amrex Documentation

Release 18.00-dev

AMReX Team

Nov 02, 2018

CONTENTS:

AMReX is a software framework library containing all the functionality to write massively parallel, block-structured adaptive mesh refinement (AMR) applications. AMReX is freely available at https://github.com/AMReX-Codes/amrex.

AMReX Tutorials are a set of small stand-alone example codes that demonstrate how to use different parts of the AMReX functionality.

We are always happy to have users contribute to AMReX Tutorials as well as the AMReX source code. To contribute, issue a pull request against the development branch (details at https://help.github.com/articles/creating-a-pull-request/).

The amrex/Tutorials directory is broken into the following categories:

CHAPTER

ONE

TUTORIALS/AMR

CHAPTER

TWO

TUTORIALS/BASIC

The tutorials in amrex/Tutorials/Basic demonstrate the most fundamental operations supported by AMReX.

2.1 HelloWorld

HelloWorld_C and HellowWorld_F demonstrate the GNU Make system – with a sample Make.package and GNUmakefile – and the amrex::Initialize and amrex::Finalize functions.

In addition, in HelloWorld_C, the amrex::Print() operation, which only prints from the I/O processor, is used to print out the AMReX version (as defined by amrex::Version()) being used.

HelloWorld_F is a simple example of how to use the F_Interface routines, which are Fortran wrappers for the underlying C++ data strutures and iterators. Here, for example, rather than calling amrex::Print() in C++, we test on whether amrex_parallel_ioprocessor() is true, and if so, invoke the usual Fortran print call.

2.2 main

main_C and main_F introduce the following:

1. By default, AMReX initializes MPI and uses MPI_COMM_WORLD as its communicator. However, applications could choose to initialize MPI themselves and pass in an existing communicator.

2. By default, AMReX treats command line arguments as inputs parameters. The expected format of argv is

   executable inputs_file parm=value

   Here, *executable* is the filename of the executable, *inputs_file* is the file containing runtime parameters used to build AMReX ParmParse database, and *parm=value* is an input parameter that will override its value in *inputs_file*. Both *inputs_file* and *parm=value* are optional. At most one *inputs_file* is allowed. Howeer, there can be multiple ‘parm=value‘s.

   The parsing of the command line arguments is performed in amrex::Initialize. Applications can choose to skip command line parsing. Applications can also provide a function that adds parameters to AMReX ParmParse database.

2.3 HeatEquation

The HeatEquation examples solve a 2D or 3D (determined by how you set DIM in the GNUmakefile) heat equation explicitly on a domain-decomposed mesh. This example is described in detail in the Basics chapter of the amrex Documentation

CHAPTER

THREE

TUTORIALS/CVODE

There are two CVODE tutorials in the `amrex/Tutorials/CVODE` directory, called `EX1` and `EX2`. `EX1` consists of a single ODE that is integrated with CVODE within each cell of a 3-D grid. It demonstrates how to initialize the CVODE solver, how to call the ODE right-hand-side (RHS), and, more importantly, how to *re*-initialize the solver between cells, which avoids allocating and freeing solver memory between each cell (see the call to `FCVReInit()` in the `integrate_ode.f90` file in the `EX1` directory.)

The `EX2` example demonstrates the slightly more complicated case of integrating a system of coupled ODEs within each cell. Similarly to `EX1`, it provides an RHS and some solver initialization. However, it also demonstrates the performance effect of providing an analytic Jacobian matrix for the system of ODEs, rather than requiring the solver to compute the Jacobian matrix numerically using a finite-difference approach. The tutorial integrates the same system of ODEs on the same 3-D grid, but in one sweep it instructs CVODE to use the analytic function that computes the Jacobian matrix, and in the other case, it does not, which requires CVODE to compute it manually. One observes a significant performance gain by providing the analytic Jacobian function.

See the CVODE section of the AMReX documentation for general instructions on how to include CVODE in an AMReX application.

CHAPTER

FOUR

TUTORIALS/EB

CHAPTER

FIVE

TUTORIALS/HYPRE

CHAPTER

SIX

TUTORIALS/LINEARSOLVERS

CHAPTER

SEVEN

TUTORIALS/PARTICLES

There are two tutorials in amrex/Tutorials/Particles that demonstrate the basic usage of AMReX's particle data structures.

7.1 ElectrostaticPIC

This tutorial demonstrates how to perform an electrostatic Particle-in-Cell calculation using AMReX. The code initializes a single particle in a conducting box (i.e. Dirichlet zero boundary conditions) that is slightly off-center in one direction. Because of the boundary conditions, the particle sees an image charge and is accelerated in this direction.

The code is currently set up to use one level of static mesh refinement. The charge density, electric field, and electrostatic potential are all defined on the mesh nodes. To solve Poisson's equation, we use AMReX's Fortran-based multigrid solver. The Fortran routines for performing charge deposition, field gathering, and the particle push are all defined in `electrostatic_pic_2d.f90` and `electrostatic_pic_3d.f90` for 2D and 3D, respectively.

The particle container in this example using a Struct-of-Arrays layout, with `1 + 2*BL_SPACEDIM` real components to store the particle weight, velocity, and the electric field interpolated to the particle position. To see how to set up such a particle container, see `ElectrostaticParticleContainer.H`.

7.2 NeighborList

This tutorial demonstrates how to have AMReX's particles undergo short-range collisions with each other. To facilite this, a neighbor list data structure is created, in which all of the partners that could potentially collide with a given particle are pre-computed. This is done by first constructing a cell-linked list, and then looping over all 27 neighbor cells to test for potential collision partners. The Fortran subroutine `amrex_compute_forces_nl` defined in `neighbor_list_2d.f90` and `neighbor_list_3d.f90` demonstrates how to loop over the resulting data structure.

The particles in this example store velocity and acceleration in addition to the default components. They are initially placed at cell centers and given random velocities. When a particle reaches the domain boundary, it is specularly reflected back into the domain. To see how the particle data structures are set up, see `NeighborListParticleContainer.cpp`.

CHAPTER

EIGHT

TUTORIALS/SWFFT

This Tutorial demonstrates how to call the SWFFT wrapper to the FFTW3 solver.

Note that the SWFFT source code was developed by Adrian Pope and colleagues and is available at:

https://xgitlab.cels.anl.gov/hacc/SWFFT

In this test case we set up a right hand side (rhs), call the forward transform, modify the coefficients, then call the backward solver and output the solution to the discrete Poisson equation.

To build the code, type 'make' in amrex/Tutorials/SWFFT. This will include code from amrex/Src/Extern/SWFFT and you will need to link to the FFT solvers themselves (on NERSC's Cori machine, for example, you would need to "module load fft")

To run the code, type 'main3d.gnu.MPI.ex inputs' in this directory

To visualize the output, set the bool write_data to true, then use amrvis3d (source available at https://github.com/AMReX-Codes/Amrvis):

amrvis3d -mf RHS SOL_EXACT SOL_COMP

to visualize the rhs, the exact solution and the computed solution.

The max norm of the difference between the exact and computed solution is also printed.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

The copyright notice of AMReX is included in the AMReX home directory as README.txt.

Your use of this software is under a 3-clause BSD license with additional modification – the license agreement is included in the AMReX home directory as license.txt.

For a pdf version of this documentation, click `here`.