FUN3D v13.4 Training

Session 15: Overset-Grid Simulations

Li Wang

Bob Biedron

Session Scope

- What this will cover
  - Static and dynamic simulations in FUN3D using overset meshes and SUGGAR++ /DiRTlib
  - Key inputs and procedures
- What will not be covered
  - SUGGAR++ operation (referred to Ralph Noack's 2015 Training materials)
- What should you already be familiar with
  - Basic time-accurate simulations
  - dynamic-mesh solver operation and control

Introduction

- Background
  - Many moving-body problems of interest involve large relative motion - rotorcraft, store separation are prime examples
    - Single rigid mesh allows only one body; no relative motion
    - Deforming meshes allow limited relative motion before mesh degenerates
    - Use overset grids to overcome these limitations
- Compatibility
  - Requires DiRTlib and SUGGAR++ from Celeritas Simulation Tech.
  - Grid formats: VGRID, AFLR3, FieldView (FV)
- Status
  - *Overset grids generally limit FUN3D scalability for dynamic meshes*
    - *SUGGAR++ executes in FUN3D on single core if connectivity data is computed during the flow solve*

Overset - General Info

- Configuring FUN3D for overset
  - Use **`--with-dirtlib=/path/to/dirtlib`** and **`--with-suggar=/path/to/suggar`**
  - FUN3D will expect to find the following libraries in those locations:
    - **`libdirt.a, libdirt_mpich.a`** and **`libp3d.a`** (these may be soft links to the actual serial and mpi builds of DiRTlib)
    - **`libsuggar.a`** and **`libsuggar_mpi.a`** (may be soft links)
- You will also need a "stand-alone" SUGGAR++ executable in addition to the library files that FUN3D will link to
- Recommend to use **`SPARSKIT`** package for mesh deformation solver
- Grids
  - A *composite* overset grid is comprised of 2 or more *component* grids - independently generated - but with similar cell sizes in the fringe areas
  - SUGGAR++ assembles the composite grid from the component grids, and determines overset connectivity data for the composite mesh

Overset Preprocessing (1/4)

- Overset simulations start with an execution of SUGGAR++ to generate a composite grid and initial (t=0) connectivity data
  - When generating component meshes, try to make cell sizes “similar” in the overlap regions - i.e. fringe (receptor) and donor have same size
  - Create an XML input file for SUGGAR++
    - Use the name of your FUN3D project for the names appearing in **`<composite_grid>`** and **`<domain_connectivity>`**
    - Can mix and match component grid types (VGRID, FV, AFLR) and select one of the types for the output composite grid - but note VGRID only supports tetrahedra
  - Run SUGGAR++ and make sure it all works as expected. You should now have a **`[project].dci`** file; this ***D***omain ***C***onnectivity ***I***nformation file contains all necessary overset data for solver interpolation between the component meshes at t=0
    - % **`suggar++ Input.xml_0`**

Overset Preprocessing (2/4)

- Example of XML input for assembling 2 component grids (static)

```
<global>
  <donor_quality value="0.9" />
  <minimize_overlap keep_inner_fringe="yes"/>
  <thin_cut set_to="out"/>
  <output>
    <composite_grid style="aflr3" filename="robin.b8.ugrid" format="binary/>
    <domain_connectivity style="unformatted_gen_drt_pairs" filename="robin.dci"/>
  </output>
  <body name="complete">
    <body name="fuselage">
        <transform>
          <scale value="2.0000E+00"/>
        </transform>
      <volume_grid name="rob_merged" style="aflr3" filename="rob_merged.b8.ugrid"
                                                    format="binary">
      </volume_grid>
    </body>
    <body name="outerbox">
      <volume_grid name="box_test" style="vgrid_set" filename="box_test">
      </volume_grid>
    </body>
  </body>
</global>
```

Component grid 1

Component grid 2

Overset Preprocessing (3/4)

- Example of XML input for assembling 2 component grids (static), contd.
  - Place component- grid files in run/working directory

Component grid 1 (AFLR)	Component grid 2 (VGRID)
rob_merged.b8.ugrid	**box_test.cogsg**
rob_merged.mapbc	**box_test.bc**
rob_merged.suggar_mapbc	**box_test.mapbc**

For nonVGRID grids, auxiliary SUGGAR++ BC file: [**component**]**.suggar_mapbc** can be created by using [**component**]**.mapbc** file

  - With successful SUGGAR++ execution, check composite-grid files

robin.b8.ugrid
robin.dci
robin.b8.ugrid.suggar_mapbc

    - If composite grid is VGRID format, the FUN3D mapbc file is created automatically; otherwise, you need to manually create **[project].mapbc** file - Can start with composite-grid **suggar_mapbc** file and modify
    - Create "Family" entries in **mapbc** file to use with **patch_lumping="family"** in **fun3d.nml** file

