics fidelity, for performance, and for design space flexibility. These applications are incorporated as software packages in the structured AMR framework. Distribution is centralized and the new packages are subjected to the framework software engineering process (regression testing, porting, source code control, etc.). This step corresponds to an **interoperability** milestone.

• Performance tuning of the benchmark code is carried out and user feedback is obtained regarding the physics fidelity and design space flexibility of the code. The the performance enhancements and other improvements addressing the deficiencies identified in the process are migrated to the framework. In addition, appropriate reusable components from the package are factored out and moved into lower layers to increase their reuse. This corresponds to **code improvement** milestones.

At the beginning of each design phase, we will also evaluate user feedback regarding the capabilities developed in previous design cycles, and schedule framework improvements to those capabilities as appropriate. As discussed above, the pricipal stakeholders will have continuous access to the latest versions of the code, and will be able to provide continuous feedback via access to TTPro.

3.3.3 Testing Methodology

There are three types of testing that are appropriate for the kinds of scientific software under development here.

- **Unit Testing**. Unit tests are written for individual classes in Layers 1-3 and and the Utility Layer of the Chombo software architecture. They are designed to test that the class is implemented as specified in the documentation.
- **Verification Testing**. These tests are written for solver and application classes developed in Chombo. They test whether the algorithm is implemented as specified in the algorithm design documentation. Typical verification tests include grid convergence studies for both truncation error and solution error, conservation and freestream-preservation testing, and checks of independence of the computed solution on the number of processors used.

- **Validation Testing**. These tests are written for applications classes developed in Chombo, They are intended to test the fidelity of integrated simulation capabilities on realistic examples of stakeholder applications. Validation testing will be performed primarily by our applications stakeholders, in close collaboration with the development team. Such testing is performed after interoperability milestones are met, but prior to the release of software under the code improvement milestones.

In standard software engineering terms, our verification and validation testing corresponds to system testing. Due to the small size of the development team, integration testing is not an issue.

In what follows below, we give detailed examples of our testing process in Chombo and ChomboVis. However, these examples are not exhaustive, and do not discuss in detail the specific tests for the new software being developed. The specification of these tests is a substantial undertaking, and constitute deliverables in the milestones given below.

Testing for Chombo

Each design unit in Chombo infrastructure is accompanied by a set of unit tests. Most unit tests are specific to a particular C++ class. For instance for class IntVectSet, a small sample of the 700 lines of unit test code for this class:

`Chombo/lib/test/BoxTools/testIntVectSet.cpp`

```
IntVectSet ivs;
IntVect a(-1*IntVect::Unit), b(0*IntVect::Unit);
ivs |= a;
if(!ivs.contains(a))
  { cout << indent << pgmname
         << ": failed -1 -1 -1 add test"
         << endl;
    eekflag = true;
  }
if(ivs.numPts() != 1)
  {
    cout << indent << pgmname
         << ": failed single IntVect count test"
         << endl;
    eekflag = true;
  }
```

All unit tests have a return code of zero on success. They are all run by the makefile target "tests" in the configuration of the code that a user is using:

```
>make tests DIM=3 DEBUG=FALSE
>make tests DIM=2
.
.
.
```

