Hands-on with OpenFOAM part II

G. Amati, CINECA

R. Ponzini, CINECA

A. Memmolo, CINECA

09/22

Univerza v Ljubljani

TU WIEN TECHNISCHE UNIVERSITÄT WIEN

CINECA

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Outline

Ponzini , CINECA

Running OpenFOAM on an HPC cluster

- Running a CFD case to production
- Running OpenFOAM in parallel
- Evaluating job scaling performances for better resources usage
- Accessing and managing of a remote simulation
- Data monitoring and saving

Implementation in OpenFOAM

- In your working space you will find the directory named:

Case_Re200_IcoFoam_par/

the main dictionary are modified and adapted from Wolf Dynamics set of 2D cylinder cases (see references at the end of this document)

For visualization using paraview move vtk file locally via scp: e.g.

- Scp xxxxxxxxxxxx xxxxxxxxxxxx

Running a CFD case to production

- Once a case setup is ok for our needs, like the cylinder laminar shedding we made in the hands-on session part I, it is quite common to perform a *mesh sensitivity analysis*
- To do so at least 3 meshes with different cell size spacing are needed
- The way the mesh changes can be designed is various and out of the scope of this hands-on session
- Usually, the mesh size is doubled or halved moving from one mesh to the other
- We will play with mesh we already have in different ways to get an understanding of how mesh size can impact with time to result, the numerical time stepping (CFL limits) and how parallel computing can be used to support our needs

Running a CFD case to production

- We have a 9200 2D mesh cell, and we have arrived to obtain a stable period shedding phenomenon in our CFD model as desired.
- We can first test the given model at different number of processor
- It is always important to test the given hardware to understand if the performances are aligned with our expectations and needs
- In our hardware, in each node we can run on up 48 cores
- To perform our test we can set a given value of the physical time we want to achieve in our case and see for different processors values (1,2,4,8,…,48) what is the Execution time we obtain. Then we can rank the results according to a selected metrics. The basic concept is lower Execution time is better but also other concepts alike efficiency and speed-up are relevant.

Running OpenFOAM in parallel

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- OpenFOAM is natively suited to run in parallel on a variety of hardware using message passing (MPI)
- To do so we only need to run the *decomposePar* functionality before execute the solver section of our standard workflow.
- Several options are presents but in general we can just run it without any parameter
- The *decomposePar* functionality is instructed using a dictionary (*system/decomposeParDict*) that indeed specify how the domain decomposition should take place
- To our need we can start with a very common and robust *scotch* method that requires only the number of subdomain to be executed

```
Usage: decomposePar [OPTIONS]
options:
  -allRegions       operate on all regions in regionProperties
  -case <dir>       specify alternate case directory, default is the cwd
  -cellDist         write cell distribution as a labelList - for use with
                    'manual' decomposition method or as a volScalarField for
                    post-processing.
  -constant         include the 'constant/' dir in the times list
  -copyUniform      copy any uniform/ directories too
  -copyZero         Copy  0 directory to processor* rather than decompose the
                    fields
  -dict <file>      read control dictionary from specified location
  -fields           use existing geometry decomposition and convert fields only
  -fileHandler <handler>
                    override the fileHandler
  -force            remove existing processor*/ subdirs before decomposing the
                    geometry
  -latestTime       select the latest time
  -libs '("lib1.so" ... "libN.so")'
                    pre-load libraries
  -noFields         opposite of -fields; only decompose geometry
  -noFunctionObjects
                    do not execute functionObjects
  -noSets           skip decomposing cellSets, faceSets, pointSets
  -noZero           exclude the '0/' dir from the times list
  -region <name>    specify alternative mesh region
  -time <ranges>    comma-separated time ranges - eg, ':10,20,40:70,1000:'
  -srcDoc           display source code in browser
  -doc              display application documentation in browser
  -help             print the usage

decompose a mesh and fields of a case for parallel execution

Using: OpenFOAM-10 (see https://openfoam.org)
Build: 10-8213cb4a3f81
```

decomposeParDict

With this file you can control

- number of task (decomposition)
- Method used for decomposition
  - Scotch
  - Xxxxxx
  - Xxxxxx

