Adaptive Mesh Refinement + Curvilinear Coordinate + Dissipative Particle Dynamics CODE

A Summer Program at CCSE, LBL

Created by: TTNguyen (Grad Student), AJNonaka (PhD), TBLe (PhD)

Institute: Lawrence Berkeley National Laboratory

Course: Summer Program

Format: LaTeXBeamer

XeLaTeXed on 2023/11/09

Table of contents

Introduction

Program Mission

Mission

The ultimate mission of the summer program is for the student participant to fully understand the fundamental theories and underlying algorithms around the Adaptive Mesh Refinement (AMR) technique, implemented in the files of AMReX library.

Deliverables

Expected Outcome

The outcome product is all obtained algorithms, pseudo codes, and pieces of software that have the capability to utilize a range of methods i.e. Adaptive Mesh Refinement, Dissipative Particle Dynamics, Sharp Immersed Boundary Method, Curvilinear Coordinate System in a targeted problem.

The targeted problem is the multi-scale computational modeling of blood flow.

Composition Packages

- The first component is Virtual Flow Simulator (VFS)[1], is a computational fluidd mechanics (CFD) package that is based on the Curvilinear Immersed Boundary (CURVIB) method to handle geometrically complex and moving domains.
- The module that we are working with is a module of VFS that has the capability of fluid-structure interaction (FSI) of solid and deformable bodies. This module has been forked and personally maintained by Dr. Trung B. Le.
- Before building FSI, a set of two prerequisites are PETSC [2] (a suite of library for the scalable solution of scientific applications modeled by partial differential equations) version 3.1 patch 8, and Eigen[3] (a C++ template library for linear algebra: matrices, vectors, numerical solvers, etc.)

[1] https://www.osti.gov/biblio/1312901
[2] https://petsc.org/release/
[3] https://gitlab.com/libeigen/eigen

Composition Packages

- The second component, AMReX[4], is a software framework for massively parallel, block-structured adaptive mesh refinement (AMR) applications. Dr. Andy J. Nonaka is one of the developers.
- The source code of AMReX can be found at:

  `https://github.com/AMReX-Codes/amrex`
- A set of tutorials are created for AMReX which helps new users get started with different modules of its huge library.

  `https://github.com/AMReX-Codes/amrex-tutorials`

[4] `https://amrex-codes.github.io/amrex/`

- Other than the two main component codes, Tam Nguyen has also developed a program called JASSFIF.jl. The program is written in Julia Lang and solve the 2D incompressible Navier-Stoke equations for CFD simulations.
- This code helps develop and maintain a small portion of the FSI code (which is HUGE!!!).
- At this point of development, the code was able to modeling fluid flow in low Reynolds number by deploying Fractional Step method. Please refer to the publications below for fully details while what be presented in these slides are our summary and understanding.

  https://doi.org/10.1016/0021-9991(85)90148-2

  https://doi.org/10.1016/0021-9991(91)90215-7

Available on GitHub

This theme and all the documentation is hosted on GitHub

Download — Fork — Contribute

Julian Solver for Incompressible Navier-Stokes:
`https://github.com/milk-white-way/JASSFIF.jl`
AMR-based Code (by Tam Nguyen):
`https://github.com/milk-white-way/AMRESSIF`
Poisson Code (by Rajssekhar Dathi):
`https://github.com/milk-white-way/HAPPiTime`

Figure: Hosted on GitHub

Getting Started

Approaches

Approach 1

In this approach, several modules and algorithms from the FSI code are merged into AMReX library.

- Pros:
  - AMR capability is already implemented, no need to code from scratch.
  - Data parallelism can be automatically handled by AMReX classes.
  - AMReX has better documentation than FSI code.
- Cons:
  - FSI algorithms in AMReX works differently from FSI code. Being able to implement the Curvilinear Coordinate system in AMReX is the key to deploy Sharp Curvilinear Immersed Boundary method.
  - Aligning data structures between AMReX and FSI could be a big challenge due to incompatibilities.
  - Adding features to AMReX's documentation.

Approaches (cont.)