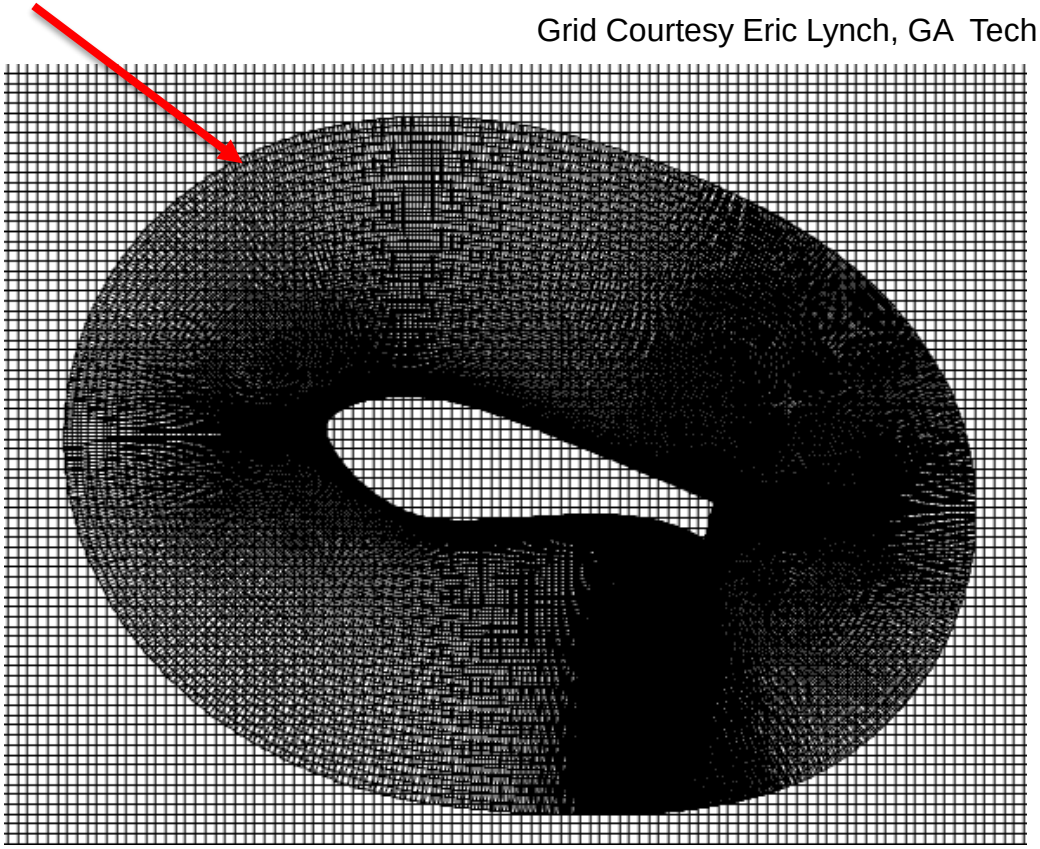
Overset Preprocessing (4/4)

- For dynamic-grid simulations, there is an additional consideration at the preprocessing stage: either precompute the overset connectivity for *ALL* time steps up front, or do this “on the fly” from within FUN3D
  - Precomputing requires up-front knowledge of the motion - *rules out 6DOF and aeroelastic cases* since the motion depends on the flow solution; *rules out deforming meshes* even if motion known
  - If the case fits these restrictions, from the point of view of flow solver run time, precomputing all connectivity is the most efficient
  - Need to ensure that SUGGAR++ motion will match FUN3D motion
  - Resulting dci files *must* be named **`[project]N.dci`** for timestep N
- If connectivity is computed at run time (by necessity or for convenience)
  - Computation of overset connectivity is performed on a single processor (the last one)
  - That processor must have enough memory (basically same memory requirements as stand alone SUGGAR++)

Overset – Boundary Conditions

- FUN3D requires only one specialized overset boundary condition - all other BCs can be applied as needed:
  - In mapbc files, set BC type to -1 for boundaries that are set via interpolation from another mesh

Grid Courtesy Eric Lynch, GA Tech



Overset – Boundary Conditions (Cont.)

- SUGGAR++ needs BC info for each component grid
  - Can be set either via the SUGGAR++ input XML file OR an auxiliary file for each component grid, **`[component].suggar_mapbc`**
  - If using the XML file approach
    - Add **`<boundary_surface>`** to **`<volume_grid>`** element

```
<boundary_surface find="yes" name="Surf=1">
  <boundary_condition type="solid"/>
</boundary_surface>
```

each surf patch

`type` can be "solid", "overlap", "farfield", etc.