- Decomposition is crucial for complex geometries and high number of tasks
- It is serial: can be very slow for big grid and/or many task

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  6                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

numberOfSubdomains 4;

method          scotch;

// ************************************************************************* //
```

Distributed Memory

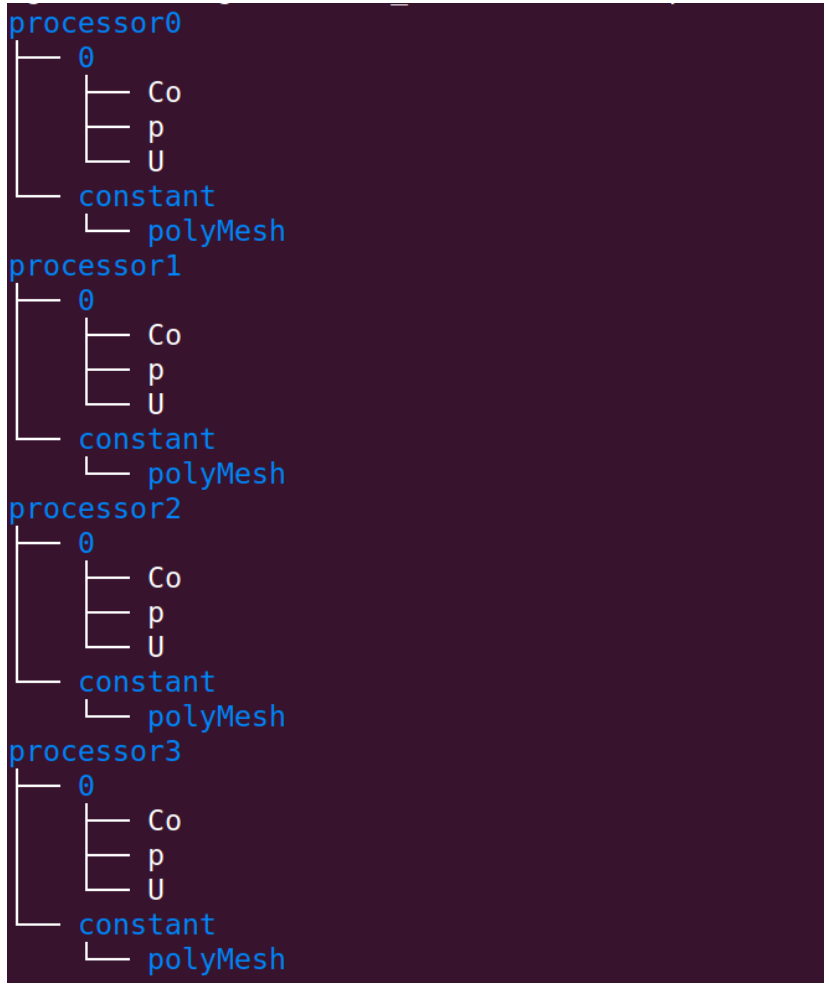
SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

Distributed Memory Parallelism:

