Incompressible Navier-Stokes Software Testing Plan

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

April 21, 2003

Contents

Chapter 1

Scope

The AMR incompressible Navier-Stokes (AMRINS) code developed for this project builds heavily on the `Chombo` infrastructure. [CGL+00] The software test plan outlined in this document will focus on the functionality developed for the Navier-Stokes code; since `Chombo` has its own software test plan, it is not necessary to provide for testing the functionality of the Chombo libraries themselves. Note, however, that since the software developed for the AMRINS project uses the `Chombo` functionality so extensively, changes and bugs in the Chombo libraries will tend to have effects on the AMRINS testing results. AMRINS developers are kept abreast of developments in `Chombo` through CVS notification (which sends e-mail whenever a change is made in the `Chombo` CVS version-control repositories), and through the ChomboUsers e-mail list.

1.1 System Overview

The AMRINS software implements an AMR algorithm for solving the incompressible Navier-Stokes equations. The algorithm used in this work is based on the inviscid algorithm presented in [MC00].

Chapter 2

Reference Documents

In addition to the `Chombo` design document [CGL+00] and the inviscid algorithm described in [MC00], we will also refer to the "Incompressible Navier-Stokes Software Design" and "Incompressible Navier-Stokes Requirements" documents.

Chapter 3

Software Test Environment

The AMRINS software, linked to the Chombo software libraries, current in the ANAG CVS repository on August 1, 2002 is being tested. As new functionality is added and current functionality is improved, testing will continue. It is expected that a given time, the AMRINS code will be in sync with the current state of the Chombo libraries.

This software is primarily intended for use on UNIX/Linux-based systems. In general, the makefiles used in both Chombo and AMRINS require GNU make (gmake). The software itself is designed to be run from a shell, with an inputs file providing run-specific inputs. For data output, the software uses hdf5, so the system must have hdf5-1.4.1 installed. The Chombo and AMRINS software is written in C++ and Fortran77, so working C++ and F77 compilers must be available. We generally use the GNU compiler: both gcc 2.95 and 3.1 have been successfully used to compile this code. In addition, the Chombo Fortran preprocessor uses PERL. If ChomboVis will be used to examine results, then it must be installed as well. ChomboVis additionally requires Python and VTK.

We have tested the AMRINS/Chombo combination in a variety of environments, with a variety of compilers. Table 3 lists the platforms and compilers we have successfully compiled and run the AMRINS code:

Testing is done by ANAG personnel, although collaborators have been useful for finding unintended functionality, primarily in the Chombo libraries themselves.

Platform	OS	C++ Compiler	Fortran Compiler
CRAY T3E	unicos	KCC 3.3d	Cray Fortran 3.5.0.4
IBM SP	AIX	KCC 4.0f, xlC 5.0.2.0	IBM XL Fortran 7.1.1.0
Pentium/AMD	Linux	gcc 2.95.3+,	g77 2.95.3+, PGI Fortran 3.3-2
		Intel C++ 6.0	Intel Fortran 5.0.1
Compaq	OSF	gcc 3.1	Compaq f77 X5.4A-1684-46B5P
Compaq	Linux	gcc 2.95.3	g77 2.95.3
SGI	IRIX	MIPS Pro CC 7.3.1.2m, gcc 2.95.3	MIPS Pro f90 7.3.1.2m

Table 3.1: Platforms and compilers on which the AMRINS code has been tested

Chapter 4

Test Identification

4.1 General Information

In general, the best way to test whether components are functioning properly is to do a convergence study. For example, in the case of a projection component, a velocity field is initialized on a series of meshes, each a factor of 2 finer than the last. The projection is applied to the velocity field, and then the divergence of the resulting velocity field is computed. If the projection component is properly implemented, the divergence should decrease at second-order rates.

4.1.1 Test Level

In general, most of the testing outlined in this document will be component testing. System-level testing will also be carried out on the entire AMRINS code. It is expected that integration testing is not necessary at this time, because of the small size of the design team.

4.1.2 Test Classes

In general, testing will be structured to evaluate correctness of the code. It is anticipated that since the next phases of code development will be focused on performance enhancement, that performance of the code will be monitored closely, so routine performance testing should be unnecessary, while testing for correctness will be important as changes are made to speed up the code.

4.2 Planned Tests

In this section, we outline the tests planned for the AMRINS code, broken down by functional algorithm unit. All testing codes are written in C++.

4.2.1 Test 1 – Advection

The second-order upwind method used for the hyperbolic advection pieces of the algorithm (steps 1a, 2a, and 3a in the pseudocode description of the algorithm) will be tested to ensure that it maintains second-order accuracy in space and time. This will be a functional test.

In the next phase of AMRINS development, we expect to factor this functional unit out of the `AMRNavierStokes` class in order to use the general hyperbolic tracing capability of the `PatchGodunov` class (currently in `Chombo/example/AMRGodunovUnsplit/src` in the `Chombo` distribution).

Once that is done, a testing module will be implemented to use a test advection problem to ensure that the advection code is second-order accurate and maintains that accuracy.

4.2.2 Test 2 – Advection-velocity (MAC) Projection

The CCProjector::levelMacProject function applies a staggered-grid projection to a staggered-grid set of advection velocities. Since the discretization used in this projection is not approximate, the divergence of the input velocity field after the projection is applied is solely dependent on the tolerance of the solver used. Because of this, it will be possible to do a single test of the projection, rather than a convergence study.