    - More cumbersome than auxiliary file, but...
    - If the auxiliary files get separated from the other files, SUGGAR may assume some defaults which can cause problems with hole cutting
    - The exception to setting SUGGAR++ BC info in the XML file is if ALL the component grids are of VGRID type - in that case both SUGGAR++ and FUN3D get BCs from the same VGRID mapbc file and can generally avoid having to explicitly set any BCs for SUGGAR++

Overset – Namelist Input

- Control of overset operations - primarily for dynamic grids - set in the **`&overset_data`** namelist in **`fun3d.nml`**

`overset_flag`	**`= .true.`**	turn on overset (default: .F.)
`dci_on_the_fly`	**`= .true.`**	compute connectivity during flow solve (.F.)
`reuse_exisiting_dci`	**`= .true.`**	if dci file for this step already exists, use it instead of computing on the fly (.F.)
`dci_period`	**`= N`**	dci data repeats every N steps (huge no.)
`reset_dci_period`	**`= .true.`**	allows change of dci_period for restart (.F.) …used for time-step change at restart
`dci_freq`	**`= M`**	compute dci data every M steps (1)
`dci_dir`	**`= 'dir'`**	look for or put dci files in this dir (./)
`skip_dci_output`	**`= .true.`**	*don't* write dci data after it's computed (.F.) …maybe this data won't be needed again
`dci_io`	**`= .true.`**	use dedicated proc(s) for fast loading of precomputed dci data (.F.) - *more later*
`dci_io_npro`	**`= P`**	use P procs for dedicated dci loading (1)

Overset Mesh Simulations – Static (1/2)

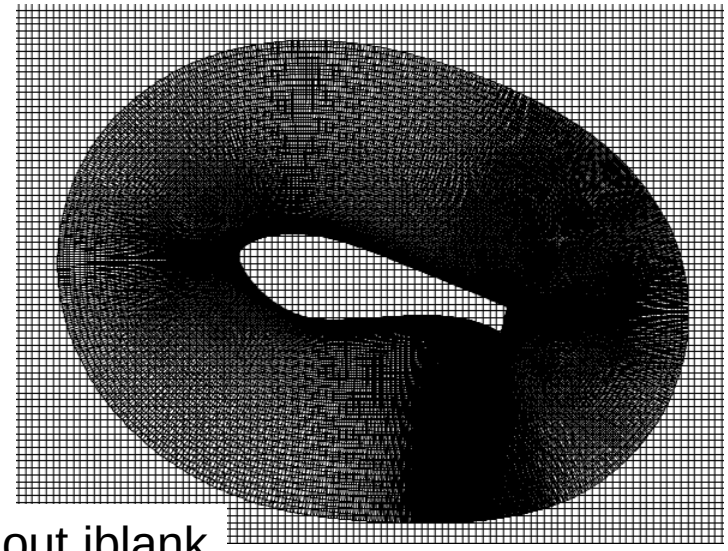
- Running FUN3D with static overset meshes:
  - Set **`overset_flag = .true.`** in the **`&overset_data`** namelist in **`fun3d.nml`** (Alt.: use the CLO **`--overset`**)
  - Require initial (t=0) DCI file **`[project].dci`** in place (preprocessed)
  - In screen output, should see something like:

```
dirtlib:init_overset Reading DCI data: ./[project].dci
Loading of dci file header took Wall Clock time = 0.002223 seconds
Loading of dci file took Wall Clock time = 0.005657 seconds
Using DiRTlib version 1.50.9 for overset capability
DiRTlib developed by Ralph Noack, Penn State University Applied Research Laboratory
```

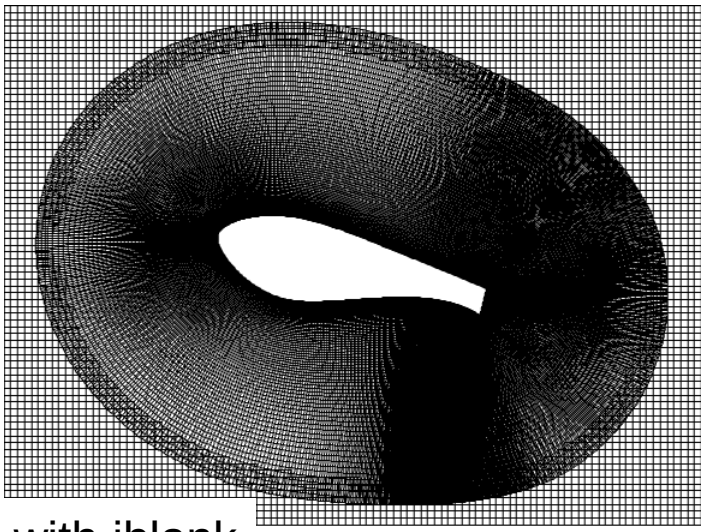
  - If you request visualization output data for an overset case, “iblank” data will automatically be output to allow blanking of the hole / out (*iblank = 0*) points for correct visualization of the solution / grid in Tecplot

