MPCUGLES Overset Version 1.1 Manual

Authors: Thomas B. Kroll, Wyatt J. Horne, Nick Morse and Krishnan Mahesh

Aerospace Engineering and Mechanics
University of Minnesota

Point of contact: Krishnan Mahesh
kmahesh@umn.edu

MPCUGLES Overset V1

A manual for compiling and running the mpcugles overset code.

More information on the overset assembly **[1]** and the numerical method **[2]** used by the code can be found in the references below.

References:

Horne, Wyatt James and Mahesh, Krishnan. "**A massively-parallel, unstructured overset method for mesh connectivity**". Journal of Computational Physics, 376:585–596, 2019. **[1]**

Horne, Wyatt James and Mahesh, Krishnan. "**A massively-parallel, unstructured overset method to simulate moving bodies in turbulent flows**". Journal of Computational Physics, 397:108790, 2019. **[2]**

Table of Contents

Running mpcugles overset

To run **mpcugles**, use the **mpcugles** executable as follows where **total_procs = nprocs+novrprocs:**

mpirun -np 'total_procs' ./mpcugles > mpcugles.out

- ★ **Note**: **mpirun** & **-np** are machine or queue system specific. Make sure to check these and match them to your machine.

Run Directory

The following are the main contents of a **run** directory. More detailed information on the contents is shown in this section.

- ➔ global.nml
- ➔ user_hooks.f90
- ➔ **Grid Directory**: grid files, overset_grid_orientation.dat
- ➔ **Other**: mpc_main.f90, scalar.dat, kill_switch.dat, interp.f90, CMakeList.txt, propeller.in
- ➔ **Post-Processing**: dump_tecplot_files.f90, post.in, dump_exodus_files.f90, circum_avg_new.f90

global.nml

The global namelist file gives the user control of the many features of the code using flags.

- ★ **Note**: (d) means the default specification for the specific flag.

&global

This section has the main control flags for the code.

General:

nprocs: The number of total processors for use by the background grid partitions. These processors are only assigned as one processor to one background partition.
novrprocs: The number of total processors for use by the overset grid partitions. These processors don't have to be assigned as one processor to one overset partition. Check the Overset section for more details.
reynolds_no: The simulation Reynolds number. In the code, this value is inserted as $1/\nu = Re/L$

body_force_x: A force source term that can be added to the equations to drive flow. Change the final letter to **y** or **z** to insert a force in the other directions. This value is typically theoretically derived from the Navier-Stokes equations and applied, for example, to a Channel Flow case.

- **0.0** (d): All three components (**body_force_x body_force_y body_force_z**) are set to 0.0 by default.

implicit_zero: Tolerance for the momentum equation solve. This value is case specific and should always be checked to achieve the desired convergence of the momentum equations.

- $1e-9$ (d):

pressure_zero: Tolerance for the pressure equation solve. This value is case specific and should always be checked to achieve the desired convergence of the momentum equations.

- $1e-9$ (d):

Runtime Settings:

scalar_cfl: The time step value. Change to match a specific case. Make sure to set it to be the same value as **scalar_four**.

- **1.0** (d):

scalar_four: The time step value. Change to match a specific case. Make sure to set it to be the same value as **scalar_cfl**.

- **1.0** (d):

time_advance_implicit: Select implicit or explicit solve.

- **0** (d): Use Explicit solve.
- **1**: Use implicit solve.

fix_dt_flag: This flag gives the option to keep the solution time step fixed.

- **1** (d): Fix the time step. It is recommended to always use a fixed time step.
- **0**: Use CFL condition with the smallest control volume and the highest velocity to calculate a time step. This flag is used together with **scalar_cfl**.

nsteps: The number of time steps for the case to run.

save_restart_interval: The interval of time steps in which to save a solution.

Grid Information:

grid_input_type: There are two options for grid types. We primarily work with the **nc** (**NetCDF**) format which results from the partitioning of a grid software generated mesh. A uniform or nonuniform, in code structured grid is also an option for use as a background grid only.

- **'nc'** (d):
- **'struct_uniform'**: An in code, uniform structured grid created from user given parameters. Available for use for the background grid.
- **'struct_nonuniform'**: An in code, nonuniform structured grid created from user given parameters. Available for use for the background grid.

input_style: Choose the file type to read in for a restart.

- **'nc'** (d): **NetCDF** format.
- **'hdf5'**: **XDMF+HDF5** format.

output_style: Choose the file type to output or write for the result.

- ➢ **'nc'** (d): **NetCDF** format.
- ➢ **'hdf5'**: **XDMF+HDF5** format.

nout_procs: This is the number of output **XDMF+HDF5** files. Depending on your case, it's recommended this number is about total processors divided by 10 once cases get very large. It's advised to keep this number close to one divisible by the total number of case processors. Otherwise, viewing in parallel might overburden a processor if not evenly distributed.

- ➢ **nprocs** (d): It's defaulted to the number of background grid processors.

File Format Information:

froot: The root name for the mesh files.

- ➢ **'mesh'** (d): Eg. **mesh001.nc**.

fext: The file extension of the **NetCDF** files.

- ➢ **'.nc'** (d): This should never change.

sol_fext: The solution file extension.

- ➢ **'sol.nc'** (d): Eg. **mesh001sol.nc**.

integerinformat: This changes the input grid, integer format that is used to create the grid file naming format within the code. Make sure that this matches the grid integer format you prefer.

- ➢ **'I3.3'** (d): Fortran input/output edit descriptor. The first number after '**I**' is the number of digits to use, and the second number means the print space to use while writing a string. This would result in **mesh001.nc** for a background grid or **mesh001001.nc** for an overset grid. Use this default value if there are at most less than **1001** background or overset grid partitions.

integeroutformat: This changes the output solution, integer format that is used to create the solution file naming format within the code. Make sure that this matches the solution integer format you prefer, ensuring all grids are accounted for.

- ➢ **'I4.4'** (d): The first number after '**I**' is the number of digits to use, and the second number means the print space to use while writing a string. This would result in **mesh0001sol.nc**. Use this default value if there are at most less than **10001** background or overset grid partitions.

result_directory: The name of the result directory.

- ➢ **'result'** (d):

restart_directory: The name of the restart directory.

- ➢ **'restart'** (d):

scalar_data_file: The name of the files used to control additional scalars for the simulation. It is recommended that this file isn't edited. Check the scalar.dat section for more information.

- ➢ **'scalar.dat'** (d):

Overset:

has_overset:

- ➢ **'yes'** (d): Turn on overset capabilities.
- ➢ **'no'**: Turns off overset capabilities to run base mpcugles for a single grid. Always make sure you set the propeller.in file correctly to turn on rotation for a propeller.

numovrgridunique The number of unique overset grids for the simulation. Adjust this number if using multiple unique overset grids.

overlap: This flag controls globally what percentage of the overset mesh's volume should be removed from the background grid. Ranges (0.0-1.0). A value of 1.0 would mean no background control volumes (CVs) are removed within the overset grid confines and turns off volume cutting.The Volume Cutting section has more information.

nfaces_to_test: The number of faces to test while using surface cutting to remove redundant CVs.

num_interp_parts_max: This is the maximum number of interpolation partners. It is important to increase this value as ratios between grid CV sizes increase at interpolation regions. This would result in increasing the search for interpolation partners and also possibly slow down the case. Try to design grids that don't exceed a ratio of 2 to 1 in independent directions at interpolation regions. If a CV can't find interpolation partners, the code will stop. This can also happen at different configurations as meshes move. Check the Cutting and Interpolation section for more details.

- ➢ **1** (d):

chimera_interp_style: This selects the type of interpolation to use.

- ➢ **'supercell-lsq'**: Use volume conserving supercell reconstruction to perform least squares interpolation.
- ➢ **'supercell-mapped-lsq'** (d): Use volume conserving supercell reconstruction with mapped coordinates to perform least squares interpolation. This option improves results for high aspect ratios & poor quality meshes. It works for interpolation regions with Hexahedrons, otherwise the code automatically defaults to **'supercell-lsq'**.
- ➢ **'supercell-rbf'**: Use volume conserving supercell reconstruction to perform interpolation using a polyharmonic spline.
- ➢ **'lsq'**: Use least squares to fit a straight line at all interpolation points. This flag isn't recommended as it has been shown the supercell is required to conserve kinetic energy.

penalty_wt: The penalty weight is used to penalize the equations of CVs involving interpolation partners. This is only done for the pressure equation solve during the corrector step and doesn't affect the momentum equation directly. Increasing this weight penalizes interpolation CVs more, making the solution more continuous but at much more computational expense. It is highly recommended to leave its value at 1.0.

- ➢ **1.0** (d): This sets a penalty factor of 1.0. Though local parameters help determine the local penalty weight, this value acts as a global factor and is multiplied to all locally determined penalty weights for an overall effect on convergence of the case.

tau_stab: This flag controls pressure stabilization in time. It's recommended that it's value is equal to the time step value if in use. This will in effect use a symmetric 50% of the previous and 50% of the current time step pressure values as the full result. Values larger than the time step result in using a higher percentage of the previous timestep pressure value while lower values result in using a lower percentage. Use only when the specific case involves overset mesh movement and time step to time step pressure fluctuations affect force calculations, otherwise set to zero.

- **0.0** (d): This turns off pressure stabilization.

Overset Solvers:

overset_time_step_style:

- **'ab2' (d)**: Adam-Bashforth 2nd order. (Explicit Solve).
- **'cnicol'**: Crank-Nicholson. (Implicit Solve).
- **'bdf2'**: Backwards Differencing 2nd order. (Implicit Solve).
- **'cnicol_node'**: Crank-Nicholson. (Implicit Solve). With nodal reconstruction.
- **'bdf2_node'**: Backwards Differencing 2nd order. (Implicit Solve). With nodal reconstruction.

max_hypre_iterations: The number of iterations to converge pressure in **HYPRE**. Set this value to get desired convergence.

- **50 (d):**

hypre_pressure_solver:

- **'gmres' (d)**: **'FlexGMRES'**. This is a generalized minimal residual method. Iterative method for numerical solution of non-symmetric system of linear equations. Normally used with a preconditioner. For pressure, this is best when used with the **amg** preconditioner. This is the recommended solver.
- **'amg'**: **'BoomerAMG'**
- **'bicgstab'**: **'BICGStab'**
- **'pcg'**: **'Preconditioned Conjugate Gradient'**

hypre_pressure_precond:

- **'none' (d)**: no preconditioning.
- **'diag'**: diagonal preconditioning.
- **'Ilu'**: euclid preconditioner.
- **'psails'**: parasails preconditioner.
- **'amg'**: **AMG** preconditioner.

hypre_momentum_solver:

- **'gmres'** (d): **'FlexGMRES'**. This is a generalized minimal residual method. Iterative method for numerical solution of non-symmetric system of linear equations. Normally used with a preconditioner. For pressure, this is best when used with the **diag** preconditioner. This is the recommended solver.

- ➢ **'amg'**: **'BoomerAMG'**
- ➢ **'bicgstab'**: **'BICGStab'**
- ➢ **'pcg'**: **'Preconditioned Conjugate Gradient'**

hypre_momentum_precond:

- ➢ **'none' (d)**: no preconditioning.
- ➢ **'diag'**: diagonal preconditioning.
- ➢ **'llu'**: euclid preconditioner.
- ➢ **'psails'**: parasails preconditioner.
- ➢ **'amg'**: **AMG** preconditioner.

★ More information on **HYPRE** solvers and preconditioners can be found at the HYPRE documentation page. There's also some more information in this document in the HYPRE section.

Single Grid Solvers:

pressure_solver: These are the different solvers available to solve the pressure equation.

- ➢ **'mpc_cg'** (d): The Conjugate Gradient method with diagonal preconditioning.
- ➢ **'mpc_sor'**: The Point Successive Over-Relaxation method.
- ➢ **'mpc_jac'**: The Point Jacobi method.

mpc_phi_omega_sor: (0.0-1.0) values to use for segregated solve and **mpc_sor**

- ➢ **1.0** (d): It's recommended to use 0.5.

max_pressure_iterations: number of iterations to converge pressure

- ➢ **1000** (d)

max_outer_iterations: number of iterations to converge pressure on the outer loop for segregated solve.

- ➢ **25** (d)

Animations:

These flags work together to turn on animations. Make sure to check all before running a simulation with an animation.
When animations are turned on, the simulation will save a copy of the solution at a specified interval. An animation will only begin if **anim_stop** > **anim_start**. The user can turn off animations by setting **anim_stop** < **anim_start,** this is defaulted this way as shown below. To turn on animations, just simply choose the beginning, interval and end of your animation and edit the following flags.

anim_start: The time step to start saving an animation copy. Must be less than **anim_stop** to turn on animations.

- **2** (d):

anim_steps: The time step interval for saving an animation copy.

- **1** (d):

anim_stop: The time step to stop saving an animation copy. Must be greater than **anim_stop** to turn on animations.

- **1** (d):

frame_num: This flag sets the number used for the first written animation frame. For example, 1 would mean the first written animation frame is named **frame001**. This flag is useful when restarting simulations.

- **1** (d):

Statistics:

These flags work together to turn on statistics. Make sure to check all before running a simulation.
When statistics are turned on, samples of the main variables are collected. Statistics will only begin sampling if **stats_stop** > **stats_start** and **statistics_flag** is set to 1 or 2. The user can turn off statistics by setting **stats_stop** < **stats_start** and **statistics_flag** to 0**,** this is the defaulted setting as shown below.

statistics_flag: The main flag controlling statistics.

- **0** (d): Statistics Off
- **1**: Start statistics afresh.
- **2**: Restart statistics and continue sampling.

stats_start: The time step to start sampling. Must be less than **stats_stop** to turn on statistics.

- **2** (d):

stats_steps: The time step interval for sampling.

- **1** (d):

stats_stop: The time step to stop sampling. Must be greater than **stats_stop** to turn on statistics.

- **1** (d):

&les

This section contains flags that control the large-eddy simulation (LES) models and settings.

subgrid_term: This turns the subgrid term on or off.

- **0** (d): LES Off
- **1**: LES On

les_model: This selects the model to be used for LES.

- **'dynamic'** (d): The Dynamic model for eddy viscosity.

- ➢ **'smagorinsky'**: The Smagorinsky for eddy viscosity.
- ➢ **'control'**: The Controlled Dynamic model for eddy viscosity.

les_avg: This selects the local eddy viscosity averaging/smoothing method. Without any kind of averaging, the local dynamic model is known to predict a highly variable eddy viscosity field that sometimes has negative values. This is caused by the model coefficient C_s which is dynamically computed to vary in space and time.

- ➢ **'no'** (d): Localised dynamic model. This runs the dynamic model without any averaging.
- ➢ **'neighbor'**: Neighbor averaged dynamic model. This method averages C_s using neighbor CVs to help get a less variable eddy viscosity field.
- ➢ **'plane'**: Homogeneous directions/plane averaged dynamic model. This method averages C_s along homogeneous flow directions, along planes. It is used for channel flow where it is possible to set that restriction. The requirement of averaging over at least one homogeneous direction is impractical for complex inhomogeneous flows.
- ➢ **'lagrangian'**: Lagrangian averaged dynamic model. This method averages C_s along fluid trajectories. Lagrangian averaging has been shown to perform best for inhomogeneous flows such as flow past a marine propeller.

les_test_filter:

- ➢ **'trapezoidal'** (d): Trapezoidal FIltering.
- ➢ **'tophat'**: Tophat FIltering.

clip_nu_t: This changes the eddy viscosity clipping settings. Any eddy viscosity values larger than a set value are set to that value. For marine propellers this value is usually set to 2.

- ➢ **1** (d): The subgrid filter value is set to 1/Re.
- ➢ **2**: The subgrid filter value set to 0.01.

&struct_grid

The user can create a structured background grid for their case using this section.

- ★ Make sure to set the **grid_input_type** flag to **'struct_uniform'**
- ★ Make sure to follow the correct ordering in specifying variables
- ★ Currently, this only works with **HDF5** output.
- ★ Any BC name with **'wall'** in the name is a no slip BC for LES since the FLUENT BC information isn't available for this grid.

struct_type: The BC type for the 6 faces of the grid. (x-1, x-2. y-1. y-2. z-1, z-2)

- ➢ **1**: Dirichlet BC
- ➢ **2**: Outflow BC
- ➢ **3**: Periodicity BC

struct_bcs_r: The BC names. For eg. "wall".
struct_lx: The length of the grid in a specific direction (x,y,z)
struct_or: The offset in a specific direction. Used to determine the centering of the grid (x,y,z)
struct_nx: The number of CVs in a specific direction (x,y,z)

&solver:

user_hooks.f90

This file contains **"hooks"** supplied to the user for implementing. More information on the specific implementations are shown below. Please check the test cases for implementation examples.

hook_data module:

This module is used to define relevant variables that will be used in the **user_hooks.f90** file.

initial_report_hook:

This is used to write out of any relevant information before the case begins.

temporal_report_hook:

This is used to write out of any relevant information or do calculations as the case runs. It is called after a time step is finished. For example, maximum velocity and pressure values can be calculated and printed to output here.

final_report_hook:

This is used to write out any relevant information before the case is finished.

initial_condition_hook:

This is used to set initial condition values. The most important variable that needs to be initialized is the velocity field. The user has the capability of manually creating many different initial conditions for any variables as needed.

boundary_condition_hook:

This is used to set case boundary conditions. It contains a do loop through all boundary zones read from the grid files. It is important that the boundary names specific to the case are entered here.
The different boundary conditions are listed below:

- **Overset Fringe**: This is the overset fringe boundary condition where interpolation occurs. By design, the user can create surfaces that have interpolation CVs. Insert the names and then nothing from the user is required here.
- **Periodic**: Insert the names and then nothing from the user is required here.
- **Outflow**: Insert the names and then nothing from the user is required here.

- **Inflow**: Insert the names. The inflow velocity at the inflow faces for the case is required to be entered.
- **Wall**: Insert the names. Distinguish the no slip boundary conditions to moving surfaces. Any surfaces in the background are defaulted to no slip. Overset surfaces are defaulted to the specific grid velocity.

It is also possible to edit this section and add more complicated boundary conditions manually.

definitions_hook:

This is used to set or define important overset and plotting variables.
Important variables to always be defined:

- Overset fringe boundary names
- Wall cut names
- Wall names for plotting

Grid Directory

Background

The background grid is contained in the **grid** folder. The background grid is always assumed to be stationary and should be large enough to contain all of the other overset meshes. The background grid partitions are always assigned one to one to background processors or in other words the number of background processors should always equal the number of background partitions. Check the figure in this section or a visual representation of this.

Naming Convention

The numbers start from zero and represent the specific partition of the grid.
mesh000.nc, mesh001.nc, **mesh002.nc**: The background grid with 3 partitions.