Approach 2

In this approach, AMR algorithms are implemented into JESSFIF.jl program.

- Pros:
  - Do not require the understanding of the whole AMReX library of classes in order to implement AMR.
  - Developers have less rigid data structure.
  - The code could be easier to develop or maintain in the without dependence on AMReX.
- Cons:
  - Time constraint since:
    - LOTS of coding!!!
    - LOTS of documentation!!!
  - Julia Lang might not have the proficient libraries for certain features.

Methodologies

Problem: The governing equations for imcompressible fluid consist of Navier-Stokes (momentum) equation and the continuity equation such that, in general coordinate system:

$$\frac{\partial u_i}{\partial t} = -\frac{\partial}{\partial x_j} u_i u_j + \frac{1}{\mathscr{Re}} \frac{\partial^2 u_i}{\partial x_i \partial x_j} - \frac{\partial P}{\partial x_i} \tag{1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2}$$

Boundary Condition

Eqs. (1) and (2) are non-dimenstionalized by a characteristic length and velocity scale.

Consider a two dimensional problem in Cartesian coordinate system, let us denote u and v so that they are velocity components in, respectively, x- and y-direction. Then the Navier-Stokes equation become:

$$\frac{\partial u}{\partial t} = -\left(\frac{\partial}{\partial x}uu + \frac{\partial}{\partial y}uv\right) + \frac{1}{\mathscr{Re}}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \frac{\partial P_x}{\partial x} \tag{3}$$

$$\frac{\partial v}{\partial t} = -\left(\frac{\partial}{\partial x}vu + \frac{\partial}{\partial y}vv\right) + \frac{1}{\mathscr{Re}}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) - \frac{\partial P_y}{\partial y} \tag{4}$$

And the Continuity equation is as follow:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{5}$$

Solution: In developing a solution for the 3D Incompressible Fluid Modeling Problem, we use a scheme called:

KM (Kim and Moin) Method with LM (Le and Moin) Modification.

The full paper can be found at:
https://doi.org/10.1016/0021-9991(85)90148-2
https://doi.org/10.1016/0021-9991(91)90215-7

Methodologies

LM Mod to KM Scheme

The changes in Le and Moin modification includes:

- Each time step is advanced in three sub-steps.
- Velocity field is advanced through a sub-step without the need to satisfy the continuity equation.
- The Poisson equation is used to project the predicted vector field into a divergence-free velocity field only at the final sub-step.
- Boundary conditions for the intermediate velocity field are derived using the method similar to that of LeVeque and Oliger.
- The method is implemented for a staggered grid.

Remark: The modification has been proved to saved 49% CPU time in comparison to the original KM scheme in a flow over backward-facing step benchmark test.

The modification from Le and Moin is such that before the application of the fractional step method, a three-step advancement scheme is used in the discretization of Eq. (1):

$$\frac{u^k - u^{k-1}}{\Delta t} = \alpha_k L\left(u^{k-1}\right) + \beta_k L\left(u^k\right) - \gamma_k N\left(u^{k-1}\right) - \zeta_k N\left(u^{k-2}\right) - (\alpha_k + \beta_k)\frac{\delta P_x^k}{\delta x} \tag{6}$$

$$\frac{v^k - v^{k-1}}{\Delta t} = \alpha_k L\left(v^{k-1}\right) + \beta_k L\left(v^k\right) - \gamma_k N\left(v^{k-1}\right) - \zeta_k N\left(v^{k-2}\right) - (\alpha_k + \beta_k)\frac{\delta P_y^k}{\delta y} \tag{7}$$

where:

- $k = 1, 2, 3$ denote the sub-step ($k - 2$ is ignored for $k = 1$);
- u_i^0 and u_i^3 are the velocities at time step n and $n + 1$; and
- $\frac{\delta}{\delta x_i}$ is the finite difference operator.