Overset Mesh Simulations – Static (2/2)

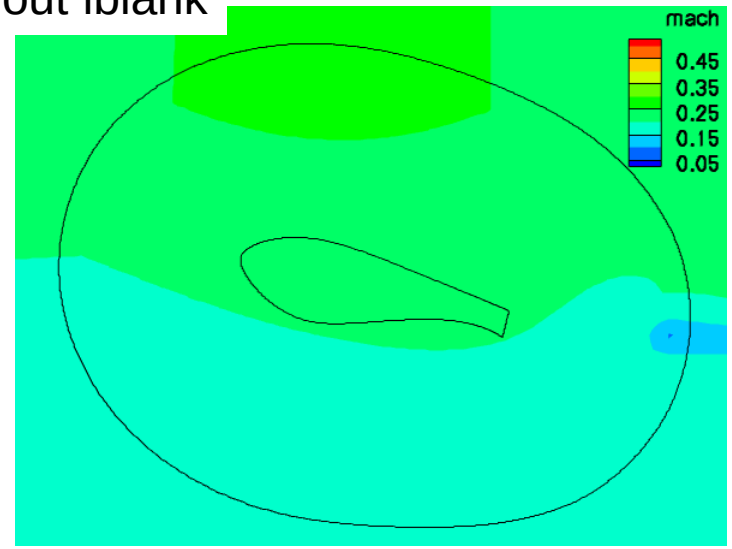
- Wind-turbine airfoil in tunnel

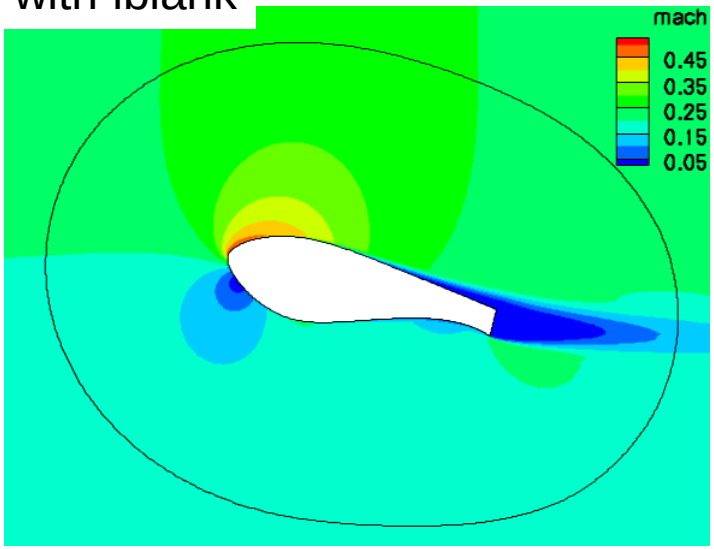


without iblank



with iblank





Overset Mesh Simulations – Dynamic (1/5)

- SUGGAR++ setup (details referred to Ralph Noack's 2015 training materials)
  - Starting from a static-grid XML file:
    - Add **<dynamic/>** to **<body>** elements that are to move, e.g.,

```
<body name="airfoil">
  <dynamic/>
    <transform>
      <translate axis="x" value="  1.3800E+00"/>
      <translate axis="y" value="  0.0000E+00"/>
      <translate axis="z" value="  5.4800E-01"/>
    </transform>
  <volume_grid name="airfoil" style="fvuns"
                              filename="airfoil_2p.fvgrid_fmt"/>
</body>
```

    - Note: use a self-terminated **<dynamic/>** so that any **<transform>** elements of **<body>** are applied to the initial component grid position when assembling the composite grid
  - Use SUGGAR++ to generate the initial (t = 0) composite grid

```
% suggar++ Input.xml_0
```

Overset Mesh Simulations – Dynamic (2/5)

- In the FUN3D **`moving_body.input`** file
  - Define the bodies and specify motion as usual; boundary numbers correspond to those in the *composite* mesh mapbc file, accounting for any boundary lumping that may be selected at run time
  - Use the component body names from the **`Input.xml_0`** file
  - Add name of the xml file used to generate the t = 0 composite mesh:

```
&composite_overset_mesh
  input_xml_file = 'Input.xml_0'
/
```

- Running FUN3D
  - Set **`moving_grid = .true.`** in **`&global`** namelist and **`overset_flag=.true. dci_on_the_fly = .true.`** in **`&overset_data`** namelist
  - When **`dci_on_the_fly = .T.`**, FUN3D calls libSUGGAR++ to compute new overset data when the grids are moved; if **`.false.`** (default), solver will try to read the corresponding dci files from disk

Overset Mesh Simulations – Dynamic (3/5)

- Running FUN3D (cont)
  - Note: when **`dci_on_the_fly = .true.`**, the *component* grids and mapbc files must be available (can be soft linked) in the FUN3D run directory, in *addition* to the t = 0 *composite*-grid and mapbc files
  - When using **`--dci_on_the_fly`**, specify *one* additional processor for SUGGAR++
    - The *last* processor gets assigned the SUGGAR++ task
    - *This processor must have enough memory for entire overset problem* (same as needed for SUGGAR++ alone)
  - There are a number of other overset-grid CLOs that may be useful for dynamic overset meshes (see “Overset – Namelist Input” slide).