Overset

The overset meshes are located in the **overset** folder inside the **grid** folder.

A big difference compared to the background grid is that overset processors can have multiple unique grids with numerous partitions assigned to the overset processors. This is because the code allows for the overset processors to work on multiple partitions at a time. This results in overset partitions then cyclically being assigned to overset processors. The figure below from **[1]** shows this visually. For more in depth details, the user can look at the communication section of **[1]**.

Though this capability is very useful, it is recommended to keep in mind the load balancing of the case. It is advised to aim for CVs to processor ratios that are approximately constant along all processors to get the best balanced performance. For example, if the CVs per

processor/partition ratio is 25,000 for the background grid, the user should aim for that ratio for all the overset processors as well.

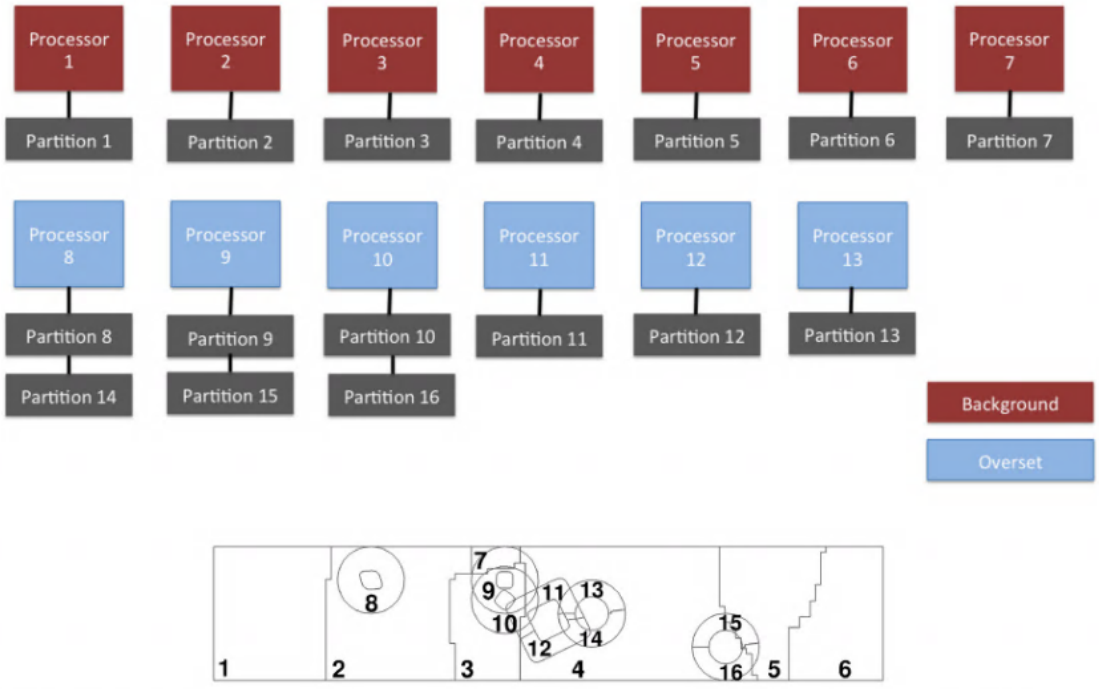


Fig. 1. Partitioned background and overset meshes in channel with partition numbering and resulting processor assignment color coded by processor type for 13 processors and 16 partitions. Numbering begins on the background channel mesh and continues onto the overset mesh partitions. Note the cyclical processor assignment is only applied to the overset partitions.

- ★ **Note**: In the above figure, **6** overset processors are working on all the **9** overset partitions. The user has to make sure that every overset processor has a partition to work on, otherwise the code will not run. This limitation would mean that the most overset processors that can be used is **9**, where one overset processor is assigned to one partition.
- ★ **Note**: On the opposite end, you need a minimum of **2** overset processors to run this case. This is because the limitation is the maximum number of partitions on a unique overset mesh.
- ★ **Note**: The user needs to be aware of the additional load on processors when using too many overset partitions with too little overset processors. For example, the case in the figure above can be run with only **2** overset processors working on all **9** overset partitions. However, the load would be great and potentially unbalanced if these partitions have too many CVs individually, yielding a high CVs per processor ratio. This might be fine, however, if the partitions are very small and the CVs per processor ratio ends up matching that of the background grid, yielding a well load balanced case.

Naming Convention

The first number represents the unique overset grid and the second number represents the specific partition of the grid.
mesh001000.nc, mesh001001.nc, mesh001002.nc: The first overset grid with 3 partitions.
mesh002000.nc, mesh002001.nc: The second overset grid, with 2 partitions.

- ★ **Note**: The unique grid number starts with 1 and the partition number starts with 0 for visual clarity.

Grid Information

With two unique overset grids, the user needs to set the global flag **numovrgridunique** = 2. The user also needs to set up the overset mesh information in the overset_grid_orientation.dat file. Below is the detailed information in the file for the specific grids.

File Contents: overset_grid_orientation.dat

1
0.000000 0.000000 0.000000 5.0 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
1
0.000000 0.000000 0.000000 3.7 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1

- ★ **Note**: The 3 lines below the number of clones here represent 1 line in the file. Each line is color coded to represent the specific clone. Here there is only one clone or copy of each grid.
- ★ **Note**: The first grid has 1 clone of itself and the second grid has 1 clone of itself.

Cloning

If the user has an overset mesh and they want to replicate it, there is a capability to clone a mesh internally. The user would need to edit the overset_grid_orientation.dat file. More information on this file is shown in the linked section.

What enables cloning? The answer is that the code allows for the overset processors to work on multiple unique grids with numerous partitions. When cloning, the code will assign the cloned grid partitions cyclically to the overset processors only. The user can have numerous unique overset grids with each having multiple clones. This figure from **[1]** shows this visually.

An example of setting the overset_grid_orientation.dat file so that the above grid from this section has clones is shown below. The first grid has 3 clones of itself with 3 lines below this number giving detailed information on the specific clones. The second grid has 2 clones of itself with 2 lines below this number giving detailed information on the specific clones.

File Contents: overset_grid_orientation.dat

3
0.000000 0.000000 0.000000 5.0 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
0.000000 0.000000 0.000000 2.5 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
0.000000 0.000000 0.000000 7.5 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
2
0.000000 0.000000 0.000000 3.7 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
0.000000 0.000000 0.000000 6.2 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1

★ **Note**: The 3 lines below the number of clones here represent 1 line in the file. Each line is color coded to represent the specific clone.

overset_grid_orientation.dat

This file gives local instructions and information for all the overset grids in the simulation. It is important that it is checked and edited before starting a new case. It is located in two locations. The main copy is in the **grid/overset** folder and this gives initial instructions . A different copy of the file is also saved every time a solution is saved in the **result** folder with instantaneous overset mesh information. This file is very important in order to restart a solution especially one with movement involved.

File contents: overset_grid_orientation.dat

---

clone_num
gridxcom(1:3), gridcc(1:3), gridth(1:3), gridv(1:3), gridomega(1:3), gridaccel(1:3), gridalpha(1:3), mbody, rhobody, Ibody(1:3), mesh_overlap, volume_cut_style, body_num, body_vol_cut, body_surf_cut, grid_color

---

- ★ **Note**: The 3 lines below the number of clones here represent 1 line in the file. This is a general representation for information on one grid only, as will be seen in examples, this file will have multiple lines for multiple grids.
- ★ **Note**: This file only gives information for overset grids as they are the only grids that have any of the overset capabilities enabled unlike the background grid.
- ★ **Note**: The background grid's center (0,0,0) is used as the case/global center.

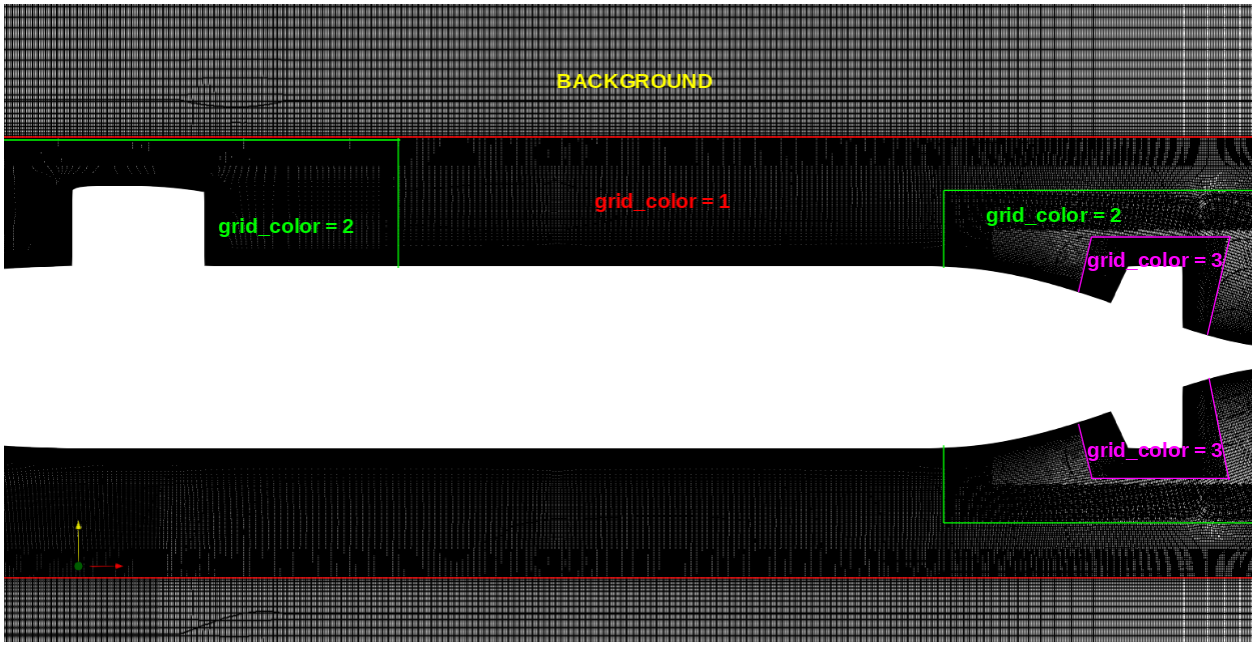
Summary of variables:

- ❖ **clone_num:** The number of clones of the specific grid.
- ❖ **gridxcom(1:3)**: The x, y, z location of the grid center of mass according to itself. Normally (0,0,0).
- ❖ **gridcc(1:3)**: The x, y, z location according to the background grid. Use this to move the grid position around. This is useful to move a grid that doesn't have the same center as the background grid. The background grid's center (0,0,0) is used as the case/global center.
- ❖ **gridth(1:3)**: The grid euler angle of orientation in radians. The convention is the right-hand rule. The rotations are about the background grid's center axes which are used as the global axes.
- ❖ **gridv(1:3)**: The grid velocity. **Note**: **Be careful about how high this is per time step!
- ❖ **gridomega(1:3)**: The grid angular velocity.
- ❖ **gridaccel(1:3)**: The grid linear acceleration.
- ❖ **gridalpha(1:3)**: The grid angular acceleration.
- ❖ **mbody**: The grid body mass. Turn to zero to turn off gravitational forces.
- ❖ **rhobody**: The grid density.
- ❖ **ibody(1:3)**: The grid principal moments of inertia.
- ❖ **mesh_overlap**: Specifies local, grid overlap (0.0 to 1.0). Set to 0.0 to revert to overlap specified in **global.nml**.
- ❖ **volume_cut_style**: **1** (Rectangular Cut), **2** (Spherical Cut), **3** (Projected Cut, from all surfaces located in overset grid and outward), **4** (Cylindrical cut, only works along the **x**-axis). These cuts are always applied on the background grids first and centered on the overset grid's center.
- ❖ **body_num**: Specifies body number for meshes on the same body. Set to 0 for no assignment. Volume cutting between overset grids will only occur for overset grids with the same body number.
- ❖ **body_vol_cut**: Specifies whether this mesh will be volume cut by other overset meshes on the same body (1 for yes, 0 for no). Does not affect background volume cutting.

- **body_surf_cut**: Specifies whether this mesh will be surface cut by other meshes on the same body (1 for yes, 0 for no).
- **grid_color**: Each grid has a "**color**" or "**hierarchy level**" at the end of each line in **overset_grid_orientation.dat**. Grids of the same body number will only be cut by grids of equal color or their color plus one. This allows for layering of many overset grids. To reproduce regular overset cutting behavior, simply make this value 1.

Always make sure to set this file correctly and use the **dump_grid** flag to examine the case before attempting to run.

An example of a case where grid coloring or hierarchy levels can be relevant is shown below. Without this, some of the grids using volume cutting might remove entire grids that are within their confines.



Cutting

Cutting is an essential part of the overset code, especially while running with multiple overset meshes. While using the overset method, there are redundant Control Volumes (CVs) in locations with multiple meshes. It is imperative that they are removed from being part of the solution and the solve. CVs that are removed from the solution are given a **mask** value of **0** while CVs in the solution have the **mask** value of **1**. There are two types of cutting that occur. Details are given below.

Volume Cutting

This is the removal of redundant CVs on the background grid that are within an overset mesh. In other words, overset grid CVs take priority and background grid CVs are removed.

Below are relevant global flags that control volume cutting:
overlap: This flag controls globally what percentage of the overset mesh's volume should be removed from the background grid that intercepts it. Ranges (0.0-1.0). A value of 1.0 would mean no background CVs are removed within the overset grid confines.

- ★ **Note**: As the value of the **overlap** flag reduces from 1.0, (1.0 - overlap) fraction of the overset grid volume of CVs are removed starting from the center and expanding outward towards the edges of the overset grid. Different types of cuts can have other varying effects. Try changing this value in the test cases to visually inspect how it changes.
- ★ **Note**: Volume cutting of other local overset meshes or between overset meshes is also possible and that can be controlled using the overset_grid_orientation.dat file and the hierarchical definitions described in that section. In this file you can also control the different types of volume cuts.

Surface Cutting

This is the removal of CVs that are within surfaces. Think of any CVs that would be located within the confines of a spherical surface for example. These need to be removed as part of the solution and solve.

Below are relevant global flags that control surface cutting:
nfaces_to_test: The number of faces to test while cutting. The code will test this number of faces and if a majority of them pass geometric tests to determine whether a CV is inside a surface, that CV will be removed.

- ★ **Note**: Sometimes surface cutting can fail if the number of faces used to do geometric tests is too small while working with a complex surface. Make sure to use the **dump_grid** flag to inspect the meshes.
- ★ **Note**: After setting up your cutting, always make sure that there is enough overlap between meshes in interpolation regions before starting a case. Little overlap can lead to interpolation CVs interpolating from other CVs too close to other interpolation CVs which increases errors. It can also lead to failure to find partners. Use the **dump_grid** flag to inspect the meshes and their setup before running.

Interpolation

The unmasked CVs directly next to masked CVs are called interpolation CVs and have an **interpolation mask** (**interp_mask**) value which is set to 1.0, otherwise the value is 0.0 for all other CVs. These CVs can be thought of as ghost CVs and need boundary condition information. The faces next to masked CVs are called fringe boundaries and the user can design grids with these boundaries.

Interpolation Partner Search

The code goes through a process to search for interpolation partner CVs for the interpolation/fringe CVs. This is a point search based on centroids.
For an interpolation CV, the code will search for CVs that:

1. Are from other partitions or unique grids.
2. Intercept the specific interpolation CV.
3. Aren't masked out of the solution.
4. Aren't interpolation CVs themselves.

The CVs that match this description are the candidates for interpolation partners and their information as well as their neighbors are sent to the interpolation CV. Even with temporal coherence to minimize its effects, the interpolation partner search is the costliest part of the mesh assembly process that can affect the length of a time step. For more details on the interpolation partner search, check the publication **[1]**.

num_interp_parts_max

num_interp_parts_max is an important flag to control the maximum number of interpolation partners that the code searches for an interpolation/fringe CV. In other words, this is a global variable that controls the maximum CV candidates to be used as interpolation partners for an interpolation/fringe CV.

Selecting a value for **num_interp_parts_max** can be simplified to the user knowing their grid design very well and also to the quality of solution they are interested in. While designing grids and their volume/surface cuts, you should know at what locations there will be large CV size differences. If at this location, there are approximately **3** CVs inside one, then set **num_interp_parts_max** to **3**. You have now increased the number of potential partner candidate CVs for all interpolation CVs globally and in this bad region it will take longer to find the **3** CVs that overlap the interpolation CV. This will give the highest quality solution as it allows for the correct use of the volume conserving, supercell interpolation. However, the negative effect would be that you have slowed down the case due to the interpolation search process for this one region taking longer.

The overall per time step case speed is dependent on **load balancing**, the **time step**, and **momentum/pressure equation convergence**. For this case, the slowing down of the case might not be as noticeable but it starts becoming more noticeable when using **num_interp_parts_max** greater than 10 and becomes part of the things that matter to case speed. Note that if a CV is only intercepted by one partner CV, then increasing **num_interp_parts_max** will have no effect on it. It’s only effect is to increase the search range and is applicable in regions like this where **3** partners will be found.

If the user isn’t interested in the highest quality solution for the reason that maybe this interpolation region is near the freestream where there aren't high gradients or they want to

improve the case speed etc. they can select **num_interp_parts_max** less than **3**. This will still run but it will be a technically lesser quality solution. Even in areas like the freestream, errors can build up enough to affect the local flow if the interpolation CV ratios are too high. A good way to check for this issue is plotting the **penalty force** variable, **hdf5** solution files should have this variable. For **nc** solution files, **ncdump -h mesh000sol.nc** will show the maximum penalty force for all background partitions if it's a background solution file and the same applies for an overset partition file. Areas without a good supercell from correct ratios will have higher values of the penalty force. It is up to the user to balance these pros and cons to best match their case and situation.

Interpolation CV size differences

It is generally recommended to avoid large differences in CV sizes (aim for max 2.0 ratio in each direction) at interpolation regions, especially in areas of high gradients. There are several reasons for this limitation.

Supercell Volume Conservation

For the case of a large, good quality interpolation CV with many potential, good quality partners that intercept it, it will always find an interpolation partner. It is however important to increase **num_interp_parts_max**. This is important in order to find enough partners to perform the volume conserving supercell interpolation for the highest quality solution. It needs enough partners to match it's volume and this could mean using many partner CVs. Even though the supercell interpolation gives a higher quality solution, there is a limit. Note that if the differences get too high, even the supercell interpolation can lead to a bad quality solution. For more details on the volume conserving supercell interpolation, check the publication **[2]**.

