User Guide for AMR Incompressible Navier-Stokes with Particles (PAmrNS) Code

Dan Martin and Phil Colella
Applied Numerical Algorithms Group

January 19, 2005

1 Overview

The PAmrNS code implements the algorithm described in the "Incompressible Navier-Stokes with Particles Algorithm Design Document" [3] using the software design described in the "Incompressible Navier-Stokes Software Design" [2] and "Software Design for Particles in Incompressible Flow, Non-Subcycled Case" [4] documents.

2 Obtaining and compiling the code

The AMR Incompressible Navier-Stokes with Particles code may be obtained from the ANAG NASA CAN website (`http://davis.lbl.gov/NASA`); using the code also requires the Chombo framework [1], the AMRINS software package (available at `http://seesar.lbl.gov/NASA`), and the fftw 3.0.1 FFT software package, downloadable from `http://www.fftw.org`. The Chombo framework also uses HDF5 for I/O.

To compile the code, first install the HDF5, fftw, and Chombo libraries according to the instructions in each package. Note that at th current time, the particle code only runs in 3D (so Chombo must be compied with DIM=3). Then, unpack the AMRINS code (this will unpack into a Chombo-IAMR directory) and the PAmrNS tarfile, which will unpack into a particle-IAMR directory. Compilation will be simpler if the Chombo, AMRINS, and PAmrNS tarfiles are all unpacked in the same location. Change to the `particle-IAMR/exec-nonSubcycled` directory and edit the GNUmakefile appropriately to supply locations for the Chombo libraries and the fftw libraries.

Finally, compile the code: `make all DIM=3 [DEBUG=<TRUE,FALSE>]` with "DEBUG=FALSE" producing optimized code.

An executable of the form `ns3d.<config-string>.ex` , where the `<config-string>` contains information about how the code was compiled.

3 Running the code

To run the code, type `ns3d.<config-string>.ex <inputs-file>` , where `inputs-file` is a file containing the parameters needed to specify the run parameters.

3.1 Inputs file options

The format for the inputs file is generally of the form `<group>.<variable> = <value(s) >` , where `<group>` generally indicates what part of the code uses a given input. Everything following a "#" on a line is ignored, and in the case of multiple instances of a variable in an inputs file, the last instance is used. A sample inputs file is in `particle-IAMR/exec-nonSubcycled/inputs`.

Some input parameters are required, while others have default values if they are not specified in the inputs file. Required variables are listed first, followed by optional ones.

3.1.1 Input parameters for `main`

- `main.num_cells` (required) SpaceDim integers – number of cells in each direction in the base computational domain.

- `main.domainLength` (required) SpaceDim Reals – length of physical domain in each direction (dx on the coarsest level will be `domainLength`[0]/`num_cells`[0])

- `main.max_step` (default = 0) integer – maximum number of timesteps to compute.

- `main.max_time` (default = 0) Real – maximum solution time to compute to.

- `main.verbosity` (default = 0) integer – higher number results in more verbose text output.

- `main.fixed_dt` (default = -1) Real – if positive, code will use this value for the timestep.

- `main.is_periodic` (default = 0) SpaceDim integers. In each coordinate direction, if 1, domain is periodic in that direction; if 0, nonperiodic.

- `main.max_level` (default = 0) integer – finest allowable refinement level. 0 means there will be no refinement.

- `main.ref_ratio` (required if `max_level` ¿ 0) `max_level` integers (one for each level). refinement ratios between levels. First number is ratio between levels 0 and 1, second is between levels 1 and 2, etc.

- `main.regrid_interval` (default = -1) integer – number of timesteps to compute between regridding. A Negative value means there will be no regridding.

- `main.block_factor` (default = 2) integer – the block factor is the number of times that grids will be coarsenable by a factor of 2. This can have implications on how well multigrid solvers work. A higher number produces "blockier" grids.

- `main.max_grid_size` (default = 32) integer – the largest allowable size of a grid in any direction. Any boxes larger than that will be split up to satisfy this constraint.

- `main.fill_ratio` (default = 0.75) Real between 0 and 1. – the efficiency of the grid generation process. Lower number means that more extra cells which aren't tagged for refinement wind up being refined along with tagged cells. The tradeoff is that higher fill ratios lead to more complicated grids, and the extra coarse-fine interface work may outweigh the savings due to the reduced number of fine-level cells.

