Fully Unstructured Navier-Stokes

Current Release: **12.4-70371**
Site Last Updated: **Wed Jun 04 16:41:00 -0400 2014**

Previous (C4: Boundary Conditions) | Up | Next (C6: Analysis)

5. Pre/Post Processing

5.1. Grid/Solution Processing with v11.0 and Higher

As of FUN3D v11.0 and higher, there is generally not a need to use the traditional party/pparty tools associated with FUN3D (the pparty executable has actually been discontinued). The solvers for v11.0 and higher now load raw grid files (VGRID, FAST, etc.) directly and partition/pre-process them internally prior to the solution phase. This new paradigm is much more scalable, eliminates the need to run a separate pre-processing step, and also eliminates the need for a set of grid partition files. When running on hundreds or thousands of processors, these sets of files became very burdensome to deal with. Moreover, existing runs/solutions may now be seamlessly restarted on any number of processors, independent of the number used in the previous run – the solvers will now load the existing restart data and redistribute it internally as necessary amongst the specified resources.

Users should find the new internal preprocessing paradigm extremely efficient as well; great care has been taken to optimize its performance and memory footprint. As a point of reference, a grid from the recent AIAA Drag Prediction Workshop with 105 million gridpoints and 600 million elements formerly required 2 weeks of wallclock time and 800 GB of shared memory (on a large shared memory machine) to preprocess for 1,024 partitions with the traditional party preprocessor. The new paradigm partitions this same test case using 1,024 cores in 5 minutes wallclock time (on an SGI ICE distributed memory machine running the Lustre file system) without the need for a large segment of shared memory.

In addition to the changes to the “front end” of the solvers, the “back end” has also been revamped. Restart information is now contained in a single file, rather than a file for each partition. Generation of visualization files for post-processing purposes should now be done with the co-processing options available in the solver. These currently include options for Tecplot; future implementations will also include Fieldview options. Party is unable to post-process the new single-file restarts.

Although party must still be used for some less common features, all of its functionality will eventually be migrated to the new paradigm, eliminating the party application entirely. Existing partition files generated by party can still be run in v11.0, and existing 10.8 restart files can also be used with v11.0, but these options will soon be phased out. For these reasons, users should begin migrating to use the new paradigm for both grid processing and restart files.

Please report any issues to the support team.

Loading Raw Grids Directly into the Solvers

The solvers in FUN3D v11.x now load raw grid formats such as VGRID, FAST, etc. directly. To do this, the user simply adds a namelist called &raw_grid to their fun3d.nml input deck, where the variables and default values are as shown below.

Loading Raw Grids Directly into the v11.0 Solvers

The basic form of the &raw_grid namelist for v11.0 is as follows:

```
&raw_grid
  grid_format  = "vgrid"
  formatted    = .false.
  lump         = 0
  swap_yz_axes = .false.
  twod_mode    = .false.
/
```

The actual project name is still specified as before, in the &project namelist.

The currently supported values of `grid_format` are “vgrid” (traditional VGRID .cogsg/.bc/.mapbc files, both single- and multi-segmented), “vgrid4” (mixed element VGRID .gridu/.mapbc files), “fast” (FAST .fgrid/.mapbc files), “fun2d” (FUN2D .faces files), and “aflr3” (formatted or unformatted .ugrid/.mapbc files).

Other formats traditionally supported by party, such as Fieldview and Plot3D, have not been implemented yet. Users must use the old party paradigm for such grids, or translate them to a supported format. These other formats will be supported in this namelist in future releases of FUN3D.

(Note: The FUN3D team has found that the use of stream IO formats is considerably more efficient in terms of time and memory, particularly for larger grids. We have internal options to convert grids to such formats; however, since none of the common grid generation tools have an established standard format, we do not discuss these options in detail here. If you are having trouble loading large grids using the standard file formats listed above, contact us for recommendations on converting them to a more amenable format for FUN3D.)

The currently supported values of `formatted` are .true. or .false. FUN3D will stop with an error message if this value is inconsistent with the value of `grid_format`. For example, if VGRID files are specified, `formatted` must be .false.

The currently supported values of `lump` match those used by the traditional party preprocessor: 0 for no boundary group lumping, 1 for lumping by physical boundary condition type, or 2 for lumping by family name (if the grid format allows the specification of family names – if it does not, FUN3D will stop with an error message).