Failed Interpolation Partner Search

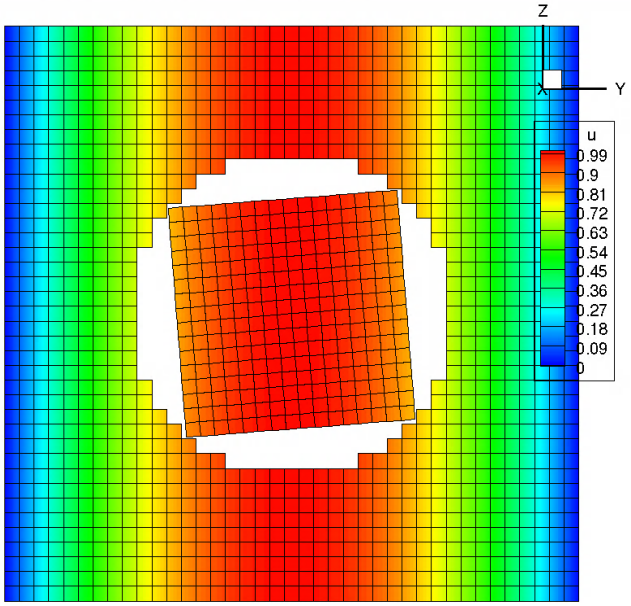
When interpolation CVs fail to find partners, an error with information will be given. This error will write out **"Could not find connectivity information for a cv"** with additional information on the rank and location of the CV. This error can occur due to reasons from this section and the due to lack of overlap which is discussed in the next section.

Sometimes interpolation CVs fail to find partners even though there are candidate interpolation partner CVs that intersect it. This is a rare and special case that can be avoided by good grid designs. The cause for this is bad quality high aspect ratio CVs near good quality low aspect ratio ones in an interpolation region with large CV size ratios. Due to the point based centroid search, there can be cases where a CV centroid is closest to an interpolation CV centroid but it actually doesn't intercept it. With **num_interp_parts_max** set to only one, this will lead to the conclusion that no partners were found for that one CV. This issue can be exacerbated when ratios between grid sizes are too large and in the presence of many overset grids in the same interpolation region. To fix this, it is recommended to increment **num_interp_parts_max** up by one until all interpolation CVs can find partners.

Lack of Overlap

The user should make sure there is enough overlap between grids for CVs to find enough interpolation partners. If a surface cut or volume cut is used incorrectly and removes all CVs around an interpolation CV, the interpolation CV becomes an **orphan** and will fail to find partners that intercept it.

The case can be that there aren't any unmasked CVs that intercept the interpolation CV. If this is the case, **num_interp_parts_max** can not help. The user can use the **dump_grid** flag to observe the meshes and fix the **overlap** flag or change the design of the meshes. Below is an example where interpolation CVs fail to find partners since no unmasked CVs intercept them.



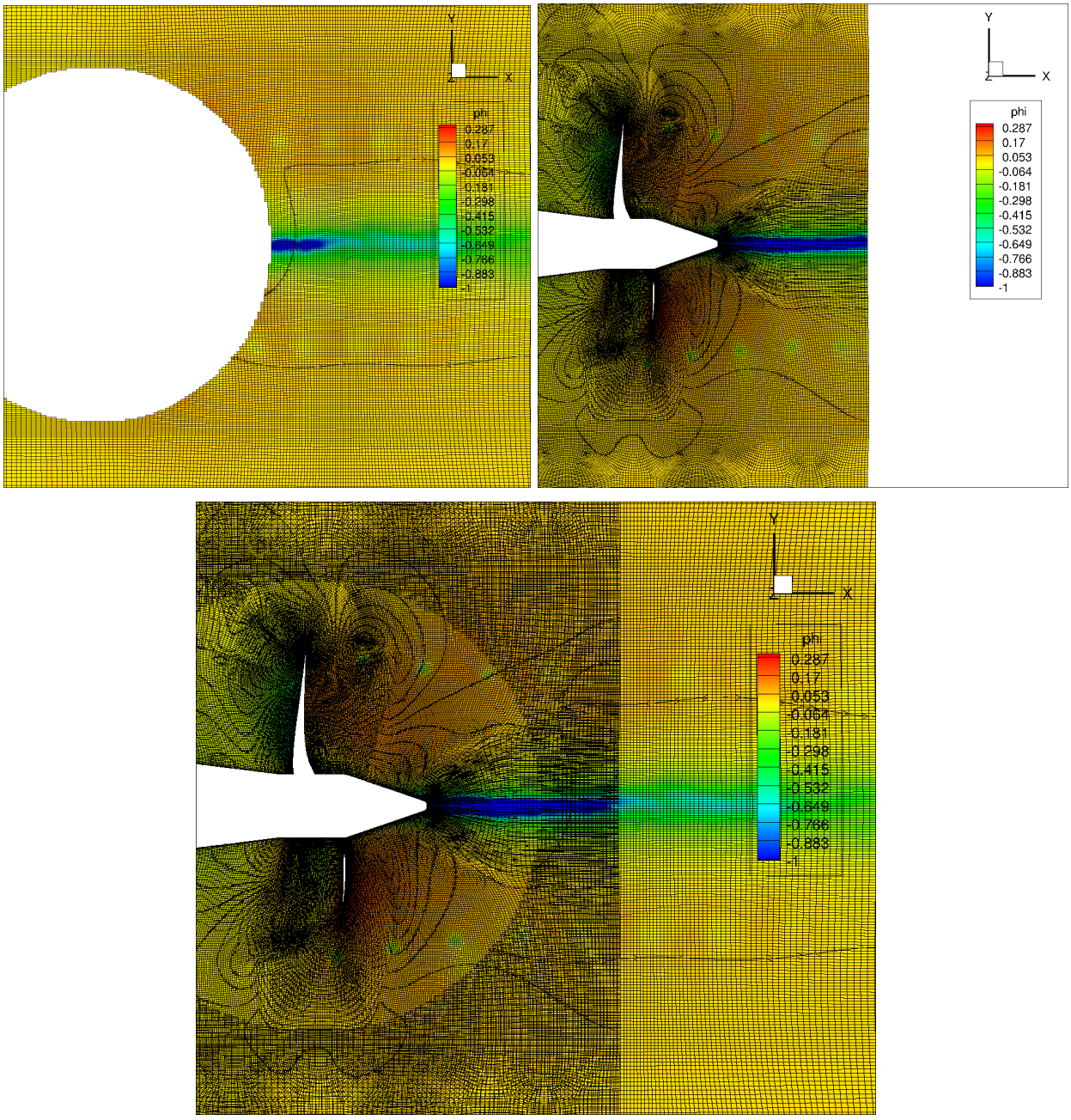
In addition, it is recommended to have at least 5 interior control volumes between two areas that are fringe/interpolation CV surfaces. This is to avoid using interpolation partner CVs too close to other interpolation CVs which increases interpolation error.

Fringe/Interpolation Surfaces Example

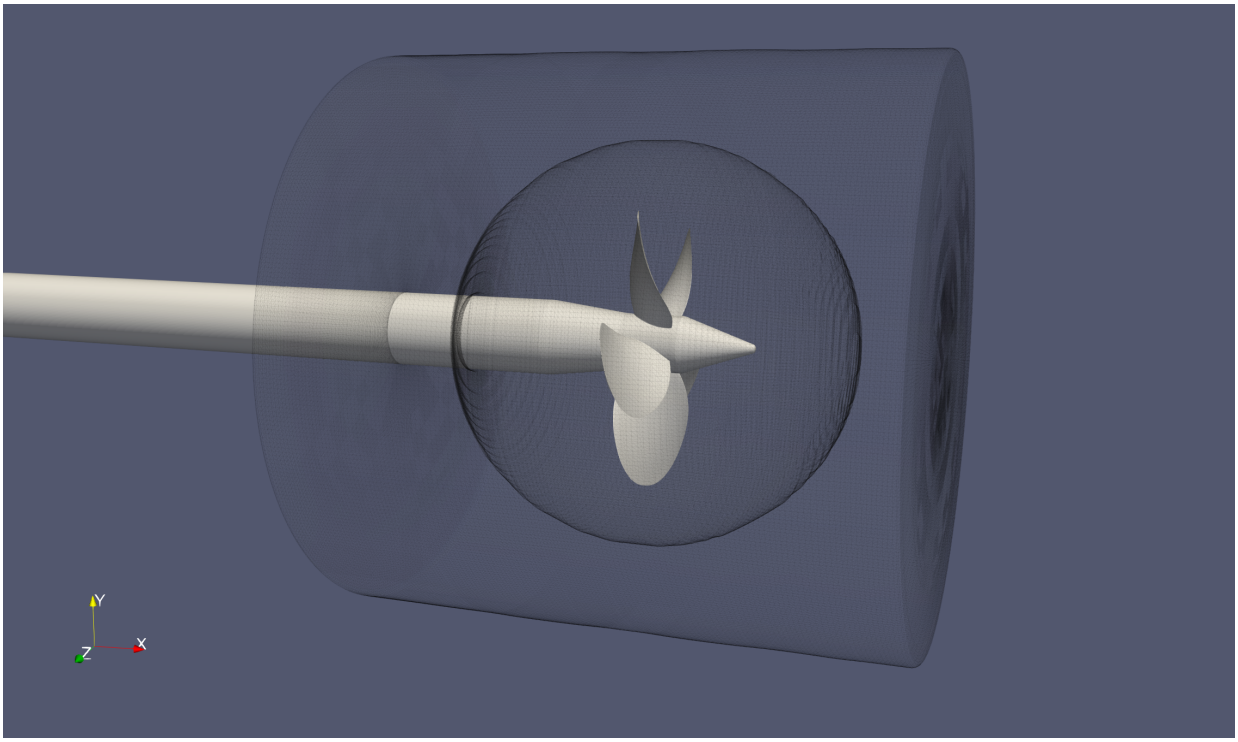
Fringe BCs can occur either by design or from cutting. All overset grids by design always have some fringe outer edge BC, and the user can set this BC on the background grid as well like in the Crashback test case. After cutting, the code will internally take care of fringe BCs.

To better visualize fringe/interpolation surface BCs, an example case with a background grid and a cylindrical, propeller overset grid is shown below. A spherical cut is used to remove redundant CVs on the background grid. The overlap region is where CVs exist that are in the solution of both grids. The spherical cut edges on the background grid and the cylinder edges of the propeller mesh are the location of interpolation CVs or the fringe boundaries. For the overlap region, it is set up so that the two edges aren't too close that they use interpolation partner CVs

too close to other interpolation CVs as interpolation partners. CVs in interpolation regions are designed to have minimal ratios in size, at most about a ratio of 2:1 in each direction.



The CVs bordering the dark surface visualized below are the interpolation CVs. They will all locally search for interpolation partner CVs to use from other grids. Only candidate CVs that are not masked out of the solution and aren't interpolation CVs can be used as interpolation partners.



Boundary Conditions with FLUENT

Boundary conditions in **FLUENT** format have information that is important for running the code. Some of this information is the boundary condition ID label. The user should always make sure to set the correct boundary condition type in the grid generation software. Overset Fringe BCs can be set to a wall type.

The user can use the following to double check that they have the correct boundary condition type on their partitioned grid.
Using the command: **ncdump mesh000.nc | less**
Scroll to the bottom of the file to view the information. Below is an example of that output.

```
zone_bc = 0, 4, 1, 1 ;

faozn_iv = 1, 222112, 225927, 229301, 231437 ;

zone_name =
  "default-interior                                    ",
  "wall                                                ",
  "periodic-shadow                                     ",
  "periodic                                            " ;
}
```

BCs from FLUENT and the corresponding ID.

- **Interior**: ID = 0
- **Periodic**: ID = 1
- **Inflow**: ID = 2
- **Outflow**: ID = 3
- **No Slip Wall:** ID = 4
- **Slip Wall**: ID = 5
- **Symmetry**: ID = 6

Other

All these files can be found in the source code **bin** directory. Make sure to copy them when setting up a case directory.

mpc_main.f90

This file is important because it initiales the **mpcugles** executable. It should never be edited. Always make sure it is in the **run** directory.

CMakeLists.txt

This file is important in order to successfully build and install the source code. It tells **cmake** about the different **.f90** files located in the **run** directory to compile together with the source code (eg. mpc_main.f90, user_hooks.f90, dump_tecplot_files.f90 etc.). Always make sure it is in the **run** directory.

scalar.dat

This file contains options on using passive scalars in the code. This first number in the file is changed to 1 to turn on passive scalars. Always make sure it is in the **run** directory regardless.

kill_switch.dat

This file gives the user the ability to turn the case off cleanly. Just change the value in this file to 1 and the case should end and save the result at that time the user edited the file.

interp.f90

This file is used to interpolate a solution from a solution to a new grid using **nc** files. This file will need to be read and edited by the user to fit their specific case of grids that need to be interpolated. Note that this solution interpolation can have memory limitations when grids get too large. Always make sure it is in the **run** directory.

propeller.in

This file is used while running the propeller case. It is used together with **stats_coef_hook** to calculate the force coefficient history. In order to use it, the **user_hooks.f90** file has to read it in.

This file is also very important while running the single grid code when the flag **has_overset='no'**. This is where the advanced coefficient is read and thus sets the rotation for the equations to solve in the rotating frame of reference. Set the advanced coefficient to zero if you don't need to solve in the rotating frame of reference.

Always make sure you have this file in the **run** directory when running a propeller case, especially when running the single grid code. Always make sure that you read it in **user_hooks.f90** if it is being used just like in the regular mpcugles.

Plotting

There are several options for plotting depending on user preference. The different options have their pros and cons. More details are given in the sections below.

XDMF+HDF5

XDMF (eXtensible Data Model and Format) is a library providing a standard way to access data produced by HPC codes. This data is stored using HDF5. **HDF** (Hierarchical Data Format) is a set of file formats (**HDF4**, HDF5) designed to store and organize large amounts of data. ParaView, VisIt and EnSight visualization programs are able to read XDMF.

XDMF+HDF5 enables the capability to arbitrarily select the number of output blocks. Visualization software like **Paraview** then reads the results in parallel. Use the **nout_procs** flag to change the number of output files.

XDMF+HDF5 eliminates the need for post-processing to visualize results. It also enables the capability to restart from these same files. **NetCDF** is only used as the grid input.

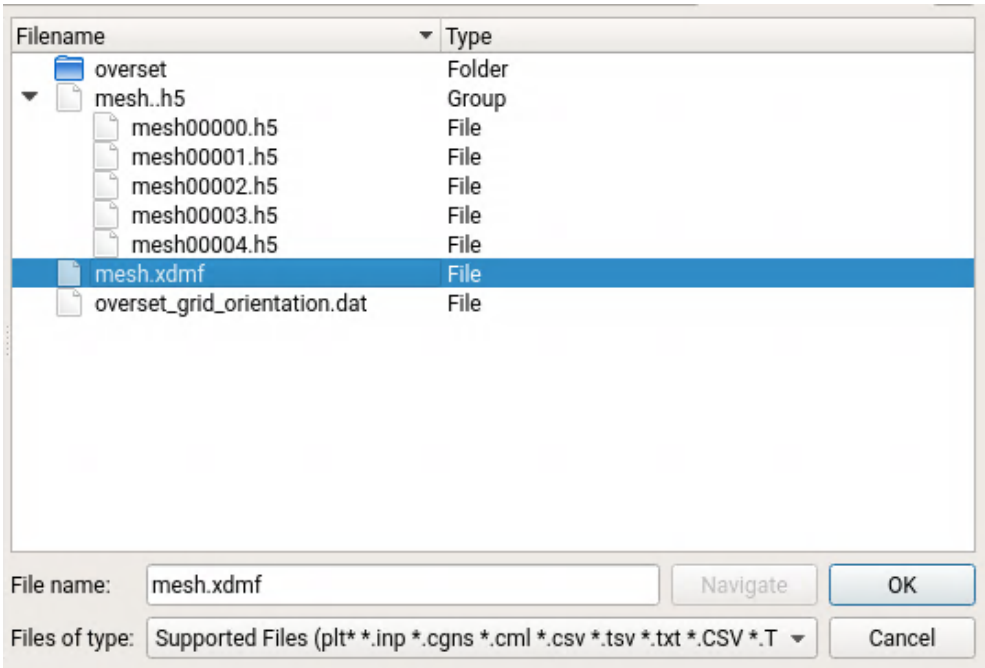
The user can change the following flags to utilize this format **input_style='hdf5'** and **output_style='hdf5'**. Make sure the source code is compiled with the option to use **HDF5** libraries turned on.

Paraview

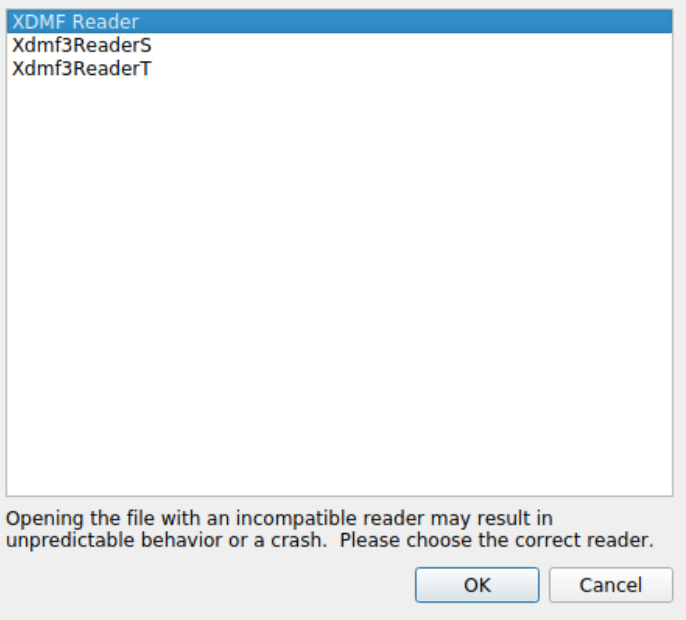
Paraview is used to visualize the **XDMF+HDF5** results.

A mesh partitioned to a specific number of partitions can be partitioned to a user preferred number of **HDF5** result files. The **nout_procs** flag is important to set the number of **HDF5** written files. In the example below it is set to **5**.

