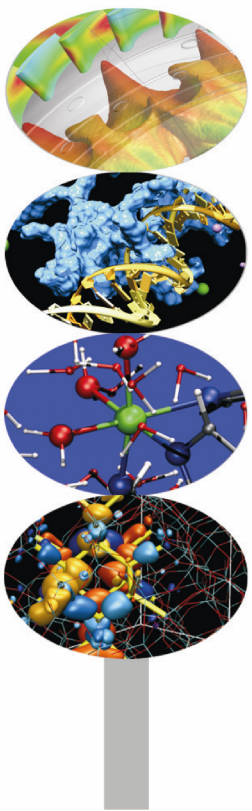
CINECA SCAI
SuperComputing Applications and Innovation

CINECA COURSES High Performance Computing 2014

HPC enabling of OpenFOAM® for CFD applications



How to submit an OpenFOAM job to the cluster

26-28 March 2014, Casalecchio di Reno, BOLOGNA.

SuperComputing Applications and Innovation Department, CINECA

PRACE CINECA

Table of Contents

Objectives and Topics

- Objective
  Show how to set-up and submit your OpenFOAM job in HPC environment:
- Topics
  - Case directory structure
  - Run the `icoFoam cavity` tutorial in your shell
  - Run the `icoFoam cavity` tutorial via batch job
  - Run the `icoFoam pitzDaily` tutorial via batch job in parallel

Table of Contents

OpenFOAM environment

Check that OpenFOAM is loaded in your shell environment with the command

```
[a08tra89@node343 ~]$ module list
```

if not, loaded it

```
[a08tra89@node343 ~]$ module load autoload openfoam/2.3.0-gnu-4.7.2
```

Test the path to the `icoFoam` executable:

```
which icoFoam
```

You will see the full path to the selected executable.
Check that your `$WM_PROJECT_USER_DIR` and your `run` `$WM_PROJECT_USER_DIR/run` exists

```
echo WM_PROJECT_USER_DIR
/plx/usertrain/a08tra89/OpenFOAM/a08tra89-2.3.0
echo $WM_PROJECT_USER_DIR/run
/plx/usertrain/a08tra89/OpenFOAM/a08tra89-2.3.0/run
```

- Use the predefined alias `foam` and `tut` to go, respectively, to the `$WM_PROJECT_USER_DIR` and your `tutorial` directory, where there are complete set-ups of cases for all the solvers.
- Make a copy of the tutorial to your `$WM_PROJECT_USER_DIR/run` directory before running. You have the permission only to read the installed tutorial directory.
- There is no specific tutorials for the utilities,
  but some solver tutorials also show how to use the utilities.

Case directory structure

- We will use the `icoFoam cavity` tutorial as a general example of case directory structure, how to set up and run the applications in OpenFOAM.
- Start by copying the tutorial to your `run` directory

```
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity $FOAM_RUN
cd $FOAM_RUN/cavity
```

Go to your `FOAM_RUN` and have a look to the case directory structure, as follow

```
<case>
|
|-- system
|     |
|     |-controlDict (control parameters: time step, grid spacing, max Courant number)
|     |
|     |-fvSchemes (discretization schemes for grad, div, laplacian, time integration, interpolation)
|     |
|     |-fvSolution (linear algebra solvers for the discretized linear system)
|
|-- constant
|     |
|     |-transportProperties (viscosity, gravity, etc.)
|     |
|     |- polyMesh (mesh generation files by BlockMeshDict)
|          |
|          |-blockMeshDict
|          |
|          |-points
|          |
|          |-cells
|          |
|          |-faces
|          |
|          |-boundary
|-- time directories (initial 0 and boundary conditions)
```

Table of Contents

Run tutorial in your shell

Remember: interactive job in your shell max 10 minutes. Use with caution. Do not overload the login with memory intensive operations (it is not our case)

- The mesh is defined by a dictionary that is read by the `blockMesh` utility. Create the mesh by typing

  ```
  [a08tra89@node343 ~]$ blockMesh
  ```

  You have now generated the mesh in OpenFOAM format.

- Check the mesh by typing

  ```
  [a08tra89@node343 ~]$ checkMesh
  ```

  You will see the mesh size, the geometrical size and some checks.

- This is the case for the `icoFoam` solver, so run it with the command

  ```
  [a08tra89@node343 ~]$ icoFoam > icofoam.log&
  ```

  You will run the simulation, using 1 proc, in background in the `login2` node, using the settings specified in the case dir. The `log` file reports the Courant numbers and the residuals.

