Module 5

Lecture 1

Roadmap

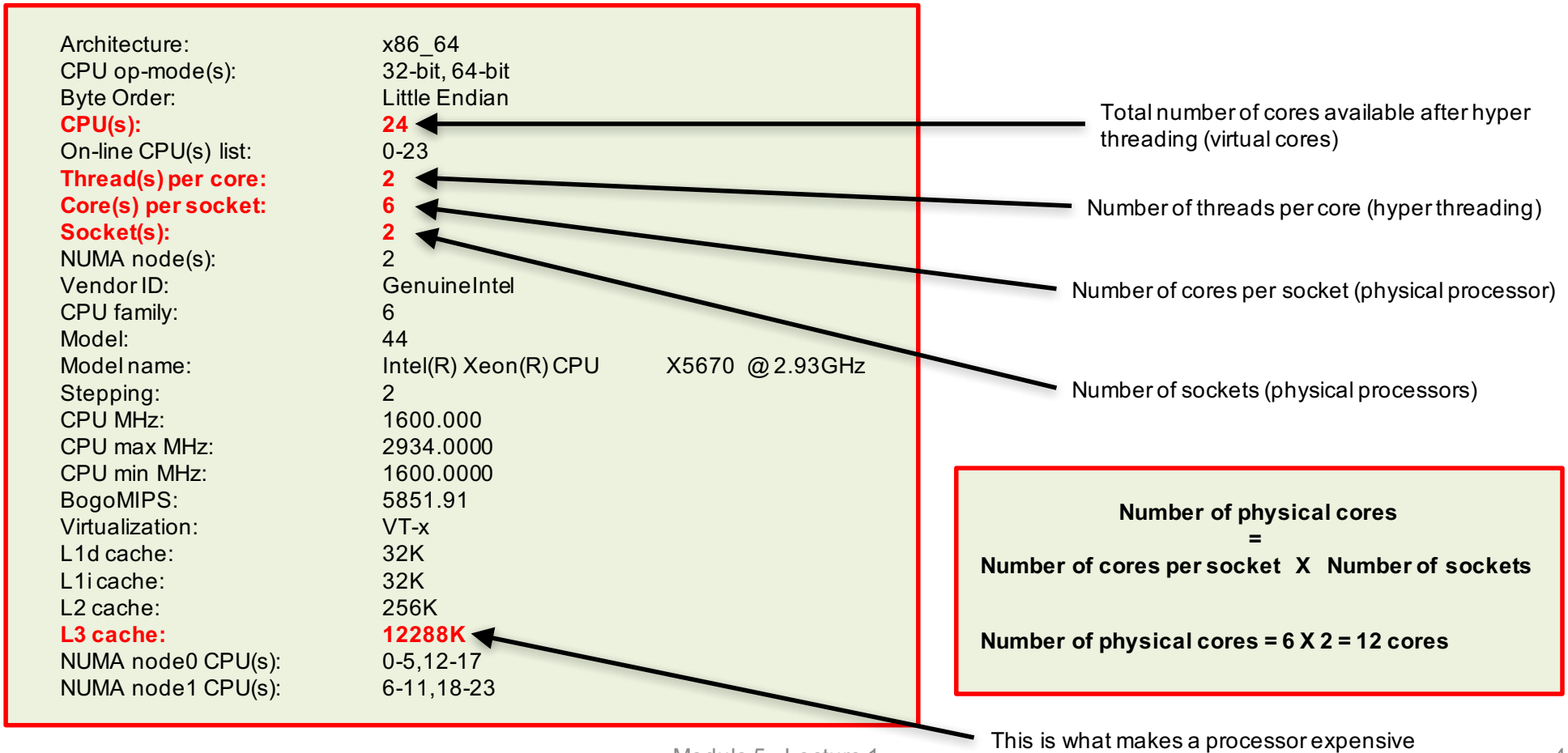
1. Running in parallel
2. Running in a cluster using a job scheduler
3. Running on the cloud

Roadmap

1. **Running in parallel**
2. **Running in a cluster using a job scheduler**
3. **Running on the cloud**

Running in parallel

- First at all, to know how many processors/cores you have available in your computer, type in the terminal:
  - `$> lscpu`
- The output for this particular workstation is the following:



Running in parallel

- OpenFOAM® does not take advantage of hyper threading technology (HT).
- Hyper threading technology (HT) is basically used by the OS to improve multitasking performance.
- This is what we have in the workstation:
    - 24 virtual cores (hyper threaded)
    - 12 physical cores
- To take full advantage of the hardware, we use the maximum number of physical cores (12 physical cores in this case) when running in parallel.
- If you use the maximum number of virtual cores, OpenFOAM® will run but it will be slower in comparison to running with the maximum number of physical cores (or even less cores).
- Same rule applies when running in super computers, so always read the hardware specifications to know the limitations.