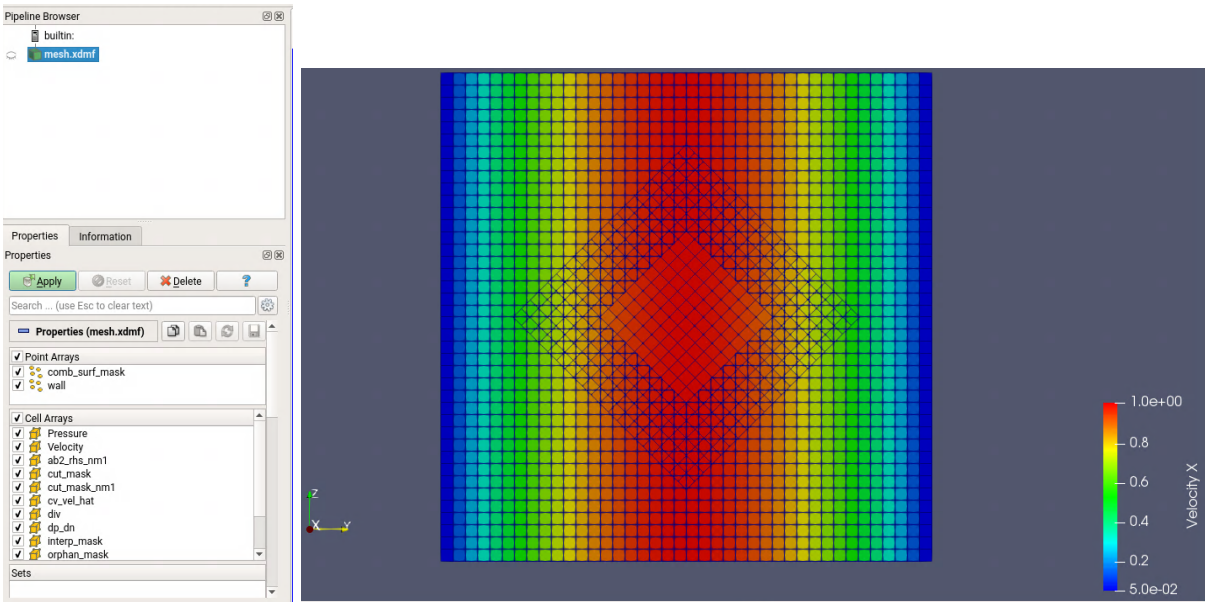
To open the results, open the **.xdmf** file. There should always be one **.xdmf** file together with **.h5** files as shown below. The user can also restart a case using these same files.



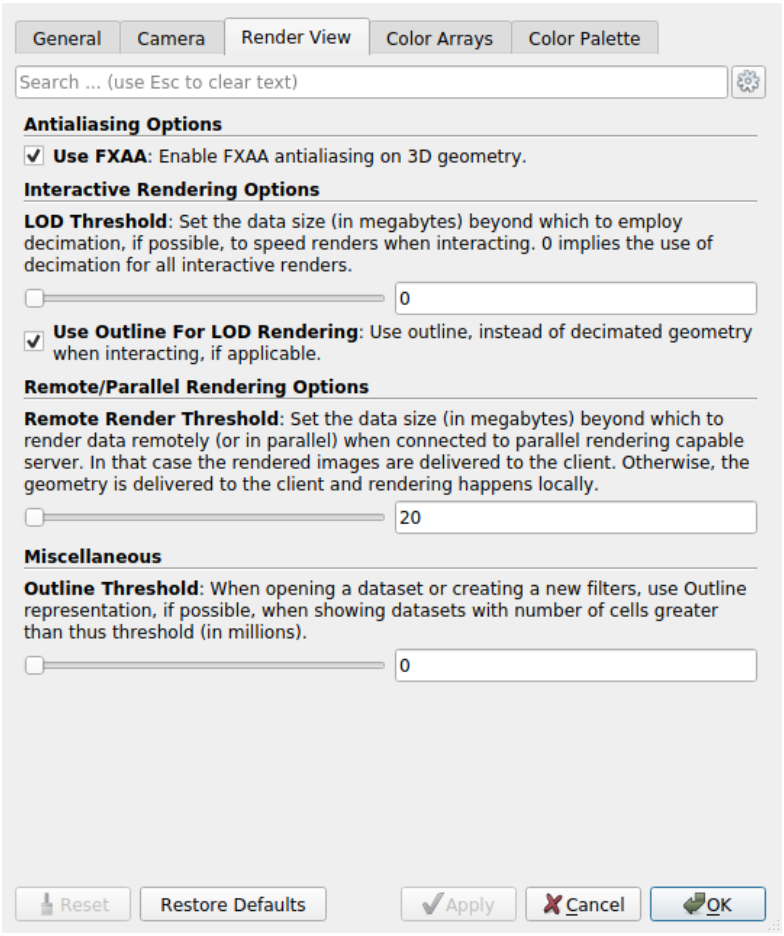
When prompted for a reader, the user should select either the **XDMF Reader** or **Xdmf3ReaderT**. Never use the **Xdmf3ReaderS**, it has issues with parallelization and will crash in parallel.



The user can then continue using selecting the relevant variables and visualizing. Make sure to use the **Threshold** filter to remove cut, redundant CVs from the visualization using the **cut_mask** variable. Note that wall surfaces show up as face variables.



When visualizing large cases in parallel, try to change the Render View options to speed up visualization and prevent lags. Rendering can be an expensive process and can slow the visualization of a data set. Lower thresholds to very low values for very large cases as shown below.



Recommendations:

- If the user is interested in seeing exact CV center values, it is recommended to use this output format and view using **Paraview**.
- If the user is interested in creating an animation it is recommended to use this output format and view using **Paraview**.

Tecplot

This is the format used with .**nc** result files and is a post-processing of these files using the **dump_tecplot_files** executable.

This creates **.dat** (ascii) files. It is often that the **.dat** files are converted to **.plt** (binary) files by an additional post-processing step using the preplot tool provided by Tecplot. Tecplot also suggests this once cases start getting large because **.plt** (binary) files speed up loading and also use less storage.

Due to a nodal based writing format, it is important to note that the results are interpolated from CV centers to nodes before being saved in the tecplot **.dat** format.

Use the blanking feature to blank out masked CVs from being displayed in the solution. Select Plot->Blanking->Value Blanking. Then select **mask** as the value to blank any values less than 1. Due to interpolation of CV values to nodes, the user might have to use a value less than 1 to show the correct solution CVs.

dump_tecplot_files.f90

This file contains information on converting **.nc** results files to Tecplot **.dat** files. It is located in the **run** directory and after code compilation creates an executable called **dump_tecplot_files**.

post.in

This file is used for additional flags paired with **dump_tecplot_files.f90**. The **flag_lamba2** flag can be turned on by inserting the value of 1 and this will also calculate $Q - criterion$ & $\lambda_2 - criterion$ variables that will be written in the **.dat** files.

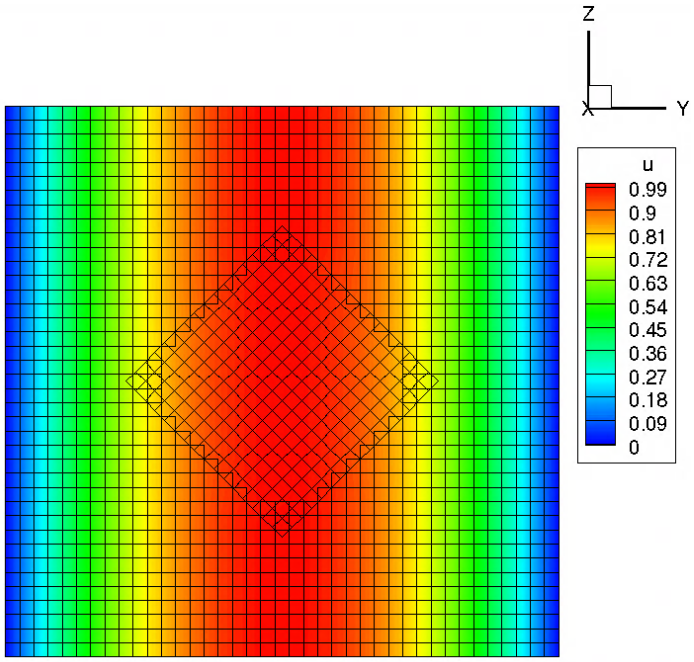
Creating Tecplot .dat files

Make sure the **output_style** flag is set as **nc** and the **nc** files are in the result folder. Run the **dump_tecplot_files** script matched correctly with the number of processors as used for **mpcugles (total_procs = nprocs+novrprocs)**.

For example, a case with 4 background processors and 1 overset processor the user executes the below command.

mpirun -np 5 ./dump_tecplot_files

An example of a result in tecplot is shown below.



circum_avg_new.f90

This file is used to perform a circumferential average of the case flow field. This file will need to be read and edited by the user to fit their specific case. It creates a Tecplot **.dat** file that can be viewed by the user. It was created for the propeller case and isn't generally applicable without some editing. Always make sure it is in the **run** directory.

Recommendations:

- If the user is interested in seeing exact CV center values, it is recommended to use the **HDF5** output format and view using **Paraview**. **Paraview** can also interpolate to nodes as a post-processing through the Cell Data to Point Data Filter.
- If the user is interested in creating an animation it is recommended to use the **HDF5** output format and view using **Paraview**.

Exodus

This is the format used with **nc** result files and is a post-processing of these files. Make sure **output_style** is set as **nc**. Then run the **dump_exodus_files** script matched correctly with the number of processors. This will produce **.exo** files that can be loaded using **Paraview**.

dump_exodus_files.f90

This file creates exodus files to visualize using **Paraview** etc. It is located in the **run** directory and after code compilation creates an executable called **dump_exodus_files**. Make sure the source code is compiled with the option to use **EXODUS** libraries turned on and also a local **EXODUS** compiled library.

Recommendations:

- Though this format is useful and the user is free to try it, it is recommended to use **HDF5** since it improves on **EXODUS**. If not using it, make sure to turn off in **custoptions.cmake**.
- If the user is interested in seeing exact CV center values, it is recommended to use the **EXODUS** output format or **HDF5** output format and view using **Paraview**. **Paraview** can also interpolate to nodes as a post-processing through the Cell Data to Point Data Filter.
- If the user is interested in creating an animation it is recommended to use the **HDF5** output format and view using **Paraview**.

Test Cases

Below is information on running some test cases to familiarize with the overset code.

Channel Flow

This case is a channel flow driven by a body force with no-slip walls and periodic boundary conditions. This case is used as a quick way to test the many capabilities of the overset code. The background grid is a fully structured channel grid with no-slip wall surfaces. An overset patch grid is introduced into the channel. This case can also be a good debugging and testing case before trying larger cases. Copy the grid and specific modified files from the source code **bin/Channel_Flow** folder to get started with this case.

Flow with one rotating overset patch

This case is a flow in a channel with an overset patch. The $Re = 50$, the freestream velocity is $U = 1.0$, and the channel length is $L = 1.0$. The patch will be rotating at an angular velocity about the x-axis $\omega_x = \pi$. This case will have a total of 5 processors to run using mpi.

grid: Contains the 4 grid partitions for the background and the overset grid directory.

- mesh000.nc
- mesh001.nc
- mesh002.nc
- mesh003.nc
- **overset:** Contains the overset grids and orientation file.
  - mesh001000.nc
  - overset_grid_orientation.dat

user_hooks.f90

Initial Condition

Specify the initial conditions for the case. Here we initialize the correct theoretical solution as shown below in **initial_condition_hook**. The flow will be only in the x-direction.

```
! ----------------------
!
! initial_condition_hook
!
! ----------------------
subroutine initial_condition_hook(gp_d, sp_d)

  use precision_m
  use data_struct_defs_m
  use grid_part_defs_m
  use sol_part_defs_m
  use global_m
  use hook_data_m
  use string_m

  implicit none
  include 'mpif.h'

  type (grid_partition_t) :: gp_d
  type (solution_partition_t) :: sp_d

  integer :: ifa, icv, izn

  ! Velocity
  sp_d%cv_velocity(:,:) = 0.d0
  sp_d%cv_velocity(1,1:gp_d%ncv_ib) = -u_centerline*(gp_d%cv_cc(2,1:gp_d%ncv_ib)**2-1.d0)

end subroutine initial_condition_hook
```

Boundary Conditions

We don't use any Inflow or Outflow boundary conditions for this case. Periodic boundary conditions are taken care of in the code so nothing is required of the user other than inserting the names in **boundary_condition_hook**.

Wall boundary conditions are set to no-slip at the faces (This is also done for nodes for plotting with tecplot). Note that if an overset grid is moving and has a wall surface, we use the wall velocity as its boundary condition. This can be changed by the user to match specific cases.

```
!!! Wall BC !!!
     else if ((string_compare(gp_d%zone_name(izn), 'wall01')).or.&
          (string_compare(gp_d%zone_name(izn), 'wall')))then

        ifa_f = gp_d%faozn_iv(izn);  ifa_l = gp_d%faozn_iv(izn+1)-1

        if (face_vel_bc) then
           do ifa = ifa_f, ifa_l
              !No Slip BC
              sp_d%face_bdy_vel(1,ifa) = 0.d0
              sp_d%face_bdy_vel(2,ifa) = 0.d0
              sp_d%face_bdy_vel(3,ifa) = 0.d0

              !Set Moving BC on Overset Walls Only
              if (myrank .ge. nprocs) then
              call cross_product(rot_vel,gp_d%gridomega,gp_d%face_cc(:,ifa)-gp_d%gridcc(:))
                 sp_d%face_bdy_vel(1,ifa) = gp_d%gridv(1) + rot_vel(1)
                 sp_d%face_bdy_vel(2,ifa) = gp_d%gridv(2) + rot_vel(2)
                 sp_d%face_bdy_vel(3,ifa) = gp_d%gridv(3) + rot_vel(3)
              endif

           end do

        endif

        if (node_vel_bc) then
            do ifa = ifa_f, ifa_l
              nof_f = gp_d%noofa_i(ifa);  nof_l = gp_d%noofa_i(ifa+1)-1

              do nof = nof_f, nof_l
                 ino =  gp_d%noofa_v(nof)
                 x = gp_d%node_cc(1,ino); y = gp_d%node_cc(2,ino)

                 nvel_p%d(1,ino) = 0.0
                 nvel_p%d(2,ino) = 0.0
                 nvel_p%d(3,ino) = 0.0

                 if (myrank .ge. nprocs) then
                 call cross_product(rot_vel,gp_d%gridomega,gp_d%node_cc(:,ino)-gp_d%gridcc(:))
                    nvel_p%d(1,ino) = rot_vel(1)+gp_d%gridv(1)
                    nvel_p%d(2,ino) = rot_vel(2)+gp_d%gridv(2)
                    nvel_p%d(3,ino) = rot_vel(3)+gp_d%gridv(3)
                 endif

              enddo
           enddo
        endif
```

definitions_hook

Here we define some important variables for the overset case.
The fringe boundary condition for overset grid can be manually set for designed grids. Here we design a patch with its walls named **‘wall05’**. This must be defined otherwise the code won’t know that these edges need interpolation.
There are no walls that cut/remove CVs for this case, thus these are commented out.
In order to plot surfaces, we insert only the surface names we’re interested in. Here we insert **‘wall’**. For more information visit the **definitions_hook** section.

```
! ----------------
!
! definitions hook
!
! ----------------

subroutine definitions_hook()

  use precision_m
  use data_struct_defs_m
  use grid_part_defs_m
  use sol_part_defs_m
  use global_m
  use hook_data_m
  use string_m
  use computational_geometry_m
  use message_passing_m

  implicit none

  !Set Fringe BCs, Cut Walls & Plot BC names
  if(has_overset == 'yes') then
    num_fringe_bcs = 2
    !num_cut_walls = 1
    num_plot_bcs = 1
    if ( associated(fringe_bc_name) )deallocate(fringe_bc_name)
    if ( associated(wall_cut_name) )deallocate(wall_cut_name)
    if ( associated(plot_bc_name) )deallocate(plot_bc_name)
    allocate(fringe_bc_name(num_fringe_bcs))
    allocate(wall_cut_name(num_cut_walls))
    allocate(plot_bc_name(num_plot_bcs))

    fringe_bc_name(1) =  'wall05'
    fringe_bc_name(2) =  '''wall05'
    !wall_cut_name(1) = ''
    plot_bc_name(1) = 'wall'
  endif

end subroutine
```

overset_grid_orientation.dat

We are only using 1 clone of the overset grid. The x-coordinate is changed to 5.0 or the center of the background grid. This is because the overset patch is centered at 0.0.
Change the ω_x or rotation rate about the x-axis to π. This means that in 1 time unit the patch should make a full rotation.
Please go to the overset_grid_orientation.dat section for more details on the contents of this.
File Contents: overset_grid_orientation.dat

1
0.000000 0.000000 0.000000 5.0 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1

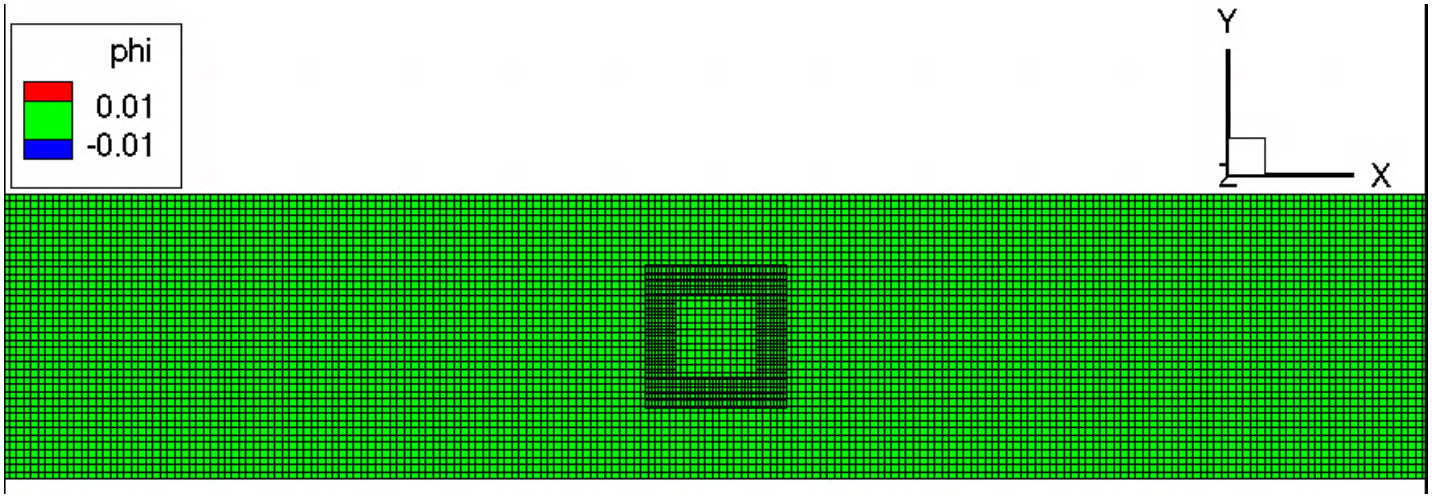
---

global.nml

The aim is to have a maximum velocity of 1.0 at the center of the channel. The selected Reynolds number inserted in **global.nml** is $\frac{1}{\nu} = \frac{Re}{L} = 50$. The selected Reynolds number is set to 50 and the corresponding body force in the **x**-direction used to drive the flow is 0.04. Set the time step to $5e-3$ and the number of steps to 50. This would mean that time has advanced 0.25 time units by the end of the run and thus the patch has rotated$45°$. Choose a preferred output format by changing the **output_style** flag, **'nc'** or **'hdf5**'. The **global.nml** file from the **bin/ChannelFlow** folder in the base code should have all these settings.

