NASA/TM—2010-216739

Preprocessor that Enables the Use of GridPro™ Grids for Unsteady Reynolds-Averaged Navier-Stokes Code TURBO

Vikram Shyam
Glenn Research Center, Cleveland, Ohio

June 2010

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

- E-mail your question via the Internet to *help@ sti.nasa.gov*

- Fax your question to the NASA STI Help Desk at 443–757–5803

- Telephone the NASA STI Help Desk at 443–757–5802

- Write to:
  NASA Center for AeroSpace Information (CASI)
  7115 Standard Drive
  Hanover, MD 21076–1320
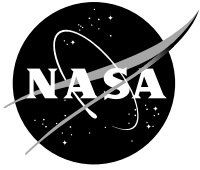
Preprocessor that Enables the Use of GridPro™ Grids for Unsteady Reynolds-Averaged Navier-Stokes Code TURBO

Vikram Shyam
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

June 2010

Acknowledgments

Preprocessor that Enables the Use of GridPro^{TM} Grids for Unsteady Reynolds-Averaged Navier-Stokes Code TURBO

Vikram Shyam
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

A preprocessor for the Computational Fluid Dynamics (CFD) code TURBO has been developed and tested. The preprocessor converts grids produced by GridPro (Program Development Company (PDC)) into a format readable by TURBO and generates the necessary input files associated with the grid. The preprocessor also generates information that enables the user to decide how to allocate the computational load in a multiple block per processor scenario.

Introduction

The Computational Fluid Dynamics (CFD) code TURBO (Refs. 1 to 3) has traditionally been used to simulate flows in axial compressors. The computational domains are traditionally discretized using a grid generator that produces H-grids. For geometries such as those in a high pressure stage of a turbine, rotors usually have large turning angles (Ref. 4). For such geometries, better grid quality is achieved by generating O-H type grids instead of H-grids (Ref. 5). In order to generate O-H grids it was decided to utilize the grid generation software GridPro. This would provide more control over the grid quality. In order to facilitate the use of grids generated by GridPro for use in TURBO, a preprocessor was created using the programming language FORTRAN (Appendix A contains a complete program listing.)

The grid generation software GridPro generates unstructured multiblock grids (the grid within each block is structured but the block layout is unstructured) (Ref. 6). The computational coordinates (i, j, k) of the blocks are not ordered according to the specifications required by TURBO (Refs. 5, 7, and 10). This introduces the need for a preprocessor. Moreover, the boundary conditions and connectivity files generated by GridPro need to be converted to formats that are amenable to TURBO.

Starting with a GridPro grid and connectivity file, the preprocessor accepts user inputs that detail boundary conditions and blade row information to produce input files (Ref. 7) that can be utilized to run TURBO. Although there are instances in which manual intervention is required (for example, when opposing faces in a block do not follow the same physical coordinate direction), the procedure is, to a great extent, automated.

Usage

This section defines the manner in which one may utilize a grid generated by GridPro for the purpose of simulating a flow using TURBO. Once a grid is generated in GridPro using a suitable topology (Ref. 6) and by assigning the desired boundary conditions to the geometric surfaces, a file with the extension '.conn' is generated that is associated with the grid. This file contains the connectivity information required to link the blocks together. For the purposes of illustration, assume that the grid file is named 'grid.tmp' and the '.conn' file is named 'grid.tmp.conn'. Using the GridPro command 'mrgb' (see Ref. 6) the '.conn' file is used to create a file with extension '.conn_n'. This file contains both the connectivity and boundary conditions required to

completely define the computational domain. For example, typing the command 'mrgb grid.tmp -maxb 1' in a terminal will produce the file 'grid.tmp.tmp.conn_n' and a grid file 'grid.tmp.tmp' that is identical to grid.tmp. The parameter '-maxb' determines how many blocks of the original grid, 'grid.tmp', are to be merged to form the new grid, 'grid.tmp.tmp'. In the above example no merging takes place. The preprocessor uses the merged grid and '.conn_n' file along with input files to generate 'GU' files (grid files formatted for use with TURBO), 'input00', 'bc.in', 'dmap.in' and 'turbo.in' (these files are required as input for TURBO). For details regarding these files and their formats refer to Reference 7. First, the GridPro grid is converted to plot3d (Ref. 8) format. In this format it is easier to verify connectivity information and the grid can also be viewed in postprocessors such as FIELDVIEW (Intelligent Light). Once the connectivity information is verified, the plot3d file is converted to GU files (one GU file for every block.) The 'conn_n' file is used to create the TURBO boundary condition file, 'bc.in' and connectivity file, 'dmap.in'. The preprocessor can operate on multiple blade rows and is therefore capable of processing grids for unsteady simulations. If the simulation involves multiple blade rows, a 'turbo.in' file is generated that contains information on the sliding interface locations. In order to conform to the boundary condition specifications of TURBO, the preprocessor checks blocks for the orientation of their computational coordinates and reorients them to satisfy the specifications. If it is unable to determine the correct block orientation a list of such blocks is printed out so that the user may manually inspect the blocks. If a manual inspection is required a separate utility called 'reorient.f' may be utilized to reorient the blocks in question. The reorienting operations are accompanied by suitable modifications to the boundary condition and connectivity files. In the event that a user would wish to run multiple blocks on a single processor, various schedule files are generated. These contain various groupings of blocks to allow the user to determine the most efficient way to run the simulation. Figure 1 shows a flowchart of the process at a high level.

Method for Reorienting Blocks

In order to determine whether a block requires reorientation the preprocessor cycles through the boundary conditions file and creates an array containing the block numbers of blocks that have one or more boundary condition. The three computational coordinate directions are assigned indices (i, j, and k). These are related to the physical coordinates x, y, and z through a coordinate transformation (Ref. 5). TURBO requires that an inlet be on the minimum i-index, i_{min}. If a block contains an inlet boundary condition, the preprocessor attempts to determine the axial direction and reorients the block such that the inlet is on an i_{min} face. It accomplishes this by searching for the direction of increasing x-coordinate. A similar procedure is used for a block containing an exit. The block containing an exit boundary condition is reoriented so that the exit lies on an i_{max} (maximum i-index value) face. Next, the preprocessor looks for periodic faces and assigns faces with a 'ref_periodic_fwd' (see Ref. 7) boundary condition to a k_{max} face and faces with a ref_periodic_bak boundary condition to a k_{min} face. Blocks that have already been operated on to align inlets and exits are manipulated in a way that ensures the inlet and exit faces are not changed. The preprocessor then attempts to determine the radial direction within every block in the grid that contains at least one no slip boundary condition and that has not been operated on before. If it is found that the extremities of a particular computational coordinate correspond to the minimum and maximum average radii within a block, the block is reoriented so that the face with the minimum radius is a j_{min} face and the face with the maximum average radius is a j_{max} face.

Once the GU files for a multiple blade row case are obtained it might be necessary to match the radial lines at the interface of the multiple rows. These are both sliding interfaces. According to TURBO specifications the radial lines at this interface must match. A simple interpolation may be performed across the interface to match the grid lines in the radial direction (see Ref. 9).

```
                         ┌─────────────┐
                         │    Start    │
                         └──────┬──────┘
                                │
                                ▼
                    ┌───────────────────────┐
                    │ Read input files       │
                    │ setup.in, gplist.in    │
                    └───────────┬───────────┘
                                │
                                ▼
                    ┌───────────────────────┐
                    │ Convert GridPro grid   │
                    │ to plot3d format       │
                    └───────────┬───────────┘
                                │
                                ▼
                    ┌───────────────────────┐
                    │ Convert GridPro        │
                    │ connectivity and       │
                    │ boundary files to      │
                    │ bc.r, dmap.r           │
                    │ (where r is the blade  │
                    │ row number)            │
                    └───────────┬───────────┘
                                │
                                ▼
┌───────────────────────┐   Yes  ◇
│ Check grid connectivity│◄──── If connectivity to
│ and display results    │      be verified
└───────────┬───────────┘      ◇
            │                   │ No
            │                   ▼
            │          ┌───────────────────────┐
            └─────────►│ Generate GU grid files │
                       │ for TURBO              │
                       └───────────┬───────────┘
                                   │
                                   ▼
┌───────────────────────┐  Yes   ◇
│ Reorient blocks for    │◄──── Reorient blocks
│ TURBO                  │      automatically?
└───────────┬───────────┘      ◇
            │                   │ No
            │                   ▼
            │          ┌───────────────────────┐     ┌─────────┐
            └─────────►│ Group blocks for       │────►│  Stop   │
                       │ multiblock per         │     └─────────┘
                       │ processor simulation   │
                       └───────────────────────┘
```
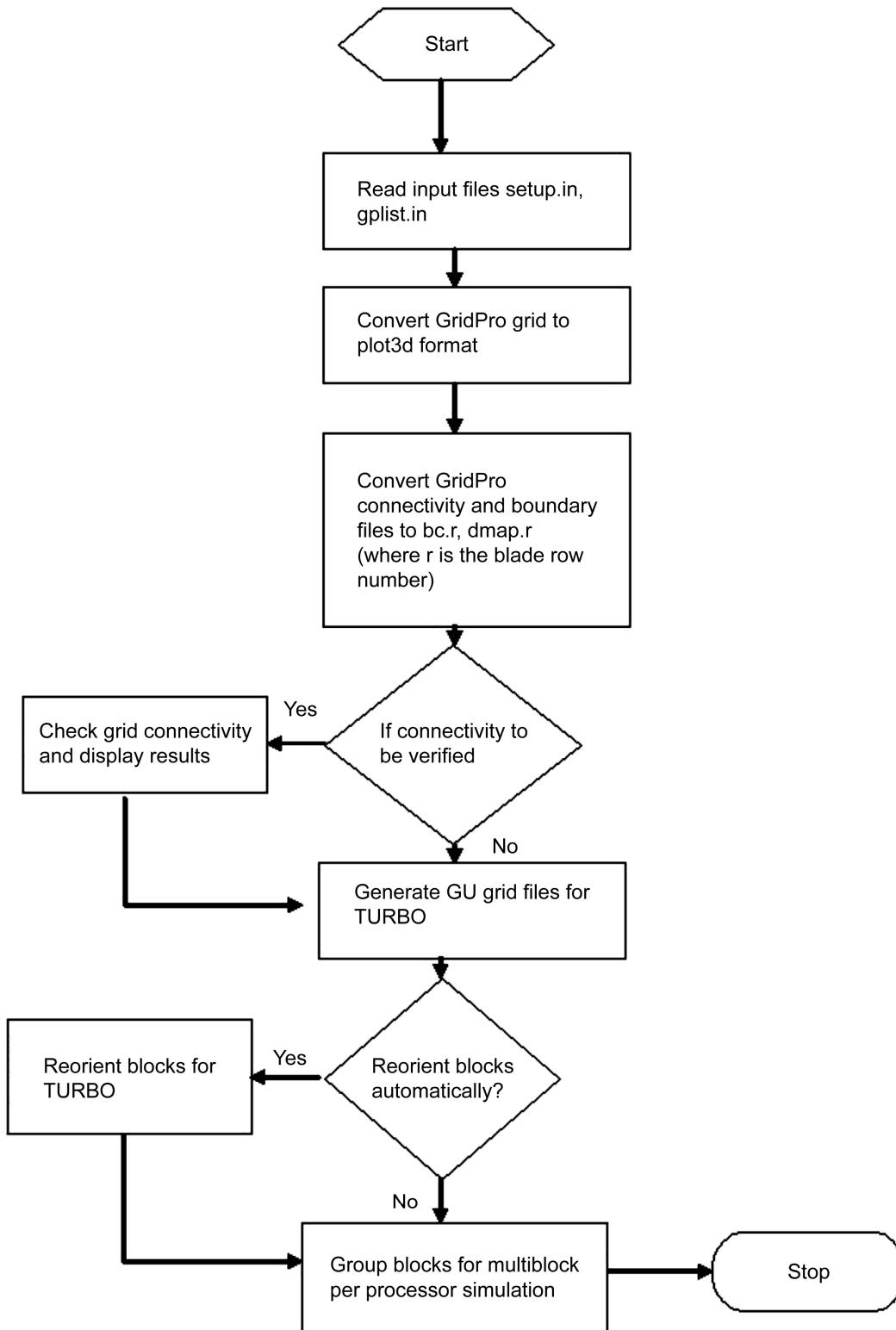
Figure 1.—Flowchart of major processes in preprocessor.

Input Files

In order to use the preprocessor, two input files are required. The first is named 'setup.in'. It contains a list of parameters that specify the input grid files for each blade row (row_names), the number of blades per blade row (num_blades), whether connectivity information should be verified (checkconn), tolerance to use for connectivity verification (conn_tol) and whether or not the preprocessor should attempt to reorient the blocks to satisfy TURBO specifications (turbo_friendly). Table 1 lists the variables and their possible values and formats that are specified in the namelist (Ref. 9) of 'setup.in'. The second input file required to run the preprocessor is a file containing a mapping between GridPro and TURBO boundary conditions. The file is named 'gplist.in'. A sample file is shown in Figure 2. The variables to the left of the '=' are formed by adding the prefix 'g' to a TURBO boundary condition name. The value to the right of the '=' refers to the number assigned to the boundary condition in GridPro. Boundary conditions that are not used in a simulation are assigned the value '999'.

TABLE 1.—CONTENTS OF PREPROCESSOR INPUT FILE SETUP.IN

Namelist	SETUP_PARAMS
variable names	Allowable values
num_blade_rows	Integer value indicating number of blade rows to be processed
num_blades	Integer array of blade counts for each blade row (int1 int2 … or int1, int2, …)
checkconn	0 or 1 (no connectivity checking or connectivity checking)
conn_tol	Real number indicating tolerance to use while checking connectivity
turbo_friendly	.TRUE. Or .FALSE. (reorient blocks to meet TURBO criteria or not)
row_names	List of grid file names for each blade row (grd1.tmp grd2.tmp …)
	Assumes that corresponding connectivity files are named grd1.tmp.conn_n etc.

```
&GP_PROPS
gslip=4
gno_slip=2
gno_slip_iso=8
grad_eq_exit=6
gperiodic=999
gpressure_exit=999
gplenum_in=999
gref_clearance=999
gts=3
gcvbc_in=7
gisentropic_in=5
gwb_steady_in=999
gwb_unsteady_in=999
gwb_steady_exit=999
gwb_unsteady_exit=999
gcvbc_sub_exit=999
gcvbc_super_exit=999
gslide=999
gslide_ts_i=10
gslide_ts_j=999
ginter_blk=1
/
&This file contains the property conversion list
&values to the left are TURBO BC names
&values to the right are gridpro values that have been assigned to boundaries
&properties that are not used are assigned 999.
&You only need to sepcify periodic or ts and NOT ref_preiodic or ref_ts
&The converter figures out whether to use ref_ and fwd or bak directions
```

Figure 2.—Contents of input file gplist.in

Examples

This section shows the usage of the preprocessor through two examples.

Example 1: Flat plate with film cooling hole

The geometry for this exercise is shown in Figure 3. Flow enters the domain from the left (minimum x face) and exits through the right (maximum x). There is an additional inlet at the minimum y face (plenum inlet). The grey inlet patch belongs to block 8 of the 19-block grid. The blue inlet patch belongs to block 9. The grey exit patch belongs to block 11 while the red exit patch belongs to block 12. Figures 4(a) and (b) show the contents of the input files setup.in and gplist.in respectively for this case. Figure 5 shows the log file created after running the preprocessor. In Figure 5, the inlet blocks 8 and 9, and the exit blocks 11 and 12 are indicated as blocks that need to be reoriented. This is clear from looking at Figure 6 that shows an excerpt from the boundary condition file for this case. Here, the first column refers to the block number and the second column is the boundary condition. Boundary condition number 202 is an inlet boundary and 305 is an exit boundary. The remaining columns are extents of the boundary within the block given in the order 'is js ks ie je ke' (Ref. 7). The reoriented blocks have is = ie for the inlet and exit boundaries. This shows that they are at i-faces. The inlets are at i_{min} faces while the exits are at i_{max} faces. The plenum boundary is not in the axial direction and must therefore be reoriented manually using a module of the preprocessor. The user provides the block number and the type of operation to perform as input to the reorientation module. The warning messages in Figure 5 are expected for this case because there is no radial direction. At the end of the output shown in Figure 5 a list of files with the prefix 'pmap' are shown to be generated. These files contain a schedule to allow multiple blocks to run in parallel on a single processor. Figure 7 shows the contents of file 'pmap.report' that summarizes the contents of the files.
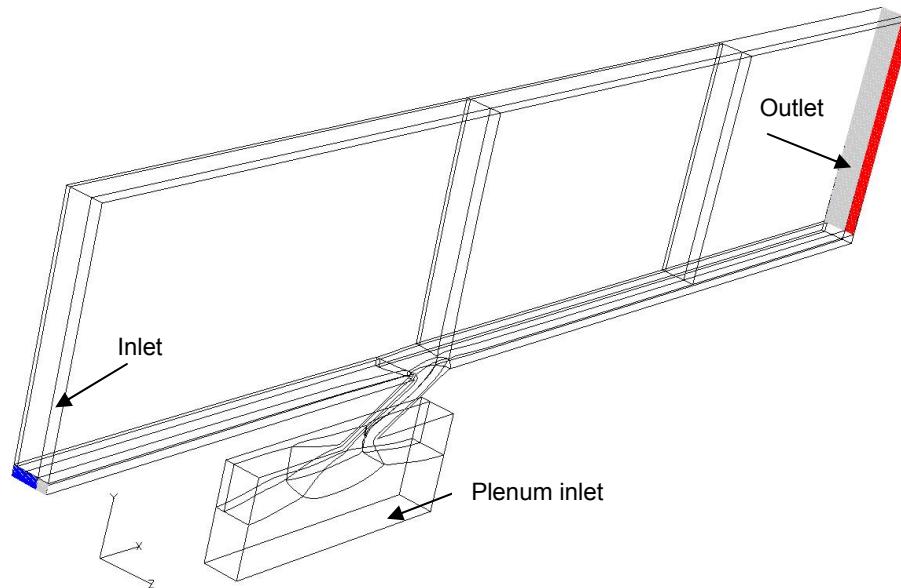
Figure 3.—Computational domain for example 1.

```
&SETUP_PARAMS                    &GP_PROPS
num_blade_rows=1                 gslip=4
num_blades=1                     gno_slip=2
checkconn=1                      gno_slip_iso=8
conn_tol=0.0000001               grad_eq_exit=6
turbo_friendly=.TRUE.            gperiodic=999
row_names=fine.tmp               gpressure_exit=999
/                                gplenum_in=999
                                 gref_clearance=999
                                 gts=3
                                 gcvbc_in=999
                                 gisentropic_in=5
                                 gwb_steady_in=999
                                 gwb_unsteady_in=999
                                 gwb_steady_exit=999
                                 gwb_unsteady_exit=999
                                 gcvbc_sub_exit=999
                                 gcvbc_super_exit=999
                                 gslide=999
                                 gslide_ts_i=10
                                 gslide_ts_j=999
                                 ginter_blk=1
                                 /

    a) setup.in                      b) gplist.in
```

Figure 4.—Input parameters for example 1.

```
Reading setup parameters from setup.in
Reading Setup Parameters from
setup.in

*************Blade row              1  *********
Converting gridpro files to plot3d
 First read for sizes ...

 Writing number of blocks and sizes to
 fine.tmp.dat


 Now read to dump plot3d file ...
Converting conn_n to bc and dmap
        43  Block interfaces found
         0 Ref Periodics found
        50  Boundary conditions found
Verifying connectivity
Grid tolerance is set at    1.000000000000000E-007
Angle of periodicity is:    360.000000000000      degrees.
File opened succesfully
        19  Blocks found
Block#  ni   nj    nk
         1        45        69        25
         2        25        69        13
         3        13        77        45
         4        13       129        97
         5        25       129        97
         6         5       129        97
         7        45        41         5
         8        45        13        89
         9        45        29        89
        10        45        13        77
        11        97        65        29
        12        97        65        13
        13        45       193        13
        14       113        97        13
        15        45        25       217
        16       113        97        25
        17        45         5       217
        18         5        97       113
        19         5        41        17
Block #        Block extents       Block size
         1         1        77625        77625
         2        77626       100050        22425
         3       100051       145095        45045
         4       145096       307764       162669
         5       307765       620589       312825
         6       620590       683154        62565
         7       683155       692379         9225
         8       692380       744444        52065
         9       744445       860589       116145
```

Figure 5.—Output upon execution of preprocessor for example 1.

```
                  10       860590       905634       45045
                  11       905635      1088479      182845
                  12      1088480      1170444       81965
                  13      1170445      1283349      112905
                  14      1283350      1425842      142493
                  15      1425843      1669967      244125
                  16      1669968      1943992      274025
                  17      1943993      1992817       48825
                  18      1992818      2047622       54805
                  19      2047623      2051107        3485
          2051107  data points will be read.
Plot3d File closed after reading
**************************************************
*****************Reading dmap.in*****************
    43
Connectivity has been verified for current row
Memory deallocation complete.
Writing GU files
Opening
 fine.tmp.p3d
                         as plot3d
        19  GU files written for BR           1 .
Combining bc and dmap files into bc.in and dmap.in
Making TURBO FRIENDLY
Blocks to change are:            8         9        11        12
        13        14        15        16        17        18
WARNING! UNABLE TO REORIENT BLOCK            8
Radial direction not detected. Manual inspection required.
WARNING! UNABLE TO REORIENT BLOCK           13
Radial direction not detected. Manual inspection required.
You should now have dmap.in, bc.in,GU files and tasklist.in(if turbo_friendly)
Creating pmap.in files for multiblock per processor options
Creating pmap files for multiblock per cpu simulations
======================================================
Average_size|total_size|maximum_size|num_procs_recmnd
     107953      2051107      312825            7
======================================================
   pmap.in.1.m3                                    has been created.
   pmap.in.2.m3                                    has been created.
   pmap.in.3.m3                                    has been created.
   pmap.in.4.m3                                    has been created.
   pmap.in.5.m3                                    has been created.
   pmap.in.6.m3                                    has been created.
   pmap.in.7.m3                                    has been created.
   pmap.in.1.m2                                    has been created.
   pmap.in.2.m2                                    has been created.
   pmap.in.3.m2                                    has been created.
   pmap.in.4.m2                                    has been created.
   pmap.in.5.m2                                    has been created.
   pmap.in.6.m2                                    has been created.
   pmap.in.7.m2                                    has been created.
===================End of all operations============
```

Blocks that will be reoriented

Warning because there is no radial direction for this case

Figure 5.—Concluded.

```
8    2   1   1   1   1  13  89/          8    2.00  1   1  45  89  13  45/
8  202   1   1   1  45  13   1/          8  202.00  1   1   1   1  13  45/
8    1   1  13   1  45  13  89/          8    1.00  1  13   1  89  13  45/
9    2   1   1   1   1  29  89/          9    2.00  1   1   1  89  29   1/
9    1   1   1   1  45   1  89/          9    1.00  1  29   1  89  29  45/
9  202   1   1   1  45  29   1/          9  202.00  1   1   1   1  29  45/
10   2   1   1   1   1  13  77/         10    2.00  1   1   1   1  77  13/
10   1   1   1   1  45  13   1/         10    1.00  1  77   1  45  77  13/
10   1   1  13   1  45  13  77/         10    1.00  1   1  13  45  77  13/
11   1   1   1   1   1  65  29/         11    1.00  1  97   1  65  97  29/
11   1   1   1   1  97  65   1/         11    1.00  1   1   1  65  97   1/
11 305   1  65   1  97  65  29/         11  305.00  65   1   1  65  97  29/
12   1   1   1   1   1  65  13/         12    1.00  1  97   1  65  97  13/
12 305   1  65   1  97  65  13/         12  305.00  65   1   1  65  97  13/
```
a) before reorientation   b) after reorientation

Figure 6.—Result of block manipulation by preprocessor for example 1.


```
**********Using Method 3***********
=============================================
Load distribution for num_procs=      1
Processor       1 has size   2051107 and      19 blocks
Total blocks assigned =      19
Percentage diff between largest and smallest:  0.000000000000000E+000
=============================================
Load distribution for num_procs=      2
Processor       1 has size   1076385 and       5 blocks
Processor       2 has size    974722 and      14 blocks
Total blocks assigned =      19
Percentage diff between largest and smallest:  9.44485476850755
=============================================
Load distribution for num_procs=      3
Processor       1 has size    703240 and       4 blocks
Processor       2 has size    715009 and       5 blocks
Processor       3 has size    632858 and      10 blocks
Total blocks assigned =      19
Percentage diff between largest and smallest:  11.4895057265013
=============================================
Load distribution for num_procs=      4
Processor       1 has size    518095 and       3 blocks
Processor       2 has size    527375 and       3 blocks
Processor       3 has size    537697 and       5 blocks
Processor       4 has size    467940 and       8 blocks
Total blocks assigned =      19
Percentage diff between largest and smallest:  12.9732916493862
=============================================
Load distribution for num_procs=      5
Processor       1 has size    428970 and       2 blocks
Processor       2 has size    416518 and       2 blocks
Processor       3 has size    430455 and       3 blocks
Processor       4 has size    429329 and       5 blocks
Processor       5 has size    345835 and       7 blocks
Total blocks assigned =      19
Percentage diff between largest and smallest:  19.6582685762739
=============================================
Load distribution for num_procs=      6
Processor       1 has size    357870 and       2 blocks
Processor       2 has size    355990 and       2 blocks
Processor       3 has size    357030 and       2 blocks
Processor       4 has size    358224 and       4 blocks
Processor       5 has size    358688 and       4 blocks
Processor       6 has size    263305 and       5 blocks
Total blocks assigned =      19
Percentage diff between      26.5921915425105
=============================================
```
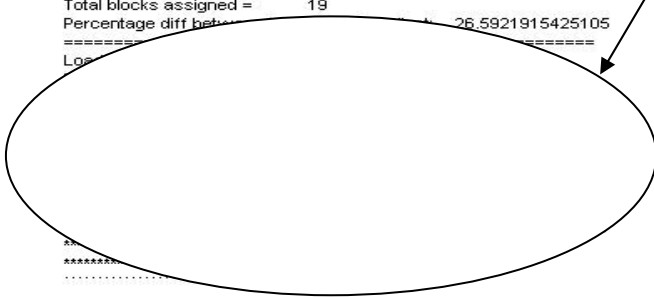
Best option

Figure 7.—Excerpt from pmap.report scheduling file.