Running in parallel

Why use parallel computing?

- **Solve larger and more complex problems (scale-up):**

  Thanks to parallel computing we can solve bigger problems (scalability). A single computer has limited physical memory, many computer interconnected have access to more memory.

- **Provide concurrency (scale-out):**

  A single computer or processor can only do one thing at a time. Multiple processors or computing resources can do many things simultaneously.

- **Save time:**

  Run faster (speed-up) and increase your productivity, with the potential of saving money in the design process.

- **Save money:**

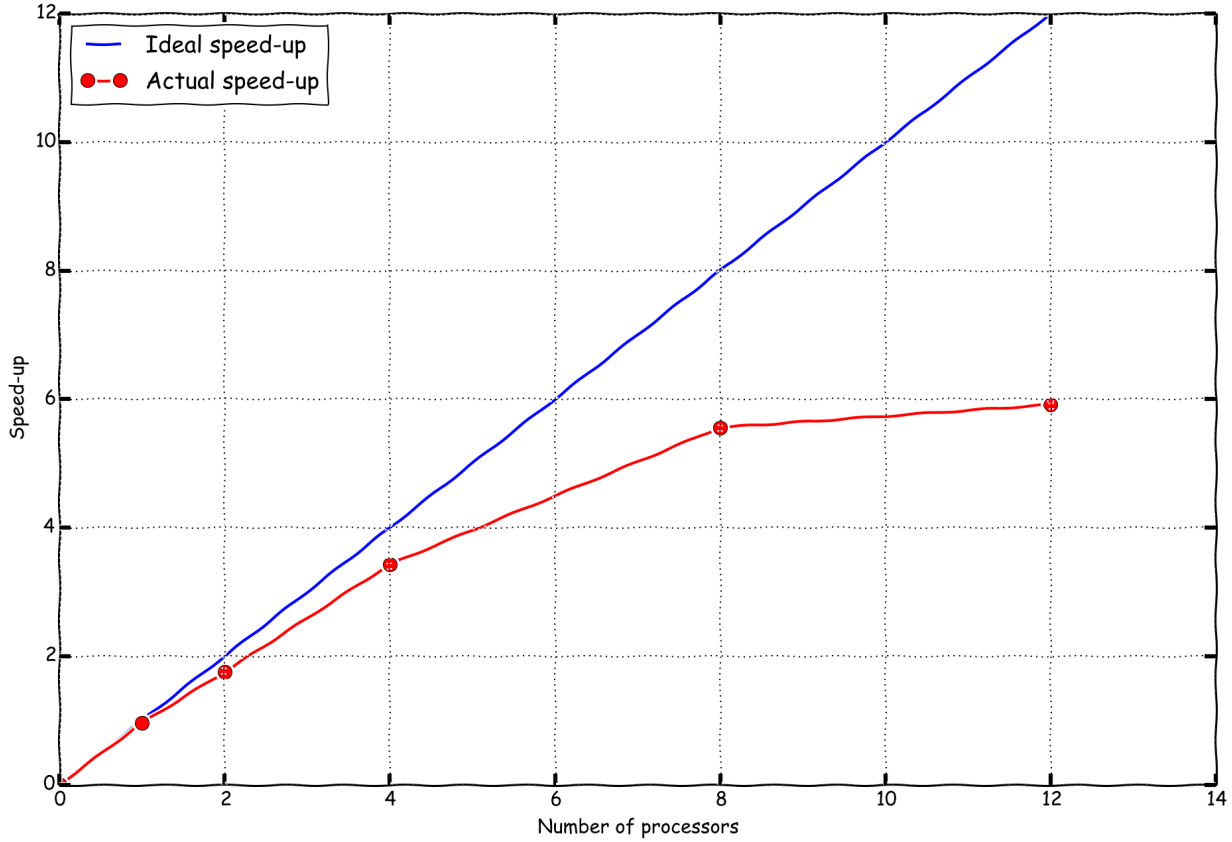
  In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel computers can be built from cheap, commodity components.

- **Limits to serial computing:**

  Both physical and practical reasons pose significant constraints to simply building ever faster serial computers (*e.g*, transmission speeds, CPU clock rate, limits to miniaturization, hardware cooling).