Regression Testing

The tests are run on an array of machines and compiler settings and configurations by another set of Perl scripts that catalog the failures and monitor return codes and run-times. Typical output for a given machine looks as follows:

```
date=Tue Jun 11 14:52:32 PDT 2002
unamea=Linux ford.lbl.gov 2.4.7-10 #1 Thu Sep 6 16:46:36 EDT 2001 i686 unknown
arch=Linux
 compiling in serial
CATFISH_DIR=/export/users/noel/catfish
  1 2D dbg0 p0   icc   g77 CL=11  24( 0) EXB=1 EXR=11111111111111111111111111 89.8%
  2 3D dbg0 p0   icc   g77 CL=11  24( 0) EXB=1 EXR=11111111111111111111111111 91.3%
  3 2D dbg1 p0   icc   g77 CL=11  24( 0) EXB=1 EXR=1111111111111111 94.3%
  4 3D dbg1 p0   icc   g77 CL=11  24( 0) EXB=1 EXR=1111111111111111 90.1%
  5 2D dbg1 p0   icc pgf77 CL=11  24( 0) EXB=1 EXR=1111111111111111 94.5%
  6 3D dbg1 p0   icc pgf77 CL=11  24( 0) EXB=1 EXR=1111111111111111 98.2%



Build01 dim=2 debug=0 mpi=0
   cxx=    icc 6.0 cfl=-O3 -ip -unroll -I/opt/intel/compiler60/ia32/include
   fc =    g77 2.95.3 ffl=-O2 -malign-double -funroll-loops -fno-second-underscore
                        compile time   cpu utilization
  lib build     OK=1    232.008166     91.27%
  test build    OK=1    162.092177     93.11%
  example build OK=1    427.923115     95.14%
  Number of tests compiled: 24  (0 had issues)
Runs:                                                                    time(cpus)
  poissonSolve            t.AMRPoisson.testgrids.2d           OK=1         3.414
  poissonSolve            t.AMRPoisson.2d                     OK=1         2.719
  poissonSolve            t.AMRPoisson.2comp.2d               OK=1         3.943
  amrGodunovSplit         t.AMRGodunovSplit.wave2d            OK=1         1.987
  amrGodunovSplit         t.AMRGodunovSplit.ramp2d            OK=1         5.741
  amrGodunovSplit         t.AMRGodunovSplit.explosion2d       OK=1         4.061
  amrGodunovUnsplit       t.AMRGodunovUnsplit.wave2d          OK=1         0.253
  amrGodunovUnsplit       t.AMRGodunovUnsplit.ramp2d          OK=1         1.468
  ns                      t.ns2d.ABCHW.64                     OK=1         8.500
  ns                      t.ns2d.ABCHW.64-128                 OK=1         3.961
  heat                                                        OK=1         0.060
  heat                                                        OK=1         0.043
  heat                                                        OK=1         0.061
  amrio                                                       OK=1         0.083
  ugio                                                        OK=1         0.034
  richardsonExtrap                                            OK=1         0.187
  poissonSolve            b.AMRPoisson.testgrids.2d           OK=1        27.220
```

```
  poissonSolve            b.AMRPoisson.2d                    OK=1        17.275
  poissonSolve            b.AMRPoisson.2comp.2d              OK=1        31.501
  amrGodunovSplit         b.AMRGodunovSplit.wave2d           OK=1        13.311
  amrGodunovSplit         b.AMRGodunovSplit.ramp2d           OK=1        25.941
  amrGodunovSplit         b.AMRGodunovSplit.explosion2d      OK=1        56.354
  amrGodunovUnsplit       b.AMRGodunovUnsplit.wave2d         OK=1         9.547
  amrGodunovUnsplit       b.AMRGodunovUnsplit.ramp2d         OK=1        14.906
  ns                      b.ns2d.ABCHW.64                    OK=1         8.809
  ns                      b.ns2d.ABCHW.64-128                OK=1        31.760

Build02 dim=3 debug=0 mpi=0
   cxx=    icc 6.0 cfl=-O3 -ip -unroll -I/opt/intel/compiler60/ia32/include
   fc =    g77 2.95.3 ffl=-O2 -malign-double -funroll-loops -fno-second-underscore
  lib build     OK=1    230.000309     93.77%
  test build    OK=1    147.988874     96.37%
  example build OK=1    345.515422     96.59%
  Number of tests compiled: 24  (0 had issues)
Runs:
  poissonSolve            t.AMRPoisson.testgrids.3d          OK=1         7.147
  poissonSolve            t.AMRPoisson.3d                    OK=1        12.124
  poissonSolve            t.AMRPoisson.2comp.3d              OK=1        19.572
  amrGodunovSplit         t.AMRGodunovSplit.wave3d           OK=1        20.469
  amrGodunovSplit         t.AMRGodunovSplit.ramp3d           OK=1        22.985
  amrGodunovSplit         t.AMRGodunovSplit.explosion3d      OK=1        13.047
  amrGodunovUnsplit       t.AMRGodunovUnsplit.wave3d         OK=1        17.149
  amrGodunovUnsplit       t.AMRGodunovUnsplit.ramp3d         OK=1        36.870
  ns                      t.ns3d.ring                        OK=1        19.583

etc.
```

Full regression testing of the Chombo libraries is performed weekly.