The `swap_yz_axes` allows the user to swap the y- and z-axes if desired.

The `twod_mode` input tells FUN3D to process and solve on the input grid in 2D mode. This option is really only relevant for AFLR input grids, since this is the only currently-supported mixed-element format (grids must be a single layer of prisms or hexes in the spanwise direction in order to run them as 2D cases). If `grid_format` is “fun2d”, `twod_mode` need not be set; the solver will automatically switch to 2D mode.

LOADING RAW GRIDS DIRECTLY INTO THE v11.1 (AND HIGHER) SOLVERS

The basic form of the &raw_grid namelist for v11.1 and higher is as follows:

```
&raw_grid
  grid_format   = "vgrid"
  data_format   = "none"   ! must be specified
  patch_lumping = "none"
  swap_yz_axes  = .false.
  twod_mode     = .false.
  fieldview_coordinate_precision = "dummy"  ! must be specified for FV grids
/
```

The actual project name is still specified as before, in the &project namelist.

The currently supported values of `grid_format` are “vgrid” (traditional VGRID .cogsg/.bc/.mapbc files, both single- and multi-segmented), “vgrid4” (mixed element VGRID .gridu/.mapbc files), “fast” (FAST .fgrid/.mapbc files), “fun2d” (FUN2D .faces files), “aflr3” (formatted, unformatted, or C-binary/Fortran-stream .ugrid/.r8.ugrid/.b8.ugrid/.mapbc files), and “fieldview” (formatted or unformatted .fvgrid_fmt/.fvgrid_unf/.mapbc files). Version 11.2 and higher can also read NSU3D grids (.mcell.unf and VGRID-style .mapbc files) by setting `grid_format` to “nsu3d”.

Other formats traditionally supported by party, such as Plot3D and CGNS have not been implemented yet. Users must use the old party paradigm for such grids, or translate them to a supported format. These other formats will be supported in this namelist in future releases of FUN3D.

(Note: The FUN3D team has found that the use of stream IO formats is considerably more efficient in terms of time and memory, particularly for larger grids. AFLR3 is the only tool we are aware of that can provide grids in this format (AFLR3's C-binary .b8.ugrid files). We have internal options to convert other grids to such formats; however, we do not discuss these options in detail here. If you are having trouble loading large grids using the standard file formats listed above, contact us for recommendations on converting them to a more amenable format for FUN3D.)

The currently supported values of `data_format` are "ascii", "unformatted", or "stream" and correspond to formatted, Fortran unformatted, and C-binary/Fortran-stream, respectively. FUN3D will stop with an error message if this value is inconsistent with the value of `grid_format`. For example, if VGRID files are specified, `data_format` must be "unformatted".

The currently supported values of `patch_lumping` are as follows: "none", "BC", and "family". This will group the patches in your raw surface grid in the specified manner (no lumping, lump by physical boundary condition, or lump by family name). Note that as of v11.1, .mapbc files for any of the supported grid formats may contain an optional third column of data, which specifies a family name. (The exception is the VGRID .mapbc file, where the family name is mandatory and appears in the 6th column.) If family names are not present in the .mapbc file, `patch_lumping` must either be "none" or "BC".

The `swap_yz_axes` allows the user to swap the y- and z-axes if desired.

The `twod_mode` input tells FUN3D to process and solve on the input grid in 2D mode. This option is really only relevant for AFLR input grids, since this is the only currently-supported mixed-element format (grids must be a single layer of prisms or hexes in the spanwise direction in order to run them as 2D cases). If `grid_format` is "fun2d", `twod_mode` need not be set; the solver will automatically switch to 2D mode.

The `fieldview_coordinate_precision` input is required for Fieldview meshes and is used to specify the precision of floating-point variables in the grid file. The available values are "single" or "double".

Unsupported Options

In addition to file formats such as NSU3D, Plot3D, and CGNS, there are several options that party must continue to be used for at this time. For example, merging tetrahedral VGRID meshes into prismatic meshes must still be performed using party. The resulting prismatic AFLR3 files can be processed with the new paradigm, but the actual merging must still be performed using party. Mirroring of an input grid is available directly in the solver as of v11.1, but must be performed with party for earlier releases. Please contact the support team with questions about unsupported options.