Example 2: OSU HPT

The geometry used in this example was utilized in the work described in References 10 and 11. Figure 8 shows the blocks in this grid. The blue mesh represents the sliding interface boundary for the stator. The green and red meshes are periodic (time shift in this case) with each other. Figure 9 shows the contents of the input files for this example. There are now two blade rows in the input file and the blade count of each row is used to verify the connectivity of the time-shift (tangential) boundaries by calculating the angle through which a tangential boundary must be rotated to match its partner. The log file from executing the preprocessor is shown in Figure 10. Due to the existence of multiple rows in this example, the interface file 'turbo.in' is also populated with necessary information (Ref. 7). Figure 11 shows the results of reorienting the blocks to satisfy TURBO specifications.

Looking at Figure 11, it is clear that the tangential boundaries on blocks 4 and 8 have been placed on k faces in accordance to TURBO specifications. The boundary types –106 and –107 that are applicable to any computational coordinate face (i,j,k) are changed to boudnary type –102 that only deals with the time-shift boundary condition on a k-face. The sliding interface on block 6 has been placed on an i_{max} face.

These examples have shown the functionality of the preprocessor. There are several independent utilities that have also been developed to perform various operations on grids. Future work would include the integration of these utilities into the preprocessor and the creation of a Graphical User Interface (GUI) to make the preprocessor more user-friendly.
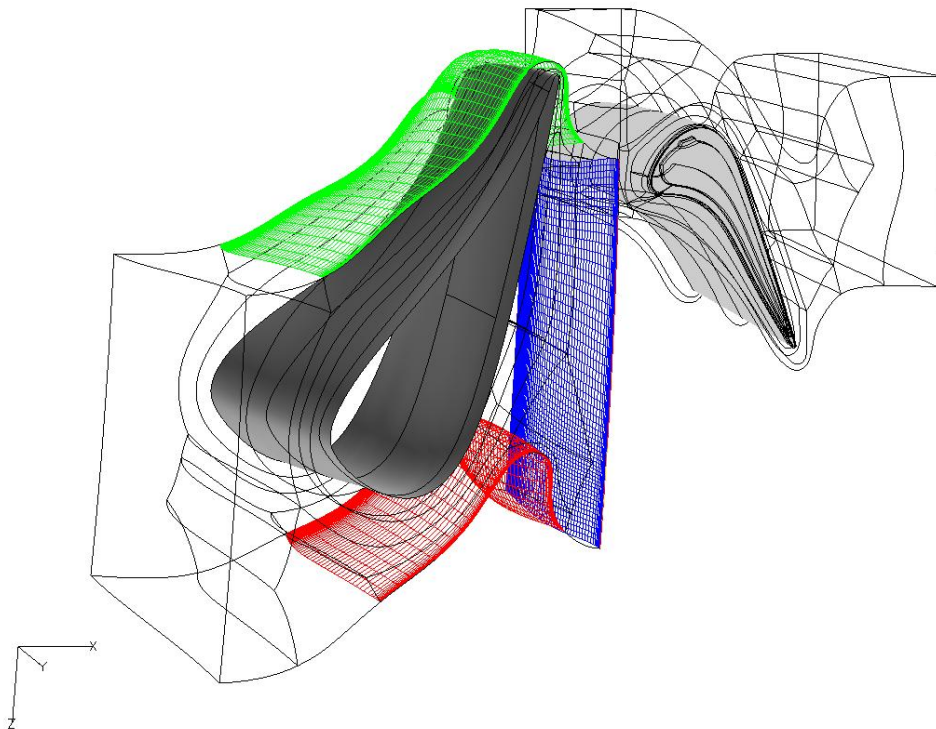

Figure 8.—Computational domain for example 2.

```
&SETUP_PARAMS                              |&GP_PROPS
num_blade_rows=2                           gslip=4
num_blades=38 72                           gno_slip=2
checkconn=1                                gno_slip_iso=8
conn_tol=0.000001                          grad_eq_exit=6
turbo_friendly=.TRUE.                      gperiodic=999
row_names= stator.tmp rotor.tmp            gpressure_exit=999
/                                          gplenum_in=999
                                           gref_clearance=999
                                           gts=3
                                           gcvbc_in=999
                                           gisentropic_in=5
                                           gwb_steady_in=999
                                           gwb_unsteady_in=999
                                           gwb_steady_exit=999
                                           gwb_unsteady_exit=999
                                           gcvbc_sub_exit=999
                                           gcvbc_super_exit=999
                                           gslide=999
                                           gslide_ts_i=10
                                           gslide_ts_j=999
                                           ginter_blk=1
                                           /

        a) setup.in                             b) gplist.in
```

Figure 9.—Input parameters for example 2.

```
Reading setup parameters from setup.in
Reading Setup Parameters from
setup.in

*************Blade row            1   *********
Converting gridpro files to plot3d
 First read for sizes ...

 Writing number of blocks and sizes to
 stator.tmp.dat


 Now read to dump plot3d file ...
Converting conn_n to bc and dmap
        34   Block interfaces found
         2 Ref Periodics found
        47   Boundary conditions found
Verifying connectivity
Grid tolerance is set at    1.000000000000000E-006
Angle of periodicity is:    9.47368421052632        degrees.
File opened succesfully
        11   Blocks found
Block#  ni   nj    nk
         1        67        27       109
         2       109        85        67
         3       109        17        67
         4         5       127        33
         5        17       127        17
         6         9       127        57
         7        13       127         9
         8        41       127         5
         9         5       127        57
        10        45       127         9
        11         5       127       109
Block #       Block extents      Block size
         1            1     197181      197181
         2       197182     817936      620755
         3       817937     942087      124151
         4       942088     963042       20955
         5       963043     999745       36703
         6       999746    1064896       65151
         7      1064897    1079755       14859
         8      1079756    1105790       26035
         9      1105791    1141985       36195
        10      1141986    1193420       51435
        11      1193421    1262635       69215
    1262635   data points will be read.
Plot3d File closed after reading
***************************************************
****************Reading dmap.in*****************
  34
Connectivity has been verified for current row
```

Figure 10.—Output upon execution of example 2.

```
Memory deallocation complete.
Writing GU files
Opening
 stator.tmp.p3d
                  as plot3d
      11  GU files written for BR          1 .
************Blade row           2  *********
Converting gridpro files to plot3d
 First read for sizes ...

 Writing number of blocks and sizes to
 rotor.tmp.dat


 Now read to dump plot3d file ...
Converting conn_n to bc and dmap
      74  Block interfaces found
       3 Ref Periodics found
      55  Boundary conditions found
Verifying connectivity
Grid tolerance is set at   1.000000000000000E-006
Angle of periodicity is:   5.00000000000000      degrees.
File opened succesfully
      17  Blocks found
Block#  ni   nj    nk
         1         13       107        25
         2         17       107        37
         3         49       107         9
         4         17       107        17
         5         77        60         9
         6         61        60         9
         7          9        28       161
         8        107         5        89
         9          5        60         9
        10         13        33       161
        11          5       107        65
        12        161        33         9
        13         81       107        13
        14        161        37        68
        15        161        37        17
        16         28        13       161
        17        107        33        57
Block #        Block extents       Block size
         1          1     34775     34775
         2      34776    102078     67303
         3     102079    149265     47187
         4     149266    180188     30923
         5     180189    221768     41580
         6     221769    254708     32940
         7     254709    295280     40572
         8     295281    342895     47615
         9     342896    345595      2700          ...
```

Figure 10.—Concluded.

```
4   -106   1    1    1    1  127  33/          4  -102.00   1    1    5   33  127   5/
8   -107   9    1    1   41  127   1/          8  -102.00   9    1    1   41  127   1/
4      2   1    1    1    5    1  33/          4     2.00   1    1    1   33    1   5/
4      2   1  127    1    5  127  33/          4     2.00   1  127    1   33  127   5/
5      2   1    1    1   17    1  17/          5     2.00   1    1    1   17    1  17/
5      2   1  127    1   17  127  17/          5     2.00   1  127    1   17  127  17/
6    402   1    1    1    1  127  57/          6   402.00   9    1    1    9  127  57/
6      2   1    1    1    9    1  57/          6     2.00   1    1    1    9    1  57/
6    106   1  101    1    5  127   1/          6   102.00   5  101   57    9  127  57/
6    107   1  101   57    5  127  57/          6   102.00   5  101    1    9  127   1/
6    106   1   85    1    5  101   1/          6   102.00   5   85   57    9  101  57/
```

        a) before reorientation   b) after reorientation

Figure 11.—Result of block manipulation by preprocessor for example 2.

Conclusions

The preprocessor was successfully tested for the geometry involved in the case of flow over a flat plate with a film cooling hole and for the geometry of a high pressure turbine stage. GridPro output (grid, boundary information and connectivity) was utilized to generate input for TURBO.