Overset Mesh Simulations – Dynamic (4/5)

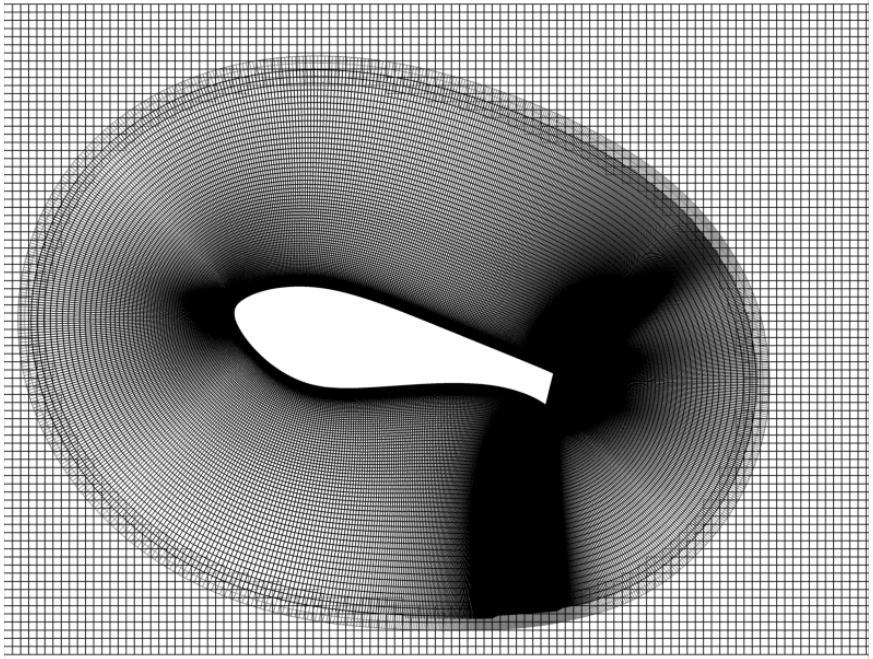
- Another option, in the **`&global_data`** namelist in **`fun3d.nml`**

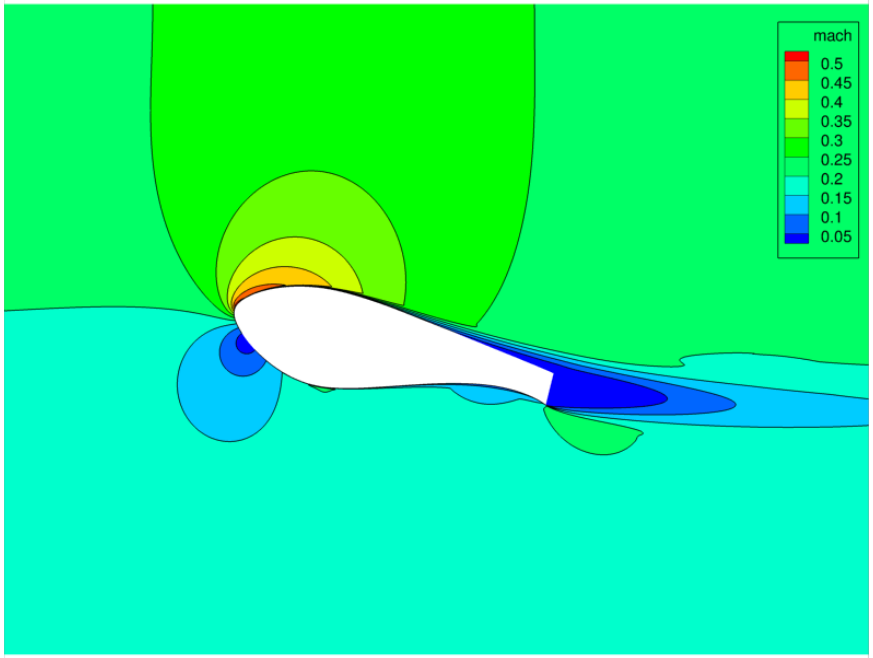
  `grid_motion_and_dci_only = .true.` (default: .F.) step through the mesh motion and compute dci data but don't solve flow eqns.