ChomboVis Testing

ChomboVis comes with two testing scripts and, orthogonal to these, a class-by-class unit testing mode.

Integration tests

cmdline_test.sh fires off ChomboVis with various legal combinations of its command line options. The text output goes to a temporary file. If that file differs in any way from canonicals/cmdline_test.txt, then cmdline_test.sh reports that an error has occurred.

api_test.sh tests most (ideally it should be all) the methods comprising the external API, that is the programming interface for user scripts. api_test.sh does its work by launching ChomboVis – twice, first on a 2D file, then on a 3D file – with the user_script option indicating examples/api_test.py. It is api_test.py that actually calls the API methods. Like cmdline_test.sh, api_test.sh compares its text output to the contents of a file in the canonicals directory. In addition, api_test.sh records an image of the VTK window,

and compares that to canonicals/api_test2D.ppm or canonicals/api_test3d.ppm. When running api_test.sh, be sure to drag your VTK window quickly away from the dialogs that pop up around it, lest pieces of those dialogs show up in the saved image.

If you have a ~/.chombovisrc, it will not affect these tests as cmdline_test.sh and api_test.sh say ignore_rc=1 on their command lines.

Unit tests

Independently of cmdline_test.sh and api_test.sh, ChomboVis can be instructed, from the command line, to cause its own classes to test themselves. The relevant command line options are test_class and test_mode.

test_class indicates which class to test. Any of the singleton classes (e.g. vtk_grid, vtk_slice, control_stream, etc) may be so tested. (For the entire list, search for self.init_data in chombovis.py.) What then happens is that if the indicated class has a method called UnitTest, ChomboVis will invoke it. At this writing, we have not done much with the various classes' UnitTest methods. Thus, the testing framework is there, but the tests are not.

When you invoke the test_class option, ChomboVis will print the names of all the classes it constructs, up to and including the one you indicated with test_class. This list comprises the minimal set of classes necessary to construct the "test class" you indicated. Thus, if you say test_class=vtk_slice, you will see this:

```
info:chombovis:_makeClasses():Constructing  saved_states
info:chombovis:_makeClasses():Constructing  vtk_vtk
info:chombovis:_makeClasses():Constructing  vtk_data
info:chombovis:_makeClasses():Constructing  vtk_cmap
info:chombovis:_makeClasses():Constructing  vtk_slice
********** vtk_slice:unitTest()
```

If you want ChomboVis to construct not the minimal set of classes but the entire set, then put test_mode=max on your command line.

If you have a ~/.chombovisrc, and you want to use test_class, then it is a good idea to add ignore_rc=1 on your command line, lest your ~/.chombovisrc invoke functionality from some class that is higher-level than your "test class".

3.3.4 Software Products and Documentation

The deliverables in this project are described in detail in the Milestones and Schedules section below. However, the deliverables in this project can be broken down into four categories, corresponding to the different phases in the spiral design process. For each category, we provide here a description, and list the milestones in section 4.2 corresponding to that category.

- **Requirements analysis**. In this phase of the project, we determine the capabilities required for each major component of the project (the two applications, and the

visualization and analysis tool). For the two applications, the analysis has been performed and documented in the proposal, and in an abbreviated form in section 2 above. For the visualization and analysis tools, this analysis will be performed as part of milestone O1.1. Products in this area corresponds to a documentation deliverable.

- **Detailed Algorithm and Software Design**. In this phase of the project, we document in detail the principal algorithm components that will be required to meet the project goals; the API's for the principal software components; and requirements testing matrix for each major component. Milestones: E, H, O2.2, O4.3. Products in this area corresponds to a documentation deliverable.

- **Initial Implementation**. This class of tasks includes a baseline implementation of a new software capability, along with the completion of unit testing, verification testing, and baseline performance measurements. Milestones: E, I, J, O2.1, O2.2, O3.1, O3.2, O4.2. Products in this area includes both software deliverables (applications made available to the stakeholder community) and documentation deliverables (complete requirements traceability matrix for unit and verification testing; baseline performance document; revisions to algorithm and software design documents; preliminary user's guide).

