Incompressible Navier-Stokes Requirements

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

April 21, 2003

Contents

Chapter 1

Introduction

1.1 Purpose

This document outlines the requirements for the Chombo-based AMR Incompressible Navier-Stokes (AMRINS) program.

1.2 Overview

The purpose of this project is to provide an adaptive mesh refinement (AMR) code capability for simulating multiphase low-Mach-number fluid dynamics in microgravity environments.

Chapter 2

General Description

2.1 Overall System Concept

The AMRINS program uses the Chombo C++ libraries to implement an AMR algorithm for the incompressible Navier-Stokes equations which uses block-structured mesh refinement to increase solution accuracy at less computational expense than the equivalent uniform grid computation. This code refines in time as well as space. Various algorithmic elements are required to ensure that the solution maintains such properties as a divergence-free velocity field, conservation of advected quantities, and freestream preservation.

2.2 Operating Environment

AMRINS and the Chombo software infrastructure are designed to work in either a single-processor environment or on a parallel distributed memory system. In the latter case, Chombo has high-level libraries which support domain decomposition across processors which are implemented using MPI.AMRINS shares the same operating environment requirements as Chombo. To compile the code, functioning C++ and F77 compilers are required, along with GNU make and PERL, which is used for preprocessing Chombo Fortran code.

Both Chombo and AMRINS use HDF5 for I/O operations. AMRINS writes data output files which can be read, visualized, and analyzed by the ChomboVis utility.

2.3 High-level Diagram

A diagram of the basic `AMRLevel/AMRNavierStokes` class relationship is shown in Figure 2.1. The `AMRNavierStokes` class, which is derived from the `AMRLevel` class, contains the basic dependent data `vel_old, vel_new, lambda_old,` and `lambda_new` for a single AMR level. A `CCprojector` object is also a member of each `AMRNavierStokes` object,

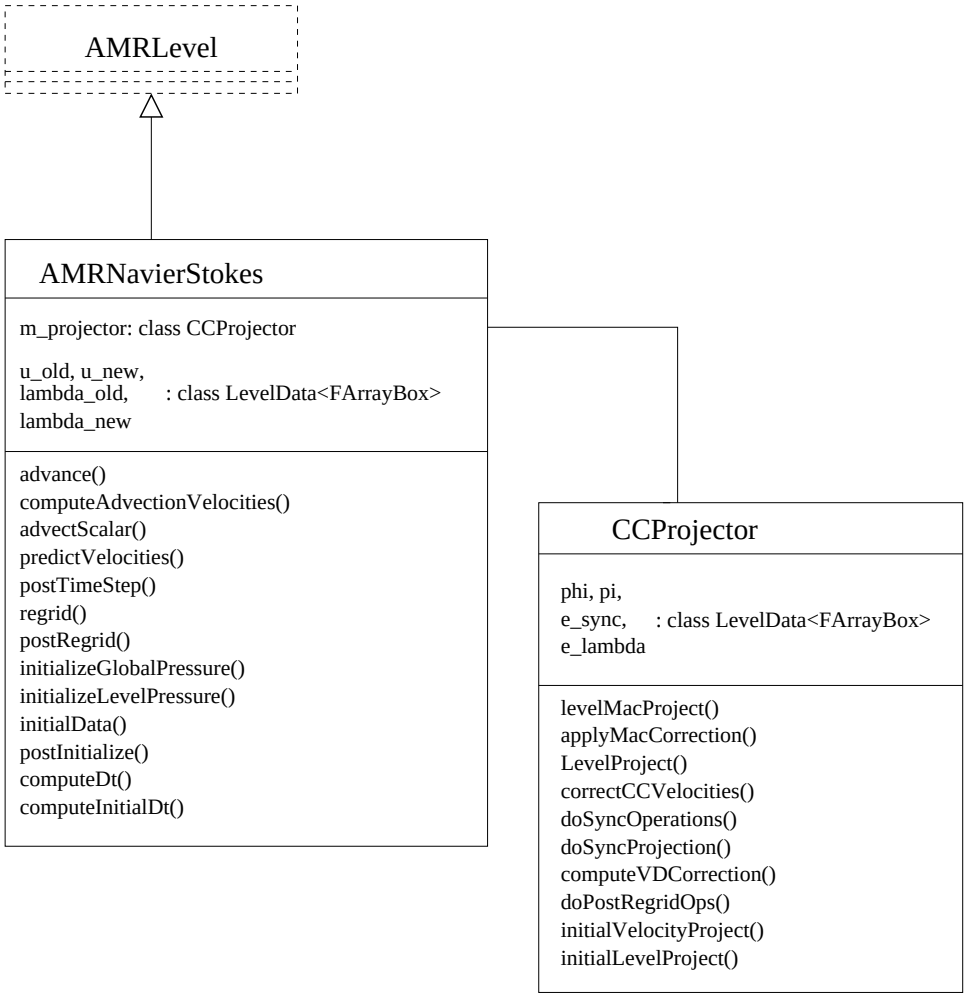

Figure 2.1: Software configuration diagram for the `AMRNavierStokes` class.

and is responsible for divergence-constraint related operations for the level's worth of data contained in the `AMRNavierStokes`. It is anticipated that as new some functionality is added (support for suspended particles, for example), the basic structure of the code will remain the same, with the AMRINS code being the basis for the extended functionality code. We anticipate that this will result in a number of child codes depending on the requirements of the physical system being modeled. Other functionalities being implemented for the NASA-CAN project, such as multifluid front tracking and self-gravity, will be implemented in separate codes which will also use much of the same basic functional building blocks and Chombo classes as the AMRINS code.