  - Useful as an easy (not the most efficient) way to precompute dci data while ensuring the motion will match exactly with FUN3D
- Solution data in hole points (governing equations *not* solved at hole pts.)
  - Starts at freestream
  - FUN3D will "fill in" flow data at hole points at each time step by averaging data at surrounding points - eventually replaces freestream
  - Averaging is important for dynamic case so a hole point that suddenly becomes a solve point has something better than freestream as an IC
  - *Best Practice*: use "keep inner fringe" option in SUGGAR++ XML file - retains extra fringe (interpolated) points near hole edges as a buffer of points that become exposed before hole pts. - interp. better than avg.

Overset Mesh Simulations – Dynamic (5/5)

- Wind-turbine airfoil in tunnel





Example – Store Separation (1/5)

- Test case located in: tutorials/flow_overset_grids
- Super-coarse grid for a 4-finned store magnetically suspended below a semi-span wing. Could be hooked up to 6DOF library but here we specify the motion for t > 0 as a constant downward velocity
- **`run_tutorial.sh`**
  - First runs stand-alone SUGGAR++ executable to generate a dci file for a static-grid / steady-state solution
  - Next runs nodet_mpi to give a steady-state solution on composite mesh - this will become the starting solution (t=0) for the moving-grid / unsteady case
  - Finally runs moving-grid case in which dci data generated “on the fly” for each of 50 time steps

Example – Store Separation (2/5)

- Set up SUGGAR++ XML file **`wingstore.xml`**

```
<global>
  <symmetry_plane axis="Y"/>
  <minimize_overlap keep_inner_fringe="yes"/>
  <output>
  <composite_grid style="unsorted_vgrid_set" filename="wingstore"/>
  <domain_connectivity style="ascii_gen_drt_pairs" filename="wingstore.dci"/>
  </output>
  <body name="wingstore">
    <body name="wing">
      <volume_grid name="wing" style="vgrid_set" filename="wing"/>
    </body>
    <body name="store">
      <dynamic/>
      <volume_grid name="store" style="vgrid_set" filename="store">
      </volume_grid>
    </body>
  </body>
</global>
```

- Add <dynamic/> tag since we will ultimately be doing moving-grid case
- Component grids are VGRID – don't need explicit BCs in the XML file

Example – Store Separation (3/5)

- Relevant **fun3d.nml** data (static grid / steady state)

```
&overset_data
   overset_flag = .true.
/
&project
   project_rootname = "wingstore"  ! same as <composite_grid> filename
/                                  ! we set in wingstore.xml
```

- Relevant **fun3d.nml** data (moving / unsteady)

```
&overset_data
   overset_flag = .true.
   dci_on_the_fly = .true.    ! Must have composite ("wingstore") and
/                             ! Component grids ("wing" and "store")
&global                       ! in the run directory
   moving_grid = .true.
/
&project
   project_rootname = "wingstore"
/
```

Example – Store Separation (4/5)

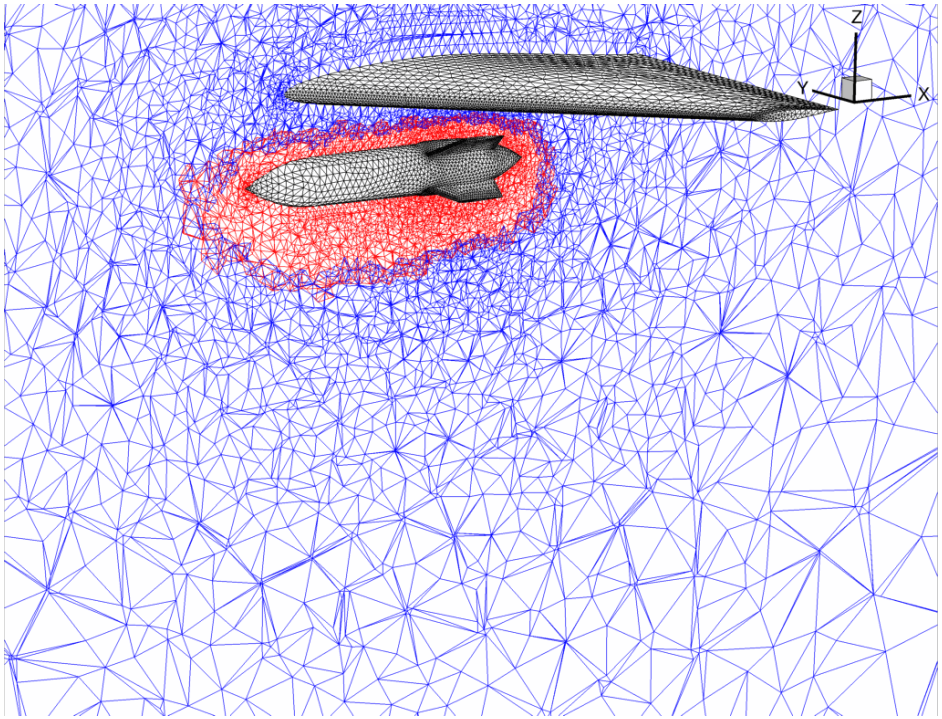
- Relevant **`moving_body.input`** data (moving / unsteady)

```
&body_definitions
   n_moving_bodies     = 1
   body_name(1)        = "store"  ! same name used in xml file
   n_defining_bndry(1)= 1
   defining_bndry(1)   = 4
   mesh_movement(1)    = "rigid"
   motion_driver(1)    = "forced"
/
&forced_motion
   translate(1) = 1              ! constant-rate translation
   translation_rate(1) = -0.2    ! Mach 0.2 downward
/
&composite_overset_mesh
   input_xml_file = "wingstore.xml"
/
```