Appendix A.—Code Listing

```
!****************************PREPROCESSOR FOR TURBO***************************
!********Converts gridpro grids and conn_n to GU, bc.in,dmap.in,turbo.in*******
!********************************Vikram Shyam*********************************
!******************************Apr 08 2008***********************************
!*********************Last modified---Jun 12 2008****************************

!     The max num of blade rows is set at 100. If more are needed for some strange
reason!!!???
!     change the number 100 to whatever it needs to be. Same true for number of sliding
interfaces which is 1000

      program preprocessor
      implicit none
      integer i,j,k,startindex,axis
      integer num_blade_rows,checkconn
      logical turbo_friendly
      integer num_blades(100)!,n_bc(100)!,indices(100) !if more than 100 blade rows, change
this
      integer num_bc_tot,total_slides,tot_bks!,slidepos(1000)!if more than 1000 blocks change
this and if_dir
!        integer,dimension(:),allocatable::slidepos
      real conn_tol
      real x,y,z,speed
      character(100)::row_names(100) !if more than 100 blade rows, change this
      character(100)::infile,ifmt1,ifmt2,yfmt1,yfmt2
      integer, dimension(:),allocatable::yn,indices
     &,n_bc,slidepos!,if_dir
      integer :: if_dir(1000)
      axis=0
      speed=0.
      if_dir(1:1000)=0

      print *,'Reading setup parameters from setup.in'
      call readfiles(num_blade_rows,num_blades,row_names,
     &checkconn,conn_tol,turbo_friendly)
      allocate(n_bc(num_blade_rows)
     &,indices(num_blade_rows+1))
! indices(i) stores the block number that ends the (i-1)th blade row
!used in mergein for updating block ids and in find_dir2change for
!determining how many inlets, slides, exits in each BR
!        indices(1:100)=0
      indices(1:(num_blade_rows+1))=0
!        n_bc(1:100)=0
      n_bc(1:num_blade_rows)=0
      startindex=0
      if (num_blade_rows>99) then
      print *,"ERROR! TOO MANY BLADE ROWS"
      print *,"Please change limit of 100 in source code"
      endif
      do i=1,num_blade_rows
    infile=row_names(i)
      print *,'************Blade row ',i,' ********'
      print*,'Converting gridpro files to plot3d'
    call pro2p3d(infile)
      print*,'Converting conn_n to bc and dmap'
        call gp2turbo(infile,i,n_bc(i))!n_bc stores num BC in each BR
        if (checkconn.eq.1) then
        print*,'Verifying connectivity'
          call checkdmap(infile,conn_tol,num_blades(i),i)!i is blade row num
    endif
       print *,'Writing GU files'
        call p3d2gu(infile,i,startindex) !startindex is the last block in the BR
        indices(i+1)=startindex
      enddo
!     print *,indices(1:5)
      tot_bks=startindex
!     print*,tot_bks
      allocate(slidepos(tot_bks))
```

```fortran
      print *,'Combining bc and dmap files into bc.in and dmap.in'
!mergein combines bc.in and dmap.in and also outputs total number of BCs as num_bc_tot
!isinlet, isexit are calculated here as well and used in find_dir2change
      call mergein(indices,num_blade_rows,n_bc,num_bc_tot)

!      print *,'Do you wish to make this case turbo friendly?'

!*****REORIENTING BLOCKS HERE**********************

      if (turbo_friendly) then
      print *,'Making TURBO FRIENDLY'
      call find_dirs_axial(tot_bks) !writes out tasklist.in.axial - a record of blocks and
task numbers used to reorient
      call find_dirs_periodic(tot_bks) !writes out tasklist.in.periodic
      call find_dirs_radial(tot_bks) !writes out tasklist.in.radial
!tasklist files can be renamed to tasklist.in, edited and used with reorient.f for manual
reorientation.

!      call operate
      call periodic_fix
      endif
!***********turbo.in written here******************
!      if (num_blade_rows>1) then
      call findslides(total_slides,slidepos,tot_bks,if_dir)
      if (total_slides>0) then
      print *,'writing turbo.in'
      open(unit=13,file='turbo.in',status='unknown',form='formatted')
      write(13,'(2x,1I3)')num_blade_rows
      allocate(yn(tot_bks))
!      allocate(if_dir(total_slides))
      yfmt1=file_name0('(1x,',tot_bks)
      yfmt2=file_cat(yfmt1,'I3,I3,1x,F4.2)')
      ifmt1=file_name0('(1x,',total_slides)
      ifmt2=file_cat(ifmt1,'I2)')
      do i=1,num_blade_rows
      yn(1:tot_bks)=0 !This enters a 1 if block is part of BR i and 0 otherwise
      yn((indices(i)+1):indices(i+1))=
     &slidepos((indices(i)+1):indices(i+1)) !Unnecessary operation so that slidepos can
remain unallocatable
                                          !slidepos is len=1000 but want only
length=tot_bks so allcoate yn and transfer

      write(13,yfmt2)yn,axis,speed
      enddo
!------WARNING assuming that sliding interfaces come in pairs..not true if faces are split---
--
      write(13,*)total_slides/2
!      if_dir(1:total_slides)=2
      write(13,ifmt2)if_dir(1:total_slides)
      close(13)
!      deallocate(if_dir,yn)
      deallocate(yn)
      else
      open(unit=13,file='turbo.in',status='unknown',form='formatted')
      write(13,'(2x,1I3)')0
      close(13)
      endif
!      endif
      print *,'You should now have dmap.in, bc.in,GU files and
     & tasklist.in(if turbo_friendly)'
      print *,'Creating pmap.in files for multiblock per processor
     & options'
      call make_pmap
      print *,'==================End of all operations==========='

      contains

      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
```

```
         character(len=*),intent (in) :: pre,post
         character(len=100) :: file_cat
         ints=len_trim(pre)
         inte=len_trim(post)
         write(file_cat,*)pre(1:ints),post(1:inte)
         return
         end function file_cat

         function file_name0(pre,n) ! copied from TURBO!!!
         implicit none
         integer n,ints

         character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                               '(a,i4)','(a,i5)','(a,i6)'/)
         character(len=*),intent (in) :: pre
         character(len=100) :: file_name0
         if (n.gt.0) ints = log10(real(n))
         if (n.eq.0) ints = 0
         ints = ints+1
         write(file_name0,form(ints))pre,n
         return
         end function file_name0

         end program preprocessor

         subroutine findslides(total_slides,slide_pos,num_bks,if_dir)
         implicit none
!    bc.in VARS
         integer :: num_bc_real,num_bc,dum,dir,if_dir(1000)
         integer, dimension(:),allocatable:: block_id,start_i,
     &                        start_j,start_k,end_i,end_j,end_k
         real,dimension(:),allocatable::bc_type_and_group
         integer, dimension(:),allocatable::gid,gname

!    local VARS
         character(len=50) :: infile,bcfile
         integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
         integer :: i1,j1,k1,i2,j2,k2,total_recs
         integer:: fid,slide_pos(1000)
!     integer,dimension(:),allocatable::slide_pos
         integer num_bks,num_blade_rows,total_slides
         bcfile='bc.in'
         total_slides=0

! READ BC.IN AND STORE
!     allocate(slide_pos(num_bks))
         slide_pos(1:num_bks)=-1
         open(unit=10,file=bcfile,FORM='formatted',status='unknown')
         num_bc=0
         do
         read(10,*),dum
!     print *,dum
         if (dum.eq.0) EXIT
         num_bc=num_bc+1
         enddo
         close(10)

         !*****TO SIMPLIFY REWRITE INTO bc.in*********
         num_bc_real=num_bc
         num_bc=num_bc+1


         allocate(block_id(num_bc),bc_type_and_group(num_bc)
     &,start_i(num_bc),
     &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
     &,end_k(num_bc),gid(num_bc),gname(num_bc))

!      print *,num_bc_real,'Boundary conditions found.'
         open(unit=10,file=bcfile,FORM='formatted',status='unknown')
         do i=1,num_bc
         read(10,*),block_id(i),
```

```
      &bc_type_and_group(i),start_i(i),
      &start_j(i),start_k(i),end_i(i),end_j(i)
      &,end_k(i)
       if (bc_type_and_group(i)>400.) then
       total_slides=total_slides+1
       slide_pos(block_id(i))=1
       call current_bc_dir(start_i(i),
      &start_j(i),start_k(i),end_i(i),end_j(i)
      &,end_k(i),dir)
!mins>0, max<0
      if (dir.eq.1) if_dir(total_slides)=1
      if (dir.eq.-1) if_dir(total_slides)=2
      if (dir.eq.2) if_dir(total_slides)=3
      if (dir.eq.-2) if_dir(total_slides)=4
      if (dir.eq.3) if_dir(total_slides)=5
      if (dir.eq.-3) if_dir(total_slides)=6
      endif
      enddo
      close(10)
!       print *,slide_pos(1:num_bks),total_slides

      end subroutine
!******************************************************************
      subroutine current_bc_dir(is,js,ks,ie,je,ke,dir)
      implicit none
      integer:: is,js,ks,ie,je,ke,dir
      if (is.eq.ie) then
          if (is.eq.1) then
                  dir=1
          else
          dir=-1
      endif
        elseif (js.eq.je) then
          if (js.eq.1) then
                  dir=2
          else
          dir=-2
      endif
        elseif (ks.eq.ke) then
          if (ks.eq.1) then
                  dir=3
          else
          dir=-3
      endif
        endif


      end subroutine current_bc_dir

!*******************************************************************************************
      subroutine new_bc_dir(bc,dir,newdir,check,inlet_exists,exit_exists
      &)
       implicit none
       integer bc,newdir,check,dir,inlet_exists,exit_exists
       if (bc>200 .and. bc<300 .and. bc/=205) then
         if (check.eq.1)then
            newdir=1
         else
         newdir=dir
         endif
        elseif ((bc>300 .and. bc<400)) then
         if (check.eq.2)then
            newdir=-1
         else
         newdir=dir
         endif
        elseif(bc.eq.402 .and. inlet_exists.eq.1 .and. exit_exists.eq.0)
       &then
         if (check.eq.2)then
            newdir=-1
         else
```

```
         newdir=dir
       endif
     elseif (bc.eq.402 .and. exit_exists.eq.1 .and. inlet_exists.eq. 0)
    & then
       if (check.eq.1)then
          newdir=1
       else
       newdir=dir
       endif
     elseif (bc.eq.403) then
       if (check.eq.3)then
          newdir=-2
       else
       newdir=dir
       endif
     else
       newdir=dir
     endif
     end subroutine new_bc_dir

!*****************************************************************
     subroutine readfiles(num_blade_rows,num_blades,row_names,
    &checkconn,conn_tol,turbo_friendly)
     implicit none
     integer i,j,k
     integer num_blade_rows,checkconn
     logical turbo_friendly
     integer num_blades(100)
     real conn_tol
     real x,y,z
     character(100)::row_names(100)
     character(100)::fname

     namelist/SETUP_PARAMS/
    &num_blade_rows,num_blades,checkconn,conn_tol,turbo_friendly
    &,row_names

     num_blade_rows=0
     num_blades(1:100)=0.0
     checkconn=0
     conn_tol=0.0
     turbo_friendly=.FALSE.

     fname = 'setup.in'
     !defaults
     print *,'Reading Setup Parameters from ',fname
     open(UNIT=7,file=fname,form='formatted')
     rewind(7)
     do while (.not. .FALSE.)
     read(7,nml=SETUP_PARAMS,err=301,end=303)
     close(7)

     goto 302

301  continue
     enddo

303  close(7)
302  continue

!     print*,num_blade_rows,num_blades,checkconn,conn_tol,turbo_friendly
     do i=1,num_blade_rows
!     print*,row_names(i)
     enddo

     end subroutine readfiles

     subroutine pro2p3d(infile)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!_____
!--                                                              #
```

```
!---- Code to read in GridPro file and output plot3d type file  #
!---- NOTE: only reads ascii and writes fortran unformatted     #
!----                                                           #
!---- rigby@lerc.nasa.gov, Initial: 030497, Revised: 030597     #
! revised (4/29/98) by VK Garg so as to read x,y,z as 1-D arrays#
! (makes it possible to read and write large grids)             #
!----                                                           #
!----  010628 Took away need to ask user number of blocks       #
!----                                                           #
      parameter(maxp=3000000, nbmx=400)
      integer im(nbmx),jm(nbmx),km(nbmx),iw(maxp)
      real x(maxp),y(maxp),z(maxp)
      character(100) infile, p3dfile, datfile
c_____
c--                            #
c---- Get name of grid file   #
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#
      open(unit=10,file=infile,status='old',form='formatted')
      p3dfile=file_cat(infile,'.p3d')
      open(unit=11,file=p3dfile,status='unknown',form='unformatted')
      datfile=file_cat(infile,'.dat')
      open(unit=82,file=datfile,status='unknown',form='formatted')
c_____
c--                                       #
c---- Read grid and output GridPro file   #
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#

      do i=1,icom
        read(10,*)
      enddo
c_____
c--                                                   #
c---- Determine size of each block and write sizes   #
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#
      print*,' First read for sizes ...'
      maxpt= 0
      nblks = 0
      do nb=1,nbmx
!        if(mod(nb,max(nblks/10,1)).eq.0) print*,' Done thru ',nb,' ...'
         read(10,*,end=88) im(nb),jm(nb),km(nb)
!        print *,'check 1'
         nblks = nblks + 1
         npt=im(nb)*jm(nb)*km(nb)
         maxpt=max(maxpt,npt)
         ii=0
         do i=1,im(nb)
         do j=1,jm(nb)
         do k=1,km(nb)
          ii=ii+1
          iw(ii)=(k-1)*jm(nb)*im(nb)+(j-1)*im(nb)+i
         enddo
         enddo
         enddo
!        print *, 'check 2'
         read(10,*) (x(iw(i)),y(iw(i)),z(iw(i)),i=1,ii)
!        print *,'check 3'
      enddo
88    continue
      if(nblks.gt.nbmx.or.maxpt.gt.maxp) then
        print*,' ERROR:'
        print*,' The maximum # of blocks is : ',nblks
        print*,' The maximum # of points in a block is : ',maxpt
        print*,' Currently compiled for ',nbmx,' blocks'
        print*,'            With ',maxp,' points in a block'
        stop
      endif
      rewind(10)
      write(11) nblks
      write(11) (im(nb),jm(nb),km(nb),nb=1,nblks)
      print*,' '
      print*,' Writing number of blocks and sizes to ',datfile
```

```
      print*,' '
      write(82,*) nblks
      do nb=1,nblks
        write(82,*) im(nb),jm(nb),km(nb)
      enddo
c_____
c--                                      #
c---- Read each block then write  #
c^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^#
      print*,' Now read to dump plot3d file ...'
      do ic=1,icom
        read(10,*)
      enddo
      do nb=1,nblks
!       if(mod(nb,max(nblks/10,1)).eq.0) print*,' Done thru ',nb,' ...'
        read(10,*) im(nb),jm(nb),km(nb)
        ii=0
        do i=1,im(nb)
        do j=1,jm(nb)
        do k=1,km(nb)
         ii=ii+1
         iw(ii)=(k-1)*jm(nb)*im(nb)+(j-1)*im(nb)+i
        enddo
        enddo
        enddo
        read(10,*) (x(iw(i)),y(iw(i)),z(iw(i)),i=1,ii)
        write(11) (x(i),i=1,ii),(y(i),i=1,ii),(z(i),i=1,ii)
      enddo

      close(10)
      close(11)
      close(82)

      contains

      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
      character(len=*),intent (in) :: pre,post
      character(len=100) :: file_cat
      ints=len_trim(pre)
      inte=len_trim(post)
      write(file_cat,*)pre(1:ints),post(1:inte)
      return
      end function file_cat

      end subroutine pro2p3d
!*************Program to use gridpro conn_n file******************
!***********to generate msuTurbo bc.in and dmap.in****************
!***********        by Vikram Shyam - 1/10/07     ****************
!***********        Last modified: 4/02/2008          ************
!****************************************************************
      subroutine gp2turbo(infile,blade_row)
      implicit none

!     SUPERBLOCK VARS
      integer, dimension(:), allocatable:: sbid,ni,nj,nk,ebid,eb2sb
      integer,dimension(:),allocatable::b_lbid,b_pty
      integer :: nblks,blade_row
      character(len=2):: SB
      character(len=12):: super
!     PATCH VARS
      integer,dimension(:),allocatable::pid,sb1,sf1,sb2,sf2,p_pty,p_lbid
      character(len=3) :: fmap
      integer,dimension(:,:),allocatable:: l1,l2,h1,h2
      integer::np
      character(len=11) :: face
      character(len=1) :: p,fi,fj,fk
!     dmap.in VARS
      integer :: num_b2b, num_special_b2b
      integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
```

```
        integer, dimension(:),allocatable:: d1e,id,dir2,lor1,lor2,p_b2b
        integer, dimension(:),allocatable:: p_special,bc,p_bc
        integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!       bc.in VARS
        integer :: num_bc
!        integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
!        integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit
        integer, dimension(:),allocatable:: block_id,bc_type,start_i,
       &                        start_j,start_k,end_i,end_j,end_k
        integer, dimension(:),allocatable::gid,gname
        integer :: sf_num !vs 1/30/08

!       local VARS
        character(len=100) :: gprofile,dmapfile,bcfile,infile,file_name0
        integer :: i,j,k,l,m,n
        integer, dimension(:),allocatable::map1,map2,map3
        integer :: checkind,update_gp_props

!       GPro property file vars
        integer:: slip,no_slip,no_slip_iso,isentropic_in,rad_eq_exit
        integer::periodic,ref_periodic,cvbc_in,pressure_exit,plenum_in
        integer::ref_ts, ref_clearance,ref_periodic_fwd,ref_periodic_bak
       & ,ref_ts_fwd,ref_ts_bak,ts,periodic_fwd,periodic_bak,
       & ts_bak,wb_steady_in,wb_unsteady_in,wb_steady_exit,
       & wb_unsteady_exit,cvbc_sub_exit,cvbc_super_exit,slide,slide_ts_i,
       & slide_ts_j,inter_blk
        integer:: gslip,gno_slip,gno_slip_iso,gisentropic_in,grad_eq_exit
        integer::gperiodic,gcvbc_in,gpressure_exit,ginter_blk,
       & gplenum_in
        integer::gref_ts, gref_clearance,
       & gts,
       & gwb_steady_in,gwb_unsteady_in,
       & gwb_steady_exit,gwb_unsteady_exit,gcvbc_sub_exit,
       & gcvbc_super_exit,gslide, gslide_ts_i,gslide_ts_j




!       print *,"Enter GridPro file to be translated: "
!        read *, gpro

        update_gp_props=1
!        print *,'Do you want to update bc properties?yes=1,no=0'
!        read *,update_gp_props

!       dmap.in is the file into which TURBO connectivity is written
!        dmap='dmap.in'
!        bcfile='bc.in'
        gprofile=file_cat(infile,'.conn_n')
        dmapfile=file_name0('dmap.',blade_row)
        bcfile=file_name0('bc.',blade_row)

        open(UNIT=7, FILE=gprofile, FORM='formatted',status='old')
        read(7,*), nblks,super


        allocate(sbid(nblks),ni(nblks),nj(nblks),nk(nblks),ebid(nblks),
       & eb2sb(nblks),b_pty(nblks),b_lbid(nblks))

        read(7,*) ! REads comment line in grid pro conn_n file

        read(7,*),(SB, sbid(i),ni(i),nj(i),nk(i),ebid(i),eb2sb(i),b_pty(i)
       & ,b_lbid(i),i=1,nblks)


        read(7,*) np,face
        allocate(pid(np),sb1(np),sb2(np),sf1(np),sf2(np),
       &         l1(3,np),l2(3,np),h1(3,np),h2(3,np),p_b2b(np)
       &         ,p_special(np),p_pty(np),p_lbid(np),p_bc(np))
        allocate(map1(np),map2(np),map3(np))
        read(7,*) ! Reads comment line in grid pro conn_n file
```

```
!*****Read patches and find how many block interfaces exist********************
!     nblks blocks are read and np patches.
!      if the connecting block id is 0, no connectivity is needed
!      if the label pty on a patch is 3, ref periodic condition is used

!      Connectivity count for regular b2b
       num_b2b=0
!      Special interfaces (Reference: pty=3)
       num_special_b2b=0

!      Boundary condition count (sb2=0) need list of p_pty -> bc_label
!      p_pty  bc.in label   bc
!      0           1        slip
!      1           2        no slip
!      2          101       periodic
!      3         -101       ref periodic
!      4          201       cvbc_in
!      5          202       isentropic in
!      6          301       rad_eq_exit
!      7          304       pressure_exit

!A list of TURBO BC values. '_k' has been ommited because only fwd and bak are used.
!For the periodic direction this code decides whetehr to use reference conditions in bc/dmap
!and will determine whether to use _fwd or _bak for both periodic and ts.

!Except for _fwd and _bak, these values are overwritten by their
       num_bc=0

       slip=1
       no_slip=2
       no_slip_iso=3
       ref_clearance=-103
       ref_periodic_fwd=-104
!       ref_periodic_fwd=-101
       ref_periodic_bak=-105
!       ref_periodic_bak=-101
       ref_ts_fwd=-106
!       ref_ts_fwd=-102
       ref_ts_bak=-107
!       ref_ts_bak=-102
       periodic_fwd=104
!       periodic_fwd=101
       periodic_bak=105
!       periodic_bak=101
       ts_fwd=106
!       ts_fwd=102
       ts_bak=107
!       ts_bak=102
       cvbc_in=201
       isentropic_in=202
       wb_steady_in=203
       wb_unsteady_in=204
       plenum_in=205
       rad_eq_exit=301
       wb_steady_exit=302
       wb_unsteady_exit=303
       pressure_exit=304
       cvbc_sub_exit=305
       cvbc_super_exit=306
       slide=401
       slide_ts_i=402
       slide_ts_j=403
       inter_blk=-555

       gslip=4
       gno_slip=2
       gno_slip_iso=8
       gisentropic_in=999
       gcvbc_in=7
       grad_eq_exit=999
```

```
      gperiodic=3
      gpressure_exit=10
      gplenum_in=5
   gref_clearance=999
   gts=999
   gcvbc_in=7
   gisentropic_in=999
   gwb_steady_in=999
   gwb_unsteady_in=999
   gplenum_in=999
   grad_eq_exit=999
   gwb_steady_exit=999
   gwb_unsteady_exit=999
   gpressure_exit=999
   gcvbc_sub_exit=6
   gcvbc_super_exit=999
   gslide=999
   gslide_ts_i=999
   gslide_ts_j=999
      ginter_blk=1

      call readgp(gslip,gno_slip,gno_slip_iso,gisentropic_in,
     &grad_eq_exit,gperiodic,gcvbc_in,gpressure_exit,
     &gplenum_in,gts,gref_clearance,gwb_steady_in,gwb_unsteady_in,
     &gwb_steady_exit,gwb_unsteady_exit,gcvbc_sub_exit,
     &gcvbc_super_exit,gslide,gslide_ts_i,gslide_ts_j,ginter_blk)
! Boundary conditions to be found when
! 1. sb2 is 0 (no connecting face)
! 2. sb2 is nonzero and the connection is a reference periodic
!    or some other reference condition
! 3. when sb2 is sb1 but protperty is periodic.


      do i=1,np
      read(7,*) p,pid(i),sb1(i),sf1(i),sb2(i),sf2(i),fmap,l1(1,i),
     &          l1(2,i),l1(3,i),h1(1,i),h1(2,i),h1(3,i),l2(1,i),
     & l2(2,i),l2(3,i),h2(1,i),h2(2,i),h2(3,i),p_pty(i),p_lbid(i)
      fi=fmap(1:1)
      fj=fmap(2:2)
      fk=fmap(3:3)
      map1(i)=string_to_int(fi)
      map2(i)=string_to_int(fj)
      map3(i)=string_to_int(fk)
      if (abs(sb2(i))>0) then ! selects block to block interfaces only
          if((p_pty(i)==ginter_blk))then ! selects only physically connected block interfaces
              num_b2b=num_b2b+1
              p_b2b(num_b2b)=i
          elseif (.not.(sb1(i)==sb2(i))) then ! selects reference connections
              num_special_b2b=num_special_b2b+1
              p_special(num_special_b2b)=i
              num_bc=num_bc+1
              p_bc(num_bc)=i
              num_bc=num_bc+1
              p_bc(num_bc)=-i
          elseif (sb1(i)==sb2(i)) then
                  if (p_pty(i)==gperiodic.or.p_pty(i)==gts)then
              num_bc=num_bc+1
              p_bc(num_bc)=i
              num_bc=num_bc+1
              p_bc(num_bc)=-i
                  else
              num_bc=num_bc+1
              p_bc(num_bc)=i
                  endif
          endif
      elseif (sb2(i)==0) then
              num_bc=num_bc+1
              p_bc(num_bc)=i
      endif
! VErify map in gridpro
!     print *,fi,fj,fk
```

```
! end verify map
      enddo


      print *, num_b2b,' Block interfaces found'
!      print *, (p_b2b(i),i=1,num_b2b)
      print *, num_special_b2b, "Ref Periodics found "
!      print *, (p_special(i),i=1,num_special_b2b)
      print *, num_bc,' Boundary conditions found'
!      print *, (p_bc(i),i=1,num_bc)

!   Verify gridpro file
!      do i=1,nblks
!      print * ,SB,sbid(i),ni(i),nj(i),nk(i),ebid(i),eb2sb(i),p_pty(i),
!     &        p_lbid(i)
!      end do
! End verify

!      do i=1,np
!      print *, p,pid(i),sb1(i),sf1(i),sb2(i),sf2(i),fmap,l1(1,i),
!     &          l1(2,i),l1(3,i),h1(1,i),h1(2,i),h1(3,i),l2(1,i),
!     & l2(2,i),l2(3,i),h2(1,i),h2(2,i),h2(3,i),pty,lbid
!      end do
      close(7)
!*******************END READ patches, bc's, b2b's************************
! Write dmap.in
      allocate(id(num_b2b),is(num_b2b),ie(num_b2b),js(num_b2b)
     & ,je(num_b2b),ks(num_b2b),ke(num_b2b),blkb(num_b2b),d1(num_b2b),
     & d1s(num_b2b),d1e(num_b2b),d2(num_b2b),d2s(num_b2b),d2e(num_b2b),
     & d3(num_b2b),d3s(num_b2b),d3e(num_b2b),dir1(num_b2b),dir2(num_b2b)
     & ,lor1(num_b2b),lor2(num_b2b))

      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
      write(8,'(I5)') num_b2b
! Wirtes out block to block interfaces only:
      do i=1,num_b2b
!      print *,i,' of ',num_b2b,' b2b complete'
      m=p_b2b(i)
      id(i)=sb1(m)
      blkb(i)=sb2(m)
      is(i)=l1(1,m)
      ie(i)=h1(1,m)
      js(i)=l1(2,m)
      je(i)=h1(2,m)
      ks(i)=l1(3,m)
      ke(i)=h1(3,m)

      if (map1(m)<3) then
          n=map1(m)+1
          d1(i)=map1(m)
          d1s(i)=l2(n,m)
          d1e(i)=h2(n,m)
      else
          n=map1(m)-2
          d1(i)=map1(m)-3
          d1s(i)=h2(n,m)
          d1e(i)=l2(n,m)
      endif

      if (map2(m)<3) then
          n=map2(m)+1
          d2(i)=map2(m)
          d2s(i)=l2(n,m)
          d2e(i)=h2(n,m)
      else
          n=map2(m)-2
          d2(i)=map2(m)-3
          d2s(i)=h2(n,m)
          d2e(i)=l2(n,m)
      endif
```

```
        if (map3(m)<3) then
            n=map3(m)+1
            d3(i)=map3(m)
            d3s(i)=l2(n,m)
            d3e(i)=h2(n,m)
        else
            n=map3(m)-2
            d3(i)=map3(m)-3
            d3s(i)=h2(n,m)
            d3e(i)=l2(n,m)
        endif
! Enforce Grid pro rules and TURBO rules
        if (d1s(i)<d1e(i)) then
            d1s(i)=d1s(i)+2
            d1e(i)=d1e(i)+1
        elseif (d1s(i)==d1e(i)) then
            if (d1s(i)==0) then
                    d1s(i)=d1s(i)+2
                    d1e(i)=d1e(i)+2
            else
                    d1s(i)=d1s(i)+1
                    d1e(i)=d1e(i)+1
            endif
        else
            d1s(i)=d1s(i)+1
            d1e(i)=d1e(i)+2
        endif

        if (d2s(i)<d2e(i)) then
            d2s(i)=d2s(i)+2
            d2e(i)=d2e(i)+1
        elseif (d2s(i)==d2e(i)) then
            if (d2s(i)==0) then
                    d2s(i)=d2s(i)+2
                    d2e(i)=d2e(i)+2
            else
                    d2s(i)=d2s(i)+1
                    d2e(i)=d2e(i)+1
            endif
        else
            d2s(i)=d2s(i)+1
            d2e(i)=d2e(i)+2
        endif

        if (d3s(i)<d3e(i)) then
            d3s(i)=d3s(i)+2
            d3e(i)=d3e(i)+1
        elseif (d3s(i)==d3e(i)) then
            if (d3s(i)==0) then
                    d3s(i)=d3s(i)+2
                    d3e(i)=d3e(i)+2
            else
                    d3s(i)=d3s(i)+1
                    d3e(i)=d3e(i)+1
            endif
        else
            d3s(i)=d3s(i)+1
            d3e(i)=d3e(i)+2
        endif

        if (is(i)<ie(i)) then
            is(i)=is(i)+2
            ie(i)=ie(i)+1
        elseif (is(i)==ie(i)) then
            if (is(i)==0) then
            is(i)=2
            ie(i)=2
            elseif (.not.is(i)==0) then
            is(i)=is(i)+1
            ie(i)=ie(i)+1
            endif
```

```
            else
                is(i)=is(i)+1
                ie(i)=ie(i)+2
            endif

            if (js(i)<je(i)) then
                js(i)=js(i)+2
                je(i)=je(i)+1
            elseif (js(i)==je(i)) then
                if (js(i)==0) then
                js(i)=2
                je(i)=2
                elseif (.not.js(i)==0) then
                js(i)=js(i)+1
                je(i)=je(i)+1
                endif
            else
                js(i)=js(i)+1
                je(i)=je(i)+2
            endif

            if (ks(i)<ke(i)) then
              ks(i)=ks(i)+2
              ke(i)=ke(i)+1
            elseif (ks(i)==ke(i)) then
              if (ks(i)==0) then
                    ks(i)=2
                    ke(i)=2
              elseif (.not.ks(i)==0) then
                    ks(i)=ks(i)+1
                    ke(i)=ke(i)+1
                endif
            else
                ks(i)=ks(i)+1
                ke(i)=ke(i)+2
            endif
            if (is(i)==ie(i)) then
                dir1(i)=1
                if (is(i)<ni(sb1(m))) then
                        lor1(i)=0
                else
                lor1(i)=1
                endif
            elseif (js(i)==je(i)) then
                dir1(i)=2
                if (js(i)<nj(sb1(m))) then
                lor1(i)=0
                else
                lor1(i)=1
                endif
            else