Running in parallel

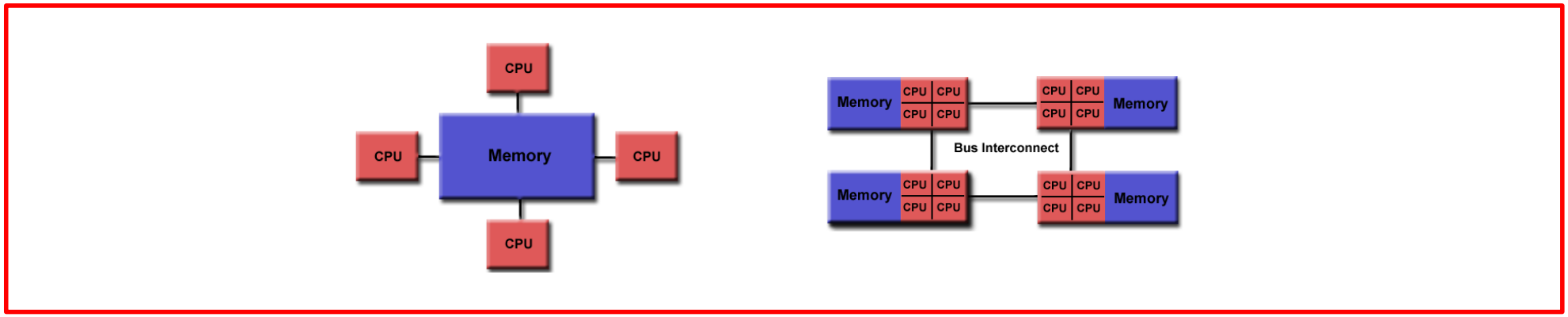
Speed-up and scalability example



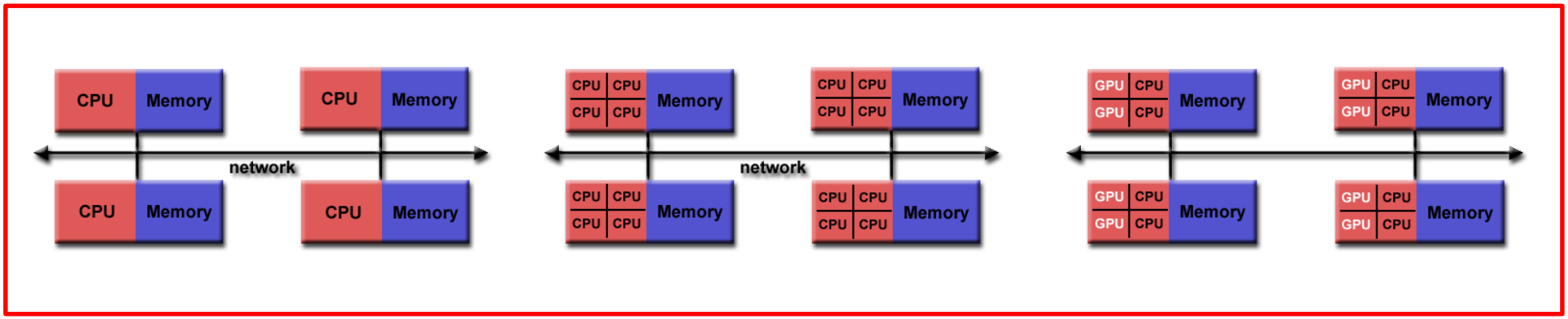
- You can find this case and the parallel and serial timing in the directory **`$PTOFC/additional_tutorials/tuts1/elbow3d/sHM_mesh`**.
- The parallel case with 1 processor runs slower than the serial case due to the extra overhead when calling the MPI library.

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 distributed among different processors.



Shared memory architectures – Workstations and portable computers



Distributed memory architectures – Cluster and super computers

Running in parallel

Some facts about running OpenFOAM® in parallel:

- Applications generally do not require parallel-specific coding. The parallel programming implementation is hidden from the user.
- Also, in order to run in parallel you will need an MPI library installation in your system.
- Most of the applications and utilities run in parallel.
- If you write a new solver, it will be in parallel (most of the times).
- We have been able to run in parallel up to 14000 processors.
- We have been able to run OpenFOAM® using single GPU and multiple GPUs.
- Do not ask me about scalability, that is problem/hardware specific.
- If you want to learn more about MPI and GPU programming, do not look in my direction.
- And of course, to run in parallel you need the computing power.