$L(u_i)$ represent second-order finite difference approximation to the viscous terms, such that:

$$L(u_i) = \frac{1}{\mathscr{Re}} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{8}$$

Meanwhile, $N(u_i)$ represent second-order finite difference approximation to the convective terms, such that:

$$N(u_i) = \frac{\partial}{\partial x_j} u_i u_j \tag{9}$$

α_k, β_k, γ_k, and ζ_k are constant coefficients. In general,

$$\sum_{k=1}^{3}(\alpha_k + \beta_k) = \sum_{k=1}^{3}(\gamma_k + \zeta_k) = 1 \tag{10}$$

For the LM algorithm, the constant coefficients are chosen as:

k	α	β	γ	ζ
1	$\frac{4}{15}$		$\frac{8}{15}$	0
2	$\frac{1}{15}$		$\frac{5}{12}$	$\frac{-17}{60}$
3	$\frac{1}{6}$		$\frac{3}{4}$	$\frac{-5}{12}$

Third-order Accuracy in N

Second-order Accuracy in L

Second-order Accuracy in Δt

Applying the Kim and Moin's Fractional Step method to the three-step time advancement scheme, one has:

$$
\begin{aligned}
\frac{\hat{u}^k - \hat{u}^{k-1}}{\Delta t} &= (\alpha_k + \beta_k) L\left(\hat{u}^{k-1}\right) - \frac{\alpha_k}{\mathcal{Re}} \frac{\delta}{\delta x}\left(\frac{\delta \hat{u}^{k-1}}{\delta x} + \frac{\delta \hat{v}^{k-1}}{\delta y}\right) \\
&\quad + \beta_k L\left(\hat{u}^k - \hat{u}^{k-1}\right) - \gamma_k N\left(u^{*k-1}\right) - \zeta_k N\left(u^{*k-2}\right) \\
\frac{u^{*k} - \hat{u}^k}{\Delta t} &= - \frac{\delta \Phi_x^{*k}}{\delta x} \\
\frac{\hat{v}^k - \hat{v}^{k-1}}{\Delta t} &= (\alpha_k + \beta_k) L\left(\hat{v}^{k-1}\right) - \frac{\alpha_k}{\mathcal{Re}} \frac{\delta}{\delta y}\left(\frac{\delta \hat{u}^{k-1}}{\delta x} + \frac{\delta \hat{v}^{k-1}}{\delta y}\right) \\
&\quad + \beta_k L\left(\hat{v}^k - \hat{v}^{k-1}\right) - \gamma_k N\left(v^{*k-1}\right) - \zeta_k N\left(v^{*k-2}\right) \\
\frac{v^{*k} - \hat{v}^k}{\Delta t} &= - \frac{\delta \Phi_y^{*k}}{\delta y}
\end{aligned}
$$

Methodologies

QUICK and related convection-diffusion schemes

B. P. Leonard's Quadratic Upstream Interpolation for Convective Kinematics (QUICK).

The full papers can be found at:
`https://doi.org/10.1016/0045-7825(79)90034-3`
`https://doi.org/10.1016/0307-904X(95)00084-W`

Problem: Consider a non-dimensional convection-diffusion equation with constant coefficients in 1D coordinate system such that:

$$\frac{\partial U}{\partial x} = \frac{1}{\mathscr{Pe}} \frac{\partial^2 U}{\partial x^2} + S(x) \tag{11}$$

Solution: QUICK(1/8) convection scheme with the "left-right" notation is as follow:

$$\left(\frac{\partial U}{\partial x}\right)_i = \frac{U_r(i) - U_l(i)}{h} \tag{12}$$

While U_r and U_l are linear interpolation of left and right wall values of U such that:

$$(U_r)^{\text{QUICK}}(i) = \frac{1}{2}(U_{i+1} + U_i) - \frac{1}{8}(U_{i+1} - 2U_i + U_{i-1}) \tag{13}$$

$$(U_l)^{\text{QUICK}}(i) = (U_r)^{\text{QUICK}}(i-1) \tag{14}$$

Substituting above values from Eqs.(13) and (14) into Eq. (12) gives:

$$\left(\frac{\partial U}{\partial x}\right)_i = \frac{1}{2}(U_{i+1} + U_i) - \frac{1}{8}(U_{i+1} - 2U_i + U_{i-1}) - \frac{1}{2}(U_i + U_{i-1}) - \frac{1}{8}(U_i - 2U_{i-1} + U_{i-2}) \tag{15}$$

$$= \frac{3U_{i+1} + 3U_i - 7U_{i-1} + U_{i-2}}{8h} \tag{16}$$

QUICK technique can also be applied for the second derivative of the diffusive terms as follow:

$$\frac{1}{\mathscr{P}e}\left(\frac{\partial^2 U}{\partial x^2}\right)_i = \frac{1}{\mathscr{P}e}\frac{U_r'(i) - U_l'(i)}{h} \tag{17}$$