                dir1(i)=3
                 if (ks(i)<nk(sb1(m))) then
                 lor1(i)=0
                 else
                 lor1(i)=1
                 endif
            endif

            if (d1s(i)==d1e(i)) then
                dir2(i)=d1(i)+1
!     print *,'1 direction is',dir2(i),d1s(i),ni(sb2(m))
!    &,nj(sb2(m)),nk(sb2(m))
              select case(dir2(i))
              case(1)
                  checkind=ni(sb2(m))
              case(2)
                  checkind=nj(sb2(m))
              case(3)
                  checkind=nk(sb2(m))
              end select
```

```
            if (d1s(i)<checkind) then
               lor2(i)=0
               else
               lor2(i)=1
               endif
         elseif (d2s(i)==d2e(i)) then
               dir2(i)=d2(i)+1
!     print *,'2 direction is',dir2(i),d2s(i),ni(sb2(m))
!     &,nj(sb2(m)),nk(sb2(m))
            select case(dir2(i))
            case(1)
                checkind=ni(sb2(m))
            case(2)
                checkind=nj(sb2(m))
            case(3)
                checkind=nk(sb2(m))
            end select

            if (d2s(i)<checkind) then
               lor2(i)=0
               else
               lor2(i)=1
               endif
         else
               dir2(i)=d3(i)+1
!     print *,'3 direction is',dir2(i),d3s(i),ni(sb2(m))
!     &,nj(sb2(m)),nk(sb2(m))
            select case(dir2(i))
            case(1)
                checkind=ni(sb2(m))
            case(2)
                checkind=nj(sb2(m))
            case(3)
                checkind=nk(sb2(m))
            end select

            if (d3s(i)<checkind) then
               lor2(i)=0
               else
               lor2(i)=1
               endif
         endif

         if (is(i)==2 .and. d1s(i)==2 .and. is(i)<ie(i)
     &     .and. d1s(i)<d1e(i)) then
               is(i)=1
                  d1s(i)=1
         endif

         if (js(i)==2 .and. d2s(i)==2 .and. js(i)<je(i)
     &     .and. d2s(i)<d2e(i)) then
               js(i)=1
                  d2s(i)=1
         endif

         if (ks(i)==2 .and. d3s(i)==2 .and. ks(i)<ke(i)
     &     .and. d3s(i)<d3e(i)) then
               ks(i)=1
                  d3s(i)=1
         endif


         write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
         end do
```

```
        deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)
! ****************Done writing normal b2b interfaces**************

!**************Writing special Block to Block interfaces***********

       write(8,'(I3)'), num_special_b2b
       allocate(id(num_special_b2b),is(num_special_b2b),
     &ie(num_special_b2b),js(num_special_b2b),je(num_special_b2b)
     &,ks(num_special_b2b),ke(num_special_b2b),blkb(num_special_b2b)
     &,d1(num_special_b2b),d1s(num_special_b2b),d1e(num_special_b2b)
     &,d2(num_special_b2b),d2s(num_special_b2b),d2e(num_special_b2b)
     &, d3(num_special_b2b),d3s(num_special_b2b),d3e(num_special_b2b)
     &,dir1(num_special_b2b),dir2(num_special_b2b),
     & lor1(num_special_b2b),lor2(num_special_b2b),bc(num_special_b2b))
       do i=1,num_special_b2b
!        print *,i,' of ',num_special_b2b,' ref_b2b complete'
         m=p_special(i)
         id(i)=sb1(m)
         blkb(i)=sb2(m)
         is(i)=l1(1,m)
         ie(i)=h1(1,m)
         js(i)=l1(2,m)
         je(i)=h1(2,m)
         ks(i)=l1(3,m)
         ke(i)=h1(3,m)

         if (map1(m)<3) then
                 n=map1(m)+1
                 d1(i)=map1(m)
                 d1s(i)=l2(n,m)
                 d1e(i)=h2(n,m)
         else
                 n=map1(m)-2
                 d1(i)=map1(m)-3
                 d1s(i)=h2(n,m)
                 d1e(i)=l2(n,m)
         endif

         if (map2(m)<3) then
                 n=map2(m)+1
                 d2(i)=map2(m)
                 d2s(i)=l2(n,m)
                 d2e(i)=h2(n,m)
         else
                 n=map2(m)-2
                 d2(i)=map2(m)-3
                 d2s(i)=h2(n,m)
                 d2e(i)=l2(n,m)
         endif

         if (map3(m)<3) then
                 n=map3(m)+1
                 d3(i)=map3(m)
                 d3s(i)=l2(n,m)
                 d3e(i)=h2(n,m)
         else
                 n=map3(m)-2
                 d3(i)=map3(m)-3
                 d3s(i)=h2(n,m)
                 d3e(i)=l2(n,m)
         endif
! Enforce Grid pro rules and TURBO rules
         if (d1s(i)<d1e(i)) then
                 d1s(i)=d1s(i)+2
                 d1e(i)=d1e(i)+1
         elseif (d1s(i)==d1e(i)) then
                 if (d1s(i)==0) then
                         d1s(i)=d1s(i)+2
                         d1e(i)=d1e(i)+2
                 else
```

```
                        d1s(i)=d1s(i)+1
                        d1e(i)=d1e(i)+1
                endif
        else
                d1s(i)=d1s(i)+1
                d1e(i)=d1e(i)+2
        endif

        if (d2s(i)<d2e(i)) then
                d2s(i)=d2s(i)+2
                d2e(i)=d2e(i)+1
        elseif (d2s(i)==d2e(i)) then
                if (d2s(i)==0) then
                        d2s(i)=d2s(i)+2
                        d2e(i)=d2e(i)+2
                else
                        d2s(i)=d2s(i)+1
                        d2e(i)=d2e(i)+1
                endif
        else
                d2s(i)=d2s(i)+1
                d2e(i)=d2e(i)+2
        endif

        if (d3s(i)<d3e(i)) then
                d3s(i)=d3s(i)+2
                d3e(i)=d3e(i)+1
        elseif (d3s(i)==d3e(i)) then
                if (d3s(i)==0) then
                        d3s(i)=d3s(i)+2
                        d3e(i)=d3e(i)+2
                else
                        d3s(i)=d3s(i)+1
                        d3e(i)=d3e(i)+1
                endif
        else
                d3s(i)=d3s(i)+1
                d3e(i)=d3e(i)+2
        endif

        if (is(i)<ie(i)) then
                        is(i)=is(i)+2
                  ie(i)=ie(i)+1
        elseif (is(i)==ie(i)) then
                if (is(i)==0) then
                        is(i)=2
                        ie(i)=2
                elseif (.not.is(i)==0) then
                        is(i)=is(i)+1
                        ie(i)=ie(i)+1
                endif
        else
                is(i)=is(i)+1
                ie(i)=ie(i)+2
        endif

        if (js(i)<je(i)) then
                        js(i)=js(i)+2
                        je(i)=je(i)+1
        elseif (js(i)==je(i)) then
                if (js(i)==0) then
                        js(i)=2
                        je(i)=2
                elseif (.not.js(i)==0) then
                        js(i)=js(i)+1
                        je(i)=je(i)+1
                endif
        else
                js(i)=js(i)+1
                je(i)=je(i)+2
        endif
```

```
           if (ks(i)<ke(i)) then
                   ks(i)=ks(i)+2
                   ke(i)=ke(i)+1
           elseif (ks(i)==ke(i)) then
                   if (ks(i)==0) then
                           ks(i)=2
                           ke(i)=2
                   elseif (.not.ks(i)==0) then
                           ks(i)=ks(i)+1
                           ke(i)=ke(i)+1
                   endif
           else
                   ks(i)=ks(i)+1
                   ke(i)=ke(i)+2
           endif

           if (is(i)==ie(i)) then
                   dir1(i)=1
                   if (is(i)<ni(sb1(m))) then
                           lor1(i)=0
                   else
                           lor1(i)=1
                   endif
           elseif (js(i)==je(i)) then
                   dir1(i)=2
                   if (js(i)<nj(sb1(m))) then
                           lor1(i)=0
                   else
                           lor1(i)=1
                   endif
           else
                   dir1(i)=3
                    if (ks(i)<nk(sb1(m))) then
                           lor1(i)=0
                    else
                           lor1(i)=1
                    endif
           endif
        if (d1s(i)==d1e(i)) then
             dir2(i)=d1(i)+1
!    print *,'1 direction is',dir2(i),d1s(i),ni(sb2(m))
!   &,nj(sb2(m)),nk(sb2(m))
        select case(dir2(i))
        case(1)
             checkind=ni(sb2(m))
        case(2)
             checkind=nj(sb2(m))
        case(3)
             checkind=nk(sb2(m))
        end select

            if (d1s(i)<checkind) then
            lor2(i)=0
            else
            lor2(i)=1
            endif
        elseif (d2s(i)==d2e(i)) then
             dir2(i)=d2(i)+1
!    print *,'2 direction is',dir2(i),d2s(i),ni(sb2(m))
!   &,nj(sb2(m)),nk(sb2(m))
        select case(dir2(i))
        case(1)
             checkind=ni(sb2(m))
        case(2)
             checkind=nj(sb2(m))
        case(3)
             checkind=nk(sb2(m))
        end select

        if (d2s(i)<checkind) then
```

```
                  lor2(i)=0
                  else
                  lor2(i)=1
                  endif
           else
                  dir2(i)=d3(i)+1
!     print *,'3 direction is',dir2(i),d3s(i),ni(sb2(m))
!     &,nj(sb2(m)),nk(sb2(m))
              select case(dir2(i))
              case(1)
                  checkind=ni(sb2(m))
              case(2)
                  checkind=nj(sb2(m))
              case(3)
                  checkind=nk(sb2(m))
              end select

              if (d3s(i)<checkind) then
                  lor2(i)=0
                  else
                  lor2(i)=1
                  endif
           endif

           if (is(i)==2 .and. d1s(i)==2 .and. is(i)<ie(i)
     &    .and. d1s(i)<d1e(i)) then
                  is(i)=1
                        d1s(i)=1
           endif

           if (js(i)==2 .and. d2s(i)==2 .and. js(i)<je(i)
     &    .and. d2s(i)<d2e(i)) then
                  js(i)=1
                        d2s(i)=1
           endif

           if (ks(i)==2 .and. d3s(i)==2 .and. ks(i)<ke(i)
     &    .and. d3s(i)<d3e(i)) then
                  ks(i)=1
                        d3s(i)=1
           endif
!         if (update_gp_props.eq.1) then


                 if (p_pty(p_special(i)).eq.gperiodic) then
                       if (sf1(p_special(i))>0) then
                             bc(i)=ref_periodic_fwd
                       elseif(sf1(p_special(i))<0) then
                             bc(i)=ref_periodic_bak
                       endif
                 elseif (p_pty(p_special(i)).eq.gts) then
                       if (sf1(p_special(i))>0) then
                             bc(i)=ref_ts_fwd
                       elseif(sf1(p_special(i))<0) then
                             bc(i)=ref_ts_bak
                       endif
                 elseif (p_pty(p_special(i)).eq.gref_clearance) then
                       bc(i)=ref_clearance
                 else
                       bc(i)=999
                       print *,'BC TYPE FROM GridPro','p_pty(p_special(i))',
     &'not defined. Check and update gplist.in'
                 endif
!         else
!                 bc(i)=bcconvert(p_pty(p_special(i)))
!         endif

      write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
```

```
      & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)
       end do
       deallocate(id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e
      &, d3,d3s,d3e,dir1,dir2,lor1,lor2,bc)
       close(8)

   !****************Boundary conditions -> bc.in****************
       open(unit=10,file=bcfile,FORM='formatted',status='unknown')
   !     write(10,'(I3)') num_bc! not in bc.in format
       allocate(block_id(num_bc),bc_type(num_bc),start_i(num_bc),
      &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
      &,end_k(num_bc),gid(num_bc),gname(num_bc))
       do i=1,num_bc
   !       print *,i,' of ',num_bc,' bc complete'
          if (p_bc(i)<0) then
                 p_bc(i)=abs(p_bc(i))
                 sf_num=sf2(p_bc(i))
                 block_id(i)=sb2(p_bc(i))
          start_i(i)=l2(1,p_bc(i))+1
          start_j(i)=l2(2,p_bc(i))+1
          start_k(i)=l2(3,p_bc(i))+1
          end_i(i)=h2(1,p_bc(i))+1
          end_j(i)=h2(2,p_bc(i))+1
          end_k(i)=h2(3,p_bc(i))+1
          else
                 sf_num=sf1(p_bc(i))
                 block_id(i)=sb1(p_bc(i))
          start_i(i)=l1(1,p_bc(i))+1
          start_j(i)=l1(2,p_bc(i))+1
          start_k(i)=l1(3,p_bc(i))+1
          end_i(i)=h1(1,p_bc(i))+1
          end_j(i)=h1(2,p_bc(i))+1
          end_k(i)=h1(3,p_bc(i))+1
          endif
   !      bc_type=bcconvert(p_pty(p_bc(i)))
   !     Use readgp to update bc property tags and assign TURBO bc values
   !          if (update_gp_props.eq.1) then
                 if (p_pty(p_bc(i)).eq.gslip) then
                       bc_type=slip
                 elseif (p_pty(p_bc(i)).eq.gno_slip) then
                       bc_type=no_slip
                 elseif (p_pty(p_bc(i)).eq.gno_slip_iso) then
                       bc_type=no_slip_iso
                 elseif (p_pty(p_bc(i)).eq.gcvbc_in) then
                       bc_type=cvbc_in
                 elseif (p_pty(p_bc(i)).eq.gisentropic_in) then
                       bc_type=isentropic_in
                 elseif (p_pty(p_bc(i)).eq.grad_eq_exit) then
                       bc_type=rad_eq_exit
                 elseif (p_pty(p_bc(i)).eq.gpressure_exit) then
                       bc_type=pressure_exit
                 elseif (p_pty(p_bc(i)).eq.gperiodic) then
   !                  print *,gperiodic,periodic_fwd,periodic_bak
                    if (sb1(p_bc(i)).eq.sb2(p_bc(i))) then !Decides whetehr to use
ref_periodic or periodic
                          if (sf_num>0) then
                                bc_type=periodic_fwd
                          elseif(sf_num<0) then
                                bc_type=periodic_bak
                          endif
                    else
                          if (sf_num>0) then
                                bc_type=ref_periodic_fwd
                          elseif(sf_num<0) then
                                bc_type=ref_periodic_bak
                          endif
                    endif
                 elseif (p_pty(p_bc(i)).eq.gts) then
                    if (sb1(p_bc(i)).eq.sb2(p_bc(i))) then !Decides whetehr to use
ref_ts or ts
                          if (sf_num>0) then
```

```
                                                bc_type=ts_fwd
                                        elseif(sf_num<0) then
                                                bc_type=ts_bak
                                        endif
                                else
                                        if (sf_num>0) then
                                                bc_type=ref_ts_fwd
                                        elseif(sf_num<0) then
                                                bc_type=ref_ts_bak
                                        endif
                                endif
                        elseif (p_pty(p_bc(i)).eq.gref_clearance) then
                                bc_type=ref_clearance
                        elseif (p_pty(p_bc(i)).eq.gplenum_in) then
                                bc_type=plenum_in
                        elseif (p_pty(p_bc(i)).eq.gwb_steady_in) then
                                bc_type=wb_steady_in
                        elseif (p_pty(p_bc(i)).eq.gwb_unsteady_in) then
                                bc_type=wb_unsteady_in
                        elseif (p_pty(p_bc(i)).eq.gwb_steady_exit) then
                                bc_type=wb_steady_exit
                        elseif (p_pty(p_bc(i)).eq.gwb_unsteady_exit) then
                                bc_type=wb_unsteady_exit
                        elseif (p_pty(p_bc(i)).eq.gcvbc_sub_exit) then
                                bc_type=cvbc_sub_exit
                        elseif (p_pty(p_bc(i)).eq.gcvbc_super_exit) then
                                bc_type=cvbc_super_exit
                        elseif (p_pty(p_bc(i)).eq.gslide) then
                                bc_type=slide
                        elseif (p_pty(p_bc(i)).eq.gslide_ts_i) then
                                bc_type=slide_ts_i
                        elseif (p_pty(p_bc(i)).eq.gslide_ts_j) then
                                bc_type=slide_ts_j
                        else
                                bc_type=999
                        print *,'BC TYPE FROM GridPro','p_pty(p_bc(i))',
        &'not defined. Check and update gplist.in'
                        endif
!               else
!                       bc_type=bcconvert(p_pty(p_bc(i)))
!               endif


!        print *,block_id(i),
!        &bc_type(i),start_i(i),
!        &start_j(i),start_k(i),end_i(i),end_j(i)
         write(10,'(2x,1I3,1I6,6I5,1("/"))'),block_id(i),
        &bc_type(i),start_i(i),
        &start_j(i),start_k(i),end_i(i),end_j(i)
        &,end_k(i)

         enddo
         write(10,'(2x,1I3,1I6,6I5,1("/"))'),0,
        &0,0,0,0,0,0,0

         close(10)
         deallocate(block_id,bc_type,start_i,gid,gname,
        &                       start_j,start_k,end_i,end_j,end_k)


!******************  Functions   ***************************
         contains

         function bcconvert(bc)
         integer::bc,bcconvert
         select case(bc)
         case(0)
         bcconvert=0
         case(1)
         bcconvert=0
```

```
      case(2)
      bcconvert=2
      case(3)
      bcconvert=101
      case(4)
      bcconvert=1
      case(5)
      bcconvert=201
      case(6)
      bcconvert=301
      case(7)
      bcconvert=304
      end select
      end function bcconvert

      function string_to_int(ch1)

      integer :: string_to_int,i
      character(LEN=1) :: ch1,ch2
      do i=0,9
      write(ch2,'(i1)') i
      if (ch2==ch1) then
      string_to_int=i
      end if
      end do

      end function string_to_int

      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
      character(len=*),intent (in) :: pre,post
      character(len=100) :: file_cat
      ints=len_trim(pre)
      inte=len_trim(post)
      write(file_cat,*)pre(1:ints),post(1:inte)
      return
      end function file_cat


      end subroutine gp2turbo
!*****************End function definitions ********************

!*****************SUBROUTINES****************************
      subroutine readgp(slip,no_slip,no_slip_iso,isentropic_in,
     &rad_eq_exit,periodic,cvbc_in,pressure_exit,
     &plenum_in,ts, ref_clearance,wb_steady_in,wb_unsteady_in,
     &wb_steady_exit,wb_unsteady_exit,cvbc_sub_exit,
     &cvbc_super_exit,slide,slide_ts_i,slide_ts_j,ginter_blk)
      implicit none


      character(len=40)  :: fname                 ! namelist input file name
      logical:: fname_exists         ! logical telling whether namelist input file exists
      logical:: file_end = .false.
      integer:: slip,no_slip,no_slip_iso,isentropic_in,rad_eq_exit
      integer::periodic,cvbc_in,pressure_exit,plenum_in
      integer::ref_clearance,ts,wb_steady_in,wb_unsteady_in,
     & wb_steady_exit,wb_unsteady_exit,cvbc_sub_exit,
     & cvbc_super_exit,slide,slide_ts_i,slide_ts_j,inter_blk

      integer:: gslip,gno_slip,gno_slip_iso,gisentropic_in,grad_eq_exit
      integer::gperiodic,gcvbc_in,gpressure_exit,gplenum_in
      integer::gref_clearance,gts,gwb_steady_in,gwb_unsteady_in,
     & gwb_steady_exit,gwb_unsteady_exit,gcvbc_sub_exit,
     & gcvbc_super_exit,gslide, gslide_ts_i,gslide_ts_j,
     & ginter_blk

      namelist/GP_PROPS/
     & gslip,gno_slip,gno_slip_iso,gisentropic_in,grad_eq_exit,
     & gperiodic,gcvbc_in,gpressure_exit,
```

```
      & gplenum_in, gref_clearance,gts,
      & gwb_steady_in,gwb_unsteady_in,gwb_steady_exit,
      & gwb_unsteady_exit,gcvbc_sub_exit,gcvbc_super_exit,gslide,
      & gslide_ts_i,gslide_ts_j,
      & ginter_blk
       fname = 'gplist.in'
       !defaults
       gslip=999
       gno_slip=999
       gno_slip_iso=999
       gisentropic_in=999
       gcvbc_in=999
       grad_eq_exit=999
       gperiodic=999
       gplenum_in=999
      gref_clearance=999
      gts=999
      gcvbc_in=999
      gisentropic_in=999
      gwb_steady_in=999
      gwb_unsteady_in=999
      gplenum_in=999
      grad_eq_exit=999
      gwb_steady_exit=999
      gwb_unsteady_exit=999
      gpressure_exit=999
      gcvbc_sub_exit=999
      gcvbc_super_exit=999
      gslide=999
      gslide_ts_i=999
      gslide_ts_j=999
         ginter_blk=1
!        print *,'Default Properties are:'
!        print *,'gslip=',gslip
!        print *,'gno_slip=',gno_slip
!        print *,'gno_slip_iso=',gno_slip_iso
!        print *,'gisentropic_in=',gisentropic_in
!        print *,'gcvbc_in=',gcvbc_in
!        print *,'grad_eq_exit=',grad_eq_exit
!        print *,'gperiodic=',gperiodic
!        print *,'gref_periodic=',gref_periodic
!        print *,'gpressure_exit=',gpressure_exit

!         print *,'Reading GridPro Properties from ',fname
         open(UNIT=17,file=fname,form='formatted')
         rewind(17)
         do while (.not. file_end)
         read(17,nml=GP_PROPS,err=301,end=303)
         close(17)

         slip=gslip
         no_slip=gno_slip
         no_slip_iso=gno_slip_iso
         isentropic_in=gisentropic_in
         cvbc_in=gcvbc_in
         rad_eq_exit=grad_eq_exit
         periodic=gperiodic
         plenum_in=gplenum_in
         ref_clearance=gref_clearance
         ts=gts
         wb_steady_in=gwb_steady_in
         wb_unsteady_in=gwb_unsteady_in
         wb_steady_exit=gwb_steady_exit
         wb_unsteady_exit=gwb_unsteady_exit
         pressure_exit=gpressure_exit
         cvbc_sub_exit=gcvbc_sub_exit
         cvbc_super_exit=gcvbc_super_exit
         slide=gslide
         slide_ts_i=gslide_ts_i
         slide_ts_j=gslide_ts_j
         inter_blk=ginter_blk
```

```
      goto 302

 301  continue
      enddo

 303  close(17)
 302  continue

!      print *,'Updated Properties are:'
!      print *,'slip=',slip
!      print *,'no_slip=',no_slip
!      print *,'no_slip_iso=',no_slip_iso
!      print *,'isentropic_in=',isentropic_in
!      print *,'cvbc_in=',cvbc_in
!      print *,'rad_eq_exit=',rad_eq_exit
!      print *,'periodic=',periodic
!      print *,'pressure_exit=',pressure_exit

      end subroutine readgp

!**************Program to use gridpro conn_n file*******************
!***********to check msuTurbo bc.in and dmap.in*******************
!*******************by Vikram Shyam 08/07/07*********************
!*****************Last Modified on 04/02/08*********************

      subroutine checkdmap(infile,tol,num_blades,blade_row)
      implicit none
!     dmap.in VARS
      integer :: num_b2b, num_special_b2b
      integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
      integer, dimension(:),allocatable :: d1e,id,dir2,lor1,lor2,p_b2b
      integer, dimension(:),allocatable:: p_special,bc,p_bc
      integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!     bc.in VARS
      integer :: num_bc
      integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
      integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit

!     local VARS
      character(len=100)::gpro,dmapfile,infile,bcfile,p3dfile,file_name0
      integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
      integer :: i1,j1,k1,i2,j2,k2
      integer, dimension(:),allocatable::ni,nj,nk,
     &         block_start,block_end,block_size
      real, dimension(:),allocatable::x,y,z
      real, dimension(:),allocatable::g1,g2,g3
      real x1,x2,y1,y2,z1,z2,angle
      logical xn1,yn1,zn1,xn2,yn2,zn2,off,on
      real :: tol !grid tolerance
      integer :: num_blades,blade_row
      integer:: all_good
      real*8:: pi
      pi=3.14159265358979323846
      off=.FALSE.
      on=.TRUE.
      all_good=1
      p3dfile=file_cat(infile,'.p3d')
!      dmapfile='dmap.in'
!      bcfile='bc.in'
!      tol=0.0000001
      dmapfile=file_name0('dmap.',blade_row)
      bcfile=file_name0('bc',blade_row)

      print *,'Grid tolerance is set at ',tol
!      print*,' Input name of P3d grid file :'
!      read *, infile
!      print*,'No. blades in blade row:'
!      read*,num_blades

      angle=360./num_blades
      print *, 'Angle of periodicity is:',angle,' degrees.'
```

```
      open(UNIT=10, FILE=p3dfile, FORM='unformatted')

      read(10), nblks
      print *,'File opened succesfully'
      print *,nblks,' Blocks found'
      allocate(ni(nblks),nj(nblks),nk(nblks),
     &block_start(nblks),block_end(nblks),block_size(nblks))

      read(10), (ni(nb),nj(nb),nk(nb),nb=1,nblks)
      print *, 'Block#  ni   nj     nk'
      do nb=1,nblks
      print *,nb,ni(nb),nj(nb),nk(nb)
      enddo

      sumn=0

      print *,'Block #       Block extents      Block size'
      do nb=1,nblks
!        print *,'nblks=',nb
        block_size(nb)=ni(nb)*nj(nb)*nk(nb)
        block_start(nb)=sumn+1
        sumn=sumn+block_size(nb)
        block_end(nb)=sumn

      print *,nb,block_start(nb),block_end(nb),block_size(nb)
      end do

      print *, sumn,' data points will be read.'

      allocate(x(sumn),y(sumn),z(sumn))

      do nb=1,nblks

!        print *,'nblks=',nb
        ii=block_end(nb)-block_start(nb)+1
        allocate(g1(ii),g2(ii),g3(ii))
        read(10) (g1(m),m=1,ii),(g2(m),m=1,ii),(g3(m),m=1,ii)
        m=0
        do ijk=block_start(nb),block_end(nb)
          m=m+1
          x(ijk)=g1(m)
          y(ijk)=g2(m)
          z(ijk)=g3(m)
        enddo
        deallocate(g1,g2,g3)
      enddo
      close(10)
      print *,'Plot3d File closed after reading'
      print *,'*************************************************'

!**************  READING dmap.in    ********************


!***************Normal b2b Checking*************************
      print *,'*****************Reading dmap.in*****************'
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
!       print *,'Number of block to block interfaces'
      read(8,*) num_b2b
      write(*,'(I5)') num_b2b
      allocate(id(num_b2b),is(num_b2b),ie(num_b2b),js(num_b2b)
     & ,je(num_b2b),ks(num_b2b),ke(num_b2b),blkb(num_b2b),d1(num_b2b),
     & d1s(num_b2b),d1e(num_b2b),d2(num_b2b),d2s(num_b2b),d2e(num_b2b),
     & d3(num_b2b),d3s(num_b2b),d3e(num_b2b),dir1(num_b2b),dir2(num_b2b)
     & ,lor1(num_b2b),lor2(num_b2b))
!       print *,'id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e,d3,d3s,
!      &d3e,dir1,dir2,lor1,lor2'
      do i=1,num_b2b
      read(8,*) id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
```

```
      & ,dir1(i),dir2(i),lor1(i),lor2(i)
!     write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
!     & ie(i),js(i)
!     & ,je(i),ks(i),ke(i),blkb(i),
!     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
!     & ,dir1(i),dir2(i),lor1(i),lor2(i)
      enddo
! 1.  CHECKING GRID CONNECTIONS
! Convert to node centered
      do i=1,num_b2b
      call shift(is(i),ie(i),lor1(i))
      call shift(js(i),je(i),lor1(i))
      call shift(ks(i),ke(i),lor1(i))
      call shift(d1s(i),d1e(i),lor2(i))
      call shift(d2s(i),d2e(i),lor2(i))
      call shift(d3s(i),d3e(i),lor2(i))
! write out to verify
!      print *,'Node centered dmap is as follows: '
!      write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
!     & ie(i),js(i)
!     & ,je(i),ks(i),ke(i),blkb(i),
!     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
!     & ,dir1(i),dir2(i),lor1(i),lor2(i)

!COMPARE LOWER DIAGONAL PT
      call find_ijk(ijk,is(i),js(i),ks(i),ni(id(i)),nj(id(i)),nk(id(i))
     &,block_start(id(i)),block_end(id(i)))
      x1=x(ijk)
      y1=y(ijk)
      z1=z(ijk)
      call mapijk(i1,j1,k1,d1(i),d1s(i),d2(i),d2s(i),d3(i),d3s(i))
      call find_ijk(ijk,i1,j1,k1,ni(blkb(i)),nj(blkb(i))
     &,nk(blkb(i)),block_start(blkb(i)),block_end(blkb(i)))
      x2=x(ijk)
      y2=y(ijk)
      z2=z(ijk)
      call compare(x1,x2,xn1,tol)
      call compare(y1,y2,yn1,tol)
      call compare(z1,z2,zn1,tol)
!COMPARE UPPER DIAGONAL PT

      call find_ijk(ijk,ie(i),je(i),ke(i),ni(id(i)),nj(id(i)),nk(id(i))
     &,block_start(id(i)),block_end(id(i)))
      x1=x(ijk)
      y1=y(ijk)
      z1=z(ijk)
      call mapijk(i2,j2,k2,d1(i),d1e(i),d2(i),d2e(i),d3(i),d3e(i))
      call find_ijk(ijk,i2,j2,k2,ni(blkb(i)),nj(blkb(i))
     &,nk(blkb(i)),block_start(blkb(i)),block_end(blkb(i)))
      x2=x(ijk)
      y2=y(ijk)
      z2=z(ijk)
      call compare(x1,x2,xn2,tol)
      call compare(y1,y2,yn2,tol)
      call compare(z1,z2,zn2,tol)

!************CHECKING MAPPING*******************
      if ((is(i).eq.ie(i)).and.(dir1(i)/=1)) then
      print *, 'ERROR!! dir1 should be 1, current value is ',dir1(i)
      all_good=0
      endif
      if ((js(i).eq.je(i)).and.(dir1(i)/=2)) then
      print *, 'ERROR!! dir1 should be 2, current value is ',dir1(i)
      all_good=0
      endif
      if ((ks(i).eq.ke(i)).and.(dir1(i)/=3)) then
      print *, 'ERROR!! dir1 should be 3, current value is ',dir1(i)
      all_good=0
      endif

      if ((d1s(i).eq.d1e(i)).and.(dir2(i)/=(d1(i)+1))) then
```

```
      print *, 'ERROR!! dir2 should be 1, current value is ',dir2(i)
      all_good=0
      endif
      if ((d2s(i).eq.d2e(i)).and.(dir2(i)/=(d2(i)+1))) then
      print *, 'ERROR!! dir2 should be 2, current value is ',dir2(i)
      all_good=0