- `main.grid_buffer_size` (default = 2) integer – number of level ℓ cells between the $\ell-(\ell+1)$ and $(\ell-1)-\ell$ interfaces required for the grids to be considered properly nested.

- `main.checkpoint_interval` (default = -1) integer – number of timesteps between writing checkpoint files. Negative number means that checkpoint files are never written, 0 means that checkpoint files are written before the initial timestep and after the final one.

- `main.plot_interval` (default = -1) integer – number of timesteps between writing plotfiles. Negative number means that plotfiles are never written, 0 means that plotfiles are written before the initial timestep and after the final one.

- `main.checkPrefix` (default "chk") string – prefix for checkpoint files.

- `main.plotPrefix` (default "plt") string – prefix used for plotfiles.

- `main.max_dt_grow` (default = 1.5) Real – maximum factor by which the timestep can increase from one timestep to the next.

- `main.gridfile` (default is none) string – if present, indicates the file from which the initial grid hierchy will be read.

- `main.cfl` (default = 0.5) Real – CFL number (maximum allowable value for max(vel)*dt/dx.

- `main.particleCFL` (defaults to main.cfl) Real – similar to main.cfl, except using max(particleVel).

3.1.2 Input parameters for `ns`

- `ns.particle_epsilon` (required) Real – parameter for the particle delta function. Essentially the width of the discrete delta function used to spread particle forces to the mesh.

- `ns.spreadingRadius` (required) Real – parameter for the particle projection – related to the size of the locus around each particle upon which the particle projector's kernel is computed.

- `ns.init_shrink` (default = 1.0) Real – factory by which to shrink the initial timestep.

- `ns.max_dt` (default = 1.0e8) Real – maximum allowable timestep.

- `ns.project_initial_vel` (default = 1) integer – if 1, project the initial velocity to ensure that it is discretely divergence-free. if 0, don't do this.
- `ns.update_velocity` (default = 1) integer – if 0, don't update the velocity field at the end of the timestep (all the computations are done, but then the velocity field is reverted to what it was at the beginning of the timestep. The result is a velocity field which is constant in time.)
- `ns.init_pressures` (default = 1) integer – if 0, don't initialize the pressure before the first timestep.
- `ns.num_init_passes` (default = 1) integer – if `init_pressuures` is not 0, then the number of initialization timesteps to compute for pressure initialization.
- `ns.tag_vorticity` (default = 0) integer – if 1, then tag for refinement based on undivided vorticity. if 0, then don't.
- `ns.vorticity_tagging_factor` (default = 0.30) Real – if `tag_vorticity` is 1, then this is the threshold value at which we tag cells for refinement during the grid generation process. if the undivided vorticity is greater than the tagging factor, then the cell is tagged for refinement to a finer level.
- `ns.tag_particles` (default = 1) integer – if 0, then don't tag for refinement based on particles. If 1, then all cells containing particles will be refined to the finest allowable level.
- `ns.tags_grow` (default = 1) integer – amount by which to grow tags (as a safety factor) before passing to MeshRefie.
- `ns.viscosity` (default = 0.0) Real – the kinematic viscolsity of the fluid.
- `ns.particle_drag_coeff` (default = 0.0) Real – Drag coefficient for the particles.
- `ns.particleProjCoarsen` (default = -1) integer – amount to coarsen particle projection before sending to the infinite domain solver. Negative number means that we use the default value in `AMRParticleProjector`.

- `ns.particle_body_force` (default = 0) SpaceDim Reals – body force on each particle in each component direction. For example, could be 0 0 w, where w is the weight of the particle.

- `ns.correctionRadius` (default = 2) Real – parameter for the particle projector.

- `ns.particleGridsGrow` (default = 2) integer – amount to grow auxiliary grids for particles (buffer to ensure that particles don't run off the grown grids in the `AMRParticleProjector` between regridding steps.

- `ns.use_image_particles` (default = 0) integer – if 1, use image particles to help enforce the no-flow condition at solid walls. If 0, don't use image particles.

- `ns.do_particles` (default = 1) integer – if 1, do particle updates and include particle drag force in the fluid computations. If 0, don't do these things.

- `ns.read_particles_from_file` (default = 1) integer – if 1, read particles from a file, given by `ns.particle_file`. If 0, particles are defined in the code.

- `ns.particle_file` (required if `ns.read_particles_from_file` is 1) string – name of file containing particle masses, initial positions, initial velocities, etc.