- Each task has its own memory.
- The OF structure is replicated for each processor (I.e. task)
- In this directory will be written the output

- Two possibility to analize results
  1. First reconstruct global field, then using foamToVTK
  2. Using foamToVTK on different processor: I.e. you need #n different vtk files do visualize the whole domain

     `foamToVTK -case processor0`

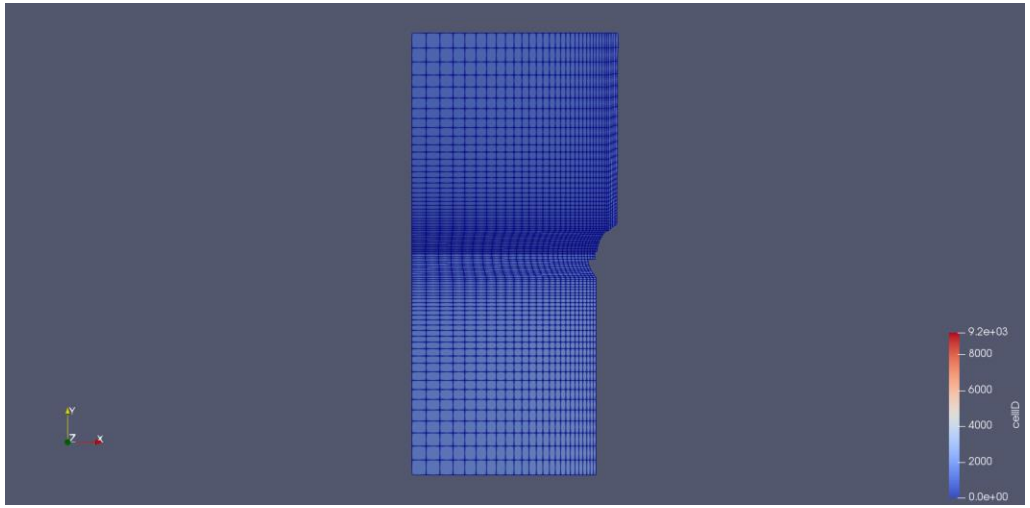


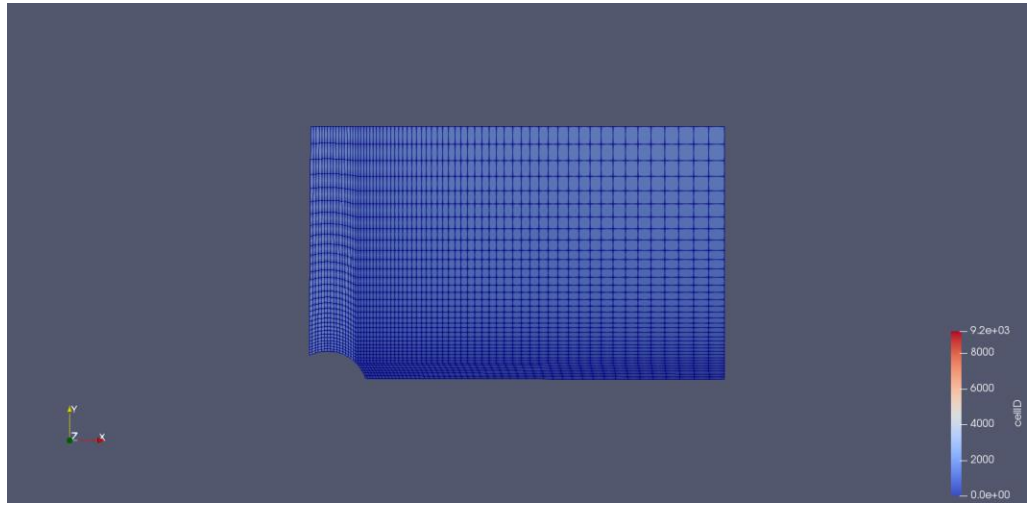
log.decomposeParDict

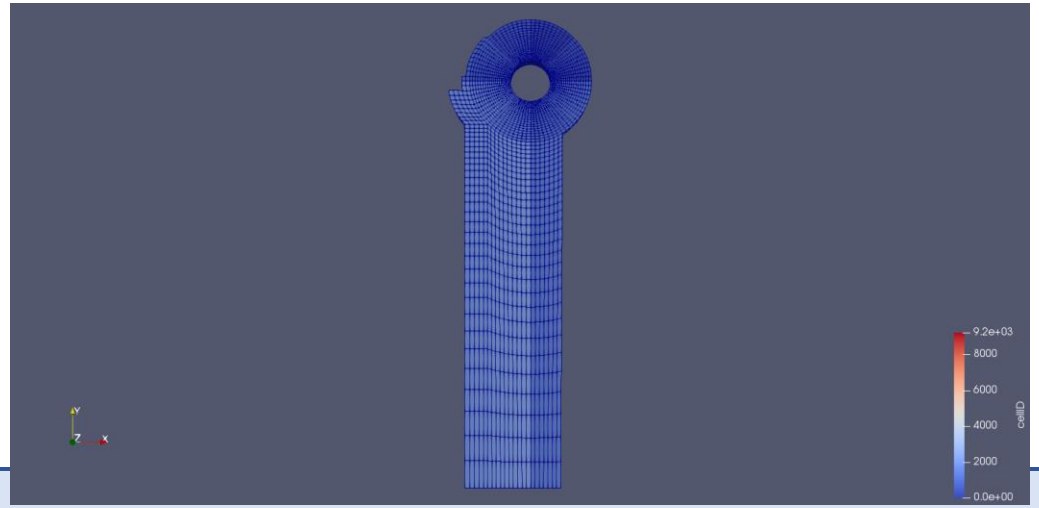