U_r' is defined by taking the central difference of two neighboring nodes, such that:

$$(U_r')^{\text{QUICK}}(i) = \frac{(U_{i+1} - U_i)}{h} \tag{18}$$

$$(U_l')^{\text{QUICK}}(i) = (U_r')^{\text{QUICK}}(i-1) \tag{19}$$

In similar manner, one has:

$$\frac{1}{\mathscr{P}e}\left(\frac{\partial^2 U}{\partial x^2}\right)_i = \frac{1}{\mathscr{P}e}\frac{U_{i+1} - 2U_i + U_{i-1}}{h^2} \tag{20}$$

Remark: Convection-diffusion operators:

Full QUICK:

$$[\mathrm{QUICK}] = \left(\frac{3U_{i+1} + 3U_i - 7U_{i-1} + U_{i-2}}{8h}\right) - \frac{1}{\mathscr{Pe}} \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2} \tag{21}$$

SPUDS-plus-CDS2:

$$[\mathrm{SPUDS} + \mathrm{CDS2}] = \left(\frac{2U_{i+1} + 3U_i - 6U_{i-1} + U_{i-2}}{6h}\right) - \frac{1}{\mathscr{Pe}} \frac{U_{i+1} - 2U_i + U_{i-1}}{h^2} \tag{22}$$

Remark: From Leonard (1995):

- QUICK operators for the convective terms, notated by QUICK[C], are third-order accurate – $O(h^3)$.
- QUICK operators for the diffusive terms, notated by QUICK[D], are only second-order accurate – $O(h^2)$.
- If a finite-volume (OA) discrete operator is viewed as an finite-difference (SP) term, there is an $O(h^2)$ discrepancy between the two. A third (or higher) order OA scheme is only second-order accurate when viewed as an SP scheme, and vice versa.

Methodologies

Implementation

Remark: Since u and v have the same mathematical expression, and are only different in notation. Let us use u for now.

Consider the general procedure for x-component velocity u such that:

$$\begin{aligned}\frac{\hat{u}^{k}-\hat{u}^{k-1}}{\Delta t} = (\alpha_k+\beta_k)L\left(\hat{u}^{k-1}\right) - \frac{\alpha_k}{\mathscr{Re}}\frac{\delta}{\delta x}\left(\frac{\delta\hat{u}^{k-1}}{\delta x}+\frac{\delta\hat{v}^{k-1}}{\delta y}\right) \\ + \beta_k L\left(\hat{u}^{k}-\hat{u}^{k-1}\right) - \gamma_k N\left(u^{*k-1}\right) - \zeta_k N\left(u^{*k-2}\right)\end{aligned} \tag{23}$$

$$\frac{u^{*k}-\hat{u}^{k}}{\Delta t} = -\frac{\delta\Phi_x^{*k}}{\delta x} \tag{24}$$

Substituting the expression for the viscous terms $L(u)$ and convective term $N(u)$ into eq. (23), one can achieve the following:

$$
\begin{aligned}
\hat{u}^k = \hat{u}^{k-1} &+ \frac{\Delta t}{\mathscr{Re}}(\alpha_k + \beta_k)\left(\frac{\delta^2 \hat{u}^{k-1}}{\delta x^2} + \frac{\delta^2 \hat{u}^{k-1}}{\delta y^2}\right) \\
&- \frac{\alpha_k \Delta t}{\mathscr{Re}} \frac{\delta}{\delta x}\left(\frac{\delta \hat{u}^{k-1}}{\delta x} + \frac{\delta \hat{v}^{k-1}}{\delta y}\right) \\
&+ \frac{\beta_k \Delta t}{\mathscr{Re}}\left[\frac{\delta^2\left(\hat{u}^k - \hat{u}^{k-1}\right)}{\delta x^2} + \frac{\delta^2\left(\hat{u}^k - \hat{u}^{k-1}\right)}{\delta y^2}\right] \\
&- \gamma_k \Delta t\left(\frac{\delta}{\delta x} u^{*k-1} u^{*k-1} + \frac{\delta}{\delta y} u^{*k-1} v^{*k-1}\right) \\
&- \zeta_k \Delta t\left(\frac{\delta}{\delta x} u^{*k-2} u^{*k-2} + \frac{\delta}{\delta y} u^{*k-2} v^{*k-2}\right)
\end{aligned}
\tag{25}
$$

Now let us find the expression for the finite difference operators in Eq.(8) and Eq.(9).

Problem: First, consider the Viscous term (8) as follows:

$$L(u_i) = \frac{1}{\mathscr{Re}} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{26}$$

For a 2 dimensional coordinate system, one has the expression such that:
In x-direction:

$$L_x(u) = \frac{1}{\mathscr{Re}} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{27}$$

In y-direction:

$$L_y(v) = \frac{1}{\mathscr{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{28}$$

The second derivative at an arbitrary point (i, j) is approximated by the second central difference:

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \tag{29}$$

$$\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} \tag{30}$$

and:

$$\frac{\partial^2 v_{i,j}}{\partial x^2} = \frac{v_{i+1,j} - 2v_{i,j} + v_{i-1,j}}{\Delta x^2} \tag{31}$$

$$\frac{\partial^2 v_{i,j}}{\partial y^2} = \frac{v_{i,j+1} - 2v_{i,j} + v_{i,j-1}}{\Delta y^2} \tag{32}$$

Viscous Flux Calculation in Code

```cpp

void viscous_flux_calc (Array<MultiFab, AMREX_SPACEDIM>& viscous_flux, Array<MultiFab, AMREX_SPACEDIM>& velocity, Geometry const& geom, Real const& ren){

    GpuArray<Real, AMREX_SPACEDIM> dx = geom.CellSizeArray();

#ifdef AMREX_USE_OMP
#pragma omp parallel if (Gpu::notInLaunchRegion())
#endif
    for ( MFIter mfi(velocity[0]); mfi.isValid(); ++mfi )
    {
        Box xbx = mfi.tilebox(IntVect(AMREX_D_DECL(1,0,0)));
        Box ybx = mfi.tilebox(IntVect(AMREX_D_DECL(0,1,0)));

        auto const& vel_x = velocity[0].array(mfi);
        auto const& vel_y = velocity[1].array(mfi);

        auto const& visc_flux_x = viscous_flux[0].array(mfi);
        auto const& visc_flux_y = viscous_flux[1].array(mfi);
#if (AMREX_SPACEDIM > 2)
        Box zbx = mfi.tilebox(IntVect(AMREX_D_DECL(0,0,1)));
        auto const& vel_z = velocity[2].array(mfi);
        auto const& visc_flux_z = viscous_flux[2].array(mfi);
#endif

        amrex::ParallelFor(xbx,
        [=] AMREX_GPU_DEVICE (int i, int j, int k)
        {
            // x-stencils
            auto const& northMAC = vel_x(i, j+1, k);
            auto const& southMAC = vel_x(i, j-1, k);

            auto const& westMAC = vel_x(i-1, j, k);
            auto const& eastMAC = vel_x(i+1, j, k);

            auto const& centerMAC = vel_x(i, j, k);

            visc_flux_x(i, j, k) = ((westMAC - 2*centerMAC + eastMAC)/(dx[0]*dx[0]) + (southMAC - 2*centerMAC + northMAC)/(dx[1]*dx[1]))/ren;
        });

        amrex::ParallelFor(ybx,
        [=] AMREX_GPU_DEVICE (int i, int j, int k)
        {
            // y-stencils
            auto const& northMAC = vel_y(i, j+1, k);
            auto const& southMAC = vel_y(i, j-1, k);

            auto const& westMAC = vel_y(i-1, j, k);
            auto const& eastMAC = vel_y(i+1, j, k);

            auto const& centerMAC = vel_y(i, j, k);

            visc_flux_y(i, j, k) = ((westMAC - 2*centerMAC + eastMAC)/(dx[0]*dx[0]) + (southMAC - 2*centerMAC + northMAC)/(dx[1]*dx[1]))/ren;
        });

#if (AMREX_SPACEDIM > 2)
        amrex::ParallelFor(zbx,
        [=] AMREX_GPU_DEVICE (int i, int j, int k)
        {
            // dummy code
            auto const& northmac = vel_z(i, j+1, k);
            auto const& southmac = vel_z(i, j-1, k);

            auto const& westmac = vel_z(i-1, j, k);
            auto const& eastmac = vel_z(i+1, j, k);

            auto const& centermac = vel_z(i, j, k);

            visc_flux_z(i, j, k) = ((westmac - 2*centermac + eastmac)/(dx[0]*dx[0]) + (southmac - 2*centermac + northmac)/(dx[1]*dx[1]))/ren;
        });
#endif
    }
}
```