Running in parallel

To run OpenFOAM® in parallel you will need to:

- **Decompose the domain.**

  To do so we use the `decomposePar` utility. You also will need a dictionary named *`decomposeParDict`* which is located in the **`system`** directory.

- **Distribute the jobs among the processors or computing nodes.**

  To do so, OpenFOAM® uses the public domain OpenMPI implementation of the standard message passing interface (MPI). By using MPI, each processor runs a copy of the solver on a separate part of the decomposed domain.

- **Finally, the solution is reconstructed to obtain the final result.**

  This is done by using the `reconstrucPar` utility.

Running in parallel

Domain Decomposition

- The mesh and fields are decomposed using the `decomposePar` utility.
- The main goal is to break up the domain with minimal effort but in such a way to guarantee a fairly economic solution.
- In other words, we want to minimize the inter-processors communication and the processor workload.
- The mesh 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.

Running in parallel

Domain Decomposition

- In the `decomposeParDict` dictionary the user must set the number of domains in which the case should be decomposed. Usually it corresponds to the number of physical cores available.
    - **numberOfSubdomains NP;**

  where **NP** is the number of cores.

- The user has a choice of eight methods of decomposition, specified by the **method** keyword in the `decomposeParDict` dictionary. And depending on the method you are using you will need to give different options.

- On completion, a set of subdirectories will be created, one for each processor. The directories are named **`processorN`** where **`N`** = 0, 1, 2, 3, and so on. Each directory contains the decomposed fields.

Running in parallel

Domain Decomposition Methods

- These are the decomposition methods available in OpenFOAM® 3.0.x
  - hierarchical
  - manual
  - metis
  - multiLevel
  - none
  - scotch
  - simple
  - structured

We highly recommend you to use this method. The only input that requires from the user is the number of subdomains/cores. This method attempts to minimize the number of processor boundaries.

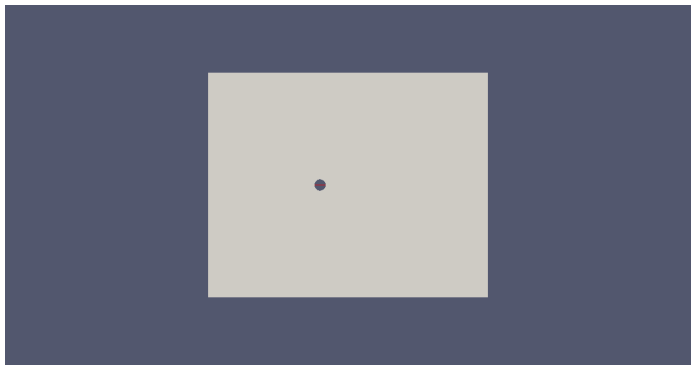
- If you want more information about each decomposition method, just read the source code:
  - **`$WM_PROJECT_DIR/src/parallel/decompose/`**