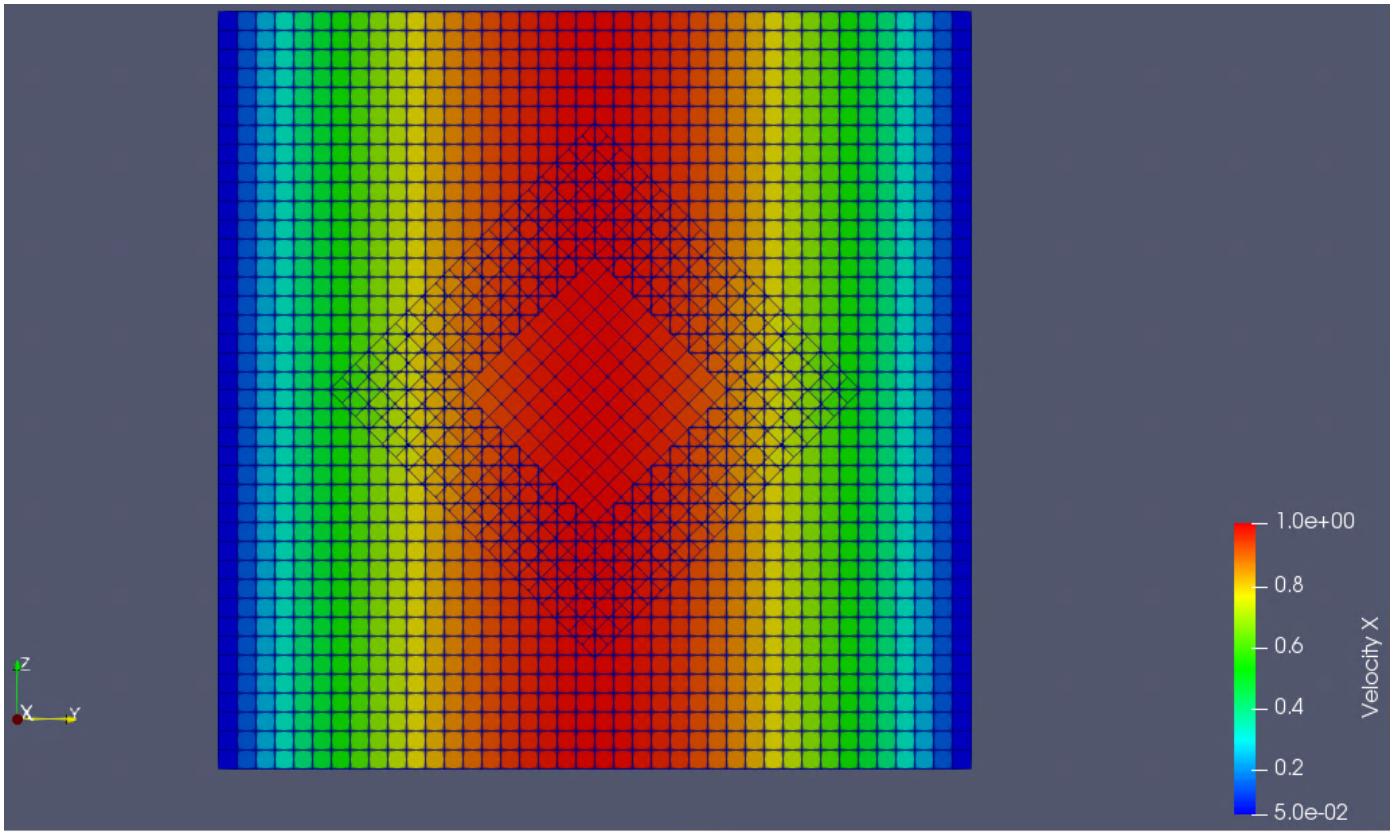
Pre-Run Checks

Before running or when trying to debug potential issues, the user can turn on the **dump_grid** flag to check that the initial condition and grids are set up correctly. Set this flag to 1 and then run the case. The solution will be in the result folder.



Results

The maximum velocity at the center of the channel is 1.0. The patch should have also rotated 45° as can be seen below. The result is plotted using **Paraview**.



Other Tests

Steady State

The user can test initializing the flow with zero velocity and run the case to a steady state.

Multiple Overset Grids

The user can test the use of multiple overset meshes by making a copy of the overset patch and editing the overset_grid_orientation.dat. More information to help in setting the case up can be found in the overset grid section.

Structured Uniform Background

The user can test using a uniform, structured background grid. Just change the **grid_input_type** flag to **'struct_uniform'** and the **output_style** flag to **'hdf5'**. The **global.nml** should have the dimension settings to match the partitioned background grid. Use the same number of processors as this case. Check the struct_grid section for more information on the structured grid settings. Check and match the **global.nml.struct** file in the source code **bin/Channel_Flow** folder.

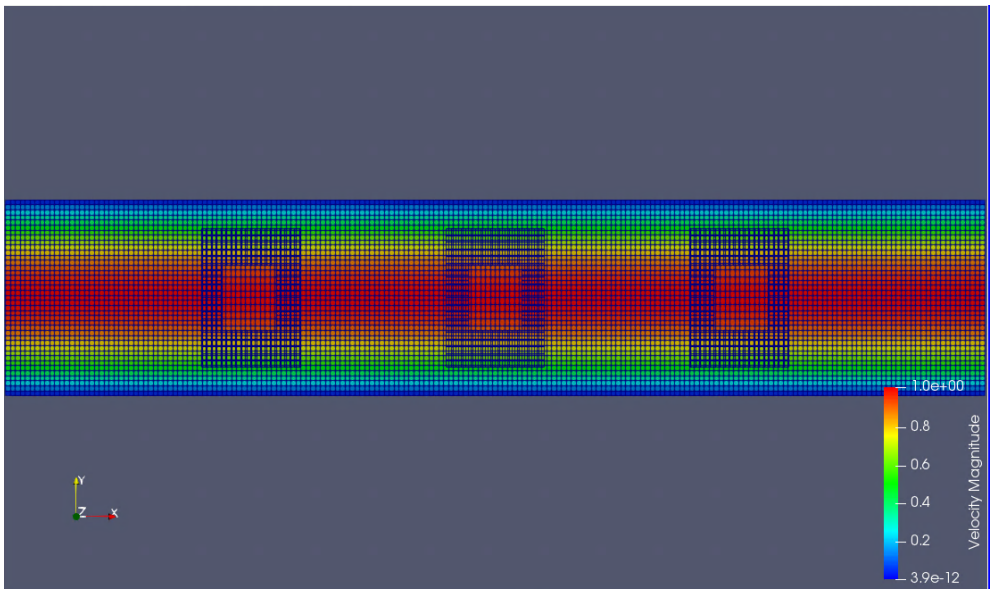
Cloning

The user can test the cloning capabilities of the code. When cloning, the code will assign the cloned mesh copy to the overset processors only. More detailed information can be found in the Cloning section.
The user can have numerous overset grids with each having their own clones. It should be noted that too many clones can overwhelm the overset processors. An example of using this test case is shown below. Edit the overset_grid_orientation.dat file as show and run again.
File Contents: overset_grid_orientation.dat

3
0.000000 0.000000 0.000000 5.0 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
0.000000 0.000000 0.000000 2.5 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1
0.000000 0.000000 0.000000 7.5 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
3.141592654 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1

★ **Note**: The 3 lines below the number of clones here represent 1 line in the file. Each line is color coded to represent the specific clone.

The result should now have 3 copies of the original patch grid, with each rotated at a 45° angle at the end of the simulation. Test using more than one overset processor to observe the capability of the overset processors being assigned multiple partitions.

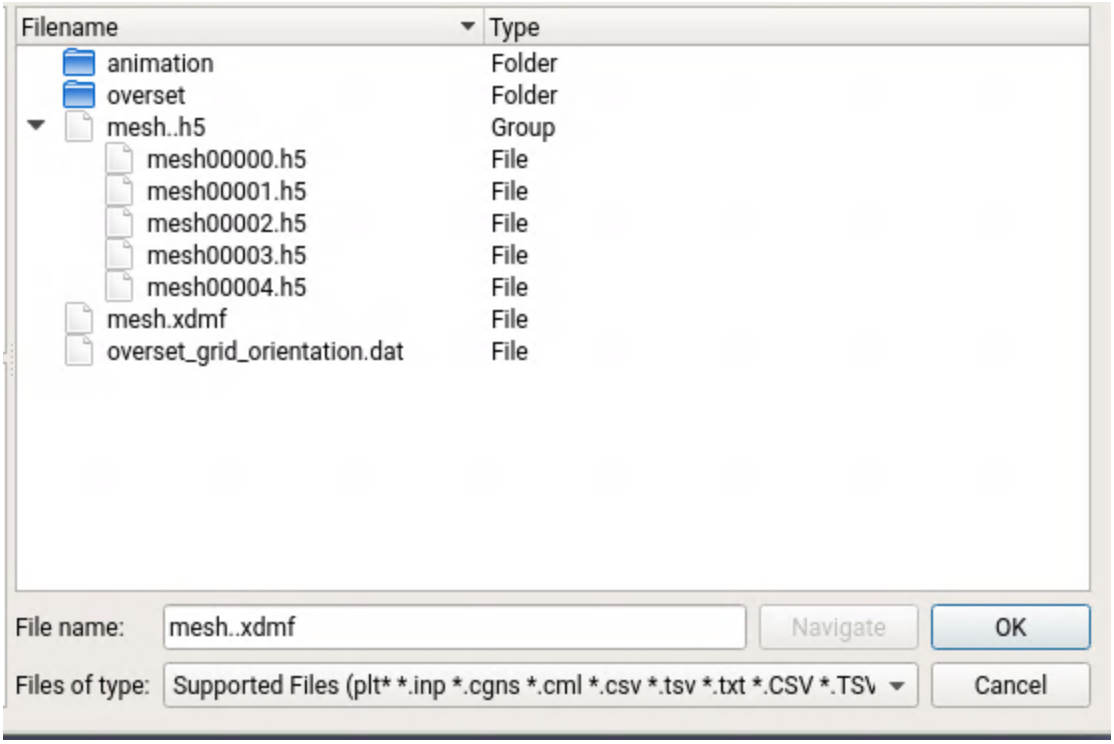


Animation

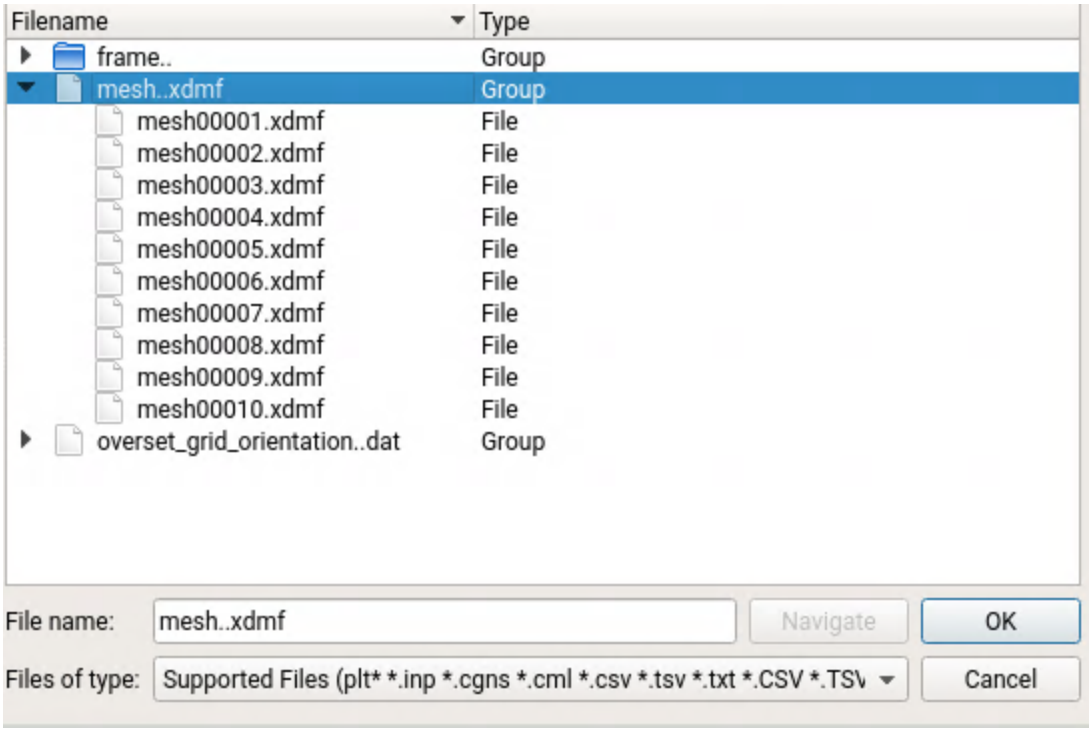
The user can test running animations using the flags detailed in the Animations section. We'll run the main case with an animation. Make sure to change **input_style** and **output_style** to **'hdf5'**. Change the animation flags, the **anim_start** to 5 and **anim_steps** to 5 and **anim_stop** to 50. The animation should start during the 5th timestep and save a frame every 5 steps and end during the last timestep. Check and match the **global.nml.anim** file in the source code **bin/Channel_Flow** folder.

The **frame_num** flag sets the number used for the first written animation frame. For example, 1 would mean the first written animation frame is named **frame001**. This flag is useful when restarting simulations. For example, if there are already 6 frames, this flag is set to 7 and the upcoming frame numbers would start at this value.

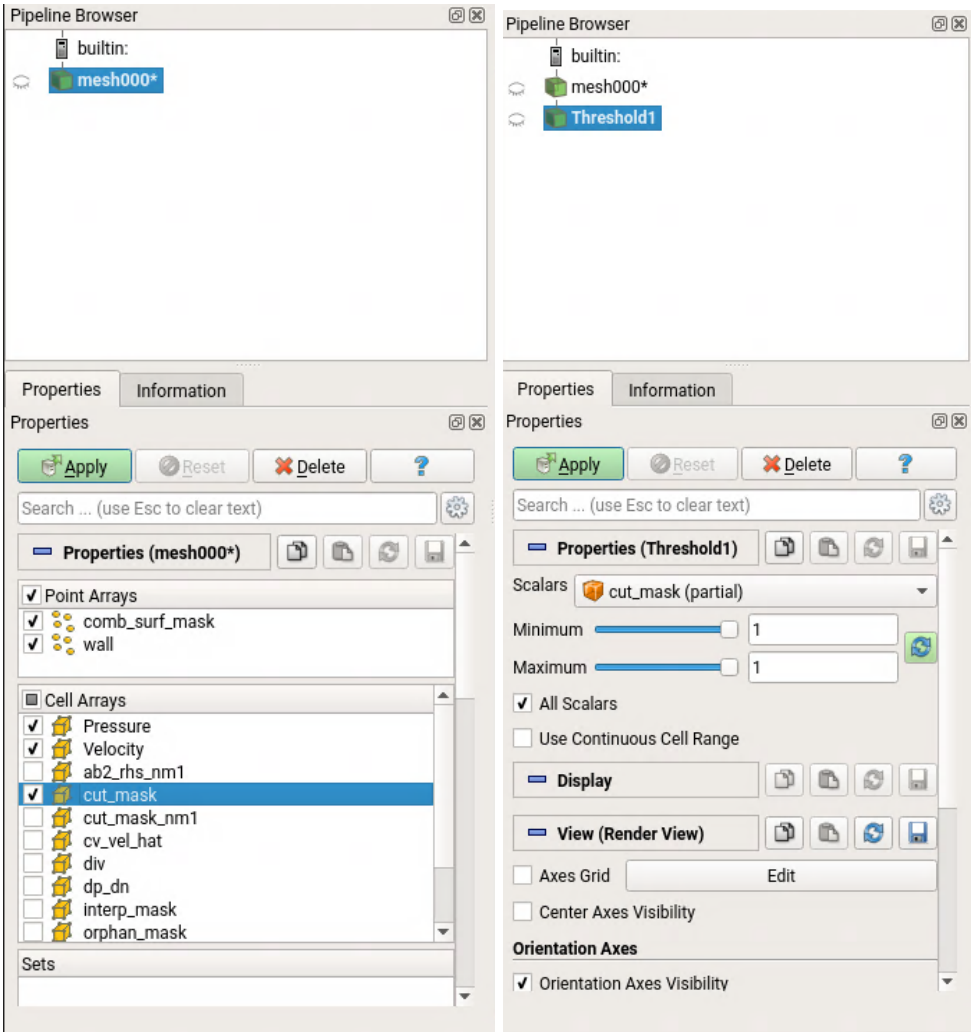
After running the case, we open the animation in **Paraview**. In the **result** folder there should be the usual **.xdmf** files and **.h5** files. There will also be an **animation** folder.



In the **animation** folder there should be 10 **frame** folders, **.xdmf** files and **overset_grid_orientation.dat** files. Select **mesh..xdmf** which should load all the frames at once.

