051176 - Computational Techniques for Thermochemical Propulsion
Master of Science in Aeronautical Engineering

Parallel running in OpenFOAM

Prof. **Federico Piscaglia**
Dept. of Aerospace Science and Technology (DAER)
Politecnico di Milano - Italy
federico.piscaglia@polimi.it

Running in parallel

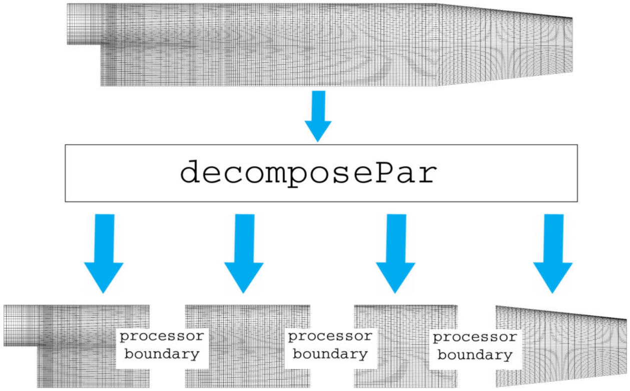
The method of parallel computing used by OpenFOAM® is known as domain decomposition, in which the geometry and associated fields are broken into pieces and allocated to separate processors for solution.

1) In a first step the domain is decomposed using the `decomposePar` utility (edit the dictionary named `decomposeParDict` which is located in the system directory of the case.)

2) The parallel running makes use of the public domain openMPI implementation of the standard message passing interface (MPI). Each processor runs a copy of the solver on a separate part of the domain mesh.

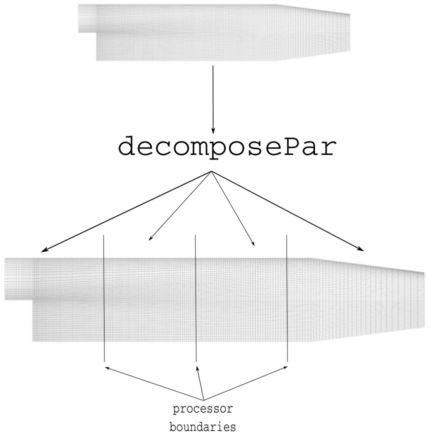
3) Finally the solution is reconstructed to obtain the final result.

1. Domain decomposition



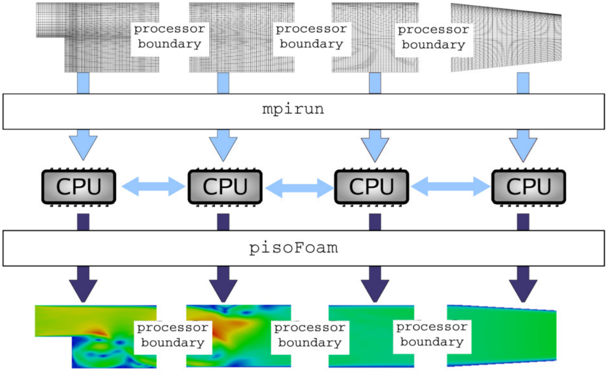
- the `decomposePar` splits the domain into different regions, connected with `processor` type interfaces to exchange information
- All sections can be located on the same machine or scattered around different nodes (distributed computing)

Domain decomposition: strategy



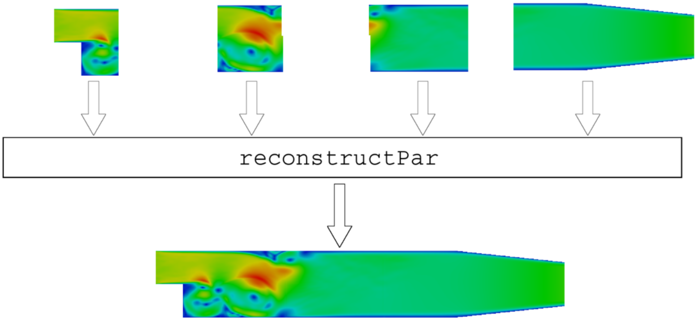
- the `decomposePar` splits the domain into different regions, connected with `processor` type interfaces to exchange information
- all sections can be located on the same machine or scattered around different nodes (distributed computing)

2. Parallel run



- Each processor run a separate copy of the solver on the selected portion of the domain.
- Information are exchanged between regions through the `processor` boundary which uses the MPI implementation

3. Solution reconstruction



- Once the simulation has completed, the whole domain can be reconstructed using the `reconstructPar` utility

Running in parallel

- We shall use the pitzDaily case for the `simpleFoam` application tutorial:

```
tut
cd incompressible/simpleFoam
```

- We can copy the pitzDaily case to the pitzDailyParallel:

```
cp -r pitzDaily $FOAM_RUN/pitzDailyParallel
```

- ...copy into the system directory the `decomposeParDict` file available in the pitzdailtExptInlet case and return to the `run` folder

```
cp pitzDailyExptInlet/system/decomposeParDict  \
$FOAM_RUN/pitzDailyParallel/system/
run
```