As shown in Figure 2.2, an `AMR` class object is constructed which manages the entire AMR multilevel solution. The `AMR` object contains a `Vector` of `AMRLevel`-derived objects, in this case `AMRNavierStokes` objects, which contain the data for each level and also encapsulate the functionality required to advance the data on a single level in time.

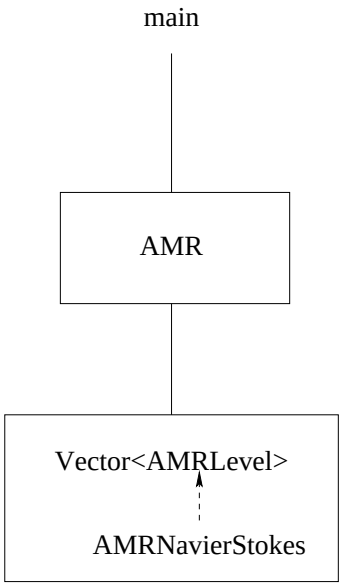

Figure 2.2: Software configuration diagram for the `AMRNavierStokes` class.

Chapter 3

Requirements

3.1 Algorithm Requirements

1. *AMR Incompressible Navier-Stokes Code (AMRINS)* – Program which implements block-structured AMR for the incompressible Navier-Stokes equations, extending the algorithm in [MC00] to the Navier-Stokes equations as documented in the Software Design Document. AMRINS uses the `AMR/AMRLevel` interface in the `AMRTimeDependent` Chombo library to manage the multilevel advance process. The following sub-requirements detail the added functionality needed to implement the Navier-Stokes algorithm in this framework. Unless otherwise noted, this requirement and all its sub-requirements have been implemented by Milestone E.

   (a) *Single-level evolution* – Code will advance a single AMR level from time t^ℓ to time $(t^\ell + \Delta t^\ell)$, where the ℓ superscript denotes the refinement level. Following [BCG89], use a semi-implicit update – explicit computation of nonlinear advection terms, implicit Runge-Kutta approach for computing the diffusive terms, and projection formulation to satisfy the divergence constraint.

      i. *Advective Terms* – Use higher-order Godunov method to compute the nonlinear advective terms, coupled with a face-centered projection and a volume-discrepancy-type correction to preserve incompressible advection.

         A. *Nonlinear advection* – Use higher-order Godunov scheme based on [Sal94, Col90] to compute nonlinear advection

         B. *Advection velocity projection* – Face-centered projection is applied to the advection velocities to ensure they satisfy the incompressible divergence constraint.

         C. *Volume discrepancy correction for freestream preservation* – use the volume discrepancy approach described in [MC00] to compute an advective correction to ensure that freestream preservation is approximately preserved at coarse-fine interfaces.

ii. *Diffusive terms* – Because of the presence of coarse-fine interfaces, L_0 stability is important when computing diffusion in an AMR context. An L_0 stable second-order Runge-Kutta method based on the one presented in [TGA96] is used to compute the diffusive terms for the single-level update.

iii. *Projection for incompressibility*. To ensure the solution approximately satisfies the incompressible divergence constraint, apply a cell-centered approximate projection [LC, MC00].

(b) *AMR solution* The AMRINS code uses the approach of Berger & Colella [BC89]; refinement is in the form of block-structured, nested grids, with refinement in time as well as space. While the `AMR/AMRLevel` interface in the Chombo `AMRTimeDependent` library manages the basic adaptive algorithm, the AMRINS implementation imposes additional algorithmic requirements, in the form of ensuring that the solution employs proper coarse-fine interface matching conditions for hyperbolic, parabolic, and elliptic operators.

i. *Hyperbolic (advection) and Parabolic (diffusion) operators* – To ensure both conservation and stability, an implicit refluxing step is carried out to ensure that fluxes are consistent across the coarse-fine interface. The presence of diffusive momentum fluxes across the coarse-fine interface requires that this flux correction step be done in an implicit manner for stability.

ii. *Elliptic matching (divergence constraint)* – To ensure that the divergence constraint is enforced properly at coarse-fine interfaces (see the discussion in [MC00], a projection based on a multilevel discretization of the cell-centered projection of [LC, MC00] is applied to the multilevel velocity field.

iii. *Freestream preservation* Refinement in time along with flux correction at coarse-fine interfaces can lead to violations of freestream preservation. A correction based on the volume discrepancy approach in [MC00] is computed based on an auxiliary passively advected scalar. This correction is then applied to the advection velocity field to correct for errors in freestream preservation.