- You will see in your working dir, the new time directories.

  ```
  [a08tra89@node343 cavity]$ ls
  0  0.1  0.2  0.3  0.4  0.5  constant  icofoam.log  system
  ```

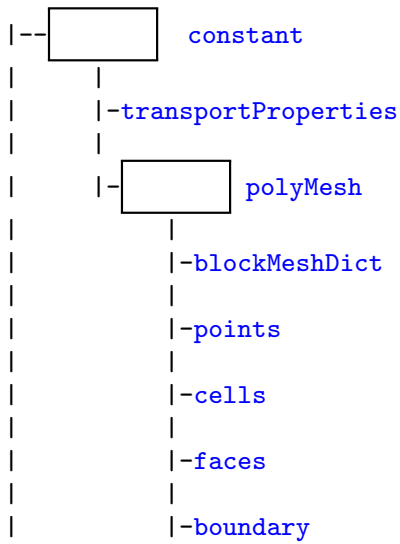
The icoFoam tutorial: a look inside

We have a look at what we did when running the cavity tutorial by looking inside the case files

- First of all, remember that the solver icoFoam is a *Transient solver for incompressible, laminar flow of Newtonian fluids*
- The case directory originally contains the following sub-directories: 0, constant and system. After the run it also contains the output directories: 0.1, 0.2, 0.3, 0.4, 0.5 and log
  - The 0* directories contain the values of all the variables at those time steps. Hence the 0 directory is the initial condition.
  - The constant directory contains the mesh and dictionaries for the thermophysical and turbulence models.
  - The system directory contains settings for the run, discretization schemes and solution procedures.
- The icoFoam solver reads the files in the case directory and runs the case according to those settings.
  In the next slides we will have a quick look the the dictionaries files to better understand how we have set the test case.

The constant directory

The constant directory has the following structure:



- The transportProperties file is the dictionary for the dimensioned scalar kinematic viscosity ν (m^2/s, in SI system)

```
FoamFile
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

nu              nu [ 0 2 -1 0 0 0 0 ] 0.01;


 ************************************************************************
```

The blockMeshDict file

The `blockMeshDict` dictionary is used to set-up the mesh generation utility `blockMesh`.
First of all, it contains a numbers of vertices:

```
FoamFile
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;  It applies a scaling factor for the vertex coordinates. Scales to dm.

vertices  There are eight vertices defininining a 3D block. OpenFOAM always uses a 3D meshes
even if the simulation is 2D
(
    (0 0 0)   // vertex number 0
    (1 0 0)   // vertex number 1
    (1 1 0)   // vertex number 2
    (0 1 0)   // vertex number 3
    (0 0 0.1) // vertex number 4
    (1 0 0.1) // vertex number 5
    (1 1 0.1) // vertex number 6
    (0 1 0.1) // vertex number 7
);
```

The blockMeshDict file

Secondly, it defines a block and the mesh for the vertices

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading (1 1 1)
);

edges
(
);
```

- hex means that is a structured hexahedral mesh
- (0 1 2 3 4 5 6 7) is the list of the vertices used to define the block. The order is important, they have to form a right-hand side system (see the User-Guide)
- (20 20 1) is the number of mesh *cells* in each direction
- simpleGrading is the cell expansion ratio, in this case equidistant. The expansion ratio enables the mesh to be graded, or refined, in specified directions. The ratio $ex_r = \frac{\delta_e}{\delta_s}$ is that of the width of the end cell δ_e along one edge of a block to the width of the start cell δ_s along that edge. There are other grading system as edgeGrading (see User-Guide for more info).

The blockMeshDict file

Finally, the boundary of the mesh is given in a list named boundary.

```
boundary
(
   movingWall // patch name (user choice) It contains two sub-dictionary
    {
        type wall; // the patch type, either a generic patch or a particular geometric condition
        faces      // a list of block faces that make up the patch. The order in which the vertices are given
        must be such that, looking from inside the block and starting with any vertex,
        the face must be traversed in a clockwise direction to define the other vertices.
        (
            (3 7 6 2)
        );
    }
    fixedWalls // patch name (user choice)
    {
        type wall;
        faces
        (
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
        );
    }
    frontAndBack // patch name (user choice)
    {
        type empty; //  2 dimensional geometry
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

The boundary is broken into patches (regions), in the example movingWall, fixedWalls and frontAndBack, where each patch in the list has its name as the keyword, which is the choice of the user. It is recommend something that conveniently identifies the patch, e.g.inlet; the name is used as an identifier for setting boundary conditions in the field data files. More info at User-Guide.