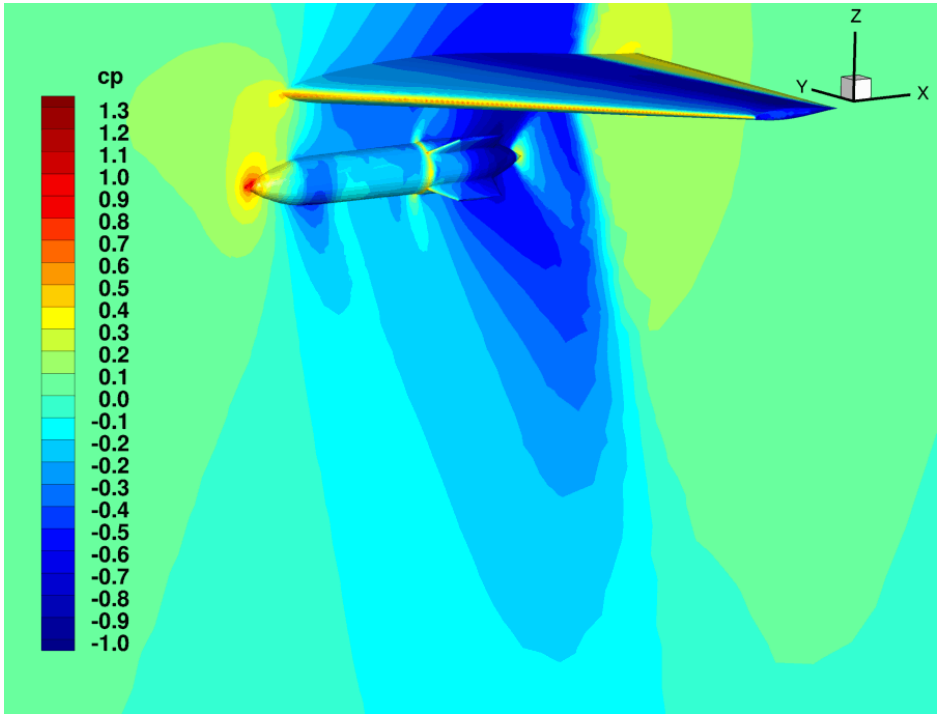
Example – Store Separation (5/5)

Slices Through Store Centerline

Hole Cutting
(mesh_animation.lay)



Pressure Coefficient
(cp_animation.lay)