```
Create time

Decomposing mesh region0

Create mesh

Calculating distribution of cells
Selecting decomposer scotch
Finished decomposition in 0.011613 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 = 2315
    Number of faces shared with processor 1 = 59
    Number of faces shared with processor 2 = 30
    Number of processor patches = 2
    Number of processor faces = 89
    Number of boundary faces = 4769

Processor 1
    Number of cells = 2315
    Number of faces shared with processor 0 = 59
    Number of faces shared with processor 2 = 22
    Number of faces shared with processor 3 = 47
    Number of processor patches = 3
    Number of processor faces = 128
    Number of boundary faces = 4732

Processor 2
    Number of cells = 2280
    Number of faces shared with processor 0 = 30
    Number of faces shared with processor 1 = 22
    Number of faces shared with processor 3 = 50
    Number of processor patches = 3
    Number of processor faces = 102
    Number of boundary faces = 4662

Processor 3
    Number of cells = 2290
    Number of faces shared with processor 1 = 47
    Number of faces shared with processor 2 = 50
    Number of processor patches = 2
    Number of processor faces = 97
    Number of boundary faces = 4677

Number of processor faces = 208
Max number of cells = 2315 (0.65217391% above average 2300)
Max number of processor patches = 3 (20% above average 2.5)
Max number of faces between processors = 128 (23.076923% above average 104)

Time = 0s

Processor 0: field transfer
Processor 1: field transfer
Processor 2: field transfer
Processor 3: field transfer

End
```

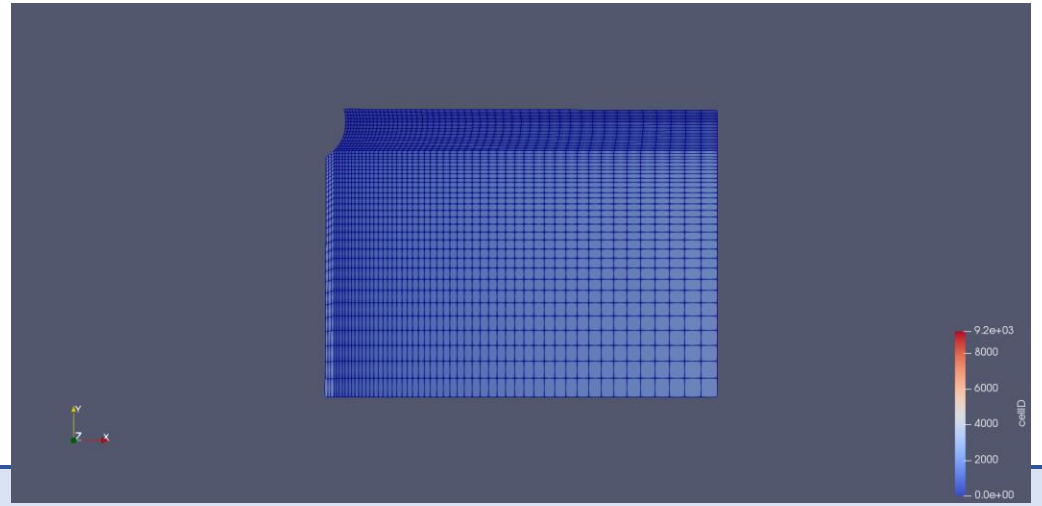
Generated grids

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP









reconstructPar

ReconstructPar allow to build a single output collecting data from different processors

- It builds the "serial" structure of output
- It is serial: can be very slow for big grid and/or many task