(c) Adaptive regridding. – After a regridding operation has altered the AMR hierarchy, the solution must be re-initialized, as detailed in the Software Design Document. Operations include:

i. re-project velocity field to ensure incompressibility

ii. initialize pressure field for new grid hierarchy

iii. recompute freestream preservation for new grid hierarchy.

(d) HDF5 I/O to plotfiles – At specified intervals, the solution is written out to hdf5 plotfiles. These plotfiles can be read and processed using the ChomboVis

utility.

(e) Checkpoint-restart capability based on hdf5 checkpoint files. – A computation may be restarted using data which is periodically written to hdf5 checkpoint files.

2. *Milestone I – suspended particles:* For the NASA CAN milestone I requirement, support for suspended particles in incompressible fluid will be added to the AMRINS code.

3.2 System Requirements

The AMRINS code is designed to use the Chombo infrastructure, which has the following characteristics:

1. *C++/Fortran* – Chombo is a C++ class library with support for interfacing with Fortran 77. In general, C++ is used for data structures and operations on irregular sets of cells, while Fortran is used for floating-point-intensive operations on regular blocks of cells.

2. *Domain decomposition for distributed memory* – The strategy employed by Chombo, which is carried over to AMRINS, is to divide the grids on each level among processors, and to use MPI for communication between processors.

3. *ParmParse user interface* – Inputs to the code are provided through a text inputs file which is processed using the Chombo `ParmParse` utility class.

3.3 Implementation Strategy

The general class structure of the AMRINS code is:

1. We make use of the `AMR` and `AMRLevel` interface classes in Chombo to manage the general AMR algorithm. The `AMR` class manages the multilevel aspects of the algorithm, such as the multilevel advance strategy, regridding operations, etc, while the `AMRLevel` class is designed to encapsulate data and operations on a single AMR level, such as the single-level advance step.

2. `AMRNavierStokes: public AMRLevel` implements `AMRLevel` class for the incompressible Navier-Stokes algorithm. The `AMRNavierStokes` class owns the primary dependent data, which is the velocity field at the old and new times for each level.

3. `CCProjector` – C++ class which implements algorithmic and data components relating to satisfying the divergence constraint, among these are the advection velocity

projection, the single-level projection, the multilevel projection, and the freestream preservation correction. Data members include the various correction fields required for these operations.

4. Diffusion Solver – At the moment, the diffusion solver is hard-coded into the `AMRNavierStokes` class; however, to improve reusability and flexibility, this is being factored out in the near future into a separate DiffusionSolver C++ class.

5. Nonlinear Advection – At the moment, the nonlinear advection prediction steps are coded into the `AMRNavierStokes` class, along with an auxiliary standalone function which computes the upwinded tracing step. In the near future, this will be replaced with the upwinded advection capability developed for the `AMRGodunov` code in the Chombo examples.

Chapter 4

Other Non-Functional Requirements/Attributes

Tied to the milestones for the NASA CAN, there are other non-functional requirements for this code:

1. *Baseline performance:* As part of milestone E of the NASA CAN, the AMRINS code has been instrumented and its performance evaluated for a benchmark problem (see the accompanying Baseline Performance document).

2. *Milestone F – performance improvement:* As part of the milestone F requirement for the NASA CAN, the performance of the AMRINS code on the baseline problem will be improved over that measured as the baseline by a five-fold reduction in time to solution, with 30% less memory usage.

3. *Milestone I – performance evalutation:* For the NASA CAN milestone I requirement, once support for suspended particles is implemented, the performance of the code will also be evaluated for a benchmark problem.

4. *Milestone G – performance improvement and release:* The performance of the new components added in milestone I will be improved by a fivefold reduction in CPU time, along with a 30% reduction in memory usage. The AMRINS code with the particle extensions will be released on the project website.

Chapter 5

Operational Scenarios

The AMRINS code is designed to be run from the command line, with user-defined options passed to the code using an inputs file using the Chombo ParmParse functionality. User-selected options include the size of the problem, how long the run should last, the number of refinement levels allowed, etc. A more complete list of inputs file options, along with a sample inputs file, appears in the AMRINS design document.

Bibliography

[BC89] M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Comput. Phys.*, 82(1):64–84, May 1989.

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

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

[LC] M. F. Lai and P. Colella. An approximate projection method for the incompressible Navier-Stokes equations. unpublished.

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

[Sal94] Jeff Saltzman. An unsplit 3d upwind method for hyperbolic conservation laws. *J. Comput. Phys.*, 115:153–168, 1994.

[TGA96] E.H. Twizell, A.B. Gumel, and M.A. Arigu. Second-order, l_0-stable methods for the heat equation with time-dependent boundary conditions. *Advances in Computational Mathmatics*, 6:333–352, 1996.