The blockMeshDict file: summary

To sum up, the `blockMeshDict` dictionary generates a block with:

- x/y/z dimensions 0.1/0.1/0.1 m
- 20 $\times$ 20 $\times$ 1 cells
- boundary conditions type: 1 `movingWall` faces, 3 `fixedWall` face, 2 `empty` `frontAndBack` faces
- the type `empty` tells OpenFOAM that is a 2 dimensional case.

OpenFOAM supports natively two mesh generators:

1. The mesh generation utility `blockMesh` to generate simple meshes of blocks of hexahedral cells.
2. The mesh generation utility `snappyHexMesh` for generating complex meshes of hexahedral and split-hexahedral cells automatically from triangulated surface geometries

For more info, please have a look to the Mesh generation and conversion User Guide.
There are different options available for conversion of a mesh that has been generated by a third-party product into that of OpenFOAM can read (`fluentMeshToFoam`, `starToFoam`, `gambitToFoam`, `ideasToFoam`, `cfx4ToFoam`).
Moreover there are commerical mesh generator tools that write the mesh directly in OpenFOAM format (ex: PointWise®).

The system directory

The system directory has the following structure:

```
|--      system
   |      |
   |      |-controlDict (control parameters: time step, grid spacing, max Courant number)
   |      |
   |      |-fvSchemes (discretization schemes for grad, div, laplacian, time integration, interpolation)
   |      |
   |      |-fvSolution (linear algebra solvers for the discretized linear system)
   |
   |
   |
```

- The controlDict contains general instructions on how to run the case
- The fvSchemes contains instructions on which discretization schemes that should be used for different terms in the equations.
- The fvSolution contains instructions on how to solve each discretized linear equation system. It also contains instructions for the PISO pressure-velocity coupling

The controlDict dictionary

The `controlDict` dictionary consist of the following lines:

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     icoFoam;  // names of the application the tutorial is set up for
startFrom       startTime;
startTime       0;          // These lines tells icoFoam to start at startTime=0,
stopAt          endTime;    // and stop at the endTime=0.5 with a TimeStep deltaT=0.005
endTime         0.5;
deltaT          0.005;
writeControl    timeStep;  // These lines telles icoFoam to write the results in separate directories
writeInterval   20;         // (purgeWrite=0) every 20 timeSteps, and that they should be written in
purgeWrite      0;          // uncompressed ascii format with write precision 6.
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;    // timeFormat and timePrecision are used for the format
timePrecision   6;          // of the naming of the time directories
runTimeModifiable true;     // Allows you to make modifications to the case while is running
```

The OpenFOAM solvers begin all runs by setting up a database. The `controlDict` dictionary sets input parameters essential for the creation of the database Only the `time control` and `writeInterval` entries are truly compulsory. For more info, please have a look to the Time and data I/O control User Guide.

The fvSchemes dictionary

The `fvSchemes` dictionary defines the discretization schemes, in particular the time marching scheme and the convection scheme for the spatial discretization

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
ddtSchemes
{
    default         Euler;
}
divSchemes
{
    default          none;
    div(phi,U)       Gauss linear;
}
```

- We use the `Euler` implicit temporal discretization, and the `linear` (central-difference) scheme for convection.
- `none` means that the scheme must be explicitly specified.
- There are more than 50 alternatives for the convection scheme, and the number is increasing.

The terms that must typically be assigned a numerical scheme in fvSchemes range from derivatives, e.g. gradient and interpolations of values from one set of points to another. The aim in OpenFOAM is to offer an unrestricted choice to the user.

For more info, please have a look to the Numerical Scheme User Guide.

The fvSolution dictionary

The `fvSolution` dictionary defines the solution procedure. The equation solvers, tolerances and algorithms are controlled by this dictionary.
The solution of the pressure `p` linear equation system required for the `icoFoam` solver looks like:

```
solvers
{
    p
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;
        relTol          0;
    }
```

- The `p` linear equation system is solved using the Preconditioned Conjugate Gradient solver `PCG`, with the preconditioner `DIC` Diagonal incomplete-Cholesky (symmetric).
- The solution is considered converged when the residual has reached the `tolerance`, or if it has been reduced by `relTol` at each time step.
- Have a look yourself the the solution of the `U` linear equation system.