      endif
      if ((d3s(i).eq.d3e(i)).and.(dir2(i)/=(d3(i)+1))) then
      print *, 'ERROR!! dir2 should be 3, current value is ',dir2(i)
      all_good=0
      endif


!******************************
      if (((xn1==.TRUE. .AND. yn1==.TRUE.) .AND. zn1==.TRUE.) .AND.
     &((xn2==.TRUE. .AND. yn2==.TRUE.) .AND. zn2==.TRUE.)) then
!      print *,'Connectivity confirmed for block ',id(i),' to ',blkb(i)
      else
      print *,'ERROR! block ',id(i),'not properly connected to block '
     &,blkb(i)
      all_good=0
      endif
      enddo

      deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)


!*****************Special b2b Checking  *************************
      read(8,*), num_special_b2b
!      print *,'Number of special block to block interfaces.'
!      write(*,'(I3)'), num_special_b2b

      allocate(id(num_special_b2b),is(num_special_b2b),
     &ie(num_special_b2b),js(num_special_b2b),je(num_special_b2b)
     &,ks(num_special_b2b),ke(num_special_b2b),blkb(num_special_b2b)
     &,d1(num_special_b2b),d1s(num_special_b2b),d1e(num_special_b2b)
     &,d2(num_special_b2b),d2s(num_special_b2b),d2e(num_special_b2b)
     &, d3(num_special_b2b),d3s(num_special_b2b),d3e(num_special_b2b)
     &,dir1(num_special_b2b),dir2(num_special_b2b),
     & lor1(num_special_b2b),lor2(num_special_b2b),bc(num_special_b2b))
!      print *,'id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e,d3,d3s,
!     &d3e,dir1,dir2,lor1,lor2,bc'

      do i=1,num_special_b2b
      read(8,*) id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

!      write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
!     &ie(i),js(i)
!     & ,je(i),ks(i),ke(i),blkb(i),
!     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
!     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

      call shift(is(i),ie(i),lor1(i))
      call shift(js(i),je(i),lor1(i))
      call shift(ks(i),ke(i),lor1(i))
      call shift(d1s(i),d1e(i),lor2(i))
      call shift(d2s(i),d2e(i),lor2(i))
      call shift(d3s(i),d3e(i),lor2(i))
! write out to verify
!      print *,'Node centered dmap is as follows: '
!      write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
!      & ie(i),js(i)
!      & ,je(i),ks(i),ke(i),blkb(i),
!      & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
!      & ,dir1(i),dir2(i),lor1(i),lor2(i)
```

```fortran
!COMPARE LOWER DIAGONAL PT
      call find_ijk(ijk,is(i),js(i),ks(i),ni(id(i)),nj(id(i)),nk(id(i))
     &,block_start(id(i)),block_end(id(i))))
       call cart2polar(x(ijk),y(ijk),z(ijk),x1,y1,z1)
       call mapijk(i1,j1,k1,d1(i),d1s(i),d2(i),d2s(i),d3(i),d3s(i))
       call find_ijk(ijk,i1,j1,k1,ni(blkb(i)),nj(blkb(i))
     &,nk(blkb(i)),block_start(blkb(i)),block_end(blkb(i))))
       call cart2polar(x(ijk),y(ijk),z(ijk),x2,y2,z2)
       call compare(x1,x2,xn1,tol)
       call compare(y1,y2,yn1,tol)
       if (bc(i).eq.-104 .or. bc(i).eq.-106) then
       z1=z1+angle*pi/180.
       if (z1>(2.*pi)) z1=z1-2.*pi
       elseif (bc(i).eq.-105.or.bc(i).eq.-107) then
       z2=z2+angle*pi/180.
       if (z2>(2.*pi)) z2=z2-2.*pi
       endif
!       print*,xn1,yn1,zn2
!       print *,x1,x2,y1,y2,z1,z2
!        zn1=.TRUE.
       call compare(z1,z2,zn1,tol)
!COMPARE UPPER DIAGONAL PT

      call find_ijk(ijk,ie(i),je(i),ke(i),ni(id(i)),nj(id(i)),nk(id(i))
     &,block_start(id(i)),block_end(id(i))))
       call cart2polar(x(ijk),y(ijk),z(ijk),x1,y1,z1)
       call mapijk(i2,j2,k2,d1(i),d1e(i),d2(i),d2e(i),d3(i),d3e(i))
       call find_ijk(ijk,i2,j2,k2,ni(blkb(i)),nj(blkb(i))
     &,nk(blkb(i)),block_start(blkb(i)),block_end(blkb(i))))
       call cart2polar(x(ijk),y(ijk),z(ijk),x2,y2,z2)
!       print *,x1,x2,y1,y2,z1,z2
       call compare(x1,x2,xn2,tol)
       call compare(y1,y2,yn2,tol)
       if (bc(i).eq.-104 .or. bc(i).eq.-106) then
       z1=z1+angle*pi/180.
       if (z1>(2.*pi)) z1=z1-2.*pi
       elseif (bc(i).eq.-105 .or. bc(i).eq.-107) then
       z2=z2+angle*pi/180.
       if (z2>(2.*pi)) z2=z2-2.*pi
       endif
!       print*,xn2,yn2,zn2
!        zn2=.TRUE.
       call compare(z1,z2,zn2,tol)
!************CHECKING MAPPING*********************
       if ((is(i).eq.ie(i)).and.(dir1(i)/=1)) then
       print *, 'ERROR!! dir1 should be 1, current value is ',dir1(i)
       all_good=0
       endif
       if ((js(i).eq.je(i)).and.(dir1(i)/=2)) then
       print *, 'ERROR!! dir1 should be 2, current value is ',dir1(i)
       all_good=0
       endif
       if ((ks(i).eq.ke(i)).and.(dir1(i)/=3)) then
       print *, 'ERROR!! dir1 should be 3, current value is ',dir1(i)
       all_good=0
       endif

       if ((d1s(i).eq.d1e(i)).and.(dir2(i)/=(d1(i)+1))) then
       print *, 'ERROR!! dir2 should be 1, current value is ',dir2(i)
       all_good=0
       endif
       if ((d2s(i).eq.d2e(i)).and.(dir2(i)/=(d2(i)+1))) then
       print *, 'ERROR!! dir2 should be 2, current value is ',dir2(i)
       all_good=0
       endif
       if ((d3s(i).eq.d3e(i)).and.(dir2(i)/=(d3(i)+1))) then
       print *, 'ERROR!! dir2 should be 3, current value is ',dir2(i)
       all_good=0
       endif
!*****************************
       if (((xn1==.TRUE. .AND. yn1==.TRUE.) .AND. zn1==.TRUE.) .AND.
```

```
      &((xn2==.TRUE. .AND. yn2==.TRUE.) .AND. zn2==.TRUE.)) then
!       print *,'Periodic Connectivity confirmed for block ',id(i),
!      &' to ',blkb(i)
       else
       print *,'ERROR! block ',id(i),'not properly connected to block '
      &,blkb(i)
       all_good=0
       endif
       enddo


       deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
      &          lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)
       deallocate(x,y,z)
       deallocate(ni,nj,nk,block_start,block_end)
       close(8)
       if (all_good.eq.1) then
       print *,'Connectivity has been verified for current row'
       else
       print*,'There are problems in the connectivity.
      &Look above for details'
       endif

       print *,'Memory deallocation complete.'


       contains

       function file_cat(pre,post)
       implicit none
       integer n,ints,inte
       character(len=*),intent (in) :: pre,post
       character(len=100) :: file_cat
       ints=len_trim(pre)
       inte=len_trim(post)
       write(file_cat,*)pre(1:ints),post(1:inte)
       return
       end function file_cat

       end subroutine checkdmap

!       contains
       subroutine compare(x1,x2,tf,tol) !tolerance of grid points at interfaces
       logical :: tf
       real:: x1,x2,del
       real, INTENT(IN),optional::tol
       if (present(tol)) then
       del=tol
       else
       del=0.
       endif
       if (abs(x1-x2).le.del) then
       tf=.TRUE.
       else
       tf=.FALSE.
       endif
       end subroutine compare

       subroutine mapijk(i,j,k,d1,d1s,d2,
      &d2s,d3,d3s)

       integer i,j,k,d1,d1s,d2,d2s,d3,d3s

       if (d1==0) then
       i=d1s
       elseif (d2==0) then
       i=d2s
       elseif (d3==0) then
       i=d3s
       endif
       if (d1==1) then
```

```
      j=d1s
      elseif (d2==1) then
      j=d2s
      elseif (d3==1) then
      j=d3s
      endif
      if (d1==2) then
      k=d1s
      elseif (d2==2) then
      k=d2s
      elseif (d3==2) then
      k=d3s
      endif
      end subroutine mapijk

      subroutine shift(is,ie,m)
      integer is,ie,m
      if (is>1) then
           if (is<ie) then
         is=is-1
        elseif (is>ie) then
             ie=ie-1
        else
            if(m==0) then
              ie=ie-1
              is=is-1
           endif
        endif
       endif
      end subroutine shift


       subroutine find_ijk(ijk,i,j,k,ni,nj,nk,bs
     &             ,be)
      integer, intent(out)::ijk
      integer, intent(in)::i,j,k
      integer ::ni,nj,nk
      integer::bs,be
      ijk=(k-1)*nj*ni+(j-1)*ni+i
      ijk=ijk+(bs-1)
      end subroutine find_ijk

       subroutine find_i_j_k(i,j,k,ijk,ni,nj,nk,m,bs
     &             ,be)
      integer::ijk,m
      integer::i,j,k
      integer ::ni,nj,nk,ri,rj,rk
      integer :: bs,be

      ijk=ijk-(bs-1)
      rk=mod(ijk,(ni*nj))
      k=(ijk-rk)/(ni*nj)+1
      rj=mod(rk,ni)
      j=(rk-rj)/ni+1
      i=rj

      end subroutine find_i_j_k

   !Program to convert multigrid plot3d files to GU
   !Vikram Shyam - 9/4/07

      subroutine p3d2gu(infile,nbr,startindex)
      implicit none
      integer:: i,j,k,n,nb,nbr,bld_psg,nbgu
      integer::ni1,nj1,nk1,fid,startindex
      integer,allocatable,dimension(:)::ni,nj,nk
      real, allocatable,dimension(:,:,:) :: x,y,z,g1,g2,g3
      real*8, allocatable,dimension(:,:,:) :: x1,y1,z1
      character(len=100)::fname,p3dfile,file_name0,infile
      real*8:: pi
      pi=3.14159265358979323846
```

```
      nbgu=1
!      nbr=1
       bld_psg=1
!      print *,'BEWARE: THIS WILL OVERWRITE ANY
!      &          GU FILES CURRENTLY IN THIS FOLDER'


!      print *,'Enter Plot3d File name (must be real*8):'
!      read *,oname
!      oname='ast.coarse.x'
       p3dfile=file_cat(infile,'.p3d')

      print *,'Opening',p3dfile,' as plot3d'
       open(UNIT=9, FILE=p3dfile, FORM='unformatted', STATUS='unknown')
       read(9) nb
       allocate(ni(nb),nj(nb),nk(nb))
       read(9) (ni(i),nj(i),nk(i),i=1,nb)
!      print *,nb,' blocks found.'
!      print *,'Block extents:'
!      print *,'block#   ni   nj   nk'
!      do n=1,nb
!      write(*,*) n,ni(n),nj(n),nk(n)
!      enddo
       do n=1,nb
       allocate(x(ni(n),nj(n),nk(n)),
      &         y(ni(n),nj(n),nk(n)),z(ni(n),nj(n),nk(n)))

       allocate(g1(ni(n),nj(n),nk(n)),
      &         g2(ni(n),nj(n),nk(n)),g3(ni(n),nj(n),nk(n)))
       allocate(x1(ni(n),nj(n),nk(n)),
      &         y1(ni(n),nj(n),nk(n)),z1(ni(n),nj(n),nk(n)))
         read(9) (((x(i,j,k),i=1,ni(n)),j=1,nj(n)),k=1,nk(n)),
      &    (((y(i,j,k),i=1,ni(n)),j=1,nj(n)),k=1,nk(n)),
      &    (((z(i,j,k),i=1,ni(n)),j=1,nj(n)),k=1,nk(n))
       x1=x
       y1=y
       z1=z
       fid = 10*(startindex+n)
       fname = file_name0('GU',fid)
!      print *,' opening',fname,' for writing as GU'
       open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
       write(7) nbgu,nbr,bld_psg
!      print *,'Blocks  blade_row_id  blade_passage'
!      print *, nbgu,nbr,bld_psg
       write(7) ni(n),nj(n),nk(n)
      do k=1,nk(n)
      do j=1,nj(n)
      do i=1,ni(n)
      call cart2polar(x(i,j,k),y(i,j,k),z(i,j,k),g1(i,j,k)
      & ,g2(i,j,k),g3(i,j,k))
      enddo
      enddo
      enddo
       write(7)
      &          (((g1(i,j,k),i=1,ni(n)),j=1,nj(n)),
      &          k=1,nk(n)),
      &          (((g2(i,j,k),i=1,ni(n)),j=1,nj(n)),
      &          k=1,nk(n)),
      &          (((g3(i,j,k),i=1,ni(n)),j=1,nj(n)),
      &          k=1,nk(n))
!      print *,'Closing ',fname
       close(7)
       deallocate(x,y,z,x1,y1,z1,g1,g2,g3)

       end do
       print *,nb,' GU files written for BR ',nbr,'.'
       startindex=startindex+nb
       deallocate(ni,nj,nk)
       close(9)

      contains
```

```fortran
      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
      character(len=*),intent (in) :: pre,post
      character(len=100) :: file_cat
      ints=len_trim(pre)
      inte=len_trim(post)
      write(file_cat,*)pre(1:ints),post(1:inte)
      return
      end function file_cat

      end subroutine p3d2gu

      subroutine cart2polar(x,y,z,x1,r,theta)
      real::x,y,z,r,theta,x1
      real*8:: pi
      pi=3.14159265358979323846

      if (y>0.0 .and. z>0.0) then
      theta=atan(y/z)
      elseif(y<0.0 .and. z>0.0) then
      theta=2.*pi+atan(y/z)
      elseif(y>0.0 .and. z<0.0) then
      theta=pi+atan(y/z)
      elseif(y<0.0 .and. z<0.0) then
      theta=pi+atan(y/z)
      elseif(y<0.0 .and. z.eq.0.0) then
      theta=3.*pi/2.
      elseif(y>0.0 .and. z.eq.0.0) then
      theta=pi/2.
      elseif(y.eq.0.0 .and. z<0.0) then
      theta=pi
      elseif(y.eq.0.0 .and. z>0.0) then
      theta=0.
      endif
      r=sqrt(y**2+z**2)
      x1=x
      end subroutine cart2polar
!*****************Program to merge bc and dmap *******************
!*****************for multiple blade rows************************
!*****************by Vikram Shyam 04/01/07**********************
!*****************Last Modified on 04/07/08*********************

      subroutine mergein(add_2_id,num_blade_rows,n_bc,num_bc_tot)
      implicit none
!     dmap.in VARS
      integer :: num_b2b,num_special_b2b,num_b2b_tot,num_special_b2b_tot
      integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
      integer, dimension(:),allocatable:: d1e,id,dir2,lor1,lor2,p_b2b
      integer, dimension(:),allocatable:: p_special,bc,p_bc
      integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!     local VARS
      character(len=50) :: gpro,dmapfile,infile,bcfile
      integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
      integer :: i1,j1,k1,i2,j2,k2
      integer :: num_2_merge,num_blade_rows
      integer :: add_2_id(num_blade_rows+1),n_bc(num_blade_rows)
!      integer :: add_2_id(100)
      integer :: bcid !dummy
!     bc.in VARS
      integer :: num_bc,num_bc_tot!,n_bc(100)
!      integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
!      integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit
      integer, dimension(:),allocatable:: block_id,bc_type,start_i,
     &                           start_j,start_k,end_i,end_j,end_k
      integer, dimension(:),allocatable::gid,gname

      num_b2b_tot=0
      num_special_b2b_tot=0
      do num_2_merge=1,num_blade_rows
```

```
      dmapfile=file_name0('dmap.',num_2_merge)
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
      read(8,*) num_b2b
      num_b2b_tot=num_b2b_tot+num_b2b
      allocate(id(num_b2b),is(num_b2b),ie(num_b2b),js(num_b2b)
     & ,je(num_b2b),ks(num_b2b),ke(num_b2b),blkb(num_b2b),d1(num_b2b),
     & d1s(num_b2b),d1e(num_b2b),d2(num_b2b),d2s(num_b2b),d2e(num_b2b),
     & d3(num_b2b),d3s(num_b2b),d3e(num_b2b),dir1(num_b2b),dir2(num_b2b)
     & ,lor1(num_b2b),lor2(num_b2b))
   do i=1,num_b2b
      read(8,*) id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
      enddo
   read(8,*), num_special_b2b
   num_special_b2b_tot=num_special_b2b_tot+num_special_b2b

      deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)
      close(8)
      enddo

      open(unit=9,file='dmap.in',FORM='formatted',status='unknown')
      write(9,'(I5)') num_b2b_tot

      do num_2_merge=1,num_blade_rows
      dmapfile=file_name0('dmap.',num_2_merge)
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
      read(8,*) num_b2b
      allocate(id(num_b2b),is(num_b2b),ie(num_b2b),js(num_b2b)
     & ,je(num_b2b),ks(num_b2b),ke(num_b2b),blkb(num_b2b),d1(num_b2b),
     & d1s(num_b2b),d1e(num_b2b),d2(num_b2b),d2s(num_b2b),d2e(num_b2b),
     & d3(num_b2b),d3s(num_b2b),d3e(num_b2b),dir1(num_b2b),dir2(num_b2b)
     & ,lor1(num_b2b),lor2(num_b2b))
      do i=1,num_b2b
      read(8,*) id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)

      id(i)=id(i)+add_2_id(num_2_merge)
      blkb(i)=blkb(i)+add_2_id(num_2_merge)


      write(9,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)

      enddo

      deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)
   close(8)
   enddo

   write(9,'(I4)') num_special_b2b_tot

      do num_2_merge=1,num_blade_rows
   dmapfile=file_name0('dmap.',num_2_merge)

   open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
   read(8,*) num_b2b

      allocate(id(num_b2b),is(num_b2b),ie(num_b2b),js(num_b2b)
     & ,je(num_b2b),ks(num_b2b),ke(num_b2b),blkb(num_b2b),d1(num_b2b),
```

```
     & d1s(num_b2b),d1e(num_b2b),d2(num_b2b),d2s(num_b2b),d2e(num_b2b),
     & d3(num_b2b),d3s(num_b2b),d3e(num_b2b),dir1(num_b2b),dir2(num_b2b)
     & ,lor1(num_b2b),lor2(num_b2b))

     do i=1,num_b2b
       read(8,*) id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
       enddo

     deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1)

     read(8,*), num_special_b2b

       allocate(id(num_special_b2b),is(num_special_b2b),
     &ie(num_special_b2b),js(num_special_b2b),je(num_special_b2b)
     &,ks(num_special_b2b),ke(num_special_b2b),blkb(num_special_b2b)
     &,d1(num_special_b2b),d1s(num_special_b2b),d1e(num_special_b2b)
     &,d2(num_special_b2b),d2s(num_special_b2b),d2e(num_special_b2b)
     &, d3(num_special_b2b),d3s(num_special_b2b),d3e(num_special_b2b)
     &,dir1(num_special_b2b),dir2(num_special_b2b),
     & lor1(num_special_b2b),lor2(num_special_b2b),bc(num_special_b2b))

       do i=1,num_special_b2b
     read(8,*) id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)
     id(i)=id(i)+add_2_id(num_2_merge)
     blkb(i)=blkb(i)+add_2_id(num_2_merge)

       write(9,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)
       end do
       deallocate(id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e
     &, d3,d3s,d3e,dir1,dir2,lor1,lor2,bc)

     close(8)
     enddo

     close(9)
! END OF DMAP.IN MERGE

!BC.IN MERGE
     num_bc_tot=0
     open(unit=10,file='bc.in',FORM='formatted',status='unknown')
     do num_2_merge=1,num_blade_rows
     num_bc=0 !num of bc in blade row num_2_merge
     bcfile=file_name0('bc.',num_2_merge)
     open(unit=8,file=bcfile,FORM='formatted',status='unknown')
 10  read(8,*) bcid
     if (bcid.eq.0) then
     goto 20
     else
     num_bc=num_bc+1
     goto 10
     endif
 20  continue
     close(8)

     num_bc_tot=num_bc_tot+num_bc

     open(unit=8,file=bcfile,FORM='formatted',status='unknown')
!      write(10,'(I3)') num_bc! not in bc.in format
```

```
      allocate(block_id(num_bc),bc_type(num_bc),start_i(num_bc),
     &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
     &,end_k(num_bc),gid(num_bc),gname(num_bc))
      do i=1,num_bc

      read(8,*)block_id(i),
     &bc_type(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
     &,end_k(i)
      block_id(i)=block_id(i)+add_2_id(num_2_merge)

      write(10,'(2x,1I3,1I6,6I5,1("/"))'),block_id(i),
     &bc_type(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
     &,end_k(i)
      enddo
      close(8)

      deallocate(block_id,bc_type,start_i,gid,gname,
     &            start_j,start_k,end_i,end_j,end_k)
      enddo

      write(10,'(2x,1I3,1I6,6I5,1("/"))'),0,
     &0,0,0,0,0,0,0
      close(10)



!END OF BC.IN MERGE

   contains


      function file_name0(pre,n) ! copied from TURBO!!!
      implicit none
      integer n,ints

      character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                             '(a,i4)','(a,i5)','(a,i6)'/)
      character(len=*),intent (in) :: pre
      character(len=100) :: file_name0
      if (n.gt.0) ints = log10(real(n))
      if (n.eq.0) ints = 0
      ints = ints+1
      write(file_name0,form(ints))pre,n
      return
      end function file_name0
      end subroutine mergein


      function file_name0(pre,n) ! copied from TURBO!!!
      implicit none
      integer n,ints

      character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                             '(a,i4)','(a,i5)','(a,i6)'/)
      character(len=*),intent (in) :: pre
      character(len=100) :: file_name0
      if (n.gt.0) ints = log10(real(n))
      if (n.eq.0) ints = 0
      ints = ints+1
      write(file_name0,form(ints))pre,n
      return
      end function file_name0
!*************Program to reorient automatically ******************
!*****************for multiple blade rows*********************
!******************by Vikram Shyam 04/01/07*********************
!******************Last Modified on 04/07/08*******************
      subroutine find_dirs_axial(tot_bks)
      implicit none
      integer xdir,rad_dir,num_checks,new_xdir,new_rad_dir
```

```
      integer i,j,k,fid,mod_blk,bcid
      character(len=100)::fname,bcfile
 !    bc.in VARS
      integer :: num_bc,num_bc_tot,tot_bks,num_tasks
!     integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
!     integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit
      integer, dimension(:),allocatable:: block_id,bc_type,start_i,
     &done_task,start_j,start_k,end_i,end_j,end_k
      integer, dimension(:),allocatable::gid,gname,flag,blk_id
      integer dir_ind,dir,newdir,task1,task2,indices(100)
      integer::task(1000),task_id(1000)
      bcfile='bc.in'
      num_bc=0
      num_checks=0
      num_tasks=0

      open(unit=18,file=bcfile,FORM='formatted',status='unknown')
 10   read(18,*) bcid
      if (bcid.eq.0) then
      goto 20
      else
      num_bc=num_bc+1
      goto 10
      endif
 20   continue
      close(18)

        open(unit=18,file=bcfile,FORM='formatted',status='unknown')
        allocate(block_id(num_bc),bc_type(num_bc),start_i(num_bc),
     &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
     &,end_k(num_bc),gid(num_bc),gname(num_bc),blk_id(num_bc))
       allocate(flag(tot_bks))
       flag(1:tot_bks)=0

        do i=1,num_bc
          read(18,*)block_id(i),
     &bc_type(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
     &,end_k(i)
        if (abs(bc_type(i))>100 .and.bc_type(i)/=205
     &.and. flag(block_id(i)).eq.0) then
         num_checks=num_checks+1
         blk_id(num_checks)=block_id(i)
         flag(block_id(i))=1
        endif
        enddo
       close(18)

!       print *,num_checks
       print *,'Blocks to change are:',blk_id(1:num_checks)

!This part is for all blocks that have a bc >100

! first make all axial i direction
      do i=1,num_checks
      mod_blk=blk_id(i)
      fid = 10*mod_blk
      fname = file_name0('GU',fid)
      call find_x_dir(xdir,fname)
!     print *,fname
!     print *,xdir
      new_xdir=1
      if (xdir/=new_xdir .and. xdir/=0) then
      call findtask(xdir,new_xdir,task1,task2)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task1
      call reorient(mod_blk,task1)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task2
```

```
      call reorient(mod_blk,task2)
      elseif (xdir.eq.0) then
      print *, 'WARNING! UNABLE TO REORIENT BLOCK ',mod_blk
      print *, 'Axial direction not detected. Manual inspection
     & required.'
      endif
      enddo

! Now do radial direction change to j
      do i=1,num_checks
      mod_blk=blk_id(i)
      fid = 10*mod_blk
      fname = file_name0('GU',fid)
      call find_rad_dir(rad_dir,fname)
!      print *,fname
!      print *,rad_dir
      new_rad_dir=2
      if (rad_dir/=new_rad_dir .and. rad_dir/=0) then
      call findtask(rad_dir,new_rad_dir,task1,task2)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task1
      call reorient(mod_blk,task1)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task2
      call reorient(mod_blk,task2)
      elseif (rad_dir.eq.0) then
      print *, 'WARNING! UNABLE TO REORIENT BLOCK ',mod_blk
      print *, 'Radial direction not detected. Manual inspection
     & required.'
      endif
      enddo

!So far only blocks with a definite axial and radial direction have been reoriented properly.
!Now that inlets are taken care of, look for radial and periodic faces


      open(unit=9,file='tasklist.in.axial',FORM='formatted',
     &status='unknown')
      write(9,*)num_tasks
      do i=1,num_tasks
      write(9,*)task_id(i),task(i)
      enddo
      close(9)

      contains

      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
      character(len=*),intent (in) :: pre,post
      character(len=100) :: file_cat
      ints=len_trim(pre)
      inte=len_trim(post)
      write(file_cat,*)pre(1:ints),post(1:inte)
      return
      end function file_cat
!********************** **********************
      function file_name0(pre,n) ! copied from TURBO!!!
      implicit none
      integer n,ints

      character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                            '(a,i4)','(a,i5)','(a,i6)'/)
      character(len=*),intent (in) :: pre
      character(len=100) :: file_name0
      if (n.gt.0) ints = log10(real(n))
      if (n.eq.0) ints = 0
      ints = ints+1
      write(file_name0,form(ints))pre,n
```

```
        return
        end function file_name0
!**************************************************
        end
!****************************************************************

        subroutine find_dirs_periodic(tot_bks)
        implicit none
        integer pdir,rad_dir,num_checks,new_pdir,new_rad_dir
        integer i,j,k,fid,mod_blk,bcid,n
        character(len=100)::fname,bcfile
 !       bc.in VARS
        integer :: num_bc,num_bc_tot,tot_bks,num_tasks
 !       integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
 !       integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit
        integer, dimension(:),allocatable:: block_id,bc_type,start_i,
       &done_task,start_j,start_k,end_i,end_j,end_k
        integer, dimension(:),allocatable::gid,gname,flag,blk_id,bcline
        integer dir_ind,dir,newdir,task1,task2,indices(100)
        integer::task(1000),task_id(1000)
        logical::p3d_exists,gu_exists
        bcfile='bc.in'
        num_bc=0
        num_checks=0
        num_tasks=0

        open(unit=8,file=bcfile,FORM='formatted',status='unknown')
 10     read(8,*) bcid
        if (bcid.eq.0) then
        goto 20
        else
        num_bc=num_bc+1
        goto 10
        endif
 20     continue
        close(8)

    !Now take care of the periodic and raidal direciton BCs
        open(unit=8,file=bcfile,FORM='formatted',status='unknown')
        allocate(block_id(num_bc),bc_type(num_bc),start_i(num_bc),
       &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
       &,end_k(num_bc),gid(num_bc),gname(num_bc),blk_id(num_bc),
       &bcline(num_bc))
        allocate(flag(tot_bks))
        flag(1:tot_bks)=0
    !First pass through BC file, read and store data, look for inlets exits and sliding

        do i=1,num_bc
          read(8,*)block_id(i),
       &bc_type(i),start_i(i),
       &start_j(i),start_k(i),end_i(i),end_j(i)
       &,end_k(i)

          if (abs(bc_type(i))>200 .and.bc_type(i)/=205 !If a block has an inlet or exit or
sliding interface
       &.and. flag(block_id(i)).eq.0) then
          flag(block_id(i))=1 ! flagged for already complete
          endif
          enddo
          close(8)
    !Now go through again and find blocks to reorient that have not been flagged and have
periodic faces
          do i=1,num_bc
          if (abs(bc_type(i))>100 .and.abs(bc_type(i))<200
       &.and. flag(block_id(i)).eq.0) then
          num_checks=num_checks+1
          blk_id(num_checks)=block_id(i)
          bcline(num_checks)=i ! save the line number of the periodic/ts bc
          if (abs(bc_type(i)).eq.104 .or. abs(bc_type(i)).eq.106) then
          flag(block_id(i))=3 !kmax
          elseif (abs(bc_type(i)).eq.105.or.abs(bc_type(i)).eq.107) then
```

```
          flag(block_id(i))=-3 !kmin
        endif
        endif
        enddo



!      print *,num_checks
!      print *,'Blocks to change are:',blk_id(1:num_checks)

! First fix periodic faces
      do i=1,num_checks
      mod_blk=blk_id(i)
      fid = 10*mod_blk
      fname = file_name0('GU',fid)
      call find_pdir(pdir,fname,bcline(i))
!       print *,fname
!       print *,xdir
      new_pdir=flag(mod_blk) !
      if (pdir/=new_pdir) then
      call findptask(pdir,new_pdir,task1,task2)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task1
      call reorient(mod_blk,task1)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task2
      call reorient(mod_blk,task2)
      endif
      enddo

! Now do radial direction change to j
      do i=1,num_checks
      mod_blk=blk_id(i)
      fid = 10*mod_blk
      fname = file_name0('GU',fid)
      call find_rad_dir(rad_dir,fname)
!       print *,fname
!       print *,rad_dir
      new_rad_dir=2
!       if (rad_dir/=new_rad_dir) then
      if (rad_dir/=new_rad_dir .and. abs(rad_dir)/=3) then
      call findrtask(rad_dir,new_rad_dir,task1,task2)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task1
      call reorient(mod_blk,task1)
      num_tasks=num_tasks+1
      task_id(num_tasks)=mod_blk
      task(num_tasks)=task2
      call reorient(mod_blk,task2)
      endif
      enddo

      open(unit=9,file='tasklist.in.periodic',FORM='formatted',
     &status='unknown')
      write(9,*)num_tasks
      do i=1,num_tasks
      write(9,*)task_id(i),task(i)
      enddo
      close(9)
      deallocate(block_id,bc_type,start_i,start_j,start_k,end_i,end_j
     &,end_k,gid,gname,blk_id,
     &bcline)
      deallocate(flag)
      contains

      function file_cat(pre,post)
      implicit none
      integer n,ints,inte
```

```fortran
      character(len=*),intent (in) :: pre,post
      character(len=100) :: file_cat
      ints=len_trim(pre)
      inte=len_trim(post)
      write(file_cat,*)pre(1:ints),post(1:inte)
      return
      end function file_cat
!*******************  *******************
      function file_name0(pre,n) ! copied from TURBO!!!
      implicit none
      integer n,ints

      character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                             '(a,i4)','(a,i5)','(a,i6)'/)
      character(len=*),intent (in) :: pre
      character(len=100) :: file_name0
      if (n.gt.0) ints = log10(real(n))