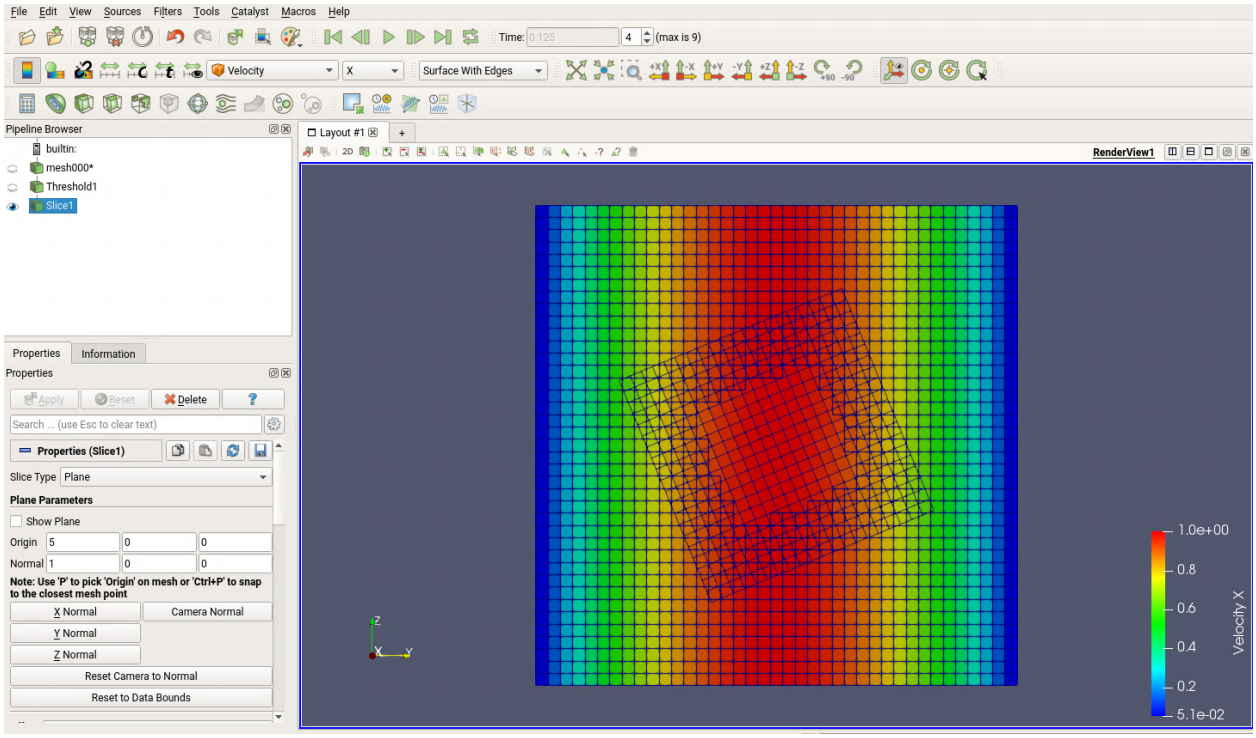


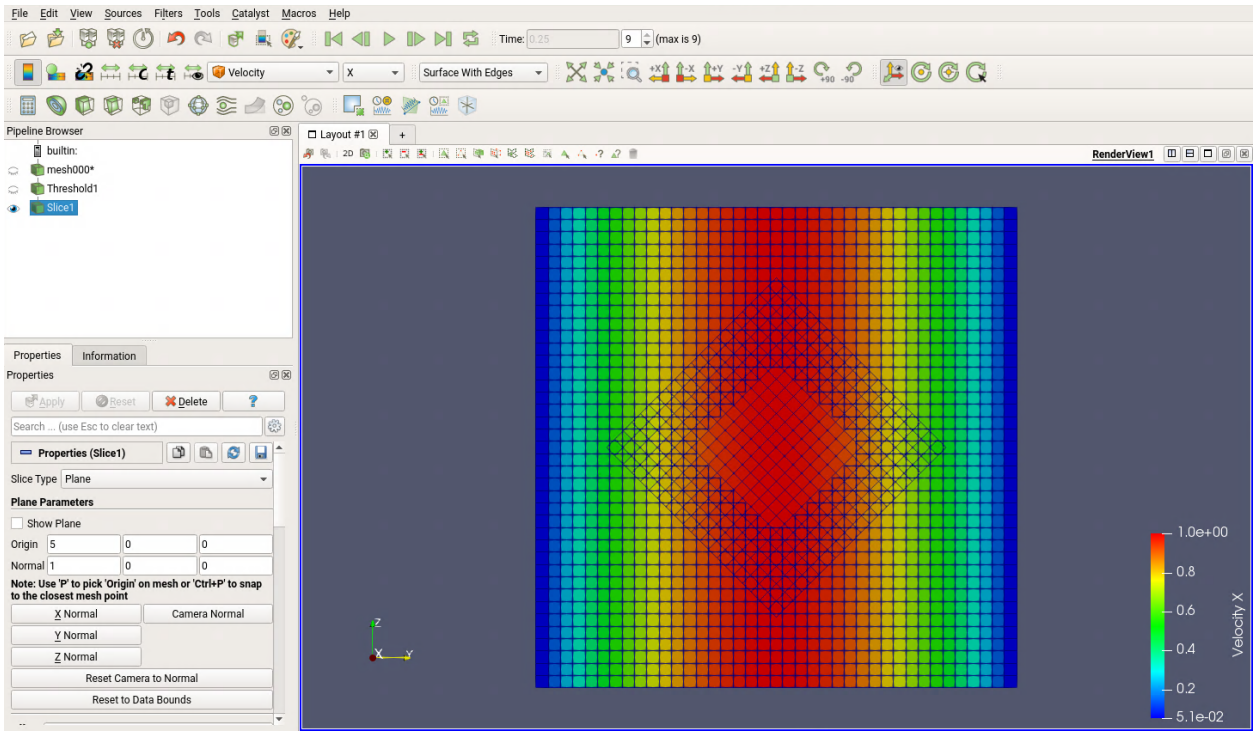
Select the variables of interest. Then use the **Threshold** filter to mask out redundant CVs.



Use the slice filter to display the Velocity in x on the y-z plane with surface edges on so you can see the meshes.The frames should be available with their timestamps and you can use the play button to animate the frames. Below are the figures showing the 4th frame and the last frame of the animation.

MPCUGLES Overset V1

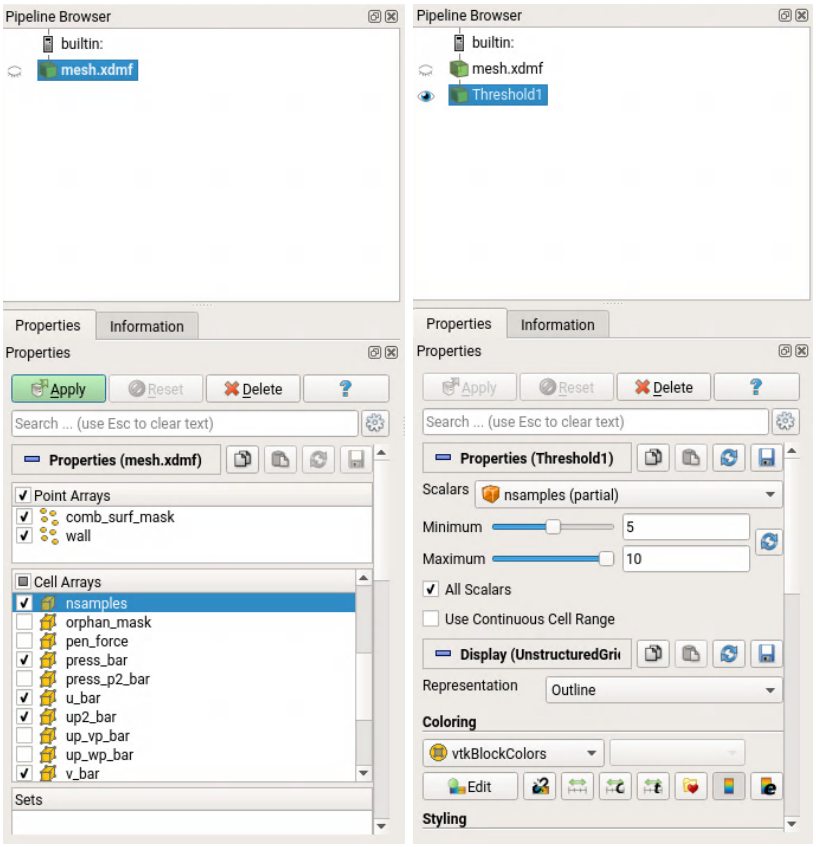




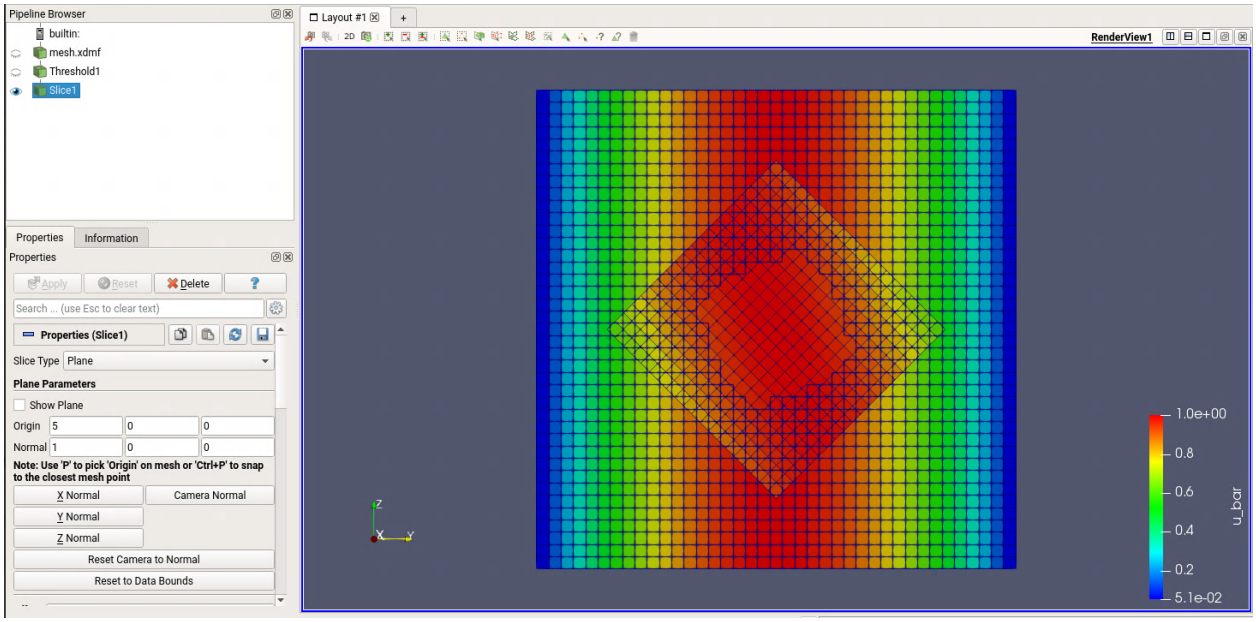
Statistics

The user can test collecting statistics using the flags detailed in the Statistics section. We'll run the same case but now with statistics enabled. First, turn on the **statistics_flag** to 1, setting **stats_start** to 5 means we would start collecting samples on the 5th timestep, setting **stats_steps** to 5 means we would take additional samples 5 timesteps after the first one, and setting **stats_stop** to 50 means the last sample must be collected by the 50th timestep, in this case a sample will be collected on this final simulation timestep. Check and match the **global.nml.stats** file in the source code **bin/Channel_Flow** folder.

The **nc** or **hdf5** results files will now have the regular variables in addition to the statistics variables. When loading a **hdf5** solution you will now see additional variables together with the normal simulation variables (For Tecplot there will be separate ***stat.dat** files). When loading and visualizing files, the user can use the number of samples variable **nsamples** with the **Threshold** filter to view relevant CVs.

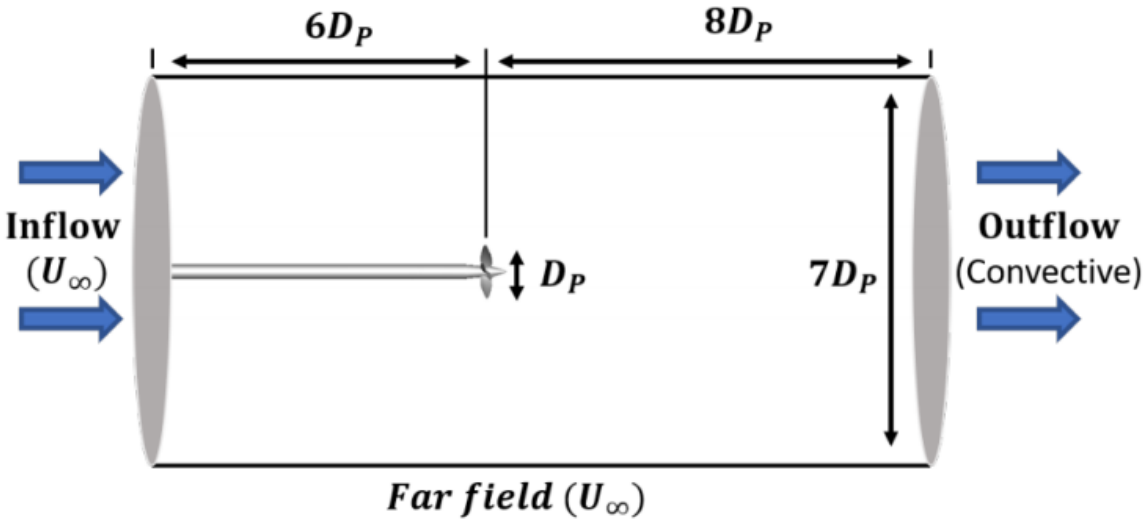


Below we visualize the mean axial velocity variable **u_bar**. Note that the overset patch grid is rotating and thus the averaging on this grid is going to be sampling both spatially and in time. Only grids that aren't moving, like the background grid will be sampling solely in time. The user needs to keep this in mind as they design a case.



Crashback for P4381 at J = -0.7

This case uses the overset code to rotate the propeller in Crashback. The advance ratio is $J = -0.7$, the freestream velocity is $U = 1.0$, and the propeller diameter is $D_p = D = 12.0$. The dimensions for the simulation are shown below. To utilize the flexibility of the method in
reducing computational cost and to use the grid generation flexibility, the background mesh is generated with a cylindrical cut in the hubcap region. A manual cut is used to remove redundant control volumes during grid generation compared to the dynamic cutting that was utilized for the channel case.



user_hooks.f90

Initial Condition

Specify the initial conditions for the case. Here we initialize the solution of the freestream velocity $U = 1.0$ as shown below in **initial_condition_hook**. The flow is only in the x-direction.

```
! ----------------------
!
! initial_condition_hook
!
! ----------------------
subroutine initial_condition_hook(gp_d, sp_d)

  use precision_m
  use data_struct_defs_m
  use grid_part_defs_m
  use sol_part_defs_m
  use global_m
  use hook_data_m
  use string_m

  implicit none
  include 'mpif.h'

  type (grid_partition_t) :: gp_d
  type (solution_partition_t) :: sp_d

  integer :: ifa, icv, izn

  ! Velocity
  sp_d%cv_velocity(:,:) = 0.d0
  sp_d%cv_velocity(1,:) = 1.d0

end subroutine initial_condition_hook
```

Boundary Conditions

In **boundary_condition_hook**, we set all the boundary condition names.

For the inflow boundary condition, we set is as the freestream velocity $U = 1.0$ in the x-direction only.

```
!!! Inflow BC !!!
    else if (string_compare(gp_d%zone_name(izn), 'inf01') .or. &
         (string_compare(gp_d%zone_name(izn), '''inf01'))) then

       ifa_f = gp_d%faozn_iv(izn);  ifa_l = gp_d%faozn_iv(izn+1)-1

       !Set Face Velocity BC
       if (face_vel_bc) then

          do ifa = ifa_f, ifa_l
             sp_d%face_bdy_vel(1,ifa) = 1.d0
             sp_d%face_bdy_vel(2,ifa) = 0.d0
             sp_d%face_bdy_vel(3,ifa) = 0.d0
          enddo
       endif

       !Match Face Velocity BC for nodes (for tecplot plotting)
       if (node_vel_bc) then
        do ifa = ifa_f, ifa_l
             nof_f = gp_d%noofa_i(ifa);  nof_l = gp_d%noofa_i(ifa+1)-1

             do nof = nof_f, nof_l
                ino =  gp_d%noofa_v(nof)
                x = gp_d%node_cc(1,ino)
                y = gp_d%node_cc(2,ino)

                nvel_p%d(1,ino) = 1.d0
                nvel_p%d(2,ino) = 0.d0
                nvel_p%d(3,ino) = 0.d0

             enddo
          enddo
       endif
```

Wall boundary conditions are set to no-slip at the faces (This is also done for nodes for plotting with tecplot). Note that if an overset grid is moving and has a wall surface, we use the wall velocity as its boundary condition. This can be changed by the user to match specific cases. Here we change the '**wall02'** BC. This surface is on the overset grid but at a redundant location on the shaft and thus it is changed to no-slip. By design, this surface is redundant to ensure enough overlap so it is treated as part of the shaft with the BC and exclusion in force calculations.

```
!!! Wall BC !!!
    else if ((string_compare(gp_d%zone_name(izn), '''wall02')).or.&
         (index(gp_d%zone_name(izn), 'hub') > 0) .or. &
         (index(gp_d%zone_name(izn), 'shaft') > 0) .or. &
         (index(gp_d%zone_name(izn), 'blade') > 0) .or. &
         (string_compare(gp_d%zone_name(izn), 'wall02')))then

       ifa_f = gp_d%faozn_iv(izn);  ifa_l = gp_d%faozn_iv(izn+1)-1

       if (face_vel_bc) then
          do ifa = ifa_f, ifa_l
             !No Slip BC
             sp_d%face_bdy_vel(1,ifa) = 0.d0
             sp_d%face_bdy_vel(2,ifa) = 0.d0
             sp_d%face_bdy_vel(3,ifa) = 0.d0

             !Set Moving BC on Overset Walls Only
             if (myrank .ge. nprocs) then
             call cross_product(rot_vel,gp_d%gridomega(:),&
                   gp_d%face_cc(:,ifa) - gp_d%gridcc(:))

                sp_d%face_bdy_vel(1,ifa) = gp_d%gridv(1) + rot_vel(1)
                sp_d%face_bdy_vel(2,ifa) = gp_d%gridv(2) + rot_vel(2)
                sp_d%face_bdy_vel(3,ifa) = gp_d%gridv(3) + rot_vel(3)

             endif

              !Overset Moving wall BCs that should be no-slip
              if (string_compare(gp_d%zone_name(izn), 'wall02') .or. &
                 (string_compare(gp_d%zone_name(izn), '''wall02')))then
                     sp_d%face_bdy_vel(1,ifa) = 0.d0
                     sp_d%face_bdy_vel(2,ifa) = 0.d0
                     sp_d%face_bdy_vel(3,ifa) = 0.d0
              endif

          enddo

       endif
```