- `ns.order_vel_interp` (default = 1) integer – if 1, do linear interpolation of fluid velocities to particle positions (for drag computation), if 0, do piecewise constant.

- `ns.limit_Vel_interp` (default =0) integer – If 1, limit slopes used in fluid velocity interpolation, if 0, don't limit these slopes. Only relevant if `ns.order_vel_interp` is nonzero.

- `ns.particle_update_type` (default = 2) Type of update scheme to use to update particle position and velocity. 0 = basic predictor-corrector, 1 = forward Euler, 2 = more accurate predictor-corrector, and 3 = forwardEuler using the predictor scheme in 2. 2 is recommended.

- `ns.viscous_solver_tolerance` (default = 1e-7) Real – tolerance to use for the `AMRSolver` for the viscous solves.

- `ns.viscous_num_smooth_up` (default = 4) integer – multigrid solver parameter for the viscous solvers.

- `ns.viscous_num_smooth_down` (default = 4) integer – multigrid solver parameter for the viscous solvers.

- `ns.num_scalars` (default = 0) integer – number of scalar components to advect/diffuse.

- `ns.scal_diffusion_coeffs` (required if `ns.num_scalars` = 1) Array of `num_scalars` Reals. Diffusion coefficients for the scalars.

- `ns.specifyInitialGrids` (default = 0) integer – if 1, specify initial grid hierarchy (which may be changed later due to regriddig). If 0, generate grids adaptively using tagging criteria.

- `ns.initialGridFile` (required if `ns.specifyInitialGrids` = 1) string – file containing initial grid hierarchy info.

- `ns.initVelFromVorticity` (default = 0) integer – if 1, initialize velocity by solving from a specified vorticity field. If 0, velocity is initialized analytically.

- `ns.backgroundVelocity` (default = 0.0) Real – if nonzero, background velocity field in the $z-$direction (value is added to velocity field in the $z-$direction.

- `ns.writeParticles` (default = 1) integer – if nonzero, write particles to plotfiles.

- `ns.writeDivergence` (default = 1) integer – if nonzero, write divergence to plotfiles.

- `ns.writeTimeDerivatives` (default = 0) integer – if nonzero, write time derivatives of velocity to plotfiles.

- `ns.writeVorticity` (default = 1) integer – if nonzero, write vorticity to plotfiles.

- `ns.writeScalars` (default = 0) integer – if nonzero, write scalar solution to plotfiles.
- `ns.writeDscalDt` (default = 0) integer – if nonzero, write time derivatives of scalars to plotfiles.
- `ns.writeProcIDs` (default = 0) integer – if nonzero, write processor distribution to plotfiles.

3.1.3 Input parameters for `projection`

- `projection.numSmoothUp` (default = -4) integer – multigrid solver parameter for the viscous solvers.
- `projection.numSmoothDown` (default = -4) integer – multigrid solver parameter for the viscous solvers.
- `projection.solverTol` (default = 1e-9) Real – tolerance to use for the `AMRSolver` for the viscous solves.

3.1.4 Input parameters for `physBC`

- `physBC.lo` (requires SpaceDim integers) Type of physical boundary condition to apply on the low side in each direction. 0 = solidWall, 1=inflow, 2=outflow, 3=symmetry, 4=noShear.
- `physBC.hi` (requires SpaceDim integers) Type of physical boundary condition to apply on the high side in each direction. 0 = solidWall, 1=inflow, 2=outflow, 3=symmetry, 4=noShear.
- `physBC.maxInflowVel` (default = 1.0e8) If inflow BC is selected, maximum value that it will take (used to compute timestep)

3.2 Visualizing the results

If `main.plot_interval` is non-negative, the `PAmrNS` code will write solutions out into hdf5 plotfiles, which are in the format used by the `ChomboVis` visualization tool.

References

[1] 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.

[2] Applied Numerical Algorithms Group. Incompressible Navier-Stokes software design. Technical report, NERSC Division, Lawrence Berkeley National Laboratory, 2002.

[3] Dan Martin and Phil Colella. Incompressible Navier-Stokes with particles design document. Technical report, Applied Numerical Algorithms Group, Lawrence Berkeley Laboratory, 2003.

[4] Dan Martin and Phil Colella. *Software Design for Particles in Incompressible Flow, Non-Subcycled Case.* Applied Numerical Algorithms Group, Lawrence Berkeley Laboratory, 2003.