- **Phase II Implementation**. This class of tasks is the point at which we incorporate improvements in the software, and release a new version. The drivers for these improvements will include performance enhancements done in response to the baseline measurements performed in the initial implementation; feedback from the stakeholder community on useability and performance; and improvements in model fidelity in response to validation testing. Milestones: F,G, O2.3, O3.3, O5.1, O5.2. Deliverables in this area include both software (applications made available to stakeholder community) and documentation (completed requirements traceability matrix for validation testing; revised performance document; revisions to algorithm and software design documents, users' guide.

- **Comprehensive Documentation of Software Internals**. This task will enable maintenance of the software after the end of the project. It will mainly consist of a Doc++ / Doxygen annotation of the C++ header files, combined with the documentation produced in the other parts of the project. In addition to the documentation, we will also deliver a comprehensive set of unit, verification, and validation tests that will also facilitate maintenance of the the software. Milestone: K.

4 Management Approach

4.1 Milestones

Milestone A: Software engineering plan ($50K - delivery 7/1/02)

We will complete and deliver to NASA the documents describing the software engineering plan, the Configuration management plan, and the quality assurance plan, following the templates provided by NASA. We will also select and install project management software.

Milestone E: Baseline code ($140K - delivery 8/31/02)

We will perform baseline measurements of an AMR incompressible Navier-Stokes code (AMR-INS), developed using the existing Chombo framework with DOE research funds. This code will be the starting point for all further microgravity development. In addition, it uses many of the software components that will be used for the star formation problem (AMR elliptic solvers, Berger-Oliger time stepping scheme, averaging and interpolation). We will deliver to NASA the design documents and requirements document for the baseline code, as well as a requirements traceability matrix for the verification, validation, and performance tests for the baseline code. The baseline code, in the form of documented source code, will be made publicly available on the project web site.

ChomboVis Milestone O1 ($90K - delivery 8/31/02)

1. We will poll the potential users of block-structured AMR software in the NASA ESS community, including users of PARAMESH and BATS-R-US, to determine their visualization needs. Based on this information, we will write a requirements document and develop a requirements traceability matrix for ChomboVis.

2. We will complete the port of ChomboVis from its current Tcl scripting environment to the Python scripting environment. This port will include a session restart capability not currently available in ChomboVis. Documented source code will be made publicly available on the project web site.

Milestone B: First annual report ($47K - delivery 8/31/02)

Milestone H: Interoperability / community delivery policy. ($90K - delivery 11/30/02)

We will develop and deliver to NASA initial versions of the design and requirements documents for all of the principal interoperability milestones in the project, including ones specifying test plans and procedures for verification and validation of all increments in modeling capability. This work will be done in collaboration with our applications stakeholders in microgravity and star formation.

Milestone F: First capability milestones ($140K - delivery 10/30/02)

We will improve the performance of the AMR-INS over the baseline obtained in milestone E by a fivefold reduction in time to solution, with 30% less memory usage. No more than a factor of two of the improvement in CPU time will be obtained by increasing the number of processors. The resulting improved AMR-INS code will be made publicly available on the project web site as documented source code.

ChomboVis Milestone O2 ($90K - delivery 11/30/02)

1. We will implement I/O interface for coupling ChomboVis with Paramesh, BATS-R-US, and other NASA AMR users as determined in milestone 1.

2. We will implement support for visualization of data with multiple centerings (cell-,face-,edge-, and node-centering).

3. We will design the analysis framework for ChomboVis, including data analysis tools and scripting interface API's, as well as the modification of the ChomboVis internals. We will provide to NASA revised design and requirements documents based on this design.

4. We make a release version of the Python-based ChomboVis available at the website, including an initial version of the users' guide. The documented source code and and other documentation, including that for the capabilities in this milestone, will be made publicly available on the project web site.