Domain decomposition

- The mesh and fields are decomposed using the `decomposePar` utility.
- The underlying aim is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution.
- The geometry and fields are broken up according to a set of parameters specified in a dictionary named `decomposeParDict` that must be located in the `system` directory of the case of interest.
- In the `decomposeParDict` file the user must set the number of domains which the case should be decomposed into: usually it corresponds to the number of cores available for the calculation:

```
numberOfSubdomains 2;
```

- The user has a choice of six methods of decomposition, specified by the method keyword.
- For each method there are a set of coefficients specified in a sub-dictionary of decompositionDict, named `<method>Coeffs`, used to instruct the decomposition process:

Domain decomposition

```
method simple/hierarchical/metis/parMetis/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 etc;
- **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 etc;
- **metis**: METIS decomposition which requires no geometric input from the user and attempts to minimize the number of processor boundaries. The user can specify a weighting for the decomposition between processors which can be useful on machines with differing performance between processors;
- **parMetis**: MPI-based version of METIS with extended functionality;
- **scotch**: similar to METIS but free for commercial use. It will completely replace METIS in future releases of OpenFOAM®;
- **manual**: Manual decomposition, where the user directly specifies the allocation of each cell to a particular processor.

NOTE: METIS and parMetis are not free for non-academic use.

Domain decomposition

- The coefficients sub-dictionary for the `simple` method are set as follows:

```
simpleCoeffs
{
     n                 (2 1 1);
     delta             0.001;
}
```

  Where **n** is the number of sub-domains in the x, y, z directions (nx ny nz) and **delta** is the cell skew factor, typically 10^{-3}.

- The coefficients sub-dictionary for the `hierarchical` method are set as follows:

```
hierarchicalCoeffs
{
    n                 (2 2 1);
    delta             0.001;
    order             xyz;
}
```

  Where **n** is the number of sub-domains in the x, y, z directions, **delta** is the cell skew factor and **order** is the order of decomposition xyz/xzy/yxz...

Domain decomposition

- The coefficients sub-dictionary for the `metis` method are set as follows:

```
metisCoeffs
{
    processorWeights
    (
        1
        ...
        1
    );
}
```

It is the list of weighting factors for allocation of cells to processors: `<wt1>` is the weighting factor for processor 1, etc.

- The coefficients sub-dictionary for the `manual` method contains the reference to a file:

```
manualCoeffs
{
    dataFile   "<fileName>";
}
```

It is the name of file containing data of allocation of cells to processors.

Domain decomposition

- Data files may need to be distributed if only local disks are used in order to improve performance.

- The root path to the case directory may differ between machines. The paths must then be specified in the decomposeParDict dictionary using `distributed` and `roots` keywords.

- The distributed entry should read:

```
distributed yes;
```

  and the roots entry is a list of root paths, `<root0>`, `<root1>`, . . . , for each node:

```
roots
<nRoots>
(
    "<root0>"
    "<root1>"
    ...
);
```

  where `<nRoots>` is the number of roots.

- Each of the processorN directories should be placed in the case directory at each of the root paths specified in the decomposeParDict dictionary.

Domain decomposition

- The decomposePar utility is executed in the normal manner by typing:

```
decomposePar -case [caseName]
```

- The output will look like:

```
Calculating distribution of cells
Selecting decompositionMethod metis
...
Processor 0
Number of cells = 6113
Number of faces shared with processor 1 = 58
Number of processor patches = 1
Number of processor faces = 58
Number of boundary faces = 12514

Processor 1
Number of cells = 6112
Number of faces shared with processor 0 = 58
Number of processor patches = 1
Number of processor faces = 58
Number of boundary faces = 12496
```

Domain decomposition

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

```
pitzDailyParallel
|------> 0
|------>  constant
|------>  processor0
|------>  processor1
|------>  system
```

- The directories are named `processor<N>`, where N = 0, 1, ... is the processor number and contains: a) constant/polyMesh directory with the decomposed mesh description; b) the time folders with the decomposed fields;

- `openMPI` can run either on a local multiprocessor machine or on multiple nodes over a network; in this latter case, a `hostile` containing the host names of the machines must be created. The file can have any name and it can be located in any path.

Parallel execution

- An application is run in parallel using the `mpirun` command:

```
    mpirun --hostfile <machines> -np <nProcs> <foamExec>  \
    <root> <case> <otherArgs> -parallel > log
```

  where: `<nProcs>` is the number of processors; `<foamExec>` is the executable, e.g. `simpleFoam`; and, the output is redirected to a file named log.

- The `<machines>` file contains the names of the machines listed, one machine per line.

- The names must correspond to a fully resolved hostname in the `/etc/hosts` file of the machine on which the openMPI is run. The `<machines>` file would contain:

```
                        hostname1
                        hostname2 cpu=2
                        hostname3
```

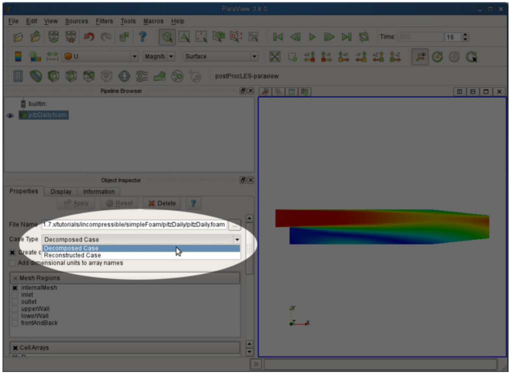
- In our case it becomes:

```
    mpirun -np 2 simpleFoam -case $FOAM_RUN/pitzDailyParallel
    -parallel > log
```

Post-processing in parallel (1/2)

- Starting from version 1.6, the OpenFOAM® reader for Paraview is capable to read decomposed cases without reconstructing them;

- In some cases, it can be preferable to post-process each subdomain separately: $\boxed{\texttt{paraFoam -case processor1}}$



File Name 1.7.x/tutorials/incompressible/simpleFoam/pitzDaily/pitzDaily.foam
Case Type Decomposed Case
[✓ Decomposed Case]
Reconstructed Case
✖ Create c
Add dimensional units to array names
✖ internalMesh
inlet
outlet
upperWall
lowerWall
frontAndBack

Post-processing in parallel (2/2)

- Almost all OpenFOAM® utilities for post-processing can be run in parallel:

```
mpirun -np 2 foamCalc mag U -parallel
```

- The decomposed case can be reconstructed using the `reconstructPar` utility:

```
Time = 0
Reconstructing FV fields
Reconstructing volScalarFields
p
.....
Reconstructing volVectorFields
U
Reconstructing volTensorFields
R
No point fields
No lagrangian fields
```

Summary

- OpenFOAM® uses **domain decomposition** to implement parallellism

  1. Mesh subdivision:
     ```
     decomposePar
     ```
  2. Parallel computation: one subdomain per processor
     ```
     mpirun -np 2 simpleFoam -parallel > log
     ```
  3. Reconstruction
     ```
     reconstructPar
     ```

- Communications are implemented using OpenMPI library
- Several possibilities of decomposition methods, including METIS and SCOTCH
- Postprocessing can be done either on the reconstructed or on the decomposed case.
- OpenFOAM® postprocessing utilities can be run in parallel.

Collated file format

Parallel I/O: new collated file format

When an OpenFOAM simulation runs in parallel, the data for decomposed fields and mesh(es) has historically been stored in multiple files within separate directories for each processor. Processor directories are named 'processorN', where N is the processor number.

- This commit introduces an alternative "collated" file format where the data for each decomposed field (and mesh) is collated into a single file, which is written and read on the master processor. The files are stored in a single directory named 'processors'.

- The new format produces significantly fewer files - one per field, instead of N per field. For large parallel cases, this avoids the restriction on the number of open files imposed by the operating system limits.

- The file writing can be threaded allowing the simulation to continue running while the data is being written to file. NFS (Network File System) is not needed when using the the collated format and additionally, there is an option to run without NFS with the original uncollated approach, known as "masterUncollated".

Parallel I/O: new collated file format

The controls for the file handling are in the subdictionary `Optimisation-Switches` of `etc/controlDict`:

```
OptimisationSwitches
{
    ...

    //- Parallel IO file handler
    //  uncollated (default), collated or masterUncollated
    fileHandler uncollated;

    //- collated: thread buffer size for queued file writes.
    //  If set to 0 or not sufficient for the file size threadin
    //  Default: 2e9
    maxThreadFileBufferSize 2e9;

    //- masterUncollated: non-blocking buffer size.
    //  If the file exceeds this buffer size scheduled transfer
    //  Default: 2e9
    maxMasterFileBufferSize 2e9;
}
```

Parallel I/O: new collated file format

When using the collated file handling, memory is allocated for the data in the thread. `maxThreadFileBufferSize` sets the maximum size of memory in bytes that is allocated. If the data exceeds this size, the write does not use threading.

When using the masterUncollated file handling, non-blocking MPI communication requires a sufficiently large memory buffer on the master node. `maxMasterFile` sets the maximum size in bytes of the buffer. If the data exceeds this size, the system uses scheduled communication.

The installation defaults for the fileHandler choice, `maxThreadFileBufferSize` and `maxMasterFileBufferSize` (set in etc/controlDict) can be over-ridden within the case controlDict file, like other parameters. Additionally the fileHandler can be set by:

- the "-fileHandler" command line argument;
- a `FOAM_FILEHANDLER` environment variable.

Parallel I/O: new collated file format

A `foamFormatConvert` utility allows users to convert files between the collated and uncollated formats, e.g.

```
mpirun -np 2 foamFormatConvert -parallel -fileHandler uncollated
```

An example case demonstrating the file handling methods is provided in:

`$FOAM_TUTORIALS/IO/fileHandling`

The work was undertaken by Mattijs Janssens, in collaboration with Henry Weller.

Thank you for your attention!

contact: federico.piscaglia@polimi.it