      if (n.eq.0) ints = 0
      ints = ints+1
      write(file_name0,form(ints))pre,n
      return
      end function file_name0
!*************************************************
      end subroutine find_dirs_periodic

!**************************************************************


      subroutine find_dirs_radial(tot_bks)
      implicit none
      integer xdir,rad_dir,num_checks,new_xdir,new_rad_dir
      integer i,j,k,fid,mod_blk,bcid,n
      character(len=100)::fname,bcfile
 !    bc.in VARS
      integer :: num_bc,num_bc_tot,tot_bks,num_tasks
!     integer :: slip,no_slip_ad,no_slip_iso,periodic,ref_periodic
!     integer :: cvbc_in,isentropic_in,rad_eq_exit,pres_exit
      integer, dimension(:),allocatable:: block_id,bc_type,start_i,
     &done_task,start_j,start_k,end_i,end_j,end_k
      integer, dimension(:),allocatable::gid,gname,flag,blk_id
      integer dir_ind,dir,newdir,task1,task2,indices(100)
      integer::task(1000),task_id(1000)
       logical::p3d_exists,gu_exists
      bcfile='bc.in'
      num_bc=0
      num_checks=0
      num_tasks=0


      open(unit=8,file=bcfile,FORM='formatted',status='unknown')
 10   read(8,*) bcid
      if (bcid.eq.0) then
      goto 20
      else
      num_bc=num_bc+1
      goto 10
      endif
 20   continue
      close(8)

        open(unit=8,file=bcfile,FORM='formatted',status='unknown')
        allocate(block_id(num_bc),bc_type(num_bc),start_i(num_bc),
     &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
     &,end_k(num_bc),gid(num_bc),gname(num_bc),blk_id(num_bc))
       allocate(flag(tot_bks))
       flag(1:tot_bks)=0

        do i=1,num_bc
          read(8,*)block_id(i),
     &bc_type(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
```

```fortran
     &,end_k(i)
       if (abs(bc_type(i))>100
     &.and. flag(block_id(i)).eq.0) then
         flag(block_id(i))=1
         endif
         enddo
      close(8)

         do i=1,num_bc
         if (abs(bc_type(i))<100
     &.and. flag(block_id(i)).eq.0) then
         num_checks=num_checks+1
         blk_id(num_checks)=block_id(i)
         flag(block_id(i))=1
         endif
         enddo




!      print *,num_checks
!      print *,'Blocks to change are:',blk_id(1:num_checks)

! Do radial direction change to j
       do i=1,num_checks
       mod_blk=blk_id(i)
       fid = 10*mod_blk
       fname = file_name0('GU',fid)
       call find_rad_dir(rad_dir,fname)
!       print *,fname
!       print *,rad_dir
       new_rad_dir=2
       if (rad_dir/=new_rad_dir) then
       call findrtask(rad_dir,new_rad_dir,task1,task2)
       num_tasks=num_tasks+1
       task_id(num_tasks)=mod_blk
       task(num_tasks)=task1
       call reorient(mod_blk,task1)
       num_tasks=num_tasks+1
       task_id(num_tasks)=mod_blk
       task(num_tasks)=task2
       call reorient(mod_blk,task2)
       endif
       enddo
       open(unit=9,file='tasklist.in.radial',FORM='formatted',
     &status='unknown')
       write(9,*)num_tasks
       do i=1,num_tasks
       write(9,*)task_id(i),task(i)
       enddo
       close(9)

       contains

       function file_cat(pre,post)
       implicit none
       integer n,ints,inte
       character(len=*),intent (in) :: pre,post
       character(len=100) :: file_cat
       ints=len_trim(pre)
       inte=len_trim(post)
       write(file_cat,*)pre(1:ints),post(1:inte)
       return
       end function file_cat
!********************* *********************
       function file_name0(pre,n) ! copied from TURBO!!!
       implicit none
       integer n,ints

       character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                             '(a,i4)','(a,i5)','(a,i6)'/)
```

```fortran
      character(len=*),intent (in) :: pre
      character(len=100) :: file_name0
      if (n.gt.0) ints = log10(real(n))
      if (n.eq.0) ints = 0
      ints = ints+1
      write(file_name0,form(ints))pre,n
      return
      end function file_name0
!*************************************************
      end subroutine find_dirs_radial
!***************************************************************


      subroutine findtask(dir,newdir,task1,task2)
      implicit none
      integer dir,newdir,task1,task2
      select case(dir)
      case(-1)
      select case(newdir)
   case(1)
   task1=1
   task2=3
   case(2)
       task1=4
   task2=2
   end select
     case(1)
     select case(newdir)
   case(2)
       task1=4
   task2=3
   end select
     case(2)
     select case(newdir)
   case(1)
   task1=4
   task2=3
   end select
     case(-2)
     select case(newdir)
   case(2)
   task1=2
   task2=3
   case(1)
       task1=4
   task2=1
   end select
     case(-3)
     select case(newdir)
   case(1)
   task1=6
   task2=1
   case(2)
       task1=5
   task2=2
   end select
     case(3)
     select case(newdir)
   case(1)
   task1=6
   task2=3
   case(2)
       task1=5
   task2=3
   end select
       end select

      end subroutine findtask

!===========================================================
```

```fortran
      subroutine findrtask(dir,newdir,task1,task2)
      implicit none
      integer dir,newdir,task1,task2
      select case(dir)
      case(-1)
         task1=4
   task2=2
      case(1)
         task1=4
   task2=1
      case(-2)
   task1=2
   task2=1
      case(-3)
         task1=5
   task2=2
      case(3)
         task1=5
   task2=1
      end select

      end subroutine findrtask


!*************************************************************
      subroutine findptask(dir,newdir,task1,task2)
      implicit none
      integer dir,newdir,task1,task2
      select case(dir)
      case(-1)
        select case(newdir)
        case(3)
        task1=6
        task2=3
        case(-3)
        task1=6
        task2=1
   end select
      case(1)
        select case(newdir)
        case(3)
        task1=6
        task2=1
        case(-3)
        task1=6
        task2=3
        end select
      case(2)
        select case(newdir)
        case(3)
        task1=5
        task2=1
        case(-3)
        task1=5
        task2=3
   end select
      case(-2)
        select case(newdir)
        case(3)
        task1=5
        task2=3
        case(-3)
        task1=5
        task2=1
   end select
      case(-3)
        select case(newdir)
        case(3)
        task1=3
        task2=1
   end select
```

```
          case(3)
            select case(newdir)
            case(-3)
            task1=3
            task2=1
       end select
          end select

          end subroutine findptask


!************************

!=========================================================
       subroutine find_pdir(pdir,fname,bcline)
!=========================================================
       implicit none
!    dmap.in VARS
       integer :: num_b2b, num_special_b2b,b2b_rec_len,pdir,bcline
       integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
       integer, dimension(:),allocatable:: d1e,id,dir2,lor1,lor2,p_b2b
       integer, dimension(:),allocatable:: p_special,bc,p_bc
       integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!    bc.in VARS
       integer :: num_bc_real,num_bc,dum
       integer, dimension(:),allocatable:: block_id,start_i,
     &                            start_j,start_k,end_i,end_j,end_k
       real,dimension(:),allocatable::bc_type_and_group
       integer, dimension(:),allocatable::gid,gname

!    local VARS
       character(len=50) :: gpro,dmapfile,infile,bcfile
       integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
       integer :: i1,j1,k1,i2,j2,k2,total_recs
       integer:: v(1:100)
       real, dimension(:),allocatable::x,y,z
       real x1,x2,y1,y2,z1,z2
       logical xn1,yn1,zn1,xn2,yn2,zn2,off,on
       integer bs,be,temp

!    Reorient Vars
       integer:: mod_blk,ni,nj,nk,task,nbgu,nbr,bld_psg
       integer::ni3new,nj3new,nk3new,nx
       integer::c4,c5,c6,pos1,pos2
       integer::dir(3),com(5)
       integer::ind1,ind2
       real,dimension(:,:,:),allocatable::x3,y3,z3,x3new,y3new,z3new

       character(len=50)::fname
       integer:: fid
       bcfile='bc.in'
       open(unit=10,file=bcfile,FORM='formatted',status='unknown')
       num_bc=0
       do
       read(10,*),dum
!     print *,dum
       if (dum.eq.0) EXIT
       num_bc=num_bc+1
       enddo
       close(10)


       num_bc_real=num_bc
       num_bc=num_bc+1


       allocate(block_id(num_bc),bc_type_and_group(num_bc)
      &,start_i(num_bc),
      &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
      &,end_k(num_bc),gid(num_bc),gname(num_bc))
```

```
!    print *,num_bc_real,'Boundary conditions found.'
     open(unit=10,file=bcfile,FORM='formatted',status='unknown')
     do i=1,num_bc
     read(10,*),block_id(i),
    &bc_type_and_group(i),start_i(i),
    &start_j(i),start_k(i),end_i(i),end_j(i)
    &,end_k(i)

!      write(*,'(2x,1I4,1x,F8.2,6I5,1("/"))'),block_id(i),
!     &bc_type_and_group(i),start_i(i),
!     &start_j(i),start_k(i),end_i(i),end_j(i)
!     &,end_k(i)
     enddo
     close(10)

!Now find ni,nj,nk for block block_id(num_bc)

     i=bcline ! this is the line at which periodic bc was found


     fid = 10*block_id(i)
     fname = file_name0('GU',fid)
     open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
     read(7) nbgu,nbr,bld_psg
     read(7) ni,nj,nk
     close(7)


      if ((start_i(i).eq.end_i(i) ).and. start_i(i).eq.ni) then
      pdir=1
      elseif ((start_i(i).eq.end_i(i) ).and. start_i(i) /= ni) then
      pdir=-1
      elseif ((start_j(i).eq.end_j(i) ).and. start_j(i).eq.nj) then
      pdir=2
      elseif ((start_j(i).eq.end_j(i) ).and. start_j(i) /= nj) then
      pdir=-2
      elseif ((start_k(i).eq.end_k(i) ).and. start_k(i).eq.nk) then
      pdir=3
      elseif ((start_k(i).eq.end_k(i) ).and. start_k(i) /= nk) then
      pdir=-3
      endif
     deallocate(block_id,bc_type_and_group
    &,start_i,
    &start_j,start_k,end_i,end_j
    &,end_k,gid,gname)
     contains

     function file_cat(pre,post)
     implicit none
     integer n,ints,inte
     character(len=*),intent (in) :: pre,post
     character(len=100) :: file_cat
     ints=len_trim(pre)
     inte=len_trim(post)
     write(file_cat,*)pre(1:ints),post(1:inte)
     return
     end function file_cat
!********************* ********************
     function file_name0(pre,n) ! copied from TURBO!!!
     implicit none
     integer n,ints

     character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
    &                            '(a,i4)','(a,i5)','(a,i6)'/)
     character(len=*),intent (in) :: pre
     character(len=100) :: file_name0
     if (n.gt.0) ints = log10(real(n))
     if (n.eq.0) ints = 0
     ints = ints+1
     write(file_name0,form(ints))pre,n
     return
```

```fortran
      end function file_name0




      end subroutine find_pdir
!=====================================================================


      subroutine find_x_dir(xdir,fname)
      implicit none
      integer:: mod_blk,ni3,nj3,nk3,task,nbgu,nbr,bld_psg
      integer::i,j,k
      real,dimension(:,:,:),allocatable::x3,y3,z3
      character(len=100)::fname
      integer:: fid,xdir
      real diffi,diffj,diffk
!      mod_blk=1
!      fid = 10*mod_blk
!      fname = file_name0('GU',fid)
!      fname='GU160'
       open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
       read(7) nbgu,nbr,bld_psg
       read(7) ni3,nj3,nk3
       allocate(x3(ni3,nj3,nk3),y3(ni3,nj3,nk3),z3(ni3,nj3,nk3))
       read(7)
     &          (((x3(i,j,k),i=1,ni3),j=1,nj3),
     &          k=1,nk3),
     &          (((y3(i,j,k),i=1,ni3),j=1,nj3),
     &          k=1,nk3),
     &          (((z3(i,j,k),i=1,ni3),j=1,nj3),
     &          k=1,nk3)
       close(7)
!      print *,'3D array of current block created.'
!      print *,'Array size is',ni3,nj3,nk3
!      print *,'Ready for manipulation.'
       diffi=0.
       diffj=0.
       diffk=0.
       !print *,'This is i'
       do j=1,nj3
       do k=1,nk3
       diffi=diffi+(x3(ni3,j,k)-x3(1,j,k))!/x3(1,j,k)
       enddo
       enddo
       diffi=diffi/(nj3*nk3)
!       !print *,diffi

!       !print *,'This is j'
       do i=1,ni3
       do k=1,nk3
       diffj=diffj+(x3(i,nj3,k)-x3(i,1,k))!/x3(i,1,k)
       enddo
       enddo
       diffj=diffj/(ni3*nk3)
       !print *, diffj

       !print *,'This is k'
       do i=1,ni3
       do j=1,nj3
       diffk=diffk+(x3(i,j,nk3)-x3(i,j,1))!/x3(i,1,k)
       enddo
       enddo
       diffk=diffk/(ni3*nj3)
       !print *, diffk

       if (abs(diffi)>abs(diffj) .and. abs(diffi)>abs(diffk)) then
       if (diffi>0.)then
       xdir=1
       else
       xdir=-1
       endif
```

```
      elseif (abs(diffj)>abs(diffi) .and. abs(diffj)>abs(diffk)) then
      if (diffj>0.)then
      xdir=2
      else
      xdir=-2
      endif
      elseif (abs(diffk)>abs(diffj) .and. abs(diffk)>abs(diffi)) then
      if (diffk>0.)then
      xdir=3
      else
      xdir=-3
      endif
      else
      xdir=0
      !print *,'No axial direction found for block ',fname
      goto 999
      endif
      !print *,'Axial direction is ',xdir

 999  continue
      deallocate(x3,y3,z3)
      end subroutine find_x_dir



!************************

!=========================================================
      subroutine find_rad_dir(rad_dir,fname)
!=========================================================

      implicit none
      integer:: mod_blk,task,nbgu,nbr,bld_psg
      character(len=100)::fname
      integer:: fid,xdir
      real diffi,diffj,diffk
      integer::i,j,k,rad_dir,nlb,ni,nj,nk
      real*8,dimension(:,:,:),allocatable::x3,y3,z3
      real*8,dimension(6):: avg
      integer, dimension(6)::hc
      integer,dimension(1)::uploc,lowloc
      real*8, dimension(1)::upval,lowval
!     fname='GU10'
      xdir=1
      open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
      read(7) nbgu,nbr,bld_psg
      read(7) ni,nj,nk
      allocate(x3(ni,nj,nk),y3(ni,nj,nk),z3(ni,nj,nk))
      read(7)
     &          (((x3(i,j,k),i=1,ni),j=1,nj),
     &          k=1,nk),
     &          (((y3(i,j,k),i=1,ni),j=1,nj),
     &          k=1,nk),
     &          (((z3(i,j,k),i=1,ni),j=1,nj),
     &          k=1,nk)
      close(7)
      !print *,'3D array of current block created.'
      !print *,'Array size is',ni,nj,nk
      !print *,'Ready for manipulation.'
      ! d=1,2,3,4,5,6
      ! d=imin,imax,jmin,jmax,kmin,kmax
      ! if hc(d)==1, case
      ! if hc(d)==0, neither
      ! if hc(d)==-1, hub
      hc(1:6)=0
      avg(1:6)=0.

      !check j faces
      j=1
      do i=1,ni
      do k=1,nk
```

```
avg(3)=avg(3)+y3(i,j,k)
end do
end do
avg(3)=avg(3)/(ni*nk)

j=nj
do i=1,ni
do k=1,nk
avg(4)=avg(4)+y3(i,j,k)
end do
end do
avg(4)= avg(4)/(ni*nk)

!check i faces

i=1
do j=1,nj
do k=1,nk
avg(1)=avg(1)+y3(i,j,k)
end do
end do
avg(1)= avg(1)/(nj*nk)

i=ni
do j=1,nj
do k=1,nk
avg(2)=avg(2)+y3(i,j,k)
end do
end do
avg(2)= avg(2)/(nj*nk)

!check k faces
k=1
do i=1,ni
do j=1,nj
avg(5)=avg(5)+y3(i,j,k)
end do
end do
avg(5)= avg(5)/(ni*nj)

k=nk
do i=1,ni
do j=1,nj
avg(6)=avg(6)+y3(i,j,k)
end do
end do
avg(6)= avg(6)/(ni*nj)


select case(abs(xdir))
case(1)
uploc=maxloc(avg(3:6))+2
lowloc=minloc(avg(3:6))+2
upval=maxval(avg(3:6))
lowval=minval(avg(3:6))
hc(uploc)=1
hc(lowloc)=-1

case(2)

avg(3)=(avg(1)+avg(2)+avg(5)+avg(6))/4 !ensures that jmin and jmax faces are not max
and min
avg(4)=avg(3)
uploc=maxloc(avg(1:6))
lowloc=minloc(avg(1:6))
upval=maxval(avg(1:6))
lowval=minval(avg(1:6))

hc(uploc)=1
hc(lowloc)=-1
case(3) ! k is inlet/exit
```

```
      !either i or j is hub-case direction
      uploc=maxloc(avg(1:4))
      lowloc=minloc(avg(1:4))
      upval=maxval(avg(1:4))
      lowval=minval(avg(1:4))
      hc(uploc)=1
      hc(lowloc)=-1

      end select
      !print *,uploc,lowloc,upval,lowval
      !print *,avg,xdir,hc

      if (hc(1).eq.1 .and. hc(2).eq.-1) then
      rad_dir=-1
      elseif(hc(1).eq.-1 .and. hc(2).eq.1) then
      rad_dir=1
      elseif (hc(3).eq.1 .and. hc(4).eq.-1) then
      rad_dir=-2
      elseif(hc(3).eq.-1 .and. hc(4).eq.1) then
      rad_dir=2
      elseif (hc(5).eq.1 .and. hc(6).eq.-1) then
      rad_dir=-3
      elseif(hc(5).eq.-1 .and. hc(6).eq.1) then
      rad_dir=3
      else
      rad_dir=0
      !print *,'no radial direction found in ',fname
      goto 999
      endif
      !print *,'Radial direction is ',rad_dir
 999  continue
      deallocate(x3,y3,z3)
      end subroutine find_rad_dir

!*************************


      subroutine operate
      implicit none
      integer i,num_oper
      character(len=50)::fname
      integer, allocatable,dimension(:)::mod_blk,task
!     integer::mod_blk,task
      open(unit=8,file='tasklist.in',FORM='formatted',status='unknown')
      read(8,*) num_oper
      allocate(mod_blk(num_oper),task(num_oper))
      do i=1,num_oper
      read(8,*) mod_blk(i),task(i)
!     print *,mod_blk(i),task(i)
      enddo
      close(8)
      do i=1,num_oper
      call reorient(mod_blk(i),task(i))
      enddo
      end subroutine



      subroutine reorient(mod_blk,task)
      implicit none
!     dmap.in VARS
      integer :: num_b2b, num_special_b2b,b2b_rec_len
      integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
      integer, dimension(:),allocatable:: d1e,id,dir2,lor1,lor2,p_b2b
      integer, dimension(:),allocatable:: p_special,bc,p_bc
      integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!     bc.in VARS
      integer :: num_bc_real,num_bc,dum
      integer, dimension(:),allocatable:: block_id,start_i,
     &                        start_j,start_k,end_i,end_j,end_k
      real,dimension(:),allocatable::bc_type_and_group
```

```
        integer, dimension(:),allocatable::gid,gname

!       local VARS
        character(len=50) :: gpro,dmapfile,infile,bcfile
        integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
        integer :: i1,j1,k1,i2,j2,k2,total_recs
        integer:: v(1:100)
        real, dimension(:),allocatable::x,y,z
        real x1,x2,y1,y2,z1,z2
        logical xn1,yn1,zn1,xn2,yn2,zn2,off,on
        integer bs,be,temp

!       Reorient Vars
        integer:: mod_blk,ni3,nj3,nk3,task,nbgu,nbr,bld_psg
        integer::ni3new,nj3new,nk3new,nx
        integer::c4,c5,c6,pos1,pos2
        integer::dir(3),com(5)
        integer::ind1,ind2
        real,dimension(:,:,:),allocatable::x3,y3,z3,x3new,y3new,z3new

        character(len=50)::fname
        integer:: fid

        off=.FALSE.
        on=.TRUE.
        b2b_rec_len=21
        dmapfile='dmap.in'
        bcfile='bc.in'

!**************  READING dmap.in    ********************

        open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
        read(8,*) num_b2b
!        write(*,'(I3)') num_b2b
        do i=1,num_b2b
        read(8,*) v(1:b2b_rec_len)
        enddo
        read(8,*), num_special_b2b
        close(8)

! Find total number of records to store
        total_recs=num_b2b+num_special_b2b


! allocate total record length
        allocate(id(total_recs),is(total_recs),
     &ie(total_recs),js(total_recs),je(total_recs)
     &,ks(total_recs),ke(total_recs),blkb(total_recs)
     &,d1(total_recs),d1s(total_recs),d1e(total_recs)
     &,d2(total_recs),d2s(total_recs),d2e(total_recs)
     &, d3(total_recs),d3s(total_recs),d3e(total_recs)
     &,dir1(total_recs),dir2(total_recs),
     & lor1(total_recs),lor2(total_recs),bc(total_recs)) ! bc is only used for special b2b

!       print *,'****************Reading dmap.in***************'
        open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
!       print *,'Number of block to block interfaces'
        read(8,*) num_b2b
!        write(*,'(I3)') num_b2b
!       print *,'id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e,d3,d3s,
!       &d3e,dir1,dir2,lor1,lor2'
        do i=1,num_b2b
        read(8,*) id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
!        write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
!       & ie(i),js(i)
!       & ,je(i),ks(i),ke(i),blkb(i),
!       & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
```

```
  !    & ,dir1(i),dir2(i),lor1(i),lor2(i)
       enddo

! 1.  CHECKING GRID CONNECTIONS

!*****************Special b2b Checking  **************************
       read(8,*), num_special_b2b
!     print *,'Number of special block to block interfaces.'
!     write(*,'(I3)'), num_special_b2b
!     print *,'id,is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,d2,d2s,d2e,d3,d3s,
!    &d3e,dir1,dir2,lor1,lor2,bc'

       do i=num_b2b+1,num_b2b+num_special_b2b
       read(8,*) id(i),is(i),
      &ie(i),js(i)
      & ,je(i),ks(i),ke(i),blkb(i),
      & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
      & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

!      write(*,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
!     &ie(i),js(i)
!     & ,je(i),ks(i),ke(i),blkb(i),
!     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
!     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

       end do


       close(8)
! dmap.in CLOSED HERE

! READ BC.IN AND STORE

       open(unit=10,file=bcfile,FORM='formatted',status='unknown')
       num_bc=0
       do
       read(10,*),dum
!       print *,dum
       if (dum.eq.0) EXIT
       num_bc=num_bc+1
       enddo
       close(10)

       !*****TO SIMPLIFY REWRITE INTO bc.in*********
       num_bc_real=num_bc
       num_bc=num_bc+1


       allocate(block_id(num_bc),bc_type_and_group(num_bc)
      &,start_i(num_bc),
      &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
      &,end_k(num_bc),gid(num_bc),gname(num_bc))

!      print *,num_bc_real,'Boundary conditions found.'
       open(unit=10,file=bcfile,FORM='formatted',status='unknown')
       do i=1,num_bc
       read(10,*),block_id(i),
      &bc_type_and_group(i),start_i(i),
      &start_j(i),start_k(i),end_i(i),end_j(i)
      &,end_k(i)

!      write(*,'(2x,1I4,1x,F8.2,6I5,1("/"))'),block_id(i),
!     &bc_type_and_group(i),start_i(i),
!     &start_j(i),start_k(i),end_i(i),end_j(i)
!     &,end_k(i)
       enddo
       close(10)

! END BC READ

       com(1)=1
```

```fortran
      com(2)=2
      com(3)=0
      com(4)=1
      com(5)=2

! ALGORITHM
! Ask user which block to be changed

      fid = 10*mod_blk
       fname = file_name0('GU',fid)
       open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
       read(7) nbgu,nbr,bld_psg
       read(7) ni3,nj3,nk3
       allocate(x3(ni3,nj3,nk3),y3(ni3,nj3,nk3),z3(ni3,nj3,nk3))
       read(7)
     &         (((x3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3),
     &         (((y3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3),
     &         (((z3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3)
       close(7)
!      print *,'3D array of current block created.'
!      print *,'Array size is',ni3,nj3,nk3
!      print *,'Ready for manipulation.'

! Now the block has been put into a 3d matrix
!
! what operation to perform?
!




! switch i-j,j-k,i-k?
! put x,y,z into temp matrix ni,nj,nk
! allocate temp matrix with dim(ninew,njnew,nknew)
! call switch routine
! write new x,y,z into linear array
!
! reverse i,j,k?
! perform operation
! PLOT3D FILE UPDATE

      select case(task)
      case(1,2,3)
      ni3new=ni3
      nj3new=nj3
      nk3new=nk3
      allocate(x3new(ni3new,nj3new,nk3new),
     &y3new(ni3new,nj3new,nk3new),z3new(ni3new,nj3new,nk3new))
       call flip(x3,x3new,ni3new,nj3new,nk3new,task)
       call flip(y3,y3new,ni3new,nj3new,nk3new,task)
       call flip(z3,z3new,ni3new,nj3new,nk3new,task)
       deallocate(x3,y3,z3)
       allocate(x3(ni3new,nj3new,nk3new),
     &y3(ni3new,nj3new,nk3new),z3(ni3new,nj3new,nk3new))
       x3=x3new
       y3=y3new
       z3=z3new
       deallocate(x3new,y3new,z3new)

       case(4,5,6)! switching i-j,j-k,k-i

       c4=(mod(6,task)*mod(5,task)/2)
       c5=(mod(4,task)*mod(6,task)/4)
       c6=(mod(5,task)*mod(4,task)/20)
       ni3new=c5*ni3+c4*nj3+c6*nk3
       nj3new=c4*ni3+c6*nj3+c5*nk3
```

```
     nk3new=c6*ni3+c5*nj3+c4*nk3
      allocate(x3new(ni3new,nj3new,nk3new),
     &y3new(ni3new,nj3new,nk3new),z3new(ni3new,nj3new,nk3new))
      call switch(x3,x3new,ni3,nj3,nk3,ni3new,nj3new,nk3new,task)
      call switch(y3,y3new,ni3,nj3,nk3,ni3new,nj3new,nk3new,task)
      call switch(z3,z3new,ni3,nj3,nk3,ni3new,nj3new,nk3new,task)
      deallocate(x3,y3,z3)
      allocate(x3(ni3new,nj3new,nk3new),
     &y3(ni3new,nj3new,nk3new),z3(ni3new,nj3new,nk3new))
      x3=x3new
      y3=y3new
      z3=z3new
      deallocate(x3new,y3new,z3new)
      end select


! find block number in dmap.in and update


! check if it is b1 or b2
! if b1
!
      do i=1,total_recs
      dir(1)=d1(i)
      dir(2)=d2(i)
      dir(3)=d3(i)
      if (.not.(id(i).eq.mod_blk .and. blkb(i).eq.mod_blk)) then
      if (id(i).eq.mod_blk) then

      select case(task)
      case(1) !reverse i
      call  flipd(is(i),ie(i),d1s(i),d1e(i),lor1(i),ni3)
      case(2) !reverse j
      call  flipd(js(i),je(i),d2s(i),d2e(i),lor1(i),nj3)
      case(3) !reverse k
      call  flipd(ks(i),ke(i),d3s(i),d3e(i),lor1(i),nk3)
      case(4)! i-j
      call switch2(is(i),js(i))
      call switch2(ie(i),je(i))
      call switch2(d1(i),d2(i))
      call switch2(d1s(i),d2s(i))
      call switch2(d1e(i),d2e(i))
      call dircheck(dir1(i),1,2)
      case(5)! j-k
      call switch2(js(i),ks(i))
      call switch2(je(i),ke(i))
      call switch2(d2(i),d3(i))
      call switch2(d2s(i),d3s(i))
      call switch2(d2e(i),d3e(i))
      call dircheck(dir1(i),2,3)
      case(6)! k-i
      call switch2(ks(i),is(i))
      call switch2(ke(i),ie(i))
      call switch2(d3(i),d1(i))
      call switch2(d3s(i),d1s(i))
      call switch2(d3e(i),d1e(i))
      call dircheck(dir1(i),3,1)
      end select


! if b2
!
      elseif (blkb(i).eq.mod_blk) then
      select case(task)
      case(1,2,3)! rev i
      c4=(mod(6,task+3)*mod(5,task+3)/2)
      c5=(mod(4,task+3)*mod(6,task+3)/4)
      c6=(mod(5,task+3)*mod(4,task+3)/20)
      nx=c4*ni3+c5*nj3+c6*nk3
      if (d1(i).eq.(task-1)) then
      call flipd2(is(i),ie(i),d1s(i),d1e(i),lor2(i),nx)
```

```
          elseif (d2(i).eq.(task-1)) then
          call flipd2(js(i),je(i),d2s(i),d2e(i),lor2(i),nx)
          elseif (d3(i).eq.(task-1)) then
          call flipd2(ks(i),ke(i),d3s(i),d3e(i),lor2(i),nx)
          endif
          case(4,5,6)!i-j
          ind1=task-2+1
          ind2=task-2-1
          call find1(pos1,com(ind1),dir,3)
          call find1(pos2,com(ind2),dir,3)
!          print *,dir(pos1),dir(pos2)
!          print *,dir2(i)
          if (dir2(i).eq.(dir(pos1)+1)) then
          dir2(i)=dir(pos2)+1
          elseif (dir2(i).eq.(dir(pos2)+1)) then
          dir2(i)=dir(pos1)+1
          endif
          call switch2(dir(pos1),dir(pos2))
          d1(i)=dir(1)
          d2(i)=dir(2)
          d3(i)=dir(3)

          end select

          endif


          elseif (id(i).eq.mod_blk .and. blkb(i).eq.mod_blk) then
! if block is circularly connected to itself
          select case(task)
          case(1,2,3) !do nothing
          case(4)
          call switch2(is(i),js(i))
          call switch2(ie(i),je(i))
          call switch2(d1s(i),d2s(i))
          call switch2(d1e(i),d2e(i))
          call dircheck(dir1(i),1,2)
          call dircheck(dir2(i),1,2)
          case(5)! j-k
          call switch2(js(i),ks(i))
          call switch2(je(i),ke(i))
          call switch2(d2s(i),d3s(i))
          call switch2(d2e(i),d3e(i))
          call dircheck(dir1(i),2,3)
          call dircheck(dir2(i),2,3)
          case(6)! k-i
          call switch2(ks(i),is(i))
          call switch2(ke(i),ie(i))
          call switch2(d3s(i),d1s(i))
          call switch2(d3e(i),d1e(i))
          call dircheck(dir1(i),3,1)
          call dircheck(dir2(i),3,1)
          end select
          endif


          enddo
! find block number in bc.in and update
          do i=1,num_bc_real
          if (block_id(i).eq.mod_blk) then
          select case(task)

          case(1)
          start_i(i)=ni3-start_i(i)+1
          end_i(i)=ni3-end_i(i)+1
          if (start_i(i)>end_i(i)) call switch2(start_i(i),end_i(i))
          case(2)
          start_j(i)=nj3-start_j(i)+1
          end_j(i)=nj3-end_j(i)+1
          if (start_j(i)>end_j(i)) call switch2(start_j(i),end_j(i))
          case(3)
          start_k(i)=nk3-start_k(i)+1
```

```
      end_k(i)=nk3-end_k(i)+1
      if (start_k(i)>end_k(i)) call switch2(start_k(i),end_k(i))
      case(4)!i-j
      temp=start_i(i)
      start_i(i)=start_j(i)
      start_j(i)=temp
      temp=end_i(i)
      end_i(i)=end_j(i)
      end_j(i)=temp

      case(5)! j-k
      temp=start_j(i)
      start_j(i)=start_k(i)
      start_k(i)=temp
      temp=end_j(i)
      end_j(i)=end_k(i)
      end_k(i)=temp

      case(6)! k-i
      temp=start_k(i)
      start_k(i)=start_i(i)
      start_i(i)=temp
      temp=end_k(i)
      end_k(i)=end_i(i)
      end_i(i)=temp

      end select
      endif
      enddo



! Update temporary variables for further manipulation
      ni3=ni3new
      nj3=nj3new
      nk3=nk3new


! ask user for next operation on block or back to main menu