```
Create time



Reconstructing fields for mesh region0

Time = 100s
Reconstructing FV fields
    Reconstructing volScalarFields

        p

    Reconstructing volVectorFields

        U_0
        U

    Reconstructing surfaceScalarFields

        phi
        phi_0

Reconstructing point fields
No point fields
No lagrangian fields

Time = 200s
Reconstructing FV fields
```

Running OpenFOAM in parallel

- Since we are starting form a given mesh and solution our parallel testing workflow will look like this:

  module purge
  module load profile/eng autoload openfoam/10
  decomposePar >& log. decomposePar
  mpirun icoFoam –parallel >& log.icoFoam

- We will also:
  - Shut off the data saving at each time
  - Shut off the function object data saving
  - set the end time at 600 (starting form time 500): 100 seconds with a dt of 0.05 seconds is equivalent to instruct for a 2000 iterations of the solver. This value should be sufficient to get meaningful time results at different parallelism.

Running OpenFOAM in batch mode

- More important in all this hands-on session we will run our CFD jobs in batch submitting the job to queuing system

>>> sbatch *-A tra22_Sctrain -p g100_usr_prod --reservation=s_tra_sc1* submit_job.sh

- With this command we are requesting to the shcheduler to execute where possible, according to Accounting and partition, our *submit_job.sh* script

Running OpenFOAM in batch mode

::
submit_job.sh file content
::

```
#!/bin/bash
#SBATCH --time=2:0:00
#SBATCH --ntasks-per-node 48
#SBATCH --ntasks-per-socket=24
#SBATCH --sockets-per-node=2
#SBATCH -N 1

echo $SLURM_JOBID > jobid.$SLURM_JOBID
cd $SLURM_SUBMIT_DIR
runjobid=$SLURM_JOB_ID
#run the workflow by means of the run_par.sh file
sh ./run_par.sh
```

Running OpenFOAM in batch mode

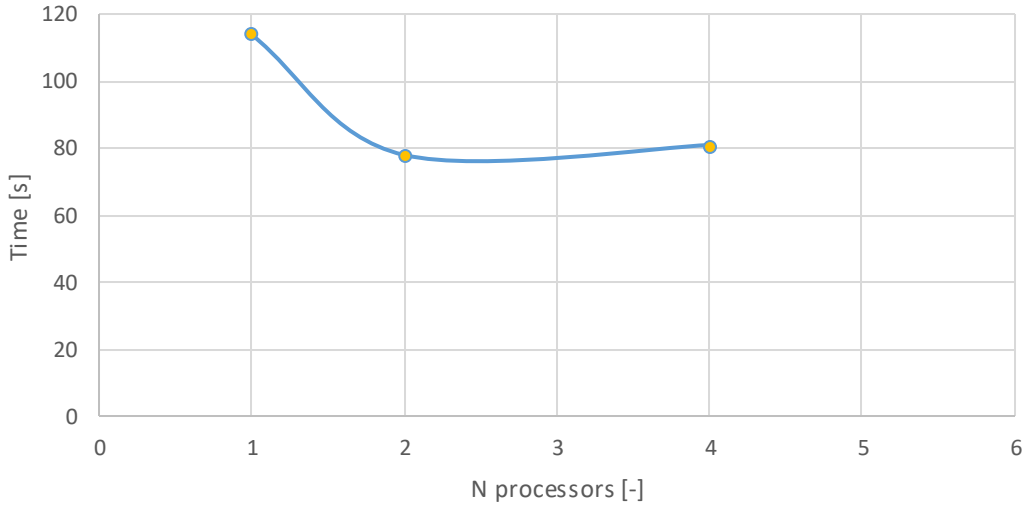
::
run_par.sh file content
::