The testMacProjection code will initialize a velocity field on a representative disjoint set of grids. This velocity field will have a divergence-free component as well as a potential component. When this velocity field is projected by the `CCProjector::levelMacProjection` function, the divergence of the resulting velocity field should be at roundoff.

4.2.3 Test 3 – Viscous solver

Currently, the viscous solver is hard-coded in the `AMRNavierStokes` class. In the next phase of AMRINS development, we expect to factor the viscous solver as a functional unit out of the `AMRNavierStokes` class to be replaced by a generalized viscous solver class. In conjunction with this effort, a baseline advection-diffusion code is being developed. Smooth scalar fields advected by this code should maintain second-order accuracy in space and time. This is currently in development, and should be implemented by the next milestone report.

4.2.4 Test 4 – Cell-centered single-level projection

To test the `CCProjector::levelProject` function, the testCCProjector test code was written. A cell-centered velocity field is initialized to be a divergence-free field (in this case a vortex) to which a potential field is added. Since this projection discretization is approximate, applying the `levelProject` function should remove the potential field with a second-order accuracy. To test this, the projection is applied to a series of refined grids,

and the divergence of the resulting velocity field should converge to 0 at second-order rates.

4.2.5 Test 5 – Cell-centered multilevel projection

The composite cell-centered multilevel projection used in the `CCProjector::initialVelocityProject` and `CCProjector::doSyncProjection` should, like the single-level cell-centered projection in `CCProjector::levelProject`, remove a potential field from a cell-centered velocity field with second-order accuracy. The test for this function is similar to that test # 4, except it is done on a hierarchy of refined grids.

4.2.6 Test 6 – AMRINS system

The AMRINS code, when run on a smooth initial velocity field, should produce results which converge at second-order rates. Also, diagnostics are generally monitored throughout runs to ensure that the code is functioning properly. For example, maximum errors in the freestream preservation quantity Λ are generally reported at every timestep, and can be monitored to ensure that the freestream preservation correction is functioning properly.

4.2.7 Test 7 – AMRINS system regression test

. To ensure that `Chombo` library changes, bug fixes, and related code changes do not cause unintended changes in code results, an AMRINS system regression test will be employed. The AMRINS code is run with the benchmark inputs file (a detailed description of the benchmark problem and the inputs to it may be found in the Baseline Performance document), and two diagnostic quantities are reported at the end of the run – the total number of cells updated, and the integral of the kinetic energy over the problem domain. Changes in these diagnostic quantities will indicate changes which will need to be investigated. The total number of cells updated in the run is sensitive to changes in the AMR grid hierarchy during the run, while the integral of the kinetic energy is sensitive to changes in the solution itself.

Chapter 5

Test Schedules

Once a capability in the code has been verified by the appropriate test, we plan to use these tests as regression tests. We plan to apply the entire test suite once each month to ensure that no unintended changes are introduced, and we also will re-run the test suite after bugs are found and corrected to ensure that new bugs are not introduced.

The AMRINS system regression test (Test 7) will be done weekly for serial runs, and monthly for the suite of parallel runs, and also after bug fixes and library changes to lessen the possibility of unintended changes in the code.

Also, acceptance tests will be run as stakeholders take possession of the software.

Chapter 6

Bug Tracking

The AMRINS developers (and the Chombo developers) use the ttpro system for bug tracking. When a bug or unexpected behavior in the code is identified, a description is entered in the ANAG ttpro database. As the bug is investigated and fixed, the description is updated and expanded. Once a bug has been fixed, the bug report is "closed" in ttpro, but it remains in the database for future reference if needed. Also, after a bug fix, the regression test (Test # 7) is re-run to ensure that no unanticipated effects have been added.

Chapter 7

Requirements Traceability

The requirements traceability matrix is presented in Figure 7.1.

Req Spec No.	Req Statement	S/W module	Test Spec	Test Case #	Verification	Mod. Field
1.a.i.A	higher-order Godunov	TraceState function	AMRINS Test Plan	1	partially verified	
1.a.i.B	Advection velocity Projection	CCProjector: levelMacProject	AMRINS test plan	2	fully verified	
1.a.i.C	Freestream preservation correction	CCProjector: computeVDCorrection	AMRINS Test Plan	6	fully verified	
1.a.ii	Viscous solver	AMRNavierStokes: computeUStar	AMRINS Test Plan	3	partially verified	
1.a.iii	level projection	CCProjector: levelProject	AMRINS Test Plan	4	partially verified	
1.b.i	flux-correction for hyperbolic AMR	AMRNavierStokes: postTimestep	AMRINS Test Plan	6	partially verified	
1.b.ii	synchronization projection	CCProjector: doSyncProject	AMRINS Test Plan	5	partially verified	
1.b.iii	freestream preservation correction	CCProjector: computeVDCorrection	AMRINS Test Plan	6	fully verified	

Table 7.1: Requirements Traceability Matrix

Bibliography

[CGL+00] P. Colella, D. T. Graves, T. J. Ligocki, D. F. Martin, D. Modiano, D. B. Serafini, and B. Van Straalen. Chombo Software Package for AMR Applications - Design Document. unpublished, 2000.

[MC00] D Martin and P Colella. A cell-centered adaptive projection method for the incompressible Euler equations. *J. Comput. Phys.*, 2000.