! change another block or quit?
! exit



! Following are global and should not be deallocated earlier


! WRITE OUT NEW GU FILE
      fid = 10*mod_blk
      fname = file_name0('GU',fid)
!      print *,' opening',fname,' for writing as GU'
      open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
      write(7) nbgu,nbr,bld_psg
!      print *,'Blocks  blade_row_id  blade_passage'
!      print *, nbgu,nbr,bld_psg
      write(7) ni3,nj3,nk3
      write(7)
     &          (((x3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3),
     &          (((y3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3),
     &          (((z3(i,j,k),i=1,ni3),j=1,nj3),
     &           k=1,nk3)
      close(7)



! WRITE OUT NEW DMAP.IN AND BC.IN

!      fname=file_cat(dmapfile,'.new')
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
```

```fortran
!       print *,'Number of block to block interfaces'
       write(8,'(I5)') num_b2b
       do i=1,num_b2b
       write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
       enddo

       write(8,'(I5)'), num_special_b2b
       do i=num_b2b+1,num_b2b+num_special_b2b
       write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

       end do
       close(8)


!        fname=file_cat(bcfile,'.new')

       open(unit=10,file=bcfile,FORM='formatted',status='unknown')
       do i=1,num_bc
       write(10,'(2x,1I4,1x,F8.2,6I5,1("/"))'),block_id(i),
     &bc_type_and_group(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
     &,end_k(i)
       enddo
       close(10)


       deallocate(is,ie,js,je,ks,ke,blkb,d1,d1s,d1e,id,dir2,lor1,
     &           lor2,d2,d2s,d2e,d3,d3s,d3e,dir1,bc)
!      deallocate(x,y,z)
!      deallocate(ni,nj,nk,block_start,block_end)
!      print *,'Memory deallocation complete.'



!***********************FUNCTION AND SUBROUTINE DEFINITIONS FOLLOW*********************


       contains

       function file_cat(pre,post)
       implicit none
       integer n,ints,inte
       character(len=*),intent (in) :: pre,post
       character(len=100) :: file_cat
       ints=len_trim(pre)
       inte=len_trim(post)
       write(file_cat,*)pre(1:ints),post(1:inte)
       return
       end function file_cat
!******************** ********************
       function file_name0(pre,n) ! copied from TURBO!!!
       implicit none
       integer n,ints

       character(8) :: form(6) = (/ '(a,i1)','(a,i2)','(a,i3)',
     &                             '(a,i4)','(a,i5)','(a,i6)'/)
       character(len=*),intent (in) :: pre
       character(len=100) :: file_name0
       if (n.gt.0) ints = log10(real(n))
       if (n.eq.0) ints = 0
       ints = ints+1
       write(file_name0,form(ints))pre,n
       return
```

```
      end function file_name0

!*******************************************************

      subroutine compare(x1,x2,tf)
      logical :: tf
      real:: x1,x2
      if (x1==x2) then
      tf=.TRUE.
      else
      tf=.FALSE.
      endif
      end subroutine compare

      subroutine mapijk(i,j,k,d1,d1s,d2,
     &d2s,d3,d3s)

      integer i,j,k,d1,d1s,d2,d2s,d3,d3s

      if (d1==0) then
      i=d1s
      elseif (d2==0) then
      i=d2s
      elseif (d3==0) then
      i=d3s
      endif
      if (d1==1) then
      j=d1s
      elseif (d2==1) then
      j=d2s
      elseif (d3==1) then
      j=d3s
      endif
      if (d1==2) then
      k=d1s
      elseif (d2==2) then
      k=d2s
      elseif (d3==2) then
      k=d3s
      endif
      end subroutine mapijk

      subroutine shift(is,ie,m) ! to change from cell center to node center
      !e.g. 2 43 becomes 1 43, 13 2 becomes 13 1 and 2 2 becomes 1 1.
      !1 23 does not change, 23 23 does not change.
      integer is,ie,m
      if (is>1) then
          if (is<ie) then
        is=is-1
        elseif (is>ie) then
            ie=ie-1
        else
           if(m==0) then
             ie=ie-1
             is=is-1
           endif
        endif
      endif
      end subroutine shift


      subroutine find_ijk(ijk,i,j,k,ni,nj,nk,bs
     &              ,be)
      integer, intent(out)::ijk
      integer, intent(in)::i,j,k
      integer ::ni,nj,nk
      integer::bs,be
      ijk=(k-1)*nj*ni+(j-1)*ni+i
      ijk=ijk+(bs-1)
      end subroutine find_ijk
```

```fortran
      subroutine find_i_j_k(i,j,k,ijk,ni,nj,nk,bs
     &          ,be)
      integer::ijk,temp
      integer::i,j,k
      integer ::ni,nj,nk,ri,rj,rk
      integer :: bs,be

      temp=ijk-(bs-1)
      rk=mod(temp,(ni*nj))
      k=(temp-rk)/(ni*nj)+1
      rj=mod(rk,ni)
      j=(rk-rj)/ni+1
      i=rj

      end subroutine find_i_j_k

!*************************

      subroutine flip(ain,aout,ni,nj,nk,dir)

      real,dimension(ni,nj,nk)::ain,aout
      integer::ni,nj,nk,i,j,k,dir
      select case(dir)
      case(1)

      do i=1,ni
      do j=1,nj
      do k=1,nk
      aout(i,j,k)=ain(ni-i+1,j,k)
      end do
      end do
      end do

      case(2)

      do i=1,ni
      do j=1,nj
      do k=1,nk
      aout(i,j,k)=ain(i,nj-j+1,k)
      end do
      end do
      end do

      case(3)

      do i=1,ni
      do j=1,nj
      do k=1,nk
      aout(i,j,k)=ain(i,j,nk-k+1)

      end do
      end do
      end do

      end select
      end subroutine flip
!*************************

      subroutine switch(ain,aout,ni,nj,nk,ninew,njnew,nknew,dir)
      !ij=4,jk=5,ik=6
      real,dimension(ninew,njnew,nknew)::aout
      real,dimension(ni,nj,nk)::ain
      integer::ni,nj,nk,i,j,k,dir
      integer::ninew,njnew,nknew

      select case(dir)
      case(4)
      do i=1,nj
      do j=1,ni
      do k=1,nk
      aout(i,j,k)=ain(j,i,k)
```

```
      end do
      end do
      end do

      case(5)
      do i=1,ni
      do j=1,nk
      do k=1,nj
      aout(i,j,k)=ain(i,k,j)
      end do
      end do
      end do

      case(6)
      do i=1,nk
      do j=1,nj
      do k=1,ni
      aout(i,j,k)=ain(k,j,i)
      end do
      end do
      end do

      end select
      end subroutine switch

!*************************
      subroutine flipd(is,ie,d1s,d1e,lor1,ni)
      integer ::isnew,ienew,d1snew,d1enew,lor1new,is,ie,d1s,d1e,lor1,ni
      integer:: temp
      lor1new=lor1
      if (is.eq.1) then ! this means d1s(i) is also 1 by Rules of TURBO
          is=2
          d1s=2
      endif
          isnew=ni-is+2
          ienew=ni-ie+2
      if (isnew>ienew) then
          temp=ienew
          ienew=isnew
          isnew=temp
          temp=d1s
          d1s=d1e
          d1e=temp
      endif
      if (isnew.eq.ienew) then
              if (lor1.eq.0) then
              lor1new=1
              else
              lor1new=0
              endif
       elseif (isnew.eq.d1s .and. isnew.eq.2 .and.
      & (.not.(isnew.eq.ienew))) then
          isnew=1
          d1s=1
      endif
      is=isnew
      ie=ienew
      lor1=lor1new
      end subroutine flipd
!*************************
      subroutine flipd2(is,ie,d1s,d1e,lor2,ni)
      integer ::isnew,ienew,d1snew,d1enew,lor2new,is,ie,d1s,d1e,lor2,ni
      integer:: temp
      lor2new=lor2
      isnew=is
      ienew=ie
      if (d1s.eq.1) then ! this means d1s(i) is also 1 by Rules of TURBO
          is=2
          d1s=2
      endif
          d1snew=ni-d1s+2
```

```
        d1enew=ni-d1e+2
     if (d1snew.eq.d1enew) then
             if (lor2.eq.0) then
             lor2new=1
             else
             lor2new=0
             endif
      elseif (d1snew.eq.is .and. is.eq.2 .and.
     & (.not.(d1snew.eq.d1enew))) then
          is=1
          d1snew=1
      endif
      d1s=d1snew
      d1e=d1enew
      lor2=lor2new
      end subroutine flipd2


!***********************************************
      subroutine dircheck(dir1,a,b)
      implicit none
      integer dir1,a,b
      if (dir1.eq.a) then
      dir1=b
      elseif (dir1.eq.b) then
      dir1=a
      else
      dir1=dir1
      endif
      end subroutine dircheck
!********************************************************
      subroutine switch2(a,b)
      integer temp,a,b
      temp=a
      a=b
      b=temp
      end subroutine switch2
!********************************************************
      subroutine find1(loc,d,l,n)
      integer :: l(n)
      integer:: d,loc,i,n
      loc=-999
      do i=1,n
      if (l(i).eq.d) then
      loc=i
      EXIT
      endif
      enddo
      end subroutine find1
!********************************************************

      end subroutine reorient



!========================================================
      subroutine check_hub_case(hc,x,y,z,ni,nj,nk,dir)
!========================================================
!     use common_area
!     use variable_area
!     use error_report
      implicit none

      real*8,allocatable, dimension(:,:)::r
      integer::i,j,k,dir,nlb,ni,nj,nk
      real*8,dimension(1:ni,1:nj,1:nk)::x,y,z
      real*8,dimension(6):: avg
      integer, dimension(6)::hc
      integer,dimension(1)::uploc,lowloc
      real*8, dimension(1)::upval,lowval
      ! d=1,2,3,4,5,6
```

```
! d=imin,imax,jmin,jmax,kmin,kmax
! if hc(d)==1, case
! if hc(d)==0, neither
! if hc(d)==-1, hub
hc(1:6)=0
avg(1:6)=0.

!check j faces
allocate(r(ni,nk))
j=1
do i=1,ni
do k=1,nk
r(i,k)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(3)= average(r,ni,nk)

j=nj
do i=1,ni
do k=1,nk
r(i,k)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(4)= average(r,ni,nk)
deallocate(r)

!check i faces

allocate(r(nj,nk))
i=1
do j=1,nj
do k=1,nk
r(j,k)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(1)= average(r,nj,nk)

i=ni
do j=1,nj
do k=1,nk
r(j,k)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(2)= average(r,nj,nk)
deallocate(r)

!check k faces
allocate(r(ni,nj))
k=1
do i=1,ni
do j=1,nj
r(i,j)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(5)= average(r,ni,nj)

k=nk
do i=1,ni
do j=1,nj
r(i,j)= radius2(x(i,j,k),y(i,j,k),z(i,j,k))
end do
end do
avg(6)= average(r,ni,nj)
!    print *,avg
deallocate(r)
select case(dir)
case(1)
uploc=maxloc(avg(3:6))+2
lowloc=minloc(avg(3:6))+2
upval=maxval(avg(3:6))
lowval=minval(avg(3:6))
```

```fortran
        hc(uploc)=1
        hc(lowloc)=-1
!        hc(4)=1
!        hc(3)=-1

        case(2)

        avg(3)=(avg(1)+avg(2)+avg(5)+avg(6))/4 !ensures that jmin and jmax faces are not max
and min
        avg(4)=avg(3)
        uploc=maxloc(avg(1:6))
        lowloc=minloc(avg(1:6))
        upval=maxval(avg(1:6))
        lowval=minval(avg(1:6))

        hc(uploc)=1
        hc(lowloc)=-1
        case(3) ! k is inlet/exit
        !either i or j is hub-case direction
        uploc=maxloc(avg(1:4))
        lowloc=minloc(avg(1:4))
        upval=maxval(avg(1:4))
        lowval=minval(avg(1:4))
        hc(uploc)=1
        hc(lowloc)=-1

        end select

        contains
        function radius2(rad1,rad2,rad3) ! use for true radius
        real*8 :: rad1,rad2,rad3,radius2
        radius2=sqrt(rad2**2+rad3**2)
        end function radius2


        function average(rrr,n1,n2)
        real*8 ::avg,sumr,average
        integer::n1,n2,n3,i,j,k
        real*8,dimension(n1,n2)::rrr

        sumr=0.
        do i=1,n1
        do j=1,n2

        sumr=sumr+rrr(i,j)
        end do
        end do
        average=sumr/(n1*n2)

        end function average


        end subroutine check_hub_case

!*********************************************************************

!*********************************************************************
        subroutine periodic_fix

        implicit none
        integer :: num_b2b, num_special_b2b,b2b_rec_len
        integer, dimension(:),allocatable :: is,ie,js,je,ks,ke,blkb,d1,d1s
        integer, dimension(:),allocatable:: d1e,id,dir2,lor1,lor2,p_b2b
        integer, dimension(:),allocatable:: p_special,bc,p_bc
        integer,dimension(:), allocatable:: d2,d2s,d2e,d3,d3s,d3e,dir1
!        bc.in VARS
        integer :: num_bc_real,num_bc,dum
        integer, dimension(:),allocatable:: block_id,start_i,
     &                            start_j,start_k,end_i,end_j,end_k
        real,dimension(:),allocatable::bc_type_and_group
        integer, dimension(:),allocatable::gid,gname
```

```
!     local VARS
      character(len=50) :: gpro,dmapfile,infile,bcfile
      integer :: i,j,k,l,m,n,nblks,sumn,nb,ii,ijk
      integer :: i1,j1,k1,i2,j2,k2,total_recs
      integer:: v(1:100)
      real, dimension(:),allocatable::x,y,z
      real x1,x2,y1,y2,z1,z2
      logical xn1,yn1,zn1,xn2,yn2,zn2,off,on
      integer bs,be,temp

!     Reorient Vars
      integer:: mod_blk,ni3,nj3,nk3,task,nbgu,nbr,bld_psg
      integer::ni3new,nj3new,nk3new,nx
      integer::c4,c5,c6,pos1,pos2
      integer::dir(3),com(5)
      integer::ind1,ind2
      real,dimension(:,:,:),allocatable::x3,y3,z3,x3new,y3new,z3new

      character(len=50)::fname
      integer:: fid
      bcfile='bc.in'




! READ BC.IN AND STORE

      open(unit=10,file=bcfile,FORM='formatted',status='unknown')
      num_bc=0
      do
      read(10,*),dum
!      print *,dum
      if (dum.eq.0) EXIT
      num_bc=num_bc+1
      enddo
      close(10)

      !*****TO SIMPLIFY REWRITE INTO bc.in*********
      num_bc_real=num_bc
      num_bc=num_bc+1


      allocate(block_id(num_bc),bc_type_and_group(num_bc)
     &,start_i(num_bc),
     &start_j(num_bc),start_k(num_bc),end_i(num_bc),end_j(num_bc)
     &,end_k(num_bc))

!       print *,num_bc_real,'Boundary conditions found.'
      open(unit=10,file=bcfile,FORM='formatted',status='unknown')
      do i=1,num_bc
      read(10,*),block_id(i),
     &bc_type_and_group(i),start_i(i),
     &start_j(i),start_k(i),end_i(i),end_j(i)
     &,end_k(i)
      if (int(abs(bc_type_and_group(i))).eq.104
     &.or.int(abs(bc_type_and_group(i))).eq.105) then
      bc_type_and_group(i)=bc_type_and_group(i)
     &/abs(bc_type_and_group(i))*101.
      endif

      if (int(abs(bc_type_and_group(i))).eq.106
     &.or.int(abs(bc_type_and_group(i))).eq.107) then
      bc_type_and_group(i)=bc_type_and_group(i)
     &/abs(bc_type_and_group(i))*102.
      endif

      enddo

      close(10)
```

```fortran
      open(unit=10,file=bcfile,FORM='formatted',status='unknown')


       do i=1,num_bc
       write(10,'(2x,1I4,1x,F8.2,6I5,1("/"))'),block_id(i),
      &bc_type_and_group(i),start_i(i),
      &start_j(i),start_k(i),end_i(i),end_j(i)
      &,end_k(i)
       enddo
       close(10)

       deallocate(block_id,bc_type_and_group,start_i,
      &                       start_j,start_k,end_i,end_j,end_k)
!**************  READING dmap.in    *********************
      dmapfile='dmap.in'
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
      read(8,*) num_b2b
!       write(*,'(I3)') num_b2b
      do i=1,num_b2b
      read(8,*) v(1:b2b_rec_len)
      enddo
      read(8,*), num_special_b2b
      close(8)

! Find total number of records to store
      total_recs=num_b2b+num_special_b2b


! allocate total record length
       allocate(id(total_recs),is(total_recs),
      &ie(total_recs),js(total_recs),je(total_recs)
      &,ks(total_recs),ke(total_recs),blkb(total_recs)
      &,d1(total_recs),d1s(total_recs),d1e(total_recs)
      &,d2(total_recs),d2s(total_recs),d2e(total_recs)
      &, d3(total_recs),d3s(total_recs),d3e(total_recs)
      &,dir1(total_recs),dir2(total_recs),
      & lor1(total_recs),lor2(total_recs),bc(total_recs)) ! bc is only used for special b2b
!       print *,'*****************Reading dmap.in*****************'
       open(unit=8,file=dmapfile,FORM='formatted',status='unknown')

       read(8,*) num_b2b

       do i=1,num_b2b
       read(8,*) id(i),is(i),
      & ie(i),js(i)
      & ,je(i),ks(i),ke(i),blkb(i),
      & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
      & ,dir1(i),dir2(i),lor1(i),lor2(i)
       enddo


        read(8,*), num_special_b2b
       do i=num_b2b+1,num_b2b+num_special_b2b
       read(8,*) id(i),is(i),
      &ie(i),js(i)
      & ,je(i),ks(i),ke(i),blkb(i),
      & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
      & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)
       if (int(abs(bc(i))).eq.104
      &.or.int(abs(bc(i))).eq.105) then
       bc(i)=bc(i)
      &/abs(bc(i))*101.
       endif

       if (int(abs(bc(i))).eq.106
      &.or.int(abs(bc(i))).eq.107) then
       bc(i)=bc(i)
```

```
     &/abs(bc(i))*102.
      endif
      end do

      close(8)
! dmap.in CLOSED HERE


! WRITE OUT NEW DMAP.IN AND BC.IN

!      fname=file_cat(dmapfile,'.new')
      open(unit=8,file=dmapfile,FORM='formatted',status='unknown')
!      print *,'Number of block to block interfaces'
      write(8,'(I5)') num_b2b
      do i=1,num_b2b
      write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1("/"))') id(i),is(i),
     & ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i)
      enddo

      write(8,'(I5)'), num_special_b2b
      do i=num_b2b+1,num_b2b+num_special_b2b
      write(8,'(1x,8I4,3(1I2,2I4),4I2,1x,1I4,1x,1("/"))') id(i),is(i),
     &ie(i),js(i)
     & ,je(i),ks(i),ke(i),blkb(i),
     & d1(i),d1s(i),d1e(i),d2(i),d2s(i),d2e(i),d3(i),d3s(i),d3e(i)
     & ,dir1(i),dir2(i),lor1(i),lor2(i),bc(i)

      end do
      close(8)

      end subroutine periodic_fix


!Routine to write pmap.in from GU file size
!2 methods used here. user can decide which one to use based on pmap.report file
!all pmap.in files genereated ...user decides which to use and renames to pmap.in
!Vikram Shyam - 3/26/09

      subroutine make_pmap
      implicit none
      integer:: i,j,k,l,n,m,nb,nbr,bld_psg,nbgu,dum1,dum2,dum3
      integer::ni1,nj1,nk1,fid,reply,abort_code,ngu
      integer::ni,nj,nk,tot_bks
      integer::lowloc,temp,low,found_max,mloc(1)
      integer,dimension(:),allocatable::ijk,guid
      integer avg_size,max_size,num_procs_rec,rec_procs
      integer total_size
      integer,dimension(:),allocatable::proc,num_procs,proc_temp
      integer,dimension(:,:),allocatable::pid
      real theta,ds
      character(len=50)::fname,oname,file_name0
      logical::p3d_exists,gu_exists
      integer out_size

      out_size=8
      nbgu=1
      nbr=1
      bld_psg=1
!      print *,'BEWARE: THIS WILL READ ALL
!     &         GU FILES CURRENTLY IN THIS FOLDER'
!      print *,'Make sure this is compiled with -r8 option'
!      oname= 'GU.r1b1.p3d.r' ! default name
      ngu=0
      n=1
      do
      fid = 10*n
      fname = file_name0('GU',fid)
      inquire(file=fname,exist=gu_exists)
!       print *,fname,gu_exists,n
```

```
          if (.not.gu_exists) EXIT
          n=n+1
          enddo

          ngu=n-1


!          print *,'Total blocks found: ',ngu

          allocate(ijk(ngu),guid(ngu))

          do n=1,ngu

          fid = 10*n
          fname = file_name0('GU',fid)
          open(unit=7,file=fname,status='UNKNOWN',form='UNFORMATTED')
          read(7) nbgu,nbr,bld_psg
          read(7) ni,nj,nk
!          print *,fname,ni,nj,nk,ngu,n
          ijk(n)=ni*nj*nk
          guid(n)=n
!          print *,fname,'size: ',ijk(n)
          close(7)
          enddo



          do i=1,ngu
          low=ijk(i)
          lowloc=i
          do j=i+1,ngu
          if (ijk(j)<low) then
          low=ijk(j)
          lowloc=j
          endif
          enddo
          temp=ijk(i)
          ijk(i)=low
          ijk(lowloc)=temp
          temp=guid(i)
          guid(i)=guid(lowloc)
          guid(lowloc)=temp
          enddo
!          print *,'Sorted List'
!          do i=1,ngu
!          print *,guid(i),ijk(i)
!          enddo
          max_size=maxval(ijk(1:ngu))
          total_size=sum(ijk)
          avg_size=((total_size)/(ngu))
          rec_procs=ceiling(total_size/real(max_size))
          print *,'Creating pmap files for multiblock per cpu simulations'
          write(*,*)'======================================================'

          print *,'Average_size|total_size|maximum_size|num_procs_recmnd'
          print *,avg_size,total_size,max_size,rec_procs
          write(*,*)'======================================================'

          open(unit=9,file='pmap.report',status='UNKNOWN',FORM='formatted')
          write(9,*) '**********Using Method 3**********'

          do num_procs_rec=1,rec_procs
          write(9,*)'======================================================'
          write(9,*)'Load distribution for num_procs=',num_procs_rec
          max_size=maxval(ijk(1:ngu))
          total_size=sum(ijk)
          avg_size=((total_size)/(ngu))
          rec_procs=ceiling(total_size/real(max_size))
!          print *,'Average total maximum number procs recommended'
!          print *,avg_size,total_size,max_size,rec_procs
```

```fortran
      if (num_procs_rec/=rec_procs) then
      max_size=ceiling(real(total_size)/real(num_procs_rec))
      max_size=max_size+ceiling(.05*real(max_size))
      !print *,'New max size is',max_size
      endif
      call pmap_m3(ijk,guid,ngu,num_procs_rec,max_size,
     &total_size,rec_procs)
      enddo

      write(9,*)'*****************************************************'
      write(9,*)'*******************Using Method 2******************'
      write(9,*)'*****************************************************'
      max_size=maxval(ijk(1:ngu))
      total_size=sum(ijk)
      avg_size=((total_size)/(ngu))
      rec_procs=ceiling(total_size/real(max_size))


      do num_procs_rec=1,rec_procs
      write(9,*)'====================================================='
      write(9,*)'Load distribution for num_procs=',num_procs_rec
      max_size=maxval(ijk(1:ngu))
      total_size=sum(ijk)
      avg_size=((total_size)/(ngu))
      rec_procs=ceiling(total_size/real(max_size))

      if (num_procs_rec/=rec_procs) then
      max_size=ceiling(real(total_size)/real(num_procs_rec))
      max_size=max_size+ceiling(.05*real(max_size))
      !print *,'New max size is',max_size
      endif
      call pmap_m2(ijk,guid,ngu,num_procs_rec,max_size,
     &total_size,rec_procs)
      enddo
      close(9)



      deallocate(ijk,guid)
!        print *,'How many procs do you want to use?'
!        read *,num_procs_rec

      end subroutine

      subroutine write_pmap(pid,num_procs_rec,ngu,num_procs,proc,method)
      implicit none
      integer:: i,j,k,l,n,m,nb,nbr,bld_psg,nbgu,dum1,dum2,dum3
      integer::ni1,nj1,nk1,fid,reply,abort_code,ngu
      integer::ni,nj,nk,tot_bks
      integer::lowloc,temp,low,found_max,mloc(1)
      integer avg_size,max_size,num_procs_rec,rec_procs
      integer total_size
      integer::proc(1:num_procs_rec),num_procs(1:num_procs_rec)
      integer::pid(1:num_procs_rec,1:ngu)
      real theta,ds
      character(len=50)::fname,oform,file_name0,file_cat
      logical::p3d_exists,gu_exists
      integer out_size,method

      fname=file_name0('pmap.in.',num_procs_rec)
      l=len_trim(fname)
      fname=fname(1:l)
!        print *,fname

      fname=file_cat(fname,'.m')
      l=len_trim(fname)
      fname=fname(1:l)
!        print *,fname
      fname=file_name0(fname(1:l),method)
      print *,fname, ' has been created.'
      open(unit=8,file=fname,status='UNKNOWN',form='FORMATTED')
```

```
            oform=file_name0('(',num_procs_rec)
            oform=file_cat(oform,'I4)')
    !       print *,oform
            write(8,oform) num_procs
            do i=1,num_procs_rec
            oform=file_name0('(',num_procs(i))
            oform=file_cat(oform,'I4)')
            write(8,oform)pid(i,1:num_procs(i))
            enddo
            close(8)

            end subroutine


            subroutine pmap_m3(ijk,guid,ngu,num_procs_rec,max_size,
           &total_size,rec_procs)
             implicit none
             integer:: i,j,k,l,n,m,nb,nbr,bld_psg,nbgu,dum1,dum2,dum3
             integer::ni1,nj1,nk1,fid,reply,abort_code,ngu
             integer::ni,nj,nk,tot_bks
             integer::lowloc,temp,low,found_max,mloc(1)
             integer::ijk(1:ngu),guid(1:ngu),flag(1:ngu)
             integer avg_size,max_size,num_procs_rec,rec_procs
             integer total_size
             integer,dimension(:),allocatable::proc,num_procs,proc_temp
             integer,dimension(:,:),allocatable::pid
             real theta,ds
             logical::p3d_exists,gu_exists
             integer out_size,method
             character(len=50)::fname,oform,file_name0,file_cat
             method=3
                    flag(1:ngu)=1
              allocate(proc(num_procs_rec),pid(num_procs_rec,ngu),
           &num_procs(num_procs_rec),proc_temp(num_procs_rec))
              proc(1:num_procs_rec)=0
              num_procs(1:num_procs_rec)=0
              proc_temp=proc
              pid(1:num_procs_rec,1:ngu)=0




    !        do i=1,num_procs_rec
             i=1
             proc(i)=ijk(ngu-i+1) !setup 10 largest blocks each on 1 proc
             flag(ngu-i+1)=0
             pid(i,1)=guid(ngu)
             num_procs(i)=1
    !         enddo

             do i=ngu,1,-1       !try to put the largest block on the largest proc to fill it up as
    much as possible
             if (flag(i).eq.1) then
    !          print *,'Unassigned block: ',guid(i),ijk(i)
             proc_temp=proc
    100        mloc=maxloc(proc_temp)

             m=mloc(1)
             if (proc(m)+ijk(i)>max_size) then !if adding this block exceeds the max size then do
    not use this proc
             proc_temp(m)=-999

             if (maxval(proc_temp).eq.-999) then ! if all processors are full, cannot accomodate
    this block at this time.
             goto 200
             else
             goto 100
             endif

             else
    !          print *,'before ', proc(m),num_procs(m)
```

```
        proc(m)=proc(m)+ijk(i)
        l=num_procs(m)+1
        pid(m,l)=guid(i)
        num_procs(m)=l
        flag(i)=0
!        print *,'Assigned to processor: ',m
        endif
200       endif
        enddo

        tot_bks=0
        do i=1,num_procs_rec
        write(9,*)'Processor ',i,' has size',
     & proc(i), ' and ',num_procs(i),' blocks'
!        print *,'Processors are:',pid(i,1:num_procs(i))
!        write(*,*)'Processor sizes for ',num_procs_rec,' processors:'
!        oform=file_name0('(',num_procs(i))
!        oform=file_cat(oform,'I10)')

!        print *,proc(1:num_procs_rec)
!         do
        tot_bks=tot_bks+num_procs(i)
        enddo
        ds=abs(real(maxval(proc)-minval(proc))/real(maxval(proc)))*100.
        write(9,*)'Total blocks assigned = ', tot_bks
        write(9,*)'Percentage diff between largest and smallest: ',ds
        call write_pmap(pid,num_procs_rec,ngu,num_procs,proc,method)

        deallocate(num_procs,proc,pid,proc_temp)



         end subroutine


         subroutine pmap_m2(ijk,guid,ngu,num_procs_rec,max_size,
        &total_size,rec_procs)
         implicit none
         integer:: i,j,k,l,n,m,nb,nbr,bld_psg,nbgu,dum1,dum2,dum3
         integer::ni1,nj1,nk1,fid,reply,abort_code,ngu
         integer::ni,nj,nk,tot_bks
         integer::lowloc,temp,low,found_max,mloc(1)
         integer::ijk(1:ngu),guid(1:ngu),flag(1:ngu)
         integer avg_size,max_size,num_procs_rec,rec_procs
         integer total_size
         integer,dimension(:),allocatable::proc,num_procs,proc_temp
         integer,dimension(:,:),allocatable::pid
         real theta,ds
         logical::p3d_exists,gu_exists
         integer out_size,method
        method=2
                flag(1:ngu)=1
         allocate(proc(num_procs_rec),pid(num_procs_rec,ngu),
        &num_procs(num_procs_rec),proc_temp(num_procs_rec))
         proc(1:num_procs_rec)=0
         num_procs(1:num_procs_rec)=0
         proc_temp=proc
         pid(1:num_procs_rec,1:ngu)=0


        do i=1,num_procs_rec
        proc(i)=ijk(ngu-i+1) !setup 10 largest blocks each on 1 proc
        flag(ngu-i+1)=0
        pid(i,1)=guid(ngu-i+1)
        num_procs(i)=1
        enddo

        do i=1,ngu
        if (flag(i).eq.1) then
!        print *,'Unassigned block: ',guid(i),ijk(i)
```

```fortran
        mloc=minloc(proc)
        m=mloc(1)
!         print *,'before ', proc(m),num_procs(m)
        proc(m)=proc(m)+ijk(i)
        l=num_procs(m)+1
        pid(m,l)=guid(i)
        num_procs(m)=l
        flag(i)=0
!         print *,'Assigned to processor: ',m
        endif
        enddo


        tot_bks=0
        do i=1,num_procs_rec
        write(9,*)'Processor ',i,' has size',
     & proc(i), ' and ',num_procs(i),' blocks'
!         print *,'Processors are:',pid(i,1:num_procs(i))
        tot_bks=tot_bks+num_procs(i)
        enddo
        ds=abs(real(maxval(proc)-minval(proc))/real(maxval(proc)))*100.
        write(9,*)'Total blocks assigned = ', tot_bks
        write(9,*)'Percentage diff between largest and smallest: ',ds
        call write_pmap(pid,num_procs_rec,ngu,num_procs,proc,method)


        deallocate(num_procs,proc,pid,proc_temp)



        end subroutine




        function file_cat(pre,post)
        implicit none
        integer n,ints,inte
        character(len=*),intent (in) :: pre,post
        character(len=50) :: file_cat
        ints=len_trim(pre)
        inte=len_trim(post)
        write(file_cat,*)pre(1:ints),post(1:inte)
        return
        end function file_cat


!***************END SUBROUTINES*****************************
```