For more info, please have a look to the Solution and algorithm control User Guide and cite Benzi.

The PISO and SIMPLE algorithms

Most fluid dynamics solver applications in OpenFOAM use the

- *Pressure-Implicit Split-Operator* `PISO` algorithm, for transient problems.
- *Semi-Implicit Method for Pressure-Linked Equations* `SIMPLE` algorithm for steady-state problems.
- These algorithms are iterative procedures for solving equations for velocity and pressure. Both algorithms are based on evaluating some initial solutions and then correcting them.

```
PISO
{
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    pRefCell        0;
    pRefValue       0;
}
```

- SIMPLE only makes 1 correction (keyword `nCorrectors`) whereas PISO requires more than 1, but typically not more than 4.
- `nNonOrthogonalCorrectors` add correctors for non-orthogonal meshes, which may sometimes influence the solution, specially for very skewed (bad quality) meshes.

The 0 directory

The 0 directory contains the dimension, the initial and boundary conditions for all primary variables. In this case p and U. As example, see below the U field

```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
dimensions      [0 1 -1 0 0 0 0]; // state the dimensions of U m/s
internalField   uniform (0 0 0);  // sets the U field to zero internally
boundaryField
{
    movingWall
    {
        type            fixedValue;
        value           uniform (1 0 0);
    }
    fixedWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }
}
// ************************************************************************* //
```

- The boundaryPatches movingWall and fixedWall are given the type fixedValue; Values uniform $U_x = 1$ m/s and $U_x = 0$, respectively.
- The frontAndBack patch is given type empty, indicating that no solution is required in the that direction since the case is $2D$.
- You can have a look yourself to the 0/p directory.

Visualization: mesh and global fields

To visual the mesh and the global fields use RCM.

- The Remote Connection Manager (RCM) is an application that allows HPC-users to perform remote visualization on Cineca HPC clusters.
- Open a session from your local workstation

```
cd run
cd cavity
touch cavity.openFOAM
```

- Open with ParaView and select data format OpenFOAM.
- You do not need to use `paraFoam` that is a parser to a specific version of `ParaView`.
- In this case you can use your version of `openFOAM` with the selected version of `ParaView`.

More details, in tomorrow presentation.

Table of Contents

Run tutorial in your batch

We will run the cavity tutorial via batch script. This tutorial is a **serial** job, we require only 1 cpu.
In the following batch script, we will execute the pre-processing `blockMesh` and the executable `icoFoam`.

- Copy the batch script in your `$FOAM_RUN/cavity`

  ```
  [a08tra89@node343 ~]$ cp $FOAM_CINECA_SCRIPT/serial.pbs $FOAM_RUN/cavity
  ```

- Move to the working dir and remove the time directories, if presents

  ```
  [a08tra89@node343 ~]$ cd $FOAM_RUN/cavity
  [a08tra89@node343 ~]$ rm -rf 0.*
  ```

- Now have a look to the script
- And modify it, according to your account and the selected queue

Serial batch script 1/2


```
#!/bin/bash
#PBS -l walltime=05:00
#PBS -l select=1:ncpus=1:mpiprocs=1
## Job name
#PBS -N foam_serial
#### Standard output and error
#PBS -o serial.out
#PBS -e serial.err
#### Submission queue
#PBS -q private -W group_list=train_copf2014
#PBS -A train_copf2014
# redirect stdout and stderr
# -o log
# -e log.err
#PBS -j oe
#PBS -m bea
#PBS -M i.spisso@cineca.it
#### end PBS directivies
```

This part of script includes the PBS directives that begins with the symbol #.
Please note that ## is a comment for PBS job scheduler

Serial batch script 2/2


```
module load autoload/0.1/verbose
module load openfoam/2.3.0-gnu-4.7.2

# move to directory where the case has been submitted
cd $PBS_O_WORKDIR

## set the solver
solver=icoFoam
blockMesh >blockMesh.$PBS_JOBID
checkMesh > checkMesh.$PBS_JOBID
$solver  > run.$PBS_JOBID
```

- The blue section load the module and move to the working dir
- The red section create the mesh with `blockMesh`, check it with `checkMesh` and execute the solver `icoFoam` in serial using the same cpu.
- The suffix `$PBS_JOBID` will append the jobid to the name of the logfile. After the execution, you will see something like `run.1595057.node351.plx.cineca.it`