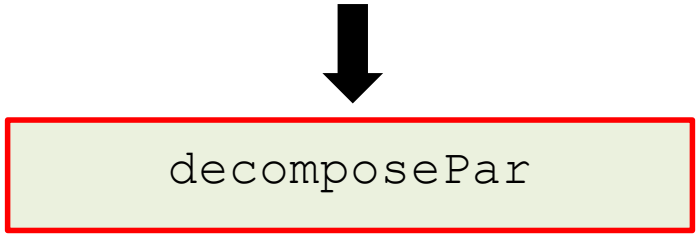
Running in parallel

Running in parallel – Gathering all together

Running in parallel

`polyMesh/` and **`0/`**

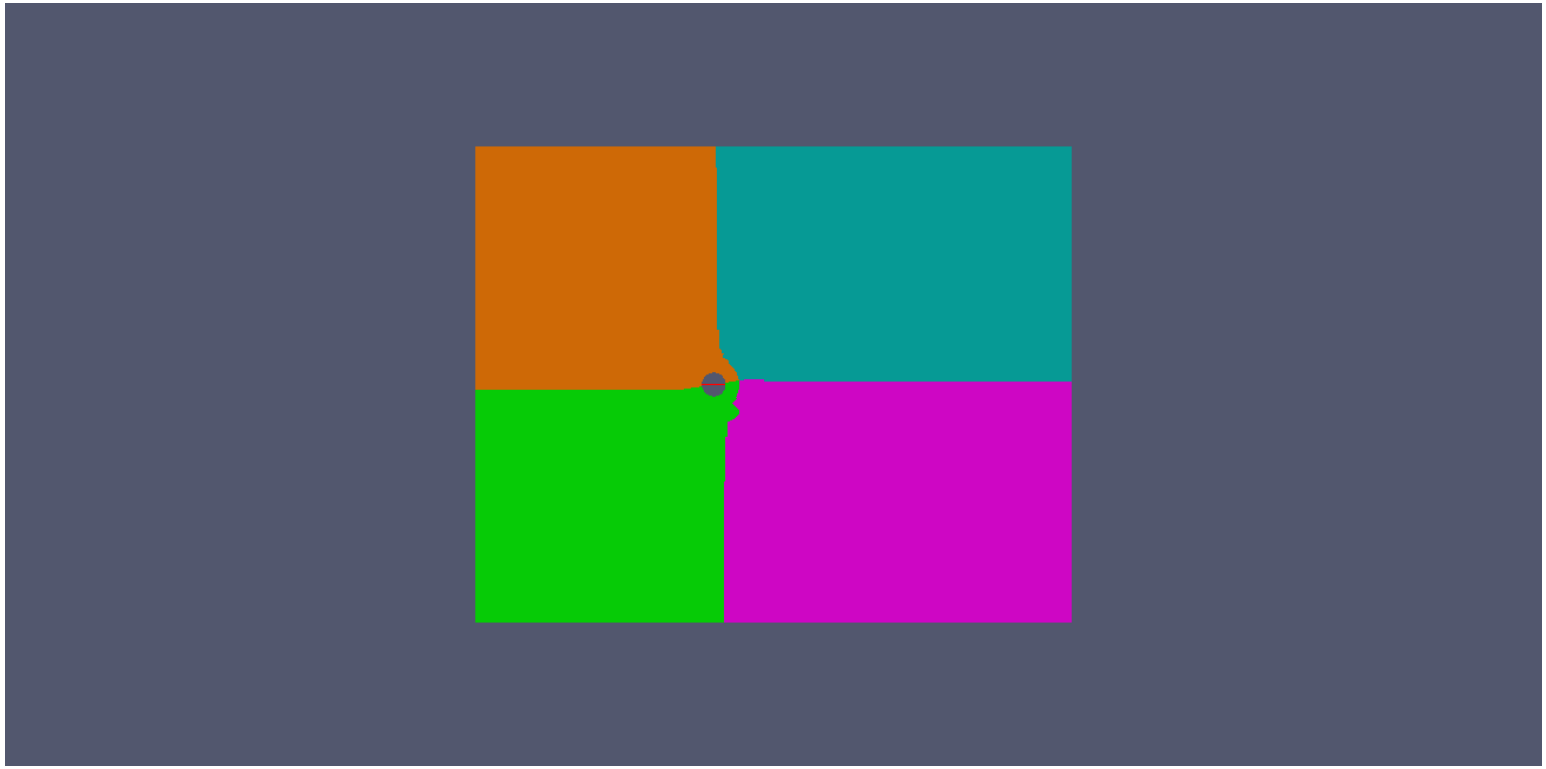




`Processor0` **`processor1`** **`processor2`** **`processor3`**

- Inside each **`processorN`** folder you will have the mesh information, boundary and initial conditions and the solution for that processor.