Milestone I: First interoperability milestone ($280K - delivery 6/30/03)

1. We will develop an extension of the AMR-INS code to support suspended particles in an incompressible fluid, based on the extension of the immersed boundary / local corrections algorithm to an adaptively-refined mesh. We will also develop the Layer 1 and Layer 2 support libraries required to implement AMR multifluid algorithms. These include software for representing the motion of the interface and for constructing the local geometry required to compute the finite volume discretizations on either side of the front; array classes and distributed data on unions of rectangles; and inter-level interpolation, boundary interpolation, and inter-level conservation tools. The development of Layer 1 and 2 support for particles and for multifluid interfaces will be performed jointly with the SciDAC ISIC effort at LBNL.

2. We will develop a hyperbolic AMR code for general systems of conservation laws based on unsplit Godunov / PPM methods. Recent results of Stone indicate that such methods are essential for the MHD simulations required for the star formation problem. We expect them also to greatly enhance the accuracy of the coupling to non-hyperbolic physics (radiation, conduction, self-gravity).

3. We will deliver to NASA revised versions of the design and requirements documents for these milestones (the initial versions were delivered in milestone H). We will perform verification, validation, and baseline performance measurements of the AMR-INS / particle code and the unsplit AMR hyperbolic code developed under these milestones, using the test suite developed in milestone H, and report the results in the requirements traceability format.

ChomboVis Milestone O3 ($90K - delivery 6/30/03)

1. We will complete a prototype implementation of the analysis tools, including a command line interface to the scripting environment.

2. We will develop and release an animation capability.

3. We will release a new version of the visualization tools that include the capabilities described above, along with an updated users'guide. The documented source code and other documentation, including those for the capabilities in this milestone, will be made publicly available on the project web site.

Milestone C: Second annual report ($47K - delivery 7/31/03)

Milestone G: Second capability milestones ($140K - delivery 11/30/03)

1. We will improve the performance of the new components developed in milestone I for representing particles by a fivefold reduction of time to solution and a 30% decrease in memory usage, over the baseline measurements performed in that milestone. We will also obtain scaled speedup of 75% efficiency up to 128 processors for the AMR-INS / particle code, and scaled speedup with 75% efficiency on 1000 processors for the unsplit AMR hyperbolic code.

2. The AMR-INS particle code and the unsplit AMR hyperbolic code developed under milestone I will be made publicly available on the project web site as documented source code, including an initial version of the users guide. We will deliver to NASA as part of the release updated versions of the requirements, design, and test documentation described in Milestone I.

ChomboVis Milestone O4 ($90K - delivery 11/30/03)

1. We will complete a revised version of the analysis tools developed in milestone 3, based on feedback from the NASA user community.

2. We will develop high-fidelity vector data display capabilities for multiply-centered data; batch processing capabilities; and presentation graphics capabilities.

3. We will design tools for handling terascale data, including out-of-core capabilities and SMP parallelism.

4. We will release a new version of the visualization tools that include the capabilities described above, along with an updated users' guide. The documented source code and and other documentation, including that for the capabilities in this milestone, will be made publicly available on the project web site.

Milestone J: Second interoperability milestone ($280K - delivery 6/30/04)

1. We will develop a multifluid interface tracking code AMR-MFINS. This includes the development of multi-fluid elliptic and hyperbolic solvers (MFElliptic, MFHyperbolic); and the integration of these libraries to obtain a Layer 4 incompressible code for two-phase flow with surface tension.

2. We will develop a coupled hyperbolic AMR code to self-gravity (AMR-SG) and an MHD version of the hyperbolic code (AMR-MHD).

3. We will deliver to NASA revised versions of the design and requirements documents, as well as an initial version of the users' guides for these codes. We will perform baseline measurements of the AMR-MFINS, AMR-SG, and AMR-MHD codes using the test suite developed in milestone H, and report the results in the requirements traceability matrix format.

ChomboVis Milestone O5 ($90K - delivery 6/30/04)

1. We will complete development of the Terascale data capability.

2. We will release the final version of ChomboVis, including the terascale capability. We will deliver to NASA documentation including a maintenance maual, updated requirements and test documents to reflect portability tests, and updates to the users guide and to the requirements and design documents to reflect the as-built system. We will release a new version of the visualization tools that include the capabilities described above, along with an updated users'guide. The documented source code and other documentation, including that for the capabilities in this milestone, will be made publicly available on the project web site.