```
#!/bin/bash


module purge
module load profile/eng autoload openfoam/10
decomposePar
mpirun icoFoam -parallel >& log.icoFoam
```

Evaluating job scaling

- The first test we made on 1,2,4 processors show that the time to solution is not getting any better after 2 processors and that even in that case the efficiency is not ideal
- This can be expected since we are using a spreading a number of cells/computational core that is too small and we are wasting the benefit of the faster number crunching by requesting too much communication

N processors [-]	Execution Time [s]	Efficiency[%]	Mesh cells /core
1	114.38	-	9200
2	78.50	73%	4600
4	81.80	35%	2300



Evaluating job scaling

- In order to try to use in a better way our cluster and see the benefit of parallelism in action we can artificially increase our mesh size
- We can do this in many ways
- For instance, we can add cells at the *blockMeshDict* level or we can use the *mirrorMesh* utility
- We will first increase by a factor of 100 our mesh requesting 10x points in each direction in XY plane (in *blockMeshDict*): we will get a mesh that is going to be: 920.000 cells
  - Copy blockMeshDict.big in blockMeshDict
- We will repeat our test and see if we can get any improvement

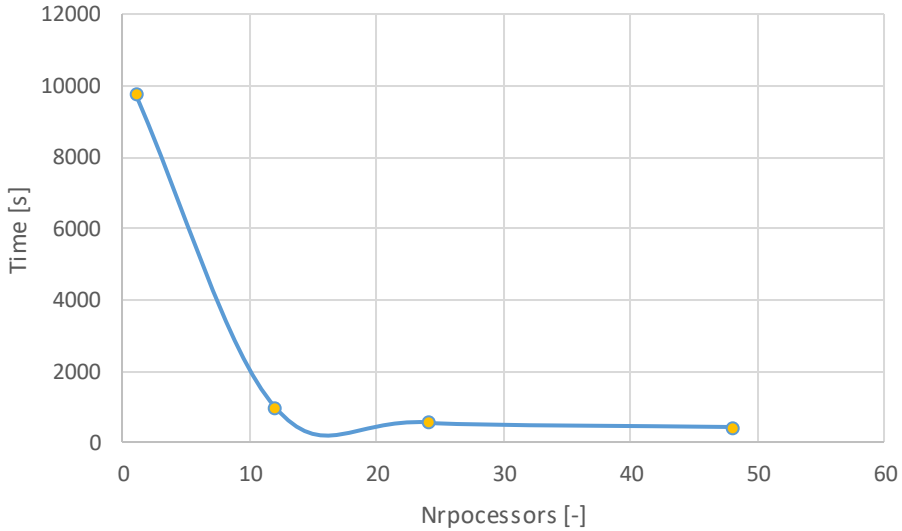
Evaluating job scaling

- In order to try to use in a better way our cluster and see the benefit of parallelism in action we can artificially increase our mesh size
- We can do this in many ways
- For instance, we can add cells at the *blockMeshDict* level or we can use the *mirrorMesh* utility
- We will see both in action
- We will first increase by a factor of 100 our mesh requesting 10x points in each direction in XY plane (in *blockMeshDict*): we will get a mesh that is going to be: 920.000 cells
- We will repeat our test and see if we can get any improvement

Evaluating job scaling

- Mesh size 920.000 cells

N processors [-]	Execution Time [s]	Efficiency [%]	Mesh cells /core
1	9795	-	920000
12	979	83%	76000
24	560	73%	38000
48	439	46%	19000



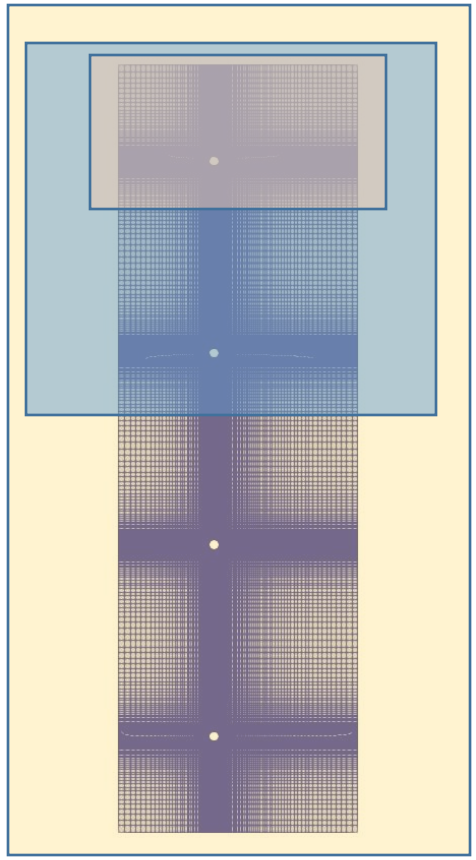
Benefit with parallel is significant: factor of about 22x in Exec Time using 48 cores.
Efficiency is 46%, ideal speedup should be 48x.

Evaluating job scaling