The Far-Field boundary condition is used for the outer walls of the simulation.

```
!!! Tunnel Far-Field BC !!!
    else if ((string_compare(gp_d%zone_name(izn), '''tunnel')).or.&
            (string_compare(gp_d%zone_name(izn), 'tunnel'))) then

       ifa_f = gp_d%faozn_iv(izn);  ifa_l = gp_d%faozn_iv(izn+1)-1
       if (face_vel_bc) then
          do ifa = ifa_f, ifa_l
             icv1 = gp_d%cv_of_face(1, ifa)
             icv2 = gp_d%cv_of_face(2, ifa)
             sp_d%face_bdy_vel(1,ifa) = 1.d0
             sp_d%face_bdy_vel(2,ifa) = 0.d0
             sp_d%face_bdy_vel(3,ifa) = 0.d0
          enddo
       endif

       if (node_vel_bc) then
          do ifa = ifa_f, ifa_l
             nof_f = gp_d%noofa_i(ifa);  nof_l = gp_d%noofa_i(ifa+1)-1
             do nof = nof_f, nof_l
                ino = gp_d%noofa_v(nof)
                nvel_p%d(1,ino) = 1.d0
                nvel_p%d(2,ino) = 0.d0
                nvel_p%d(3,ino) = 0.d0
             enddo
          enddo
       endif
```

stats_coef_hook

This subroutine calculates the forces coefficients. Make sure to enter the names of the surface names to be included in the calculations. The blade and duct surfaces are distinct for separate additional files used in analysis. Use propeller.in to insert the correct propeller information, advanced ratio and also control the printout rate as well as to include blade distribution files print in the print out. Below is the section where the blade surface names are added.

```
do izn = 2, gp_d%nzone

   ifa_f = gp_d%faozn_iv(izn);  ifa_l = gp_d%faozn_iv(izn+1)-1

   if ( (index(gp_d%zone_name(izn), 'BLADE') > 0) .or. & !Blade BC
        (index(gp_d%zone_name(izn), 'blade') > 0) ) then


         if(blade_parts_flag == 1 .and. &
           (mod(main_ts_loop_index, ncount_print) == 0)) then

            !Pressure side in fwrd mode
            if ( index(gp_d%zone_name(izn), 'p') > 0 ) then
            !if (gp_d%face_normal(1,ifa) > 0) then
             blade_idx = 0

            !Suction side in fwrd mode
            else if ( index(gp_d%zone_name(izn), 's') > 0 ) then
            !else if (gp_d%face_normal(1,ifa) < 0) then
             blade_idx = 10

           endif
         endif
```

definitions_hook

Here we define some important variables for the overset case.
The fringe boundary condition for overset can be manually set from designed grids. Here we design a manually cut propeller grid with its walls named **‘wall05’**. This must be defined otherwise the code won’t know that these edges need interpolation.
There are no walls that cut/remove CVs for this case, thus these are commented out.
In order to plot surfaces, we insert only the surface names we’re interested in. **‘wall02’** isn’t included because it is a redundant surface by design. For more information visit the definitions_hook section.

```
! ----------------------
!
! definitions_hook
!
! ----------------------
subroutine definitions_hook()
  use precision_m
  use data_struct_defs_m
  use grid_part_defs_m
  use sol_part_defs_m
  use global_m
  use hook_data_m
  use string_m
  use computational_geometry_m
  use message_passing_m

  implicit none

  num_fringe_bcs = 1
  !num_cut_walls = 1
  num_plot_bcs = 7

  if ( associated(fringe_bc_name) )deallocate(fringe_bc_name)
  !if ( associated(wall_cut_name) )deallocate(wall_cut_name)
  if ( associated(plot_bc_name) )deallocate(plot_bc_name)
  allocate(fringe_bc_name(num_fringe_bcs))
  allocate(wall_cut_name(num_cut_walls))
  allocate(plot_bc_name(num_plot_bcs))

  fringe_bc_name(1) = 'wall05'
  !wall_cut_name(1) = ''
  plot_bc_name(1) = 'blade_p'
  plot_bc_name(2) = 'blade_s'
  plot_bc_name(3) = 'hub_1'
  plot_bc_name(4) = 'hub_2'
  plot_bc_name(5) = 'hub_3'
  plot_bc_name(6) = 'shaft_1'
  plot_bc_name(7) = 'shaft_2'

end subroutine definitions_hook
```

overset_grid_orientation.dat

We are only using 1 clone of the overset grid which consists of the propeller and part of the hub. Use the equations $\omega = 2\pi n$ and $n = \frac{U}{JD}$ to solve for the $\omega_x = 0.747498$to rotate the overset mesh. Change the ω_x or rotation rate about the x-axis in this file. Note that surface and volume cutting is turned off for the grid and it is designed in a state ready to run.

Please go to the overset_grid_orientation.dat section for more details on the contents of this.
File Contents: overset_grid_orientation.dat

1
0.000000 0.000000 0.000000 0.0 0.0 0.0 0.000000 0.000000 0.0 0.000000 0.000000 0.000000
0.747498 0.000000 0.00000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0 1 0 0 0 1

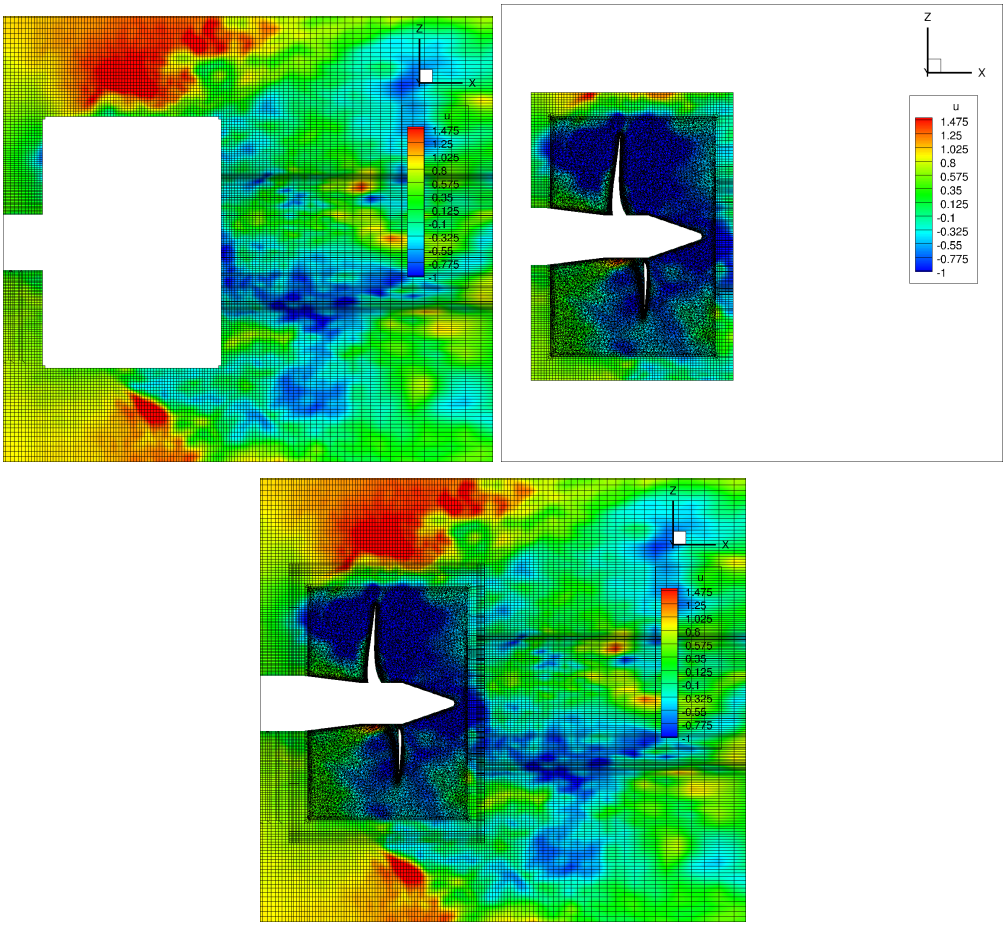
---

global.nml

The case Reynolds number is $Re = 480,000$. The selected Reynolds number inserted in global.nml is $\frac{1}{\nu} = \frac{Re}{D} = 40,000$. Set the time step to $3e-3$ and the number of steps for a revolution here is 2800. The **overlap** flag is set to1.0 to turn off volume cutting by the background because the grid has been designed with enough overlap. **Statistics** done for the propeller case are done in cylindrical coordinates due to the overset mesh sampling in both space and time. The number of overset processors (**novrprocs**) is set to match one to one to the number of overset grid partitions.

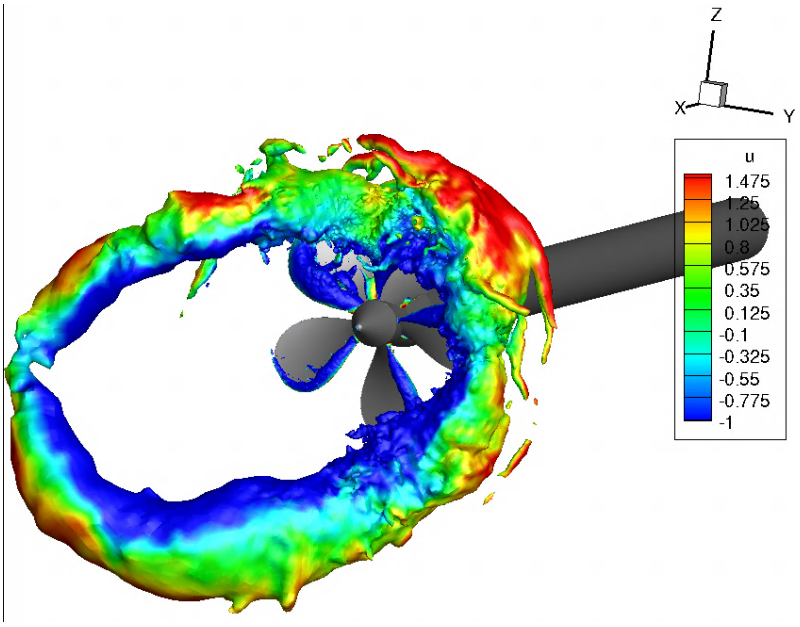
Pre-Run Checks

Before running or when trying to debug potential issues, the user can turn on the **dump_grid** flag to check that the initial condition and grids are set up correctly. Set this flag to 1 and then run the case. The solution will be in the result folder. Below we see the background grid, the overset propeller grid and the two grids together.



Results

The results should show the ring vortex, one of the main flow features of Crashback that causes force coefficients to be highly unsteady. A long validation run of about 300 revolutions is needed to converge force coefficient statistics and mean flow-field statistics.



Best Practices

Mesh Quality

- Rules on high aspect ratios and skewness for mpcugles still apply here.
- Mesh quality near blade is crucial to get force coefficients correct.
- CFL condition, time step and machine are crucial to get better run times.

Overset Mesh Configurations

- Though there is additional flexibility in grid generation while using overset, make sure to keep this balanced with the overset capabilities and limitations in mind in order to get the best results.

Interpolation

- **Penalty Weight**:
  - This is defaulted to a value of 1 and is recommended to stay at this value.
- **Overlap**: Check the Interpolation section for more details.
  - Make sure there is enough **overlap** for CVs to find enough interpolation partners.

    - It is recommended to have at least 5 interior control volumes between two areas that are interpolating. This is to avoid using interpolation partner CVs too close to other interpolation CVs which increases interpolation error.
- **Number of interpolation partners**: Check the Interpolation section for more details.
    - Avoid large differences in CV sizes (aim for max 2.0 ratio in each direction)
    - If the case is that there are large CV sizes differences, use **num_interp_parts_max** to increase the number of partners that the code searches for to include in interpolation. Note that this will slow down the case speed but improve the solution quality.
- **Large displacements**: Check that the per time step movements don't lead to skipping CVs due to large movements. For example, with a specified rotation rate, the grid displacements increases the further away from the rotation center. The propeller case here is well within that limit but keep this in mind when designing cases.

Running with overset grid motion

- Only use the following options if there is overset motion in the simulation and thus force affecting pressure fluctuations.
    - **Staggered Pressure Formulation** (**tau_stab=timestep**)
        - This is the recommended method for the solution.
        - This flag controls pressure stabilization in time. It's recommended that it's value is equal to the time step value if in use (**tau_stab=timestep**). Use only when the specific case involves overset mesh movement and time step to time step pressure fluctuations that affect force calculations, otherwise set to zero.
    - $\widehat{V}$ **LSQ Reconstruction** (**vhat_recon**=1)
        - This uses Least Squares (**LSQ**) to reconstruct face velocity ($\widehat{V}$) values between interpolation partners and interior CVs to help reduce time step to time step pressure fluctuations that can affect force calculations.
        - Generally, it is not recommended to use this method. It can have stability issues.
        - Avoid if interpolation is in areas of high gradients.
        - Avoid if interpolation CVs have too few neighbors or ratios between meshes are too high (larger than 2:1 in each direction).

LES

- Best LES Model: Dynamic Model with Lagrangian Averaging has been shown to be the best for inhomogeneous flows.

Nodal Reconstruction for Laplacian Operators on Skewed Meshes

Wyatt Horne

Finite volume discretizations often use a two point flux approximation when approximating Laplacian operators. While functionally simple, these operators are only convergent when meshes satisfy a K-orthogonality condition where cell centroids must align with gradient directions in a finite difference sense to estimate normal gradients. In addition to not converging, erroneous behavior can be introduced to the results. For the instance of the Incompressible Navier-Stokes equations, incorrect pressure fields and viscous stresses are introduced destroying accuracy of important predictions such as forces along bodies.

In general when using unstructured meshes it is functionally impossible to achieve the K-orthogonality condition. This suggests that the error of a two point flux approximation will be present in some manner no matter what is achieved during grid creation. It can at best be minimized to hopefully fall beneath other convergent errors.

Many methods have been constructed to remove this flaw. The purpose of this document is to outline the implementation of one such method which falls in the family of multi-point-flux-appoximation methods (MPFA). In these methods, fluxes are reconstructed at several points along the faces of cells using more accurate reconstructions than a two point approximation. These fluxes are used to enhance the two point approximation and remove the K-orthogonality requirement.

Method Outline

Reconstructions are performed at nodes attached to each cell using cells that are attached to each node as shown in 2D in Fig. 1. A gradient and nodal value is produced at each node using

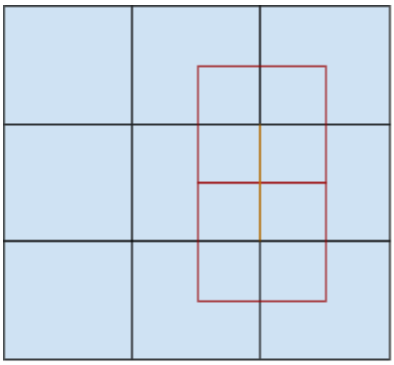


Fig 1. Dual volume (red) reconstruction around an example face (orange). Reconstruction uses the centroid values at each dual's corners.

a least squares approach to create solutions within dual volumes around each node. The values within the dual nodes are then used to evaluate values across the portions of each face that lie within the dual. Using a 2nd order approximation this corresponds to taking the average of each dual volume's value attached to a given face.

For the example of a Poisson equation, Laplacian operators are placed in the form

$$\int \partial p/\partial n \, da \;=\; \int f \, dV$$

where p is the variable being assessed, n is the normal along the faces of a cell, a is the area of the face of a cell, V is the volume of the cell, and f is a general forcing. The flux to be evaluated is therefore $\partial p/\partial n$.

The dual reconstructions could be used to directly calculate $\partial p/\partial n$ along each face discretely and then used to form a linear system for inversion. Unfortunately, the resulting linear system would be non-symmetric and not diagonally dominant resulting in poor convergence using traditional iterative techniques unless sophisticated, and likely expensive, preconditioners are employed. Instead we opt to use

$$\partial p/\partial n = (p_2 - p_1)/dx \;+\; (\partial p^d/\partial x) \cdot (n - s)$$

where the first term is the two point flux approximation across a face, $\partial p^d/\partial x$ is the reconstructed gradient using duals, and s is the normal vector between the cell centroids around a face.

The resulting linear system from this selection exactly matches the two point flux system when meshes are orthogonal. As meshes become increasingly distorted the correction from the 2nd term becomes increasingly dominant. If taken too far this will result in the same poor convergence behavior noted before, but overall the linear system is reasonable for traditional iterative techniques. If desired the corrective term can be lagged in time for temporally evolving

systems resulting in more temporal error, but a simpler to solve linear system that will still produce more accurate results than the two point flux approximation.

Implementation

To get the required cell connectivity it is necessary to create data structures which contain all cell centroid values around each node. This includes tricky corner cells across periodic boundaries and the possibility of a cell being connected to many nodes across processor boundaries and/or periodic boundaries.

To address these requirements, a new set of nodal connectivity routines were constructed which produce a data type named **node_connect**. This contains structures such as **cv_no_i**, **cv_no_v** which contain the centroids across nodes. Additionally it contains a new variable storage data type that has been termed a **halo_var**. This variable contains all centroid values of a desired variable with the inclusion of all nodal connections including values on other processors. It is coupled with an easy to use **update_halo_var()** subroutine that can acquire and store all required centroid values for nodal reconstructions across processor/periodic boundaries. A halo var can have an arbitrary number of columns and can be a double precision, integer, or 64 bit integer. It can be used for nodal or cell centroid variables.

To ease the implementation of **node_connect**, a routine was constructed which takes in general **node->cv** and **face->cv** connectivity and produces **node_connect** in parallel for arbitrary partitioned meshes. Implementation details of these routines can be found in the **/src/nodal_method** directory in the provided code.

A new set of routines have been introduced to simulate incompressible fluid flow using a predictor-corrector time advancement strategy for single and overset meshes that uses the outlined reconstruction. The routines are located in **/src/core/nodal_momentum_m.f90** and include relevant comments and explanations for the different steps of the process. Three halo vars have been introduced : **gp_d%cv_cc_n**, **sp_d%cv_press_n**, **sp_d%cv_vel_n**. These variables give cross nodal information for pressure, velocity, and spatial location of centroids. To keep the output/input identical to what has been used previously all halo variables are converted to **sp_d%temp_cv_velocity**, **sp_d%cv_velocity** and **sp_d%phi** before output and other operations that require their definition.

To invert massive linear systems HYPRE is utilized. New routines to connect the partitioned linear systems to HYPRE are introduced. Previously a general HYPRE/Fortran wrapper was created to solve general Ax=B using a CSR stored matrix. This wrapper is used here. The routines and explanations to connect to the HYPRE/Fortran wrapper can be found in **/src/core/nodal_hypre_wrapper_m.f90**

Numerical Method

A predictor-corrector method is used to simulate incompressible fluid flow. Here, you have the option to use Crank-Nicholson or BDF2 time advancement by changing the **overset_time_step_style** option in global.nml to 'cnicol_node' or 'bdf2_node' respectively. Below, details of the numerical method with BDF2 time advancement are provided.

