Multifluid Software Testing Plan
Level 0: Data holders and Infrastructure

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

June 9, 2004

Contents

Chapter 1

Scope

The multifluid code will build heavily on the MFChombo [1] infrastructure, which in turn will rely on the EBChombo [3] and Chombo [2] software. The software test plan outlined in this document will focus on the functionality developed to implement the algorithm outlined in the "Multifluid Algorithm Specification" [4] document; since MFChombo, EBChombo and Chombo have their own software test plans, it is not necessary to provide for testing the functionality of the libraries themselves. Note, however, that since the software developed for the multifluid code will use the MFChombo functionality so extensively, changes and bugs in the libraries will tend to have effects on the testing results. Multifluid code developers are kept abreast of developments in the libraries through CVS notification (which sends e-mail whenever a change is made in the CVS version-control repositories), and through the ChomboUsers e-mail list.

1.1 System Overview

The multifluid software will implement an algorithm for computing fluid dynamics for multiple immiscible fluids with surface tension [4]. This software will be built using the MFChombo software, which implements basic support for computations in a multifluid environment using an embedded interface description of the multifluid interface. The MFChombo software is built upon the Chombo and EBChombo software libraries.

Chapter 2

Reference Documents

We will refer to the multifluid algorithm described in [4] and the MFChombo software design document [1].

Chapter 3

Software Test Environment

The multifluid software, linked to the `MFChombo` and `Chombo` software libraries will be tested. As new functionality is added and functionality is improved, testing will continue. It is expected that a given time, the multifluid code will be in sync with the current state of the `MFChombo` and `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** available from NCSA. 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 will test the multifluid code in a variety of environments, with a variety of compilers. Table 3 lists the platforms and compilers we have successfully compiled and run other `Chombo` codes:

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
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

Table 3.1: Platforms and compilers on which the `Chombo` codes have been tested

Chapter 4

Test Identification

4.1 General Information

The testing for the multifluid libraries will mostly be a set of simple unit tests to verify functionality.

For much of the functionality which will be developed for this work, the best way to test whether components are functioning properly is often to do a convergence study. For example, in the case of an operator such as a gradient or a Laplacian, a field is initialized on a series of meshes, each a factor of 2 finer than the last. The operator is applied to the field. If the operator is properly implemented, the result should converge at second-order rates to an analytic solution.

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 multifluid codes once they have been fully developed. 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 future phases of code development will be focused on performance enhancement, performance of the code will then 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 multifluid software, broken down by functional algorithm component. All testing codes will be written in C++.

4.2.1 Test 1 – `MFIndexSpace`

Testing of `MFIndexSpace` creation and consistency. Topology graph coarsening. The ability to create `EBISLayout` objects for arbitrary `DisjointBoxLayout` configurations at arbitrary levels of AMR refinement for every fluid phase.

4.2.2 Test 2 – `LevelData MFCellFAB`

This require testing the `MFCellFAB` class for compliance to the templated data holder `LevelData`. This testing will require proper functioning of the following `MFCellFAB` functions

```
void copy(const Box& RegionFrom,
          const Interval& destInt,
          const Box& RegionTo,
          const MFCellFAB& source,
          const Interval& srcInt);
static int preAllocatable();
int size(const Box& R, const Interval& comps) const ;
void linearOut(void* buf, const Box& R, const Interval& comps) const ;
void linearIn(void* buf, const Box& R, const Interval& comps);
```

It will also require the proper functioning of the class `MFCellFactory` function

```
virtual MFCellFAB* create(const Box& a_box, int a_ncompsIgnored,
                          const DataIndex& a_dit) const;
```

4.2.3 Test 3 – `MFRemapper`

Both remapping functions will be tested:

```
void remap(const MFIndexSpace& a_sourceMF,
           const LevelData<MFCellFAB>& a_source,
           const MFIndexSpace& a_destMF,
           LevelData<MFCellFAB>& a_dest);

void remap(const MFIndexSpace& a_MF,
           const ProblemDomain& a_domainCoar,
           const ProblemDomain& a_domainFine,
           const LevelData<MFCellFAB>& a_source,
           const LevelData<MFCellFAB>& a_coarse,
           const int& nref,
           const int& nghost,
           LevelData<MFCellFAB>&  a_dest);
```

4.2.4 Test 4 – `Multifluid linear operations`

This involves testing algebraic operations (`+-*/`) as well as norm calculations (max norm, L1, L2). We will also show an example of calculating the divergence of a vector field (per fluid phase) in a two phase index space.

Chapter 5

Bug Tracking

The multifluid code developers (and the `MFChombo` and `Chombo` developers) use the **ttpro**^{TM}commercial system for bug tracking. When a bug or unexpected behavior in the code is identified, a description is entered in the ANAG **ttpro**^{TM}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.

Chapter 6

Requirements Traceability

The requirements traceability matrix is presented in Figure 6.1. 'Test Spec' references the tests outlined in the previous section. All tests are run in both debug and optimized mode, in both 2D and 3D configurations. Tests that have also bee verified in parallel are indicated.

Req Statement	S/W module	Test Spec	Test Case	Verification	Parallel
`MFIndexSpace` construction	`MFIndexSpace` `GeometryService`	1	sphereTest.cpp	verified	yes
Data types	`EBCellFAB` `EBFaceFAB` `LevelData<MFCellFAB>` `MFCellFactory`	2	mapperTest.cpp	verified	yes
Data remapping	`MFRemapper`	3	mapperTest.cpp	verified	no
Linear Operations	`EBCellFAB` `EBFaceFAB` `LevelData<MFCellFAB>` `MFAliasFactory`	4	levelDivTest.cpp	verified	yes

Table 6.1: Requirements Traceability Matrix

Bibliography

[1] P. Colella, D. T. Graves, T. J. Ligocki, D. Martin, D. B. Serafini, and B. Van Straalen. MFChombo Software Package for Cartesian Grid, Multifluid Applications. unpublished, 2003.

[2] 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.

[3] P. Colella, D. T. Graves, T. J. Ligocki, D. Modiano, D. B. Serafini, and B. Van Straalen. EBChombo Software Package for Cartesian Grid, Embedded Boundary Applications. unpublished, 2001.

[4] Dan Martin and Phil Colella. Multifluid algorithm specification. available at http://davis.lbl.gov/NASA, 2003.