Now let us find the expression for the finite difference operators in Eq.(8) and Eq.(9).

Problem: First, consider the Viscous term (8) as follows:

$$L(u_i) = \frac{1}{\mathcal{Re}} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \tag{33}$$

For a 2 dimensional coordinate system, one has the expression such that:
In x-direction:

$$L_x(u) = \frac{1}{\mathcal{Re}} \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{34}$$

In y-direction:

$$L_y(v) = \frac{1}{\mathcal{Re}} \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \tag{35}$$

The second derivative at an arbitrary point (i, j) is approximated by the second central difference:

$$\frac{\partial^2 u_{i,j}}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \tag{36}$$

$$\frac{\partial^2 u_{i,j}}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} \tag{37}$$

and:

$$u(x, y)^{t=0} = -\cos(2\pi x) * \sin(2\pi y) \tag{38}$$

$$v(x, y)^{t=0} = \sin(2\pi x) * \cos(2\pi y) \tag{39}$$

For inner nodes:

$$\begin{aligned} F = um\left[1/8 * (-U_{i+2} - 2U_{i+1} + 3U_i) + U_{i+1}\right] \\ + up\left[1/8 * (-U_{i-1} - 2U_i + 3U_{i+1}) + U_i\right] \end{aligned} \tag{40}$$

For begin node ($i = 1$):

$$\begin{aligned} F = um\left[1/8 * (-U_{i+2} - 2U_{i+1} + 3U_i) + U_{i+1}\right] \\ + up\left[1/8 * (-U_i - 2U_i + 3U_{i+1}) + U_i\right] \end{aligned} \tag{41}$$

For end node ($i = N$):

$$\begin{aligned} F = um\left[1/8 * (-U_{i+1} - 2U_{i+1} + 3U_i) + U_{i+1}\right] \\ + up\left[1/8 * (-U_{i-1} - 2U_i + 3U_{i+1}) + U_i\right] \end{aligned} \tag{42}$$

Methodologies

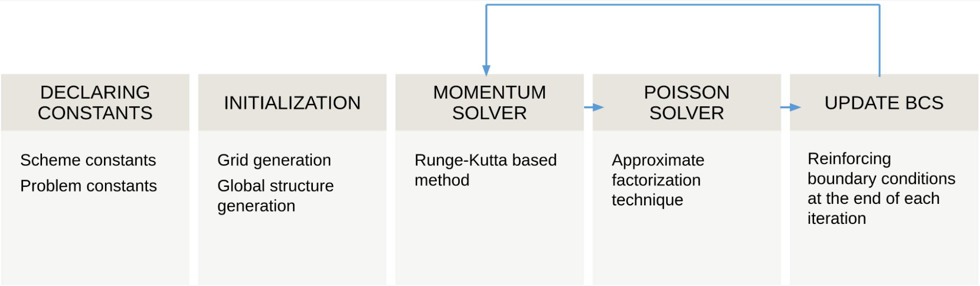
Preconditioned Jacobian-free Newton-Krylov

What is that?