Milestone D: annual report: ($37K - delivery 7/31/04)

Milestone K: Customer delivery: ($140K - delivery 11/30/04)

1. We will improve the performance of the AMR-MFINS code developed in milestone J by a fivefold reduction of time to solution, and a 30% decrease in memory usage, over the baseline measurements performed in that milestone. No more than a factor of two of the improvement in time to solution will be obtained by increasing the

number of processors. We will also obtain scaled speedup of 75% efficiency up to 128 processors for both the AMR-MFINS code and the self-gravity AMR hyperbolic code.

2. We will make publically available the codes from milestone J on the project web site as documented source code. We will deliver documentation including a maintenance manual, updated requirements and test documents to reflect portability tests, and updates to the users guide and to the requirements and design documents to reflect the as-built system.

Milestone D: Final report: ($10K - delivery 11/30/04)

4.2 Metrics

For the simulations software being developed here, are two classes of metrics that are appropriate, corresponding to the interoperability and capability milestones described above. For the capability milestones, the metrics will be the increase of performance over the baseline results. For the interoperability milestones, the metric for success will be the ability to simulate new physics. The specific goals for these simulations will be specified for the whole project as part of the deliverables in Milestone H.

For the visualization and analysis tools, the metrics for success will be the delivery of new tools required by the users, as specified in the report to be submitted under Milestone O1.1.

4.3 Risk Management

For the simulations software being developed here, there are two components of risk, both related to the complexity of the algorithms being implemented. At the algorithmic level, we are are combining a set of components (AMR, projection methods for incompressible flow, particle methods, embedded boundary / volume-of-fluid discretizations) which, by themselves, have been well-validated in their individual domains, but when combined, represent uncharted territory, particularly with respect to numerical stabilities in the coupled system. Furthermore, each of these components are high-resolution, second-order accurate methods, and are being combined in a way that preserves those properties. In particular, if there are numerical instabilities in the coupling, there are no $O(\Delta x)$ damping terms that are often used in traditional multiphysics codes to stabilize such couplings. The second component of risk comes from software complexity that follows from the algorithm complexity. The algorithms described here involve complicated combinations of regular and irregular calculations, and elaborate data and control structures. One example of this is the range of data representations that are used to implement these calculations: locally uniform grids, block-structured nested grids, particle-in-cell data, and sparse multivalued grid variables for embedded boundaries. With such complex software, there is the risk of

producing incorrect code, as well as the risks associated with getting good performance (both serial and parallel) for such calculations.

There are two aspects to our plan that are specifically intended to mitigate these two risks. The first is the use of a layered component architecture, with software components corresponding to algorithm components whose stability and accuracy properties are well-understood. This approach allows us to build and test components separately, using the test suite described in section 3.3.3. In particular, we have found that verification testing for convergence at the known rate is a fine discriminant for code correctness. Such a test regime gives us a high degree of confidence that the components are implemented correctly. Furthermore, if problems are identified later, they can be corrected in the component, and propagated throughout the software system in a systematic fashion. The second aspect of the plan is the front-loading of some of the highest-risk parts of the project in relatively early stages of the project. Specifically, we view the first capability milestone F as a stringent test of whether we will be able to meet the performance requirements of the project, and the interoperability milestone I as representing a significant portion of the new physical modeling capabilities. In each case, we will have sufficient time, if necessary, to rethink modify our approach to these critical issues within the time of the overall project.

For ChomboVis, the principal risk is meeting the needs of a diverse user base within the budget and time allowed. To mitigate this risk, we have frequent releases of new capabilities within the scope of the project, and early interactions and frequent interactions with our users. The key components to this strategy is milestone O1.1, in which we poll the users and develop a requirements document, and milestone O2.1, which will provide an I/O interface that will make ChomboVis available to a broad range of users.

A final risk specific to this project is that of being able to obtain all of the components of the software environment required for the project: specifically, the availability of standard-conforming C++ compilers and MPI and HDF5 ports on the high-end platforms procured by NASA. Our strategy for mitigating that risk is to make NASA project management aware of our requirements as they undertake the procurement process.