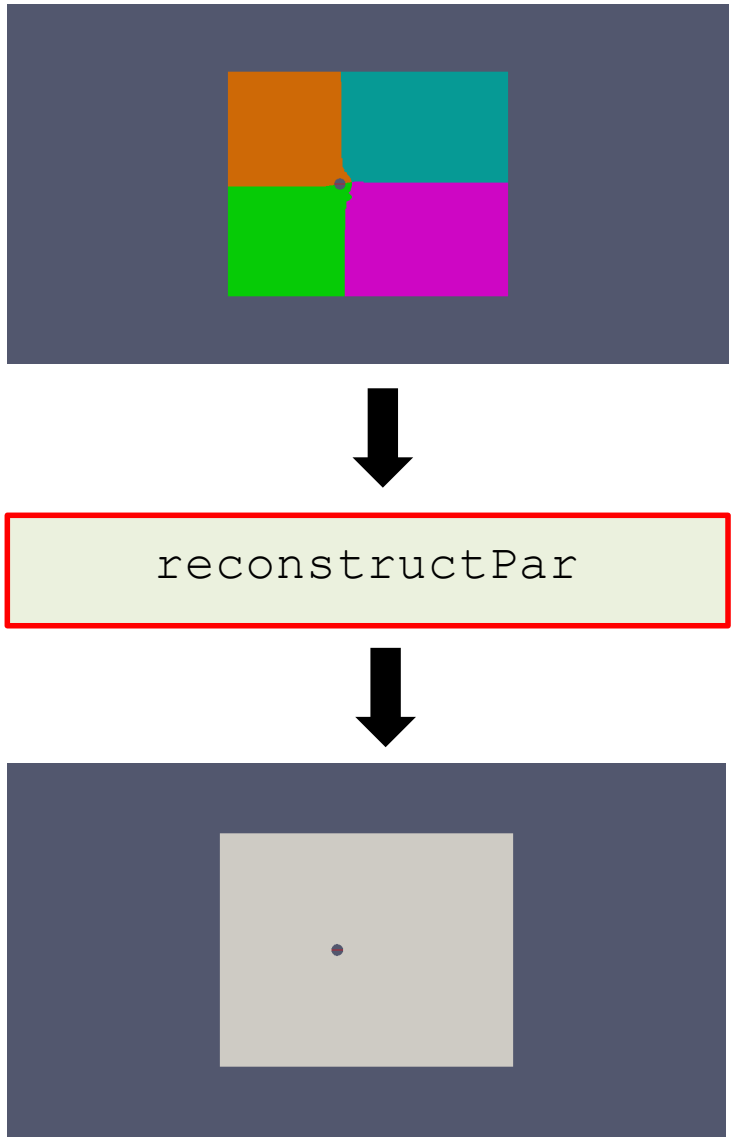
Running in parallel



```
$> mpirun -np <NPROCS> <application/utility> -parallel
```

- To run in parallel we use the MPI library.
- The number of processors to use or **<NPROCS>**, needs to be the same as the number of partitions (**numberOfSubdomains**).
- Do not forget to use the flag **`-parallel`**.

Running in parallel



- In the decomposed case, you will find the mesh information, boundary and initial conditions and the solution for every processor.
- The information is inside the directory **`processorN.`**

- When you reconstruct the case, you glue together all the information contained in the decomposed case.
- All the information (mesh, boundary and initial conditions and the solution), is inside the original case folder.

Running in parallel

- Summarizing, to run in parallel we proceed in the following way:

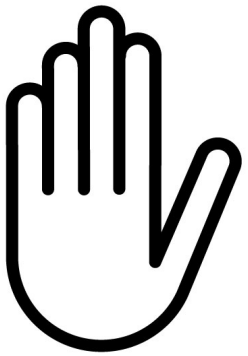
1. `$> decomposePar`
2. `$> mpirun -np <NPROCS> <application/utility> -parallel`
3. `$> reconstructPar`

- You can do the post-processing and visualization on the decomposed case or reconstructed case. We are going to address this later on.

Running in parallel

- Let's run the Rayleigh-Taylor instability case in parallel.
- Go to the directory:

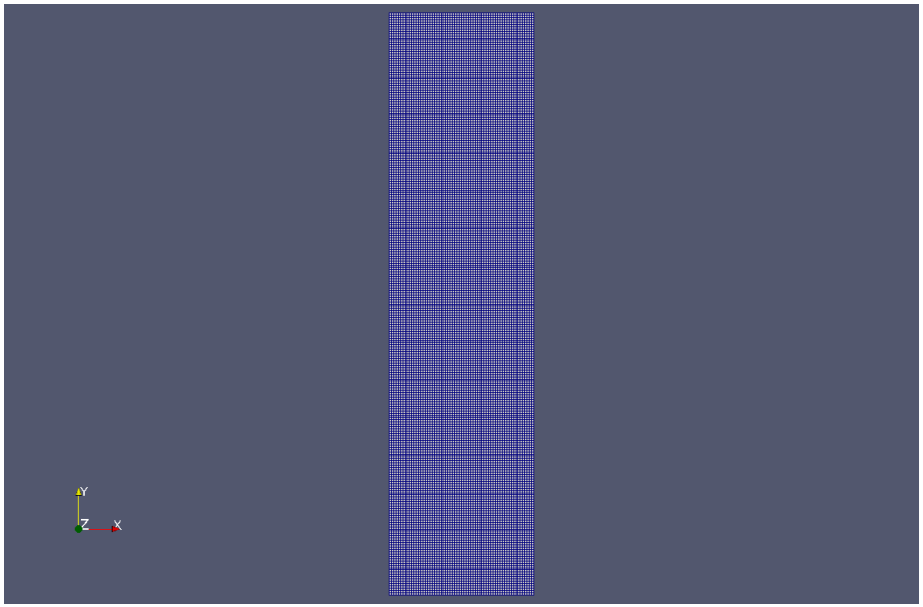
```
$PTOFC/parallel/rayleigh_taylor/c0
```