- Considering that with 0,92 mln cells we do not see any benefit to move out of a single node, i.e. increasing the number of processor over 48, we will first generate a larger mesh using the *mirrorMesh* utility and then use this mesh to make scalability test over different nodes.
- We will mirror by the Y-axis a couple of time to get the final mesh
- The benefit of using this utility is related to the effectiveness of scalability testing using mesh with the same mesh cell size but of increased number of cells. We therefore do not expect any changes to be applied to the delta-t or to other numerical parameters since the physics we are solving is exactly the same

Evaluating job scaling

- Mirrored mesh two time starting form the 920.000 mesh
- Total cells count of 3.680.000 cells
- Topology of the mesh is the same
- Min/max cell size is the same
- deltaT used to respect CFL <1 is the same



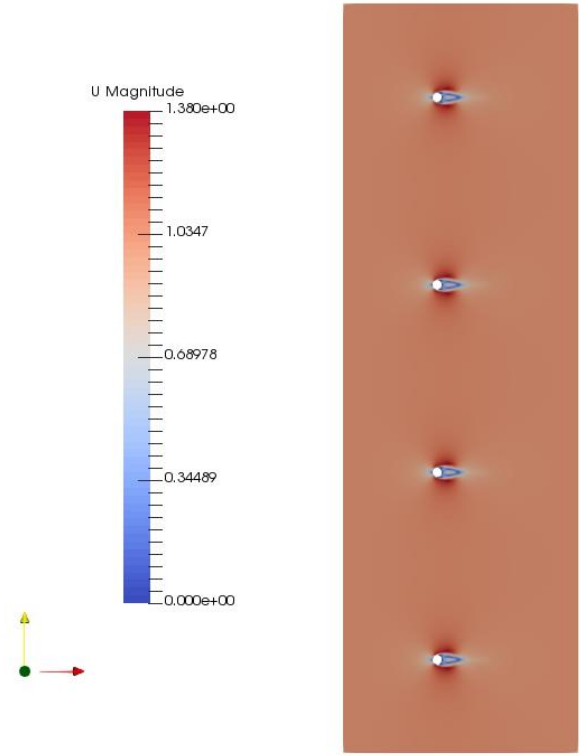
Original mesh domain: 920.000 cells

After First Mirror execution:920.000 x2 cells

After Second Mirror execution: 920.000 x2 x2 cells

Evaluating job scaling

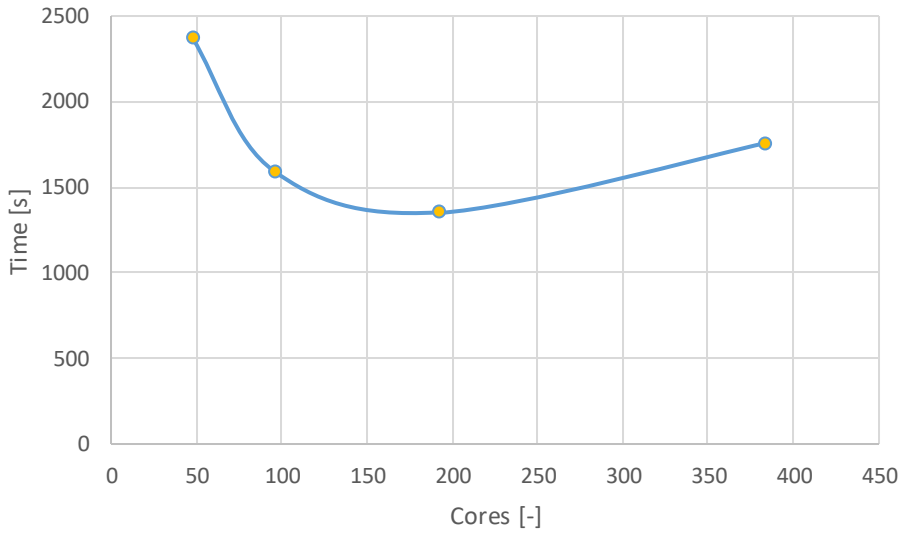
Mesh cells: 3.8 mln
Time = 10 s



Evaluating job scaling

- We will repeat our scalability test using now an incremental step based on the number of nodes, i.e we will increase each time the number of cores of 48 units

N processors [-]	Execution Time [s]	Efficiency [%]	Mesh cells /core
48	2374	-	76000
96	1591	75%	38000
192	1351	45%	19000
384	1753	18%	9500



Evaluating job scaling

- From our scalability test we can say that our case is scaling is strongly related to the mesh size
- The mesh size of a problem should be assessed using a mesh sentivity approach to understand the minimal mesh size that solve a reference problem within a certain uncertainty bound
- There is an optimal *mesh cell density / computational core* that we can define for our problem once the mesh size is given and that we can use for similar problem/solver setup/mesh topology
- In our problem we found that starting from a reasonable mesh size a value of about *20.000 mesh cells/computational core* should be fine to avoid computational resources wasting and severe sub-optimal hardware usage

Accessing a remote simulation

- When dealing with a remote simulation, i.e. runned on a hardware that is not our desktop pc, we have to understand that there are looking at a process that is hosted/runned somewhere else