References

1. Chen, J.P., and Whitfield, D.L., "Navier-Stokes Calculations for the Unsteady Flowfield of Turbomachinery," AIAA–93–0676, January 1993.
2. Chen, J.P., and Barter, J., "Comparison of Time-Accurate Calculations for the Unsteady Interaction in Turbomachinery Stage," AIAA–98–3292, July 1998.
3. Chen, J.P., and Briley, W.R., "A Parallel Flow Solver for Unsteady Multiple Blade Row Turbomachinery Simulations," ASME–2001–GT–0348. June 2001.
4. Hill, P., and Peterson, C., "Mechanics and Thermodynamics of Propulsion," Addison Wesley Publishing Company, 1992.
5. Chen, J.P., "Unsteady Three-Dimensional Thin-Layer Navier-Stokes Solutions for Turbomachinery in Transonic Flow," PhD Dissertation, Mississippi State, Mississippi, December 1991.
6. http://www.gridpro.com/, Program Development Company, White Plains NY, 10601, USA.
7. http://www.simcenter.msstate.edu/simcenter/docs/msu_turbo/, MSUTURBO Online Documentation, Engineering Research Center, MI 39762.
8. http://www.ilight.com/, Intelligent Light, Rutherford NJ, 07070, USA.
9. Nyhoff, L.R., Leestma, S.C., "Fortran 90 for Engineers and Scientists," Prentice Hall, Inc., 1997.
10. Shyam, V., "3-D Unsteady Simulation of a Modern High Pressure Turbine Stage: Analysis of Heat Transfer and Flow," PhD Thesis, The Ohio State University, Columbus, Ohio, December 2009.
11. Shyam, V., Ameri, A., Luk, D.F., and Chen, J.P., "3-D Unsteady Simulation of a Modern High Pressure Turbine Stage Using Phase Lag Periodicity: Analysis of Flow and Heat Transfer," ASME GT2009–60322.

1. REPORT DATE (DD-MM-YYYY) 01-06-2010	2. REPORT TYPE Technical Memorandum	3. DATES COVERED (From - To)

4. TITLE AND SUBTITLE
Preprocessor that Enables the Use of GridPro^{TM} Grids for Unsteady Reynolds-Averaged Navier-Stokes Code TURBO

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Shyam, Vikram

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
WBS 561581.02.08.03.21.14.03

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, Ohio 44135-3191

8. PERFORMING ORGANIZATION REPORT NUMBER
E-17316

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITOR'S ACRONYM(S)
NASA

11. SPONSORING/MONITORING REPORT NUMBER
NASA/TM-2010-216739

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Categories: 61 and 34
Available electronically at http://gltrs.grc.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 443-757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A preprocessor for the Computational Fluid Dynamics (CFD) code TURBO has been developed and tested. The preprocessor converts grids produced by GridPro (Program Development Company (PDC)) into a format readable by TURBO and generates the necessary input files associated with the grid. The preprocessor also generates information that enables the user to decide how to allocate the computational load in a multiple block per processor scenario.

15. SUBJECT TERMS
Grids; Multiblock grids

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON STI Help Desk (email:help@sti.nasa.gov)
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	UU	90	19b. TELEPHONE NUMBER (include area code) 443-757-5802

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18