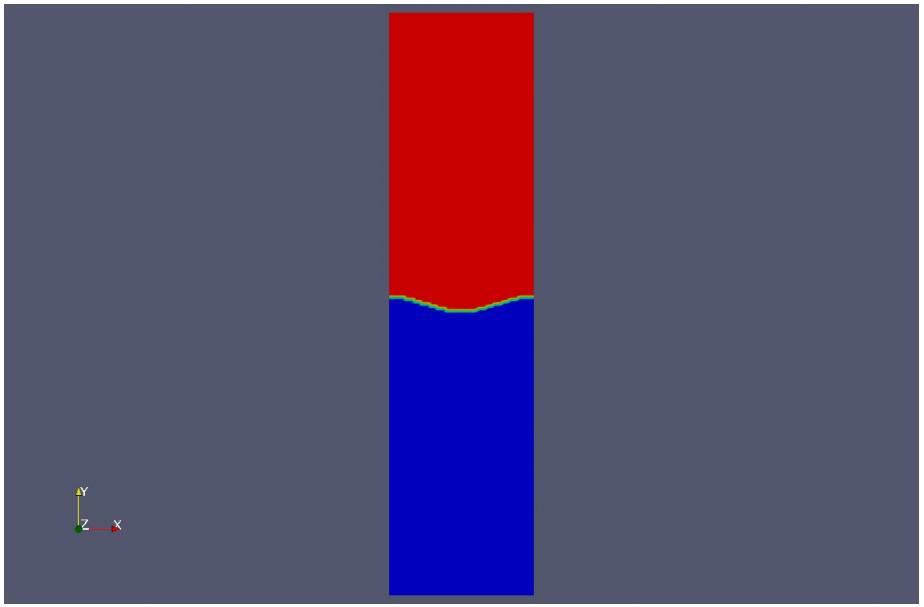
- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, `$PTOFC` is pointing to the path where you unpack the tutorials.

Running in parallel

By following the instructions, you should get something like this (remember we are running in parallel)



Mesh



VOF Fraction

Running in parallel

- In this case we are using 8 processors and the scotch method.
- Remember, if you want to change the number of processors and the decomposition method you will need to modify the *decomposeParDict* dictionary located in the **system** folder.

```
1   /*--------------------------------*- C++ -*----------------------------------*\
2   | =========                 |                                                 |
3   | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4   |  \\    /   O peration     | Version:  3.0.x                                 |
5   |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
6   |    \\/     M anipulation  |                                                 |
7   \*---------------------------------------------------------------------------*/
8   FoamFile
9   {
10      version     2.0;
11      format      ascii;
12      class       dictionary;
13      object      decomposeParDict;
14  }
15  // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17  numberOfSubdomains 8;
18
19  method          scotch;
20  //method          simple;
21
22  simpleCoeffs
23  {
24      n           (2 4 1);
25      delta       0.001;
26  }
27
28  // ************************************************************************* //
```

Running in parallel

- Go to the **`$PTOFC/parallel/rayleigh_taylor/c0`** directory. In the terminal type:

```
1.  $> cd $PTOFC/parallel/rayleigh_taylor/c0
2.  $> blockMesh
3.  $> checkMesh
4.  $> cp 0/alpha.phase1.org 0/alpha.phase1
5.  $> funkySetFields -time 0
6.  $> decomposePar
7.  $> ls -al
8.  $> mpirun -np 8 renumberMesh -overwrite -parallel
9.  $> mpirun -np 8 interFoam -parallel > log | tail -f log
10. $> reconstructPar
11. $> paraFoam
```

In this case we are using 8 processors. In your case, use the maximum number of cores available in your computer. Remember, you will need to modify the *decomposeParDict* dictionary located in the **`system`** folder. Specifically, you will need to modify the entry **numberOfSubdomains**, and set its value to the number of processors you want to use.

Running in parallel

- In step 5, we initialize the *alpha.phase1* field using the utility `funkySetFields`. If you do not have this tool, copy the file *0_init/alpha.phase1* to *0/alpha.phase1*.
- The `funkySetFields` utility is part of the external library swak4foam, we are going to talk about this library later on.
- In step 6, we decompose the case. The utility `decomposePar` reads the *decomposeParDict* dictionary that is located in the directory **`system`**.
- In step 7 we simply list all the directories and files. Do you see something different?
- In step 8, we run the utility `renumberMesh` in parallel. This utility reduce the bandwidth of the linear system to solve. In other words, the solver will run faster (at least at the beginning of the simulation). Remember, you use this utility after you decompose the domain.
- In step 9, we run the solver `interFoam` in parallel. Remember to always use the option `-parallel`. If you forget to use it, the solver will run but it will run eight copies of the same job.
- In step 10 we reconstruct the domain and then we visualize the reconstructed solution in step 11.