Submit the job in queue system

```
[a08tra89@node343 ~]$ qsub serial.pbs
```

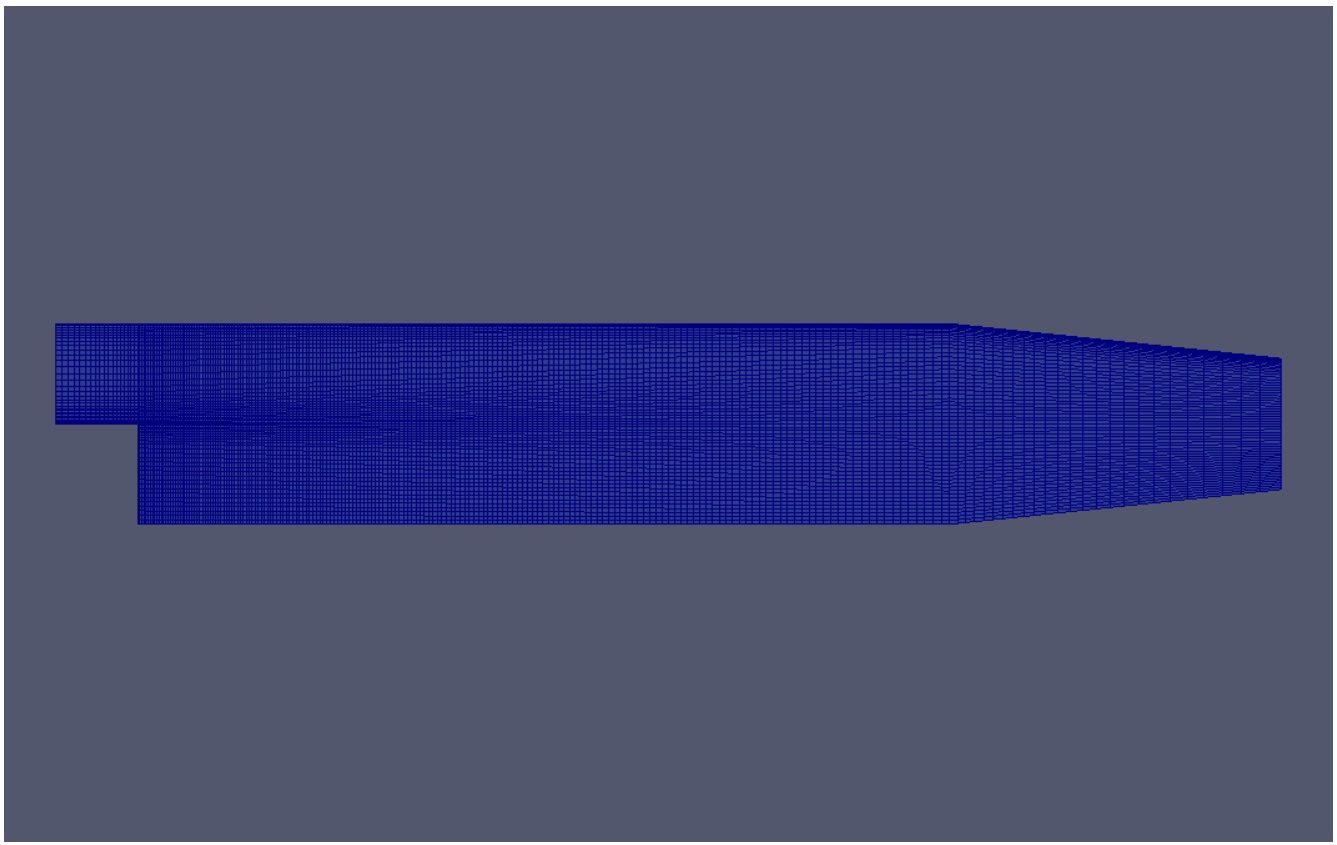
Table of Contents

Running in parallel

The method of parallel computing used by `OpenFOAM` is based on the standard Message Passing Interface (MPI) using the strategy of domain decomposition.
The geometry and the associated fields are broken into pieces and allocated to separate processors for solution.
A convenient interface, `Pstream`, is used to plug any Message Passing Interface (MPI) library into OpenFOAM