Restart Information and Parallel IO

The restart information generated at the end of a solution process is now stored in a single file, [project].flow. The default is to read and write this data in a stream format. However, if the user is running on hardware with a parallel file system (e.g., a Lustre file system), more efficient IO may be achieved for large-scale cases (thousands of cores) by taking advantage of new parallel IO options in FUN3D. These rely on the MPI-IO layer included in the MPI-2 standard. Users may engage this IO mode in FUN3D by running the solver with the command line option `--lmpi_io 1`. Restart files generated using FUN3D v10.8 may still be used with the command line option `--lmpi_io 0`, but again, the user is encouraged to move to the new paradigm.

5.2. Sequential Grid Processing

Prior to running the FUN3D solver, you will need to process your grid to get it in the FUN3D native format. This also includes domain decomposition for parallel processing if you have a multiprocessor system available. If you will not be running on more than one processor, you will (must) simply partition your grid into a single partition. A wide range of input grid formats are currently available—

see the `party` main menu for guidance. Contact FUN3D Support for assistance in converting other formats.

The tool that handles this grid conversion process is called Party. This utility also allows the user to reconstruct global solution files from a partitioned domain. A rule-of-thumb for party is 1M **nodes** take about 1.5GB memory. If you have that much memory on your machine and encounter a problem, then check the `limit` command for Csh (or `ulimit` for Bourne shell) and ensure that `datasize`, `stacksize`, and `memoryuse` are `unlimited` (or the largest your system will permit).

```
csh: limit stacksize unlimited
bsh: ulimit -s unlimited -d unlimited -m unlimited
```

Processing a Grid

To process your grid, simply run party and follow the menu prompts. You will need to have your grid in one of the supported grid formats indicated in the first block of the party main menu. (The grid formats are described in the Input Files section below.) The first set of options on the main menu is geared towards preprocessing a grid. The second set is for repartitioning an existing solution/grid. The final set is for postprocessing a solution generated by FUN3D.

FUN3D now supports 2D computations. If the grid is a FUN2D grid (see Input Files below), 2D is automatically enabled, without any additional user input. Depending on cell type (e.g. tetrahedra cannot be run as 2D), other grid formats must be forced into 2D mode by running Party with the `--twod` command line option. Note that except for FUN2D grids, the input grid must be a 3D grid, one cell wide. FUN2D grids are true 2D grids. When party processes a grid as 2D (either automatically for FUN2D meshes or with the `--twod` option for other grid types), it creates additional data in the part files that will cause the flow solver, when run, to automatically go into "2D mode", resulting in considerably faster execution than the standard 3D mode. FUN3D operating in 2D mode is roughly 1.5 to 2 times slower than FUN2D (which is no longer supported). Flow solver output for FUN2D meshes, when post-processed with the Party utility, will reside on the extruded prismatic grid, rather than the 2D triangular grid in the `[project].faces` file.

You will probably want the table of boundary conditions below available. When processing your grid, use either the indices in the first column or the second column. When you get your results back, the forces that are summarized in the `[project].forces` file will be labeled using the notation in the second column. This is done so that existing boundary condition flags may be used, while still allowing for 4-digit boundary conditions in the solver, which enable room for future growth. *Note that if you want to change a boundary condition, you must pre-process the grid over again. The BC indices are hardwired into the partition files*. This is on our to-do list to change.

Implicit Lines

The implicit lines are processed in Party with the command line —partition_lines. A formatted file with the name `[project].lines_fmt` is read; this file contains the definitions of lines emanating from viscous boundary nodes as a list of node numbers. Currently, every viscous boundary node must have an associated implicit line. A typical file, from `GnuTestCase/Grids/cylinder.lines_fmt`, is shown below. Anything beyond column 24 is strictly for information only.

```
          289        4913 Total lines and points
           17          17 Min and max points in line
           17             Points in line for line=          1
            1             Node of 1st pt in 1st line x/y/z=  0.0   0.0 -0.5
           18
           35
           52
           69
           86
          103
          120
          137
          154
          171
          188
          205
          222
          239
          256
          273             Node of last pt in 1st line x/y/z=  0.0  0.0 0.5
...
...(remaining 288 sets of node numbers defining each line)
...
```

Three files are generated by Party:

1. `[project]_line_surface_tec.dat`
2. `[project]_lines_tecplotlines.dat`
3. `[project].lines_fmt`

The first two are Tecplot visualization files. The first file is the surface formed by the top of the implicit lines, with zones corresponding to the viscous boundary subset of the file `[project]_geom.tec`. The second file is a Tecplot line file containing the individual lines (not organized by boundaries). The third file is an unformatted file used by the flow solver.