5 Product Assurance

5.1 Configuration Management

5.1.1 Configuration Control

CVS revision control All configuration control items are stored in the Concurernt Versions System (CVS) `http://www.cvshome.org/docs/` . The user can reference the CVS documentation for the specifics of this control process. All respositories are monitored by several people on every update from the development teams and each modifcation to each file requires a log entry to be created documenting the nature of the change. Other users and developers can access these updates through the `cvs update` mechanism (this

includes remote collaborators).

We will not go into too much length describing CVS operations. It will be noted that every single thing done on this project, including these reports, are stored and archived in the CVS system. All previous revisions and the entire development path are available for review.

TTPro Tracking System

for Defect and project tracking, we use the commercial software product *TTpro* (ref. `http://www.seapine.com/ttpro.html` . The tracking categories we use are

Product	Component
Chombo	Installation
	Documentation
	Code
	ChomboFortran
	Parallel
	Performance
EBChombo	Installation
	Documentation
	Code
	Parallel
	Performance
ChomboVis	Installation
	Documentation
	Code
	Performanmce
Applications	Incompressible Navier-Stokes
	Particle in Cell
WorkStations	Software
	Hardware
TTPro	

Each [Product,Component] combination has a single individual responsible for ensuring each added issue gets reviewed, categorized, and assigned. TTPro then tracks the history of each issue and can produce full and partial reports as per the state of the development effort. This tool was used to great effect in reaching the goal of zero known defects in Chombo for the recent 1.2 release.

Brian van Straalen will be responsible for overseeing configuration management issues.

5.1.2 Accounting

Most reporting is handled by the tools discussed in the previous section. TTPro has very elaborate reporting suitable for meetings, management, customers, and so forth at various

levels of detail

selecting *open:defects* produces a printable listing as a series of defects as such

06/13/02

```
  Defect No: 11
                                      Date Entered: 10/26/01
                                    Product: ChomboVis

                                      Entered by: Van Straalen, Brian
                               Component: Code

  Status: Open, assigned to Sternberg, Ted
                                Severity: Workaround
  Type: Feature Request
                                      Priority: Medium
                         Disposition: Open - Reviewed
  Reference:

  Summary: change of variable in spreadsheet
  Release Notes:
  Workaround:
-- Reported By ---
  Found by: Van Straalen, Brian
                                                 Date Found: 10/26/01
  Version Found: 3.2.2
  Description: Need to be able to change the displayed variable in a
  spreadsheet. And
  the spreadsheet should remain in the same place (same i,j,k).

  The workaround is to change variable, and then open a new spreadsheet
  for the same grid.

  P.S. If the selector is on, highlight the datum (in the spreadsheet) that
  corresponds to the precise point the selector hits in the selected plane
  of the selected box.
  Reproduced: Always
  Steps to Reproduce:
  User's Computer Configuration
  Other Hardware and Software:
  Attachments:
    none
```

:
more elaborate report filtering is available to us (after a considerable setup cost on our part).

Code and documents are tracked in this manner.

Once a week derelict (unreviewed or unassinged) defects are scanned for and reported in a group mailing to the offending individuals. The status of the workstations and development environment are brought to the weekly system administration meeting between the PI and the systems group. Once a month the current outstanding software defects are reviewed and prioritized along with current development efforts.

5.1.3 Storage and Handling

:

delivery of products

Deliverables will be made available on our NASA website: `http://davis.lbl.gov/NASA` . Documents will be made available in *Portable Document Format* (pdf). Source code will be made available as unix tape archive (tar file).

backups and retrieval

All digital material is archived using the latest archiving technology from the *High Performance Storage System (HPSS)* run by the *Nationanal Energy Research Scientific Computing Center (NERSC)* `http://hpcf.nersc.gov/storage/hpss/` . These are done on daily, weekly and monthly schedules for full and incremental backups. All documents, web pages, TTPro databases, source code respositories, and home directories are backed up. Since the NERSC HPSS facility is located in downtown Oakland, a distance of over five miles from the LBNL site, the backups will also be protected from potential catastrophic events at the LBNL site.