1. The first step is to decompose the computational domain using the `decomposePar` utility, which is set-up by the corresponding `decomposeParDict` dictionary located in the `system` directory of the case.
2. The parallel running uses the public domain openMPI implementation of the standard Message Passing Interface (MPI). In your case your have `$FOAM_MPI=openmpi-system` which use the `openmpi/1.6.3--gnu--4.7.2` library. Each processor run a copy of the solver with one separate part of the domain mesh.
3. Finally the solution is reconstructed with the `reconstructPar` utility, to obtain the final result.

pitzDaily at a glance



Run the pitzDaily tut in parallel

We will run the pitzDaily tutorial via batch script. This tutorial is a **parallel** job. In this tutorial we will:

1. execute the pre-processing, which include decomposePar, according to the selected domain decomposition strategy.
2. execute the job batch in parallel
3. Reconstruct the fields with reconstructPar

- Make a copy of the pitzDaily case to your run directory

  ```
  cp -r  $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily $FOAM_RUN/.
  ```

- copy into the system directory the decomposeParDict file available in the pitzDailyExptInlet directory

  ```
  cp  $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDailyExptInlet/system/decomposeParDict
  $FOAM_RUN/pitzDaily/system/.
  ```

- Move to the case directory

  ```
  [a08tra89@node343 ~]$ cd $FOAM_RUN/pitzDaily
  ```

- Now have a look to the decomposeParDict Dictionary.

The decomposeParDict dictionary

The geometry and fields are broken up according to a set of parameters specified in the decomposeParDict dictionary, which must be located in the system directory of the case of interest.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
numberOfSubdomains 4; // The user must set the number of domains which the case has to be decompose into
it corresponds to the number of cores available for the computations
method         hierarchical; // the user has the choice of six mothods of decomposition, specified by method.
For each method there are a set of coefficients specified in a sub-dictionary, named <method>Coeffs,
used to instruct the decomposition process
simpleCoeffs
{
    n               ( 2 1 1 );
    delta           0.001;
}
hierarchicalCoeffs
{
    n               ( 2 2 1 ); // where n is the number os sub-domains in the x,y and z directions
    delta           0.001;     // delta is the cell skew factor
    order           xyz;       // order is the order of decomposition xyz/yzx/zxy....
}
manualCoeffs
{
    dataFile        "";
}
distributed     no; // Data files may be distributed if local disks are used in order to improve I/O performace.
roots           ( ); Data may be distributed among different machines.
```

Domain decomposition method


```
method    simple/hierarchical/scotch/manual
```

- `simple` Simple geometric decomposition in which the domain is split into pieces by direction, e.g. 2 pieces in the x direction, 1 in y direction.
- `hierarchical` Hierarchical geometric decomposition which is the same as simple except the user specifies the order in which the directional split is done, e.g. first in the y direction, then the x direction
- `scotch` Scotch decomposition which requires no geometric input from the user and attempts to minimise the number of processor boundaries. The user can specify a weighting for the decomposition between processors, through an optional processorWeights keyword.
- `manual` Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor.

It is possible to plug METIS that requires no geometric input from the user and attempts to minimize the number of processor boundaries. You will need to install METIS as it is not distributed with OpenFOAM. METIS and parMetis are not free for commercial use.
For more info, please have a look to the Decomposition of mesh User Guide.

Run the pitzDaily tut in parallel

- Check that you are in your case directory

  ```
  [a08tra89@node343 ~]$ pwd
  /plx/usertrain/a08tra89/OpenFOAM/a08tra89-2.3.0/run/pitzDaily
  ```

- Generate the mesh with the blockMesh utility

  ```
  [a08tra89@node343 ~]$ blockMesh
  ```

  The end of the output lookslike

```
Writing polyMesh
----------------
Mesh Information
----------------
  boundingBox: (-0.0206 -0.0254 -0.0005) (0.29 0.0254 0.0005)
  nPoints: 25012
  nCells: 12225
  nFaces: 49180
  nInternalFaces: 24170
----------------
Patches
----------------
  patch 0 (start: 24170 size: 30) name: inlet
  patch 1 (start: 24200 size: 57) name: outlet
  patch 2 (start: 24257 size: 223) name: upperWall
  patch 3 (start: 24480 size: 250) name: lowerWall
  patch 4 (start: 24730 size: 24450) name: frontAndBack
End
```

Run the pitzDaily tut in parallel

- Execute the decomposePar utility, to decompose the domain according to the selected strategy.

  [a08tra89@node343 ~]$ decomposePar

  The output lookslike