Boundary Conditions

Boundary condition listing and input formatting is described in the previous section.

Input Files

FAST Formatted Grids

`[project].fgrid`

This file contains the complete grid stored in ASCII FAST format.

`[project].mapbc`

The first line in this file is the number of boundary groups contained in your `[project].fgrid` file. Each subsequent line in this file contains the boundary number and the type for each.

VGRID Formatted Grids

`[project].cogsg`

This file contains the complete grid stored in unformatted VGRID format. Please note that VGRID cogsg files are *always* big endian no matter what type of machine is used in grid generation. If you are running FUN3D on a native little endian machine you will have to use a compiler option or a runtime environment flag to set the default file type to big endian.

`[project].bc`

This file contains the boundary information for the grid, as well as a flag for each boundary face.

`[project].mapbc`

For each boundary flag used in `[project].bc`, this file contains the boundary type information. The boundary types are as shown in the preceding table.

FELISA Formatted Grids

`[project].gri`

This file contains the complete grid stored in formatted FELISA format.

`[project].bco`

This file contains a flag for each boundary face. If original FELISA bc flags (1, 2, or 3) are used, they are translated to the corresponding FUN3D 4-digit bc flag. Alternatively, FUN3D 4-digit bcs can be assigned directly in this file.

`[project].fro`

This file contains the surface mesh nodes and connectivities and associated boundary face tags for each surface triangle. This file can contain additional surface normal or tangent information (as output from GridEx or SURFACE mesh generation tools), but the additional data is not read or utilized by FUN3D.

FUN2D Formatted Grids

`[project].faces`

This file contains the complete grid stored in formatted FUN2D format (triangles). These meshes automatically direct FUN3D to operate in 2D mode, with no other user flags. Party will extrude the triangles into prisms in the y-direction. Output from the flow solver will be on a one-cell wide prismatic mesh. Boundary conditions are contained in the FUN2D grid file. The first time party is run, it will output a `[project].mapbc` file that contains these original boundary conditions. If you wish to change the boundary conditions from those in the `[project].faces` file, simply change them in `[project].mapbc` and rerun Party. Party will use the boundary conditions in the `[project].mapbc` file rather than the `[project].faces` file. If you wish to revert to the boundary conditions in the `[project].faces` file, you can remove the `[project].mapbc` and rerun Party.

FieldView Formatted (ASCII) Grids

`[project].fvgrid_fmt`

This file contains the complete grid stored in ASCII FieldView FV-UNS format.

Supported FV-UNS file versions: 2.4, 2.5 and 3.0 (3.0 only in FUN3D v10.7 and higher). With FV-UNS 3.0, the support is for the grid file in split format; the combined grid/results format is not supported. FUN3D does not support the Arbitrary Polyhedron elements of the FV-UNS 3.0 standard. For ASCII FV-UNS 3.0, the standard allows comment lines (line starting with !) anywhere in the file. FUN3D only allows comments immediately after line 1. Only one Grids section is allowed.

`[project].mapbc`

This file contains the boundary information for the grid. The first line is an integer corresponding to the number of boundaries. Subsequent lines, one for each boundary, contain two integers. The first is the boundary number and the second if the boundary type. The boundary types are as shown in the preceding table. If the `[project].mapbc` does not exist the first time party is run, it will create one with some dummy data based on the boundary data in the grid file. This template file can be edited to set the desired boundary conditions, and Party must be rerun to set these boundary conditions in the part files.

FieldView Unformatted Grids

`[project].fvgrid_unf`

This file contains the complete grid stored in unformatted FieldView FV-UNS format.

Supported FV-UNS file versions: 2.4, 2.5 and 3.0 (3.0 only in FUN3D v10.7 and higher). With FV-UNS 3.0, the support is for the grid file in split format; the combined grid/results format is not supported. FUN3D does not support the Arbitrary Polyhedron elements of the FV-UNS 3.0 standard. Only one Grids section is allowed.

Versions of FUN3D prior to v10.7 allowed only double-precision unformatted files; v10.7 and higher queries the user as to whether the file is single or double precision.

`[project].mapbc`

Same as above for FieldView formatted grids.

AFLR3 Formatted Grids

`[project].ugrid`

This file contains the complete grid stored in formatted AFLR3 ugrid format.