5.2 Quality Assurance

5.2.1 Coding Standards

All delivered software abides by the *ANAG Coding Standards* document included as separate document of this submission package. This includes instructions about source code documentation that are then utilized by *doc++/doxygen* to generate synchronized hyperlinked reference manuals.

5.2.2 Quality Assurance Activities

We will undertake the following quality assurance activities, coordinated with the milestones in the project.

- Design reviews. Formal design documents are early deliverables for the project (Milestones H, O1.1). They will be made available for the stakeholder community to review at that time.

- Tracking consistency between documentation and source code release. This will be done prior to the code releases representing interoperability milestones (E,F,G), capability milestones (I,J,K), and ChomboVis software release milestones O1-5.
- Ensuring tests are executed and problem reports written. The reports regarding testing are part of the deliverables for the interoperability milestones.
- Verification that problem reports are resolved and the tracking system updated. Outstanding problem reports on TTPro are reviewed weekly by the QA officers and discussed with the individuals to which they have been assigned, with new problems assigned to the appropriate developer.
- Monitoring coding standards. All codes are reviewed prior to release for conformance to coding standrds.
- Code walk-throughs and reviews. Prior to milestone releases, code reviews are held by the development team and the QA officers.

As indicated above, Brian van Straalen and Phillip Colella will act as quality assurance officers for the project.

References

[ABC94] A. S. Almgren, T. Buttke, and P. Colella. A fast adaptive vortex method in three dimensions. *J. Comput. Phys.*, 113(2):177–200, 1994.

[ABC+98] A. S. Almgren, J. B. Bell, P. Colella, L. H. Howell, and M. J. Welcome. A conservative adaptive projection method for the variable density incompressible Navier-Stokes equations. *J. Comput. Phys.*, 142(1):1–46, May 1998.

[And86] C. R. Anderson. A method of local corrections for computing the velocity field due to a distribution of vortex blobs. *J. Comput. Phys.*, 62:111–123, 1986.

[BCG89] J. B. Bell, P. Colella, and H. M. Glaz. A second order projection method for the incompressible Navier-Stokes equations. *J. Comput. Phys.*, 85(2):257–283, December 1989.

[BCW91] J. B. Bell, P. Colella, and M. Welcome. A conservative front-tracking for inviscid compressible flow. In *Proceedings of the Tenth AIAA Computational Fluid Dynamics Conference*, pages 814–822. AIAA, June 1991.

[CGM+] P. Colella, D. T. Graves, D. Modiano, D. Serafini, and B. Van Straalen. The Chombo software package for AMR applications. `http://seesar.lbl.gov/anag/chombo`.

[CM00] R. Cortez and M. Minion. The blob projection method for immersed boundary problems. *J. Comput. Phys.*, 2000. to appear.

[Col90] P. Colella. Multidimensional upwind methods for hyperbolic conservation laws. *J. Comput. Phys.*, 87:171–200, March 1990.

[GHJ+95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Grady Booch. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1995.

[JC98] H. Johansen and P. Colella. A Cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys.*, 147(2):60–85, December 1998.

[MC00a] D. F. Martin and P. Colella. A cell-centered adaptive mesh projection method for viscous incompressible flow. Technical report, Lawrence Berkeley National Laboratory, 2000. to appear.

[MC00b] D Modiano and P. Colella. A higher-order embedded boundary method for time-dependent simulation of hyperbolic conservation laws. In *Proceedings of the FEDSM 00 - ASME Fluids Engineering Simulation Meeting*, Boston, MA, June 2000.

[McC96] Steve McConnell. *Rapid Development : Taming Wild Software Schedules*. Microsoft Press, 1996.

[PC00] R. M. Propp and P. Colella. An adaptive mesh refinement algorithm for porous media flows. Technical Report LBNL-45143, Lawrence Berkeley National Laboratory, January 2000. submitted to J. Comput. Phys.

[Pes82] C. Peskin. The fluid dynamics of heart valves: Experimental, theoretical, and computational methods. *Ann. Rev. Fluid Mech.*, 14:235–259, 1982.