The predictor step is taken as

$$(3u^{*} - 4u^{n-1} + u^{n-2})/(2dt) + NL(u^{*}) + VISC(u^{*}) = f + \partial p^{n-1}/\partial x$$

where u^{*} is the predicted velocity, n-1 denotes the current time level, dt is the time step, NL is the convective term, VISC is the viscous term, f is a general body force, and $\partial p^{n-1}/\partial x$ is the pressure gradient from the previous time step. Note that the viscous term contains a Laplacian which is evaluated using the previously outlined nodal reconstruction. The convective term has been left unaltered from our previous work in order to minimize energy dissipation.

To get a divergence free solution the equation is projected to the next time level using

$$(3u^{n} - 3u^{*})/(2dt) = -\partial\phi/\partial x$$
$$3\Theta^{*}/(2dt) = \Delta\phi$$
$$\phi = p^{n} - p^{n-1}$$

where ϕ is a gauge variable and Θ^{*} is the divergence of the predicted velocity field. The second equation in this series of equations is evaluated using the Laplacian operator previously described. The cell centered pressure gradient is reconstructed using the same method as our previous work, but utilizes the corrected $\partial p/\partial n$ in the least squares minimization rather than the two point flux approximation.

Overset connectivities are introduced as was described in our previous work. A penalty force is introduced into the pressure equation to constrain the solutions to agree across meshes. This requires dynamic matrix creation for moving body cases.

Runtime considerations

- Convergence will be harsher for this method than the simple two point flux approximation, particularly pressure. Adjustments might be necessary to achieve convergences that are acceptable for complicated meshes.

- The pressure has had the dt factor removed from within the momentum routine such that all pressure residuals control the actual accuracy of pressure not pressure*dt. This means that you can adjust your pressure zero to be : **pressure_zero_new = pressure_zero_old/dt**

- To use the new method set your **overset_time_step_style** to ‘cnicol_node’ or ‘bdf2_node’ in global.nml

- In general, assembly is more expensive now due to the additional nodal connectivity. This will lead to a slow down and up to **doubling** the typical time per time step of the regular method depending on the problem. Keep this in mind while choosing to use this method for your selected problem. For example, it might be wise to use the regular method for initial transients and switch to this method for the statistics and results.

Code Compilation

This section has information on compilation of the code on different machines that we have access to.

Building mpcugles

CMake

CMake is an open-source, cross-platform family of tools designed to build, test and package software. **CMake** is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice.

To build the overset code, we utilize **CMake**. The user should make sure to load the **cmake** module. Make sure to use a version **cmake/3.1** or above. Use **cmake --version** to check the version. If a module isn't available, the user should download a **CMake** tar file for use. The **cmake** executable is then located in the bin directory after the user untars the file.

Before beginning the build, make sure to check the Libraries section and have the HYPRE, HDF5, and NetCDF & NetCDFF libraries either compiled or the available machine compiled versions and their locations known.

Building

The following are the steps to take to build the overset code.

1: Prepare the Run Directory

Make sure that all necessary files are in the **run** directory. The main files should be located in the **bin** directory of the source code. (CMakeLists.txt, mpc_main.f90, user_hooks.f90, dump_tecplot_files.f90 etc.).

2: Create the Build Directory

Create a build directory inside the **run** directory. For example, the user can name this directory **mybuild**.

This directory can be unique for each **run** directory. Suggestion is to place it within each individual **run** directory. Whenever a user makes a change in user_hooks.f90, they can come back into the **mybuild** directory and compile to update the user's changes.

3: Edit custoptions.cmake

Change the **custoptions.cmake** file to the correct settings.

This file should be located in the **'<source_code>/build/options/**' directory.

In this file, you will select options for a "**Debug**" or "**Release**" build type, set the computer specific fortran flags and the directory locations for the **run** directory and libraries like **HYPRE**, **NetCDF** and optional libraries like **HDF5** and **EXODUS**.

- ★ **Note**: Always use the "**Release"** mode when running a full case. "**Debug"** mode will significantly slow down the case and is only for debugging.
- ★ **Note**: For fortran flags, it's recommended to only use (**-fpp -DINTEL -g -traceback)** when in **Release** mode. Use more when compiling the code in **Debug** form.
- ★ **Note**: The code allows parallel **XDMF+HDF5** for output if you want to do parallel visualization using **Paraview** or **Visit**. Make sure to have the **HDF5** library compiled and its install location or the location name of the module installation in the machine then turn **HDF5** on in this file.

4: cmake

Run the following command inside the **mybuild** directory:
cmake <source_code>/build/

This will use the settings in the **custoptions.cmake** file and check to make sure that all necessary directories, libraries and flags are found. If successful, this will also generate Makefiles for your build.

5: make

If the previous step was successful, use the following command to make the executables:
make

If successful, use the following command to install the executables in the **run** directory:
make install

- ★ **Note**: The user can use the capabilities to speed up the make process by making the files in parallel using the following command: **make -j**
- ★ **Note**: The user can do all the above with just one command: **make install -j**. This will make in parallel and if successful it will also install the executables.

Machine Environment

To check which shell you're using use the command (**`which $SHELL`**). In new **HPCMP** computers you should find out what this is and sometimes make your own shell script for aliases and module loads as soon as you login. You can also request a specific shell by contacting the help desk (help@helpdesk.hpc.mil).

Intel Libraries

Intel MPI Library (**intelmpi** or **impi**)

- MPI library from intel, common in all machines.

Message Passing Toolkit MPI Library (**mpt**)

- MPI library common in **SGI** machines.
- It has been found best to use version **2.15** or **2.16**. Higher versions have issues running on multiple nodes, this will be investigated more in the future.

Compilation Flags and MPI

	Machine	CC/MPICC	CXX	FC/F90/MPIF90	F77	Running MPI
MSI	**Mangi**	mpiicc	mpiicpc	mpiifort	mpiifort	**mpirun -np**
	Mesabi	mpiicc	mpiicpc	mpiifort	mpiifort	**mpirun -np**
SGI	**Centennial**	mpicc	mpicxx	mpif90	mpif90	**mpirun -np**
	Gaffney	mpicc	mpicxx	mpif90	mpif90	**mpirun -np**
	Koehr	mpicc	mpicxx	mpif90	mpif90	**mpirun -np**
Cray	**Conrad**	cc	CC	ftn	f77	**aprun -n**
	Copper	cc	CC	ftn	f77	**aprun -n**
	Excalibur	cc	CC	ftn	f77	**aprun -n**
	Gordon	cc	CC	ftn	f77	**aprun -n**
	Onyx	cc	CC	ftn	f77	**aprun -n**

Libraries

The following libraries are essential to be compiled in a new machine. Some clusters have local already built libraries and then one can choose to use them. **HYPRE** is the library that is often not included and is important to compile by the user.

HYPRE

The Parallel High Performance Preconditioners (**HYPRE**) is a library of routines for scalable (parallel) solution of linear systems. The main strength of **HYPRE** is availability of high performance parallel multigrid preconditioners for both structured and unstructured grid problems.

The **HYPRE** solver is a standalone **Fortran 90** to **HYPRE** wrapper. It is completely general for any **AX=b** that you might want to use it for. The wrapper is located in **hypre_solve_defs_m.f90** and **hypre_solve_func_m.f90** files. The user could theoretically take those two files and drop them into any Fortran code to use parallel 64 bit **HYPRE**.

HYPRE should be compiled making sure the correct flags are used matching the mpi libraries.

runconfigure.sh

Use the following file to help automate the compilation process.

- **ACCESS**: This is the installation directory of choice. Make sure it exists.This will be the final installation location of the **lib** & **include** folders.
- Make sure to change the **Compute Node Flags** to match the machine.

```
#! /bin/sh
ACCESS=$ACCESS/p/home/username/source/koehr/hypre_installs
if [ "X$ACCESS" == "X" ] ; then
  echo "ERROR: Please set the ACCESS environment variable before executing this script."
  exit
fi

rm -f config.cache
export CFLAGS="-O3"
export CPPFLAGS="-O3"
export CXXFLAGS="-O3"
export FFLAGS="-O3"
export LDFLAGS="-O3"

#COMPUTE NODE FLAGS
```

```
export CC='cc'
export MPICC='cc'
export CXX='CC'
export MPIF90='ftn'
export FC='ftn'
export F77='f77'
export F90='ftn'

echo ${ACCESS}
ls ${ACCESS}
echo $CC $CXX $F90
./configure --prefix=${ACCESS} --with-MPI --enable-bigint
```

Compilation steps:

1. Download **hypre** tar file or use one provided
2. Untar the file: **tar -xzvf hypre.tar.gz**
3. **cd** into the **hypre/src** directory
4. Create the **runconfigure.sh.** Make sure it's executable: **chmod 755 runconfigure.sh**
5. Run the configure script: **./runconfigure.sh**
6. If step 4 is successful: **make install**
7. If step 5 is successful: **make check** (Not required but important to notify you of any errors)
8. There should now be the **lib** & **include** folders in the **ACCESS** directory.

★ **Note**: The **--with-MPI** and **--enable-bigint** options are the most important, otherwise the code won't run.

HDF5

HDF5 is available and can be loaded as a module. However, we can only use **HDF5** with parallel capabilities. Therefore, you should select and load a version with **'hdf5-parallel'**. To find the path for use in ***custoptions.cmake***, use `echo $HDF5_DIR` or `module display 'module-name'`. Otherwise, you can compile yourself, making sure the correct flags are used matching the mpi libraries.

runconfigure.sh

Use the following file to help automate the compilation process.

- **ACCESS**: This is the installation directory of choice. Make sure it exists.This will be the final installation location of the **lib** & **include** folders.
- Make sure to change the **Compute Node Flags** to match the machine.

File contents:

```
#! /bin/sh
ACCESS=$ACCESS/p/home/username/source/koehr/hdf5_installs
if [ "X$ACCESS" == "X" ] ; then
  echo "ERROR: Please set the ACCESS environment variable before executing this script."
  exit
fi

rm -f config.cache
export CFLAGS="-O3"
export CPPFLAGS="-O3"
export CXXFLAGS="-O3"
export FFLAGS="-O3"
export LDFLAGS="-O3"

#COMPUTE NODE FLAGS
export CC='cc'
export MPICC='cc'
export CXX='CC'
export MPIF90='ftn'
export FC='ftn'
export F77='f77'
export F90='ftn'

echo ${ACCESS}
ls ${ACCESS}
echo $CC $CXX $F90
./configure --prefix=${ACCESS} --enable-parallel --enable-fortran
```

Compilation steps:

1. Download **HDF5** tar file or use one provided
2. Untar the file: **tar -xzvf hdf5.tar.gz**
3. **cd** into the **hdf5** directory
4. Create the **runconfigure.sh.** Make sure it's executable: **chmod 755 runconfigure.sh**
5. Run the configure script: **./runconfigure.sh**
6. If step 4 is successful: **make install**
7. If step 5 is successful: **make check** (Not required but important to notify you of any errors, also this might take very long for **HDF5**)
8. There should now be the **lib**, **include, bin,** and **share** folders in the **ACCESS** directory.

★ **Note**: The **--enable-parallel** and **--enable-fortran** options are the most important, otherwise the code won't run correctly with **HDF5**.

NetCDFF & NetCDFF

These are sometimes available as modules. If so, you can load them both and find their paths to be used for code compilation. To find the path for use in ***custoptions.cmake****,* echo $NETCDF_DIR or module display **'module-name'**. However sometimes they're in conflict with **HDF5** and you can compile yourself choosing between **NetCDF** or **HDF5** to compile. If compiling, you should make sure the correct flags are used matching the mpi libraries. Compile **NetCDF** first then **NetCDFF** using the same installation folder.

runconfigure.sh

Use the following file to help automate the compilation process.

- **ACCESS**: This is the installation directory of choice. Make sure it exists.This will be the final installation location of the **lib** & **include** folders.
- Make sure to change the **Compute Node Flags** to match the machine.

File contents:

```
#! /bin/sh
ACCESS=$ACCESS/p/home/username/source/koehr/netcdf_installs
if [ "X$ACCESS" == "X" ] ; then
  echo "ERROR: Please set the ACCESS environment variable before executing this script."
  exit
fi

rm -f config.cache
export CFLAGS="-I${ACCESS}/include"
export CPPFLAGS="-DNDEBUG"
export LDFLAGS="-L${ACCESS}/lib"
export LD="ld"

#COMPUTE NODE FLAGS
export CC='cc'
export MPICC='cc'
export CXX='CC'
export MPIF90='ftn'
export FC='ftn'
export F77='f77'
export F90='ftn'

echo libraries in
echo ${ACCESS}
ls ${ACCESS}
echo $CC $CXX $F90
```

SHARED="--disable-shared"
./configure --disable-netcdf-4 ${SHARED} --disable-v2 --disable-fsync --prefix=${ACCESS}
--disable-dap --enable-remote-fortran-bootstrap $1

Compilation steps:

1. Download **NetCDF** or **NetCDFF** tar file or use one provided
2. Untar the file: **tar -xzvf netcdf.tar.gz**
3. **cd** into the **netcdf** or **netcdff** directory
4. Create the **runconfigure.sh.** Make sure it's executable: **chmod 755 runconfigure.sh**
5. For **NetCDFF** change the last line to: ./configure ${SHARED} --prefix=${ACCESS}
6. Run the configure script: **./runconfigure.sh**
7. If step 5 is successful: **make install**
8. If step 6 is successful: **make check** (Not required but important to notify you of any errors)
9. There should now be the **lib**, **include, bin,** and **share** folders in the **ACCESS** directory.

MSI Machines

Mangi and Mesabi environments are all in one. If you have specific questions, contact (help@msi.umn.edu).

Mangi

shell = bash
Make a **.bashrc** shell script file in the home directory if one isn't available.
module load intel/cluster
module load cmake
module load cuda

Mesabi

shell = bash
Make a **.bashrc** shell script file in the home directory if one isn't available.
module load intel/cluster
module load cmake
module load cuda

SGI Machines

The compiling environment is very similar for these machines. For some of them, you might be able to re-use **NetCDF** once compiled in one of the computers but it is suggested compiling for each machine. You should definitely re-compile **HYPRE** for each machine. It has been found that you can compile using **intelmpi** but there are several issues when running on more than

one node. This will be investigated more in the future. Therefore, the mpi libraries to use are **mpt** with the **intel** compilers.

Compiling Libraries

The following libraries are essential to be compiled in a new machine. Some clusters have local already built libraries and then one can choose to use them. **HYPRE** is the library that is often not included and is important to compile by the user.

HYPRE

This should be compiled making sure the correct flags are used matching the mpi libraries. Check the HYPRE section for more detailed information on how to compile.

HDF5

For **SGI** machines, these modules might not work together with **NetCDF** modules and also only work for **intelmpi** compilers. Because of these issues and that we have to use **mpt**, for now, we have to compile our own **HDF5**. This should be compiled making sure the correct flags are used matching the mpi libraries. Once compiled, make sure to edit the ***custoptions.cmake*** file and set the installation directory correctly as well as to set **USE_M** & **USE_Z** to **ON** with **M_DIR & Z_DIR** set to **"/lib64"** or if not available **"/usr/lib64",** then finally set **M_SHARED** and **Z_SHARED** to **ON**. Check the HDF5 section for more detailed information on how to compile.

NetCDF & NetCDFF

These are sometimes available as modules. If so, you can load them both and find their paths to be used for code compilation. To find the path for use in ***custoptions.cmake,*** echo $NETCDF_DIR or module display **'module-name'**. However sometimes they're in conflict with **HDF5** and you can compile yourself choosing between **NetCDF** or **HDF5** to compile. It is recommended to compile both **NetCDF** and **HDF5** in these machines. These libraries should be compiled making sure the correct flags are used matching the mpi libraries. Compile **NetCDF** first then **NetCDFF** using the same installation folder. Check the NetCDF & NetCDFF section for more detailed information on how to compile.

Centennial (SGI ICE XA)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
Correct modules are loaded automatically. The **mpi/sgimpt/2.15** should already be loaded. You can switch to the latest intel compilers if you wish.
module swap **compiler/intel/19.1.022**

Gaffney (HPE SGI 8600)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module unload **mpt**
module load **mpt/2.16**

Koehr (HPE SGI 8600)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module unload **mpt**
module load **mpt/2.16**

Cray Machines

The compiling environment is very similar for these machines. For some of them, you might be able to re-use **NetCDF** once compiled in one of the computers but it is suggested compiling for each machine. You should definitely re-compile **HYPRE** for each machine.

Compiling Libraries

The following libraries are essential to be compiled in a new machine. Some clusters have local already built libraries and then one can choose to use them. **HYPRE** is the library that is often not included and is important to compile by the user.

HYPRE

This should be compiled making sure the correct flags are used matching the mpi libraries. Check the HYPRE section for more detailed information on how to compile.

HDF5

HDF5 is available and can be loaded as a module. However, we can only use **HDF5** with parallel capabilities. Therefore, you should select and load a version of **cray-hdf5-parallel**. To find the location path for use in ***custoptions.cmake***, use **echo $HDF5_DIR** or **module display cray-hdf5-parallel.** Make sure to set **USE_M** & **USE_Z** in ***custoptions.cmake*** to **OFF** if you use the module. It is recommended to use the **HDF5** module in these machines. Otherwise, you can compile yourself, making sure the correct flags are used matching the mpi libraries. Check the HDF5 section for more detailed information on how to compile.

NetCDF & NetCDFF

These are sometimes available as modules. If so you can load them both and find their paths to be used for code compilation. To find the path for use in ***custoptions.cmake***, echo

$NETCDF_DIR or module display **'module-name'**. However sometimes they're in conflict with **HDF5** and you can compile yourself choosing between **NetCDF** or **HDF5** to compile. It is recommended that you compile **NetCDF** and load **HDF5** as a module. These should be compiled making sure the correct flags are used matching the mpi libraries. Compile **NetCDF** first then **NetCDFF** using the same installation folder. Check the NetCDF & NetCDFF section for more detailed information on how to compile.

Conrad (Cray XC40)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module load **intel cray-hdf5-parallel**
module swap **PrgEnv-cray PrgEnv-intel**
module unload **craype-hugepages2M**

Copper (Cray XE6m)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module load **intel cray-hdf5-parallel**
module swap **PrgEnv-cray PrgEnv-intel**
module unload **craype-hugepages2M**

Excalibur (Cray XC40)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module load **intel cray-hdf5-parallel**
module swap **PrgEnv-cray PrgEnv-intel**
module unload **craype-hugepages2M**

Gordon (Cray XC40)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module load **intel**
module swap **PrgEnv-cray PrgEnv-intel**
module load **cray-hdf5-parallel**
module unload **craype-hugepages2M**

Onyx (Cray XC40/50)

shell = tcsh
Make a **.tcshrc** shell script file in the home directory if one isn't available.
module load **intel cray-hdf5-parallel**
module swap **PrgEnv-cray PrgEnv-intel**
module unload **craype-hugepages2M**