`[project].mapbc`

Same as above for FieldView formatted grids.

NSU3D Unformatted Grids

`[project].mcell.unf`

This file contains the complete grid stored in unformatted NSU3D mcell format.

`[project].mapbc`

Same as above for FieldView formatted grids.

Pre-existing Grids/Solutions

`[project].msh`

This file contains the entire grid.

`[project]_part.n` (Optional)

These files (n=1, # of partitions) contain the grid information for each of the partitions in the domain. Sequential flow solves require one part file.

`[project]_flow.n` (Optional)

These files (n=1, # of partitions) contain the binary restart information for each grid partition. They are read by the solver for restart computations, as well as by party for solution reconstruction and plotting purposes. If you are going to read these in, the `[project]_part.n` files are required.

PLOT3D Grids

Party can read PLOT3D structured grids, and use (as hexahedral unstructured grids). The grid must be multiblock and 3-D (no iblanking) in the form:

`[project].p3d`

This file can be either formatted or unformatted. Only one-to-one connectivity is allowed with this option (no patching or overset). The grid should contain no singular (degenerate) lines or points. An additional “neutral map file” is also required:

`[project].nmf`

This file gives boundary conditions and connectivity information. The make-up of the `.nmf` file is described at geolab.larc.nasa.gov/Volume/Doc/nmf.htm

Note that for usage with FUN3D's Party software, the "Type" name in the `.nmf` file must correspond with one of FUN3D's BC types, plus it allows the Type `one-to-one`. If party does not recognize the Type, you will get errors like:

```
This may be an invalid BC index.
At a minimum, it is unknown within the module bcs_deprc.
Your data has a BC index unknown to function bc_name.
```

An example `.nmf` file is shown here for a simple 1-zone airfoil C-grid (5×257x129), with 6 exterior boundary conditions and 1 one-to-one patch (in the wake where the C-grid attaches to itself):

```
# ===== Neutral Map File generated by the V2k software of NASA Langley's GEOLA
# ============================================================================
# Block#   IDIM   JDIM   KDIM
# ----------------------------------------------------------------------------
      1

      1      5    257    129

# ============================================================================
# Type            B1   F1   S1   E1   S2   E2   B2  F2   S1   E1   S2  E2  Swap
# ----------------------------------------------------------------------------
'tangency'         1    3    1  257    1  129
'tangency'         1    4    1  257    1  129
'farfield_extr'    1    5    1  129    1    5
'farfield_extr'    1    6    1  129    1    5
'one-to-one'       1    1    1    5    1   41    1   1    1    5  257 217 false
'viscous_solid'    1    1    1    5   41  217
'farfield_riem'    1    2    1    5    1  257
```

CGNS Grids

Party has the capability to read CGNS grid files as well as write CGNS grid/solution files. The user must download the CGNS library from www.cgns.org , install it, and compile the code linked appropriately to it (i.e., `--with-CGNS=location-of-cgnslib`). You must link to CGNS Version 2.5.3 or later. If the code is not linked to the CGNS library, then Party will not allow the option of reading/writing a CGNS file. For grid input, any CGNS file to be read must be Unstructured type. The CGNS file should be named:

`[project].cgns`

The following CGNS mixed element types are supported: PENTA_6 (prisms), HEX_8 (hexes), TETRA_4 (tets), and PYRA_5 (pyramids). The CGNS BCs currently allowable are:

- 'BCWall', 'BCWallViscousHeatFlux', 'BCWallViscous', 'BCWallViscousIsothermal'—currently mapped to `viscous_solid`
- 'BCOutflow', 'BCOutflowSupersonic', 'BCExtrapolate'—currently mapped to `farfield_extr`
- 'BCOutflowSubsonic', 'BCTunnelOutflow'0—currently mapped to `farfield_pbck`
- 'BCInflow', 'BCInflowSubsonic', 'BCInflowSupersonic', 'BCFarfield'—currently mapped to `farfield_riem`
- 'BCTunnelInflow'—currently mapped to `subsonic_inflow_pt`
- 'BCSymmetryPlane'—currently mapped to `symmetry_x`, `symmetry_y`, or `symmetry_z`
- 'BCWallInviscid'—currently mapped to `tangency`

There are some additional limitations currently:

Under BC_t, Party reads the BC names, but nothing else that might be stored in the CGNS file like BCDataSet or BCData. I.e., we are only using the top-level CGNS naming convention as a "guide" to establishing the appropriate corresponding BCs in FUN3D. For the most part, this should be OK.