- There are several relevant positive side effect, like the possibility to shutdown and reconnect to the process without loosing our job
- To monitor our run, beside the mentioned function objects, we can connect to the standard output by means of Linux command like: *tee, tail, head, more*,… and so on
- We should also be able to ‘drive’ our simulation and stop it for our needs
- Stopping the remote simulation can be done by changing the *endTime* to our need or using dedicated function objects like *stopAtFile*

Accessing a remote simulation

- In order to stop and save data of our simulation in any moment we can do different things:
  - Modifying *the endTime* parameter with the *writeNow* value or with a give time we like
  - Or using the *stopAtFile functionObject*

Data monitoring and saving

- When running in parallel we are usually dealing with a mesh size that is not suitable for a desktop pc hardware (million cells)
- Data saving and monitoring of the simulation should be ideally planned in advance in order to save only necessary data
- This consideration is especially true when we are dealing with a time variant physical problem like in the case of the vortex shedding
- Preferably data should also be lighter than the full set of data output contained in a single time instant
- Lightweight data still meaningful for visualization and monitoring of the simulation are:
  - Residuals (necessary for numerical assessment of the simulation)
  - Forces and forces coefficients (necessary for physical assessment of the simulation)
  - Point/line probes
  - 2d surfaces (cut-planes,…) very useful for visualization and animation
  - 3d surfaces (iso-surfaces) very useful for visualization and animation

Data monitoring and saving

- For instance, the disk space usage for our larger mesh will look like this:
  - Single time instant data (full): 500 MB
  - Single time instant data(2D slice): 290 MB
- Please note that this is a 2d case, in 3d the difference is much higher

Hands-on what to do

1. Try to submit and monitor your simulation in parallel
2. Play with the number of cores to be used and monitor the timing

Thank you for your attention!

http://sctrain.eu/

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.