```
Create time
Decomposing mesh region0
Create mesh
Calculating distribution of cells
Selecting decompositionMethod hierarchical
Finished decomposition in 0 s
Calculating original mesh data
Distributing cells to processors
Distributing faces to processors
Distributing points to processors
Constructing processor meshes
Processor 0
    Number of cells = 3056
..
..
Processor 3
    Number of cells = 3057
...
Max number of faces between processors = 148 (4.59364% above average 141.5)
Time = 0
Processor 0: field transfer
Processor 1: field transfer
Processor 2: field transfer
Processor 3: field transfer
End.
```

- Remember: This is a **serial** process, which has decomposed the mesh for your next parallel run

Run the pitzDaily tut in parallel

- On completion, a set of subdirectories have been created, one for each processor, in the case directory

```
[a08tra89@node343 run]$ tree -L 1 pitzDaily/
pitzDaily/
|-- 0
|-- constant
|-- processor0
|-- processor1
|-- processor2
|-- processor3
`-- system
```

- The directories are named processorN where N=0,1.... represents a processor number and a time directory, containing the decomposed filed description, and a `constant/polyMesh` directory containing the decomposed mesh description.
- Copy the batch script to run in parallel in your case directory `$FOAM_RUN/pitzDaily/`

  `cp $FOAM_CINECA_SCRIPT/parallel.pbs $FOAM_RUN/pitzDaily`
- Now have a look to the script
- And modify it, according to your account and the selected queue

Parallel batch script 1/2


```
#!/bin/bash
#PBS -l walltime=05:00
#PBS -l select=1:ncpus=4:mpiprocs=4
## Job name
#PBS -N foam_par
#### Standard output and error
#PBS -o par4.out
#PBS -e par4.err
#### Submission queue
#PBS -q private -W group_list=train_copf2014
#PBS -A train_copf2014
# redirect stdout and stderr
# -o log
# -e log.err
#PBS -j oe
#PBS -m bea
##PBS -M i.spisso@cineca.it
#### end PBS directivies
```

Please note that `PBS -l select=1:ncpus=4:mpiprocs=4` is now set up for a pure MPI job with 4 cores.

Parallel batch script 2/2


```
module load autoload/0.1/verbose
module load openfoam/2.3.0-gnu-4.7.2

# move to directory where the case has been submitted
cd $PBS_O_WORKDIR

## set the solver
solver=simpleFoam
# set the number of procs
np=4
mpirun -np $np $solver -parallel > run_par.$PBS_JOBID
```

- The blue section loads the module and move to the working dir
- The red section selects the solver, the number of procs to run the parallel job, and execute the solver `simpleFoam` in parallel using the specified number of cpus.
- The suffix `$PBS_JOBID` will append the jobid to the name of the logfile. After the execution, you will see something like `run_par.1595057.node351.plx.cineca.it`

Submit the job in queue system

```
[a08tra89@node343 ~]$ qsub parallel.pbs
```

Reconstruct the fields: reconstructPar

- After the run, have a look the the processor* directories, ex. processor1

```
[a08tra89@node343 run]$ tree -L 1 processor1/
processor1/
|-- 0
|-- 100
|-- 150
|-- 200
|-- 250
|-- 300
|-- 350
|-- 400
|-- 450
|-- 50
|-- 500
|-- 550
|-- 600
|-- 650
|-- 700
|-- 750
|-- 800
|-- 825
`-- constant
```

- We can now reconstruct the entire field with the `reconstructPar` utility, by typing

```
reconstructPar
```

Reconstruct the fields: reconstructPar

- After a while, you will have the output

```
Create time
Reconstructing fields for mesh region0
Time = 50
Reconstructing FV fields
    Reconstructing volScalarFields
        p
        nut
        k
        epsilon
    Reconstructing volVectorFields
        U
    Reconstructing surfaceScalarFields
        phi
...
Reconstructing sets:
Time = 825
...
..
Reconstructing point fields
No point fields
No lagrangian fields
Reconstructing sets:
End.
```

- And the reconstructed fields in your case directory ready for post-processing and/or visualization.
  Try to load yourself the reconstructed case and visualize the time variation of the U field

```
0    150  250  350  450  500  600  700  800  constant  parallel.pbs    processor0  processor2  run_par.1596791.node351.plx.ci
100  200  300  400  50   550  650  750  825  par4.out  postProcessing  processor1  processor3  system
```