Currently it is required that the CGNS file include Elements_t nodes for all boundary faces (using type QUAD_4 or TRI_3), otherwise the code cannot recognize where boundaries are present (because it currently identifies boundaries via 2-D element types).

The best approach for the CGNS file (which is also common practice among most unstructured-type CGNS users) is not only to include all volume elements, but also to include all surface elements under separate nodes (of type TRI_3 or QUAD_4). It is also helpful to have SEPARATE ELEMENT NODES for each boundary element of a given BC type. This way, it is very easy to read and use the file. For example, using Tecplot, which can read CGNS files, one can easily distinguish the various parts (body vs. symmetry vs. farfield, if they all happen to be TRI_3's for example, as long as each part has its own node).

Under BC_t, Party requires that either ElementList / ElementRange be used, or else PointList / PointRange with GridLocation=CellCenter, to refer to the appropriate corresponding boundary elements.

Note that if the CGNS file is missing BCs (no BC_t node), Party still tries to construct the BCs based on the boundary face Elements_t information. If these boundary element nodes have been named using a recognizable BC name, then Party will try to set each BC accordingly. If the name is not recognized, you will see messages like "This may be an invalid BC index". Always check the `.mapbc` file after Party has run, to make sure that the BCs have all been interpreted and set correctly. (If not, you should edit the `.mapbc` file then re-run Party, and it will use the new corrected values in the `.mapbc` file. Party always overrides the BC info with whatever is in the `.mapbc` file when it is present!)

Output Files

Preprocessing Mode

`[project]_part.[n]`

These files (where `n` goes from 1 to the number of partitions) contain the partitioned grid in FUN3D v3 format. They also include the boundary condition information, used by the flow solver by default. If you want to override these BCs, however, it can be done through the use of a `[project].mapbc.override` file. The format of this file is the same as the `[project].mapbc` file.

`[project].msh`

This file contains the entire grid.

`[project].pr`

This file is generated when processing a FAST- or VGRID-formatted grid file. It contains some basic geometric statistics on the mesh.

`[project]_geom.tec`

This file contains the surface grid information as processed from a FAST- or VGRID-formatted input grid. This file is in ASCII Tecplot format.

`[project]_angles.dat`

This file contains some checks for big angles in the grid.

`[project]_mesh_info`

This file contains most of the grid-related screen output generated during the party run, e.g., number of cells, nodes, element types, cell volumes, and so forth.

`[project]_part_info`

This file contains detailed information about the mesh partitions, e.g., number of cells, nodes, boundary faces, boundary conditions, and so on.

Postprocessing Mode

For FUN3D Version 10.7 and higher, see Flow Visualization Output Directly From Flow Solver for an alternate to the postprocessing mode of Party for generating solution data for visualization.

When postprocessing using Party, the user has many choices for output, some of which are listed here.

`[project]_soln.tec`

This file contains the reconstructed surface mesh, along with flow variable information for plotting. This file is in ASCII Tecplot format.

`[project].fvuns`

This file contains the entire reconstructed mesh, along with flow variable information for plotting. This file is in ASCII FieldView format.

`[project]_grid.fgrid`

This file contains the entire reconstructed mesh for plotting. This file is in ASCII FAST format. (Note that the node ordering in this file will be different from the original FAST file you may have provided, since the grid is renumbered for bandwidth minimization.)

`[project]_soln.fgrid`

This file contains flow variable information for the entire reconstructed mesh for plotting. This file is in ASCII FAST format.

`[project]_vol.cgns`

This file contains the entire reconstructed mesh, including boundary elements and most typical boundary conditions (It uses UserDefined if the BC cannot be figured out.), along with flow variable information for plotting. This file is CGNS format and must be read with CGNS-compatible software. Note that if the code is not linked to the CGNS software library during compilation, then Party will not give the option to output a CGNS file.

Post-Processing/Repartitioning Moving Grid Cases

To post-process (or repartition) moving grid cases using party, you must use the command line option `--moving_grid`.

5.3. Parallel Grid Processing

Note: The traditional pParty preprocessor is obsolete as of FUN3D v11.0, as all of its functionality is now contained in the front end of the solvers. The instructions that follow are only relevant to earlier versions of FUN3D.