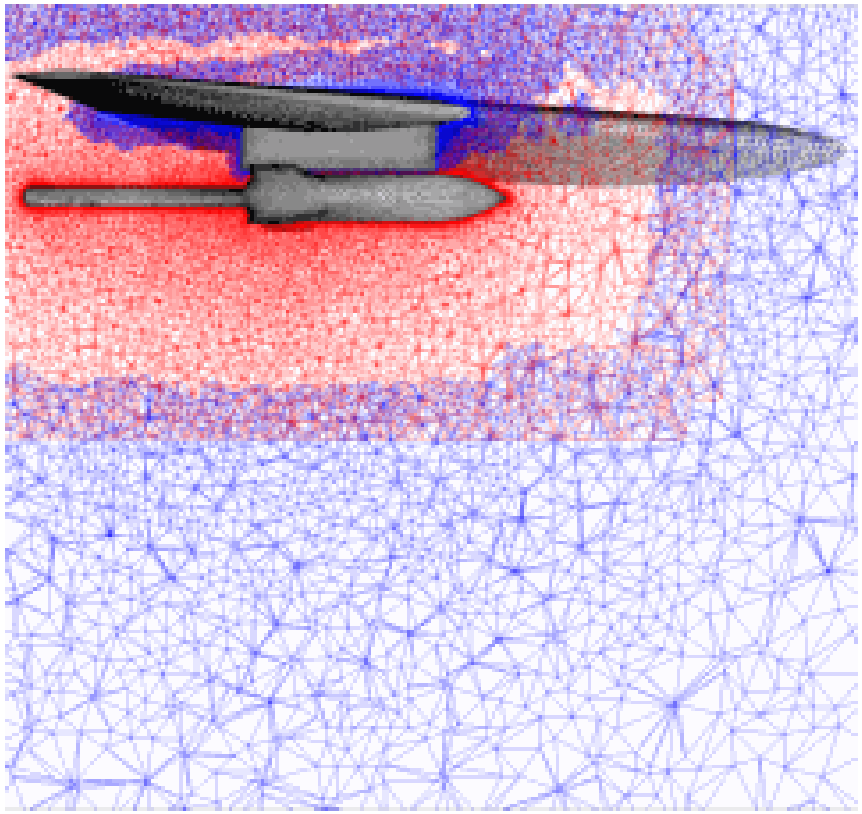


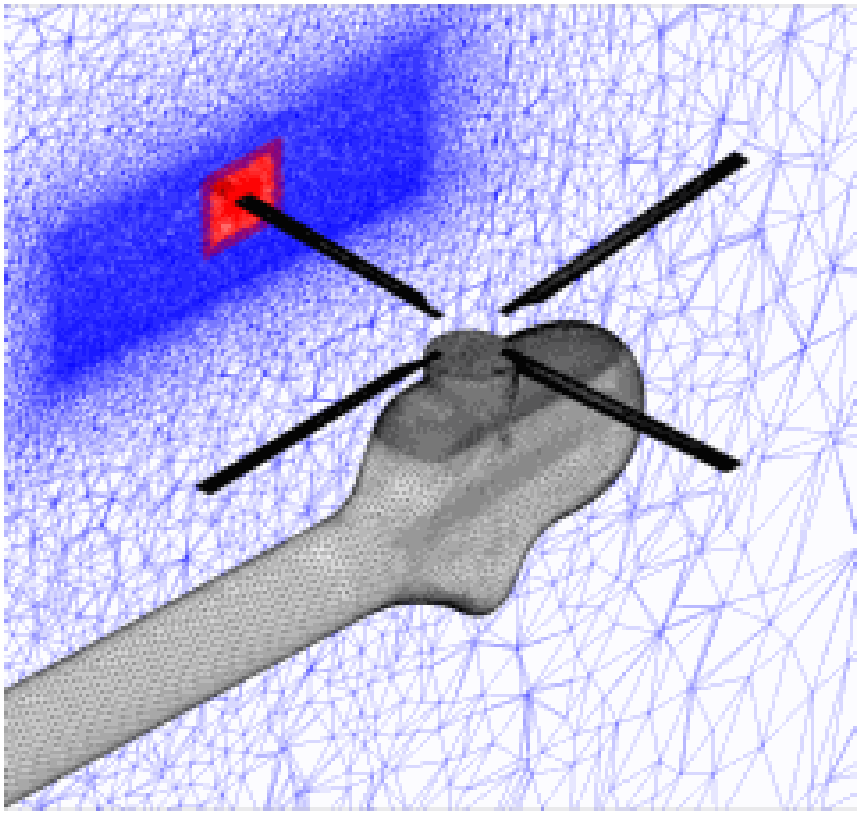
DCI_IO For Large-Scale Simulations

- Some applications are now run on many-thousand core architectures. Suggar++ does not scale to this level, but for *rigid meshes with prescribed motion*, it is possible to precompute the connectivity data in an "embarrassingly parallel" fashion, avoiding a bottleneck during FUN3D execution
- Normally FUN3D calls DiRTlib routines to load and parse this precomputed dci data. But DiRTlib reads and parses the dci file from *every* processor, which prohibits scalability beyond ~1k cores
- Instead, use dci_io = .true. and use dci_io_nprocs = P to assign P processes to read and distribute the dci data - circumvents DiRTlib
  - this is the *only* job for these processors - they operate 1 to P time steps ahead; regular flow-solve ranks work to advance flow in current step
- DCI_IO utilizes a special file containing a subset of dci data - "dcif" file
  - Convert dci generated by SUGGAR++ to dcif using utils/dci_to_dcif
- Linear scaling demonstrated up ~4K cores; P = 1 sufficient for this size

Overset Mesh Simulations – Examples

- As always, can use animation to verify; these were done using Tecplot output from FUN3D





Troubleshooting

- Orphan count is an indicator (though hardly precise) of problems - either in setup of SUGGAR++ or a poor mesh
  - Both standalone SUGGAR++ and FUN3D (“on the fly”) report orphan counts
    - should have none “due to hole-cut failures”; nonzero count a good indicator of setup issues
    - orphans “due to donor quality” perhaps an indicator or grid quality or setup
  - Visualization often the best tool to remedy
  - Celeritas’ GVIZ or Tecplot output from FUN3D can help sort out overset connectivity issues