Methodologies

Julian Solver for Incompressible Flow

Code structure:



Momentum Solver >_ Convective Flux

Results

MultiFab Tutorial

This tutorial focus on one of the most important class of AMReX, the MultiFab. The goals are:

- Create a MultiFab.
- Write data to a MultiFab.
- Plot data from a MultiFab.

Remark: What has been learned:

- Defining a MultiFab mf (ba, dm, ncomp, ngrow) takes 4 inputs:
  - `ba`: a BoxArray;
  - `dm`: a Distribution Mapping;
  - `ncomp`: the number of components to be stored in the MultiFab; and
  - `ngrow`: the number of layers of ghost cells.

Completed Module 1 >_ MultiFab Tutorial

- Accessing data in the MultiFab by loop structure: combine MFIter and ParallelFor:

- AMReX allows for selection of either 2D or 3D option at compile time using AMREX_SPACEDIM:

- A newer approach calls for explicit indices and utilize ParallelFor only:

Problem: This series solve the heat equation:

$$\frac{\partial \Phi}{\partial t} = \nabla^2 \Phi \tag{43}$$

Initial Condition

$$\Phi(\vec{r}) = 1 + e^{-r^2}$$

Boundary Condition

Periodic

Solution: A temporal discretization scheme can be used such that:

$$\frac{\partial \Phi}{\partial t} = \frac{\Phi_{i,\,j}^{n+1} - \Phi_{i,\,j}^{n}}{\Delta t} \tag{44}$$

Then, we spatially discretize the PDE by first constructing negative fluxes on cell faces, e.g:

$$F_{i+\frac{1}{2},\,j}^{n} = \frac{\Phi_{i+1,\,j}^{n} - \Phi_{i,\,j}^{n}}{\Delta x} \tag{45}$$

Substituting Eq. (44) and Eq. (45) into the heat equation (43), one can write the 2D update scheme as follow:

$$\Phi_{i,\,j}^{n+1} = \Phi_{i,\,j}^{n} + \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},\,j}^{n} - F_{i-\frac{1}{2},\,j}^{n}\right) + \frac{\Delta t}{\Delta y}\left(F_{i,\,j+\frac{1}{2}}^{n} - F_{i,\,j-\frac{1}{2}}^{n}\right) \tag{46}$$

- Heat_Equation_EX0_C : Most simple method.
- Heat_Equation_EX1_C : Custom kernel for the calculation of Fluxes.
- Heat_Equation_EX2_C : Something.
- Heat_Equation_EX3_C : Linear Solver.

Remark: Hello World!

Problem: The Example 3 of Heat Equation Series has a Initialization kernel in Fortran. Write a C equivalent Initialization kernel for this example.

Solution: The Initialization function (called init_phi(i, j, k, phi, dx, prob_lo) can be defined in a custom kernel (called mykernel.H as follow:

Then a subroutine init_phi(phi_new, geom) can be created to apply function init_phi at each valid node such that:

Meeting Minutues

Thursday, June 1

Dr. Andy J. Nonaka

AMR, Mass Conservation between AMR levels, Mass Flux Interpolation Scheme.

Dr. Trung B. Le

Hybrid Staggered+Un-Staggered Grid.

Thien-Tam Nguyen

LaTeXBeamer file.

Remark: Let's hunt for the best Beamer template.

Polar Night

- text: ■ U ■
- text: ■ U ■
- text: ■ U ■
- text: ■ U ■

Polar Storm

- text: ■ U ■
- text: ■ U ■
- : ■ U ■

Polar Frost

- text: ■ ∪ ■
- text: ■ ∪ ■
- text: ■ ∪ ■
- text: ■ ∪ ■

Polar Aurora

- text: ■ ∪ ■
- text: ■ ∪ ■
- text: ■ ∪ ■
- text: ■ ∪ ■
- text: ■ ∪ ■

Non-Nord Greens

- text: ■
- text: ■

References

References