Note: The standard sequential implementation of party is generally sufficient for most users. However, for extremely large grids and/or hardware with very limited memory, a distributed version of party is now available. Not all of the features of party are available yet, but the basic capabilities are all there.

The processing of parallel party (pParty), from a user’s perspective, works essentially the same as Party. The following section describes the *high-level differences* between using Party and pParty. Please read the section on Party before reading this section. Note, the use of pParty assumes the user knows how to execute an MPI application in the user’s environment.

The parallel version of Party (pParty) initially reads and distributes the grid in parallel, thereby reducing the initial and overall memory and computational requirements. Then pParty internally calls a parallel partitioning tool (ParMETIS) to compute an efficient partition vector used to distribute the memory and computational requirements over a set of processors for the remaining steps (and bulk) of the preprocessing.

pParty can be executed using interactive prompts, input either interactively or redirected from an input file. Additionally, a series of command line options are available, type `pparty --help` for a list.

For pParty, all the inputs can be provided by command line options and is often the recommended approach for pParty as various MPI environments often handle standard input differently. For example, to pre-process a FAST mesh with a project name of `om6viscous` and not group common boundary condition types:

```
mpirun -np 4 -machinefile machines pparty \
             --pparty_iwhere 1 \
             --pparty_project om6viscous \
             --pparty_ilump 0 \
             --partition_lines \
             --no_renum
```

where `-np 4` informs MPI of the number of processors while `-machinefile` specifies that the file `machines` should be used to determine on which machines the job is run. Information required by pParty consists of `--pparty_iwhere`, which is the grid partitioning option, `--pparty_project`, which is the project name, and `--pparty_ilump`, which is the boundary grouping option. Optional information passed to pParty in this case consists of `--partition_lines`, which keeps implicit lines intact during partitioning, and `--no_renum`, which turns off Cuthill-Mckee renumbering.

Preprocessing

For parallel party, the **number of partitions** created is the same as the **number of processors** that the user specified in the parallel job. Thus, pParty does not prompt the user, nor accepts input for the number of partitions through standard input. See the Concluding Remarks section below on how to create more partitions than actual (underlying) processors.

Currently, the following are the major **preprocessing constructs not supported** by pParty:

- 2D computations
- mixed elements
- multigrid
- Cuthill-Mckee renumbering (currently, an identity vector)
- FIELDVIEW, AFLR3, NSU3D meshes
- Boundary grouping is not supported with partition_lines
- does not create (nor subsequently need) a mesh file

Repartitioning

Preprocessing in Party typically creates a mesh file, which can be used in repartitioning. The mesh file is typically large and has been eliminated in pParty. The pParty preprocessor *does not write mesh files*.

To repartition in pParty, the user starts with an existing set of partition files (either created by Party or pParty) and a set of solution files (`[project].flow`). The user *renames the original partition* files to `[project]_orig_part.n`. Then *reruns the preprocessor* using the desired number of partitions on which to map the solution. The new files will be `[project]_part.n`.

Finally, pParty is called for repartitioning. pParty reads the original partition files and solution files, extracts the grid information, reads the new partition files, and creates the repartitioned solution files. As a result, the new solution files (`[project]_flow.n`) and new partition files (`[project]_part.n`) will be available.

Example: Repartitioning from 2 to 4 processors

Input:

- `[project]_orig_part.(1,2)` (renamed by user with old partition files)
- `[project]_flow.(1,2)`
- `[project]_part.(1-4)` (created by second call to preprocessor)
- `../Adjoint/[project]_adj.(1,2)` (if adjoint is to be repartitioned)

Output:

- `[project]_flow.(1-4)` (overwrites old solution files)
- `../Adjoint/[project]_adj.(1-4)` (overwrites old adjoint files in ../Adjoint if adjoint is requested to be partitioned)

```
  mpirun -np 4 -machinefile machines pparty < inp_init_24
```

where `inp_init_24` is

```
   11
   om6viscous
   1
   2
   4
```

Note: a user can repartition the solution from a larger number of partitions to a smaller number; or from a smaller number to a larger number. But, the *number of processors specified to MPI for repartitioning must be the larger of the two numbers* (i.e., the larger of the original partitions or number of new partitions). Both restart and adjoint files can be repartitioned. A command line option, `--pparty_repart_adjoint`, is used to turn on repartitioning of adjoint files. The command line option, `--pparty_repart_restart`, is used to turn on repartitioning of restart files.

An example session of repartitioning.

Repartitioning assumes that the user has existing partition and solution files; additionally, restart and/or adjoint files may also be available.

Create the original (2) partition file

```
 mpirun -np 2 ./pparty
   1
   om6viscous
   0

   ==> creates om6viscous_part.1 and om6viscous_part.2
```

Create the flow files

```
    mpirun -np 2 ./nodet_mpi

       <== uses @fun3d.nml@ (or @ginput.faces@ prior to release 10.5.0), om6vi
       ==> creates om6viscous_flow.1 and om6viscous_flow.2
```

Step 1. Rename the original partitions files

```
    mv om6viscous_part.1 om6viscous_orig_part.1
    mv om6viscous_part.2 om6viscous_orig_part.2
```

Step 2. Create the new partition files

```
    mpirun -np 4 ./pparty
       1
       om6viscous
       0
```

```
        ==> creates om6viscous_part.(1-4)
```

Step 3. Create the new flow files from the previous flow files.

Based on the previous steps, the following files should be available in the current directory. (If adjoint is to be repartitioned, then the om6viscous_adj.(1-2) should already exist in the ../Adjoint directory.)

```
   om6viscous_part.(1-4), om6viscous_orig_part.(1-2),  and
   om6viscous_flow.(1-2)
```

```
 mpirun -np 4 ./pparty

 Enter your selection:
   11

 Enter project name for input file
    om6viscous

 Do you have restart files that you would like to repartition? (yes=1 no=0)
    1

 Do you also have some adjoint files that you want converted? (yes=1 no=0)
    0

 How many partitions does your mesh have?
     (For mesh movement cases only: use a negative number
     to read new x, y, z coordinates from partition files.)
    2

   --- user input requested ---
  How many partitions do you want?
    4

 Reading om6viscous_flow.* (version           3 )
 Writing om6viscous_flow.* (version           3 )

   ==> om6viscous_flow.(1-4)
```

Repartitioning Adjoint files

The partitioning of adjoint works in the same manner. Starting from step 3 in the previous example, enter a “1” (yes) to have some adjoint files to be converted. The input for the Adjoint will be read from ../Adjoint and overwritten in the same directory (../Adjoint).

Note, if the user enters a “0” (no) for restart files to not be repartitioned, the question for the adjoint files will not be presented. If one wishes to repartition adjoint files only, then use the “—pparty_repart_adjoint 1” command line option, which will force the adjoint to be repartitioned even if the restart files are not.

Postprocessing

All of the non-experimental options (except global residual files) are available and work identically to Party.

For FUN3D Version 10.7 and higher, see Flow Visualization Output Directly From Flow Solver for an alternate to the postprocessing mode of pParty for generating solution data for visualization.

Summary of pParty Command Line Options

Option	Description	Argument	Party	pParty
`--party_nomsh`	Turn off writing `.msh`; use partition files instead	–	Yes	Yes
`--party_partonly`	Stop after writing the partition file	–	Yes	Yes
`--pparty_iwhere`	Grid Partitioning	integer	No	Yes

`--pparty_project`	Project Name	string	No	Yes
`--pparty_ilump`	Boundary Grouping	integer	No	Yes
`--pparty_outformat`	Turn on ASCII output of partition files	–	No	Yes
`--pparty_metisin`	Turn on the reading of metis partition file	–	No	Yes
`--pparty_metisout`	Turn on the writing of metis partition file	–	No	Yes
`--pparty_repart_resize_from`	Number of partitions to repartition from	integer	No	Yes
`--pparty_repart_resize_to`	Number of partitions to repartition to	integer	No	Yes
`--interleaf`	Write parallel partitioned file in groups of interleaf factor	integer	No	Yes

Concluding remarks

Currently, the preprocessing of pParty creates the same number of partitions as the user specified processors. If a user wants to create more partitions than processors, then the user should run multiple instances of pParty on the same processor. Thus, the number passed to `-np` will be the same as the number of partitions, but the actual number of underlying processors will be smaller. For more details, please refer to the MPI documentation for the target machine.

The `--interleaf` command line option controls the number of files being written out at once. The default is to write all files out at once. This may swamp a file system, thus a user can control the number of files written out concurrently. For instance, `--interleaf 2` will write only 2 files at the same time.

Previous (C4: Boundary Conditions) | Up | Next (C6: Analysis)