Running in parallel

Visualization of a parallel case

Running in parallel

- The traditional way is to first reconstruct the case and then do the post-process and visualization on the reconstructed case.

- To do so, we type in the terminal:

1. `$> reconstructPar`
2. `$> paraFoam`

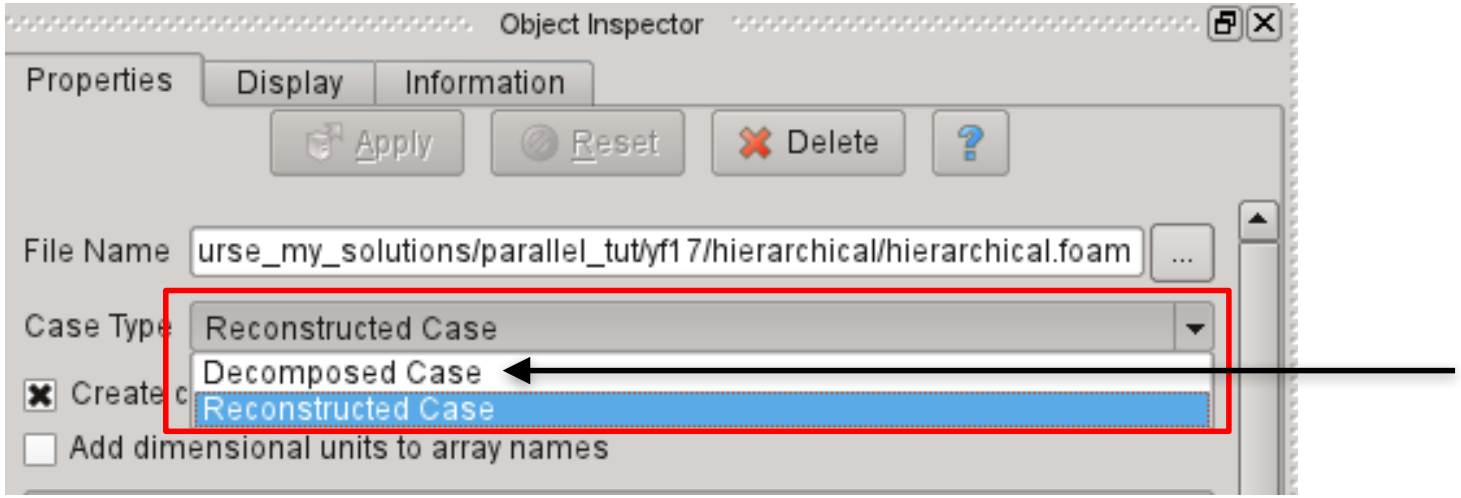
- Step 1 reconstruct the case. Remember, you can choose to reconstruct all the time steps, the last time step or a range of time steps.

- In step 2, we use `paraFoam` to visualize the reconstructed case.

Running in parallel

- An alternative way to visualize the solution, is by proceeding in the following way
    - `$> paraFoam -builtin`

- The option `-builtin` let's post-process the decomposed case directly.
- Remember, you will need to select on the object inspector the `Decomposed Case` option.



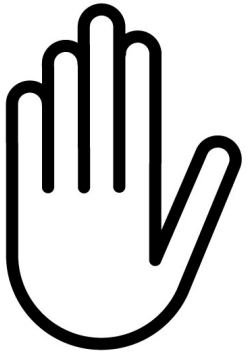
Running in parallel

- Both of the previous methods are valid.
- When we use the option `-builtin` with `paraFoam`, we work on the decomposed case directly.
- That is to say, we do not need to reconstruct the case.
- But wait, there is a third option.
- The third option consist in post-processing each decomposed domain individually.
- To load all processor directories, you will need to manually create the file *`processorN.OpenFOAM`* (where **N** is the processor number) in each processor folder.
- After creating all *`processorN.OpenFOAM`* files, you can launch `paraFoam` and load each file (the *`processorN.OpenFOAM`* files).
- As you can see, this option requires more effort from the user.

Running in parallel

- Remember the cylinder case? We told you not to forget about it. Let's run this case in parallel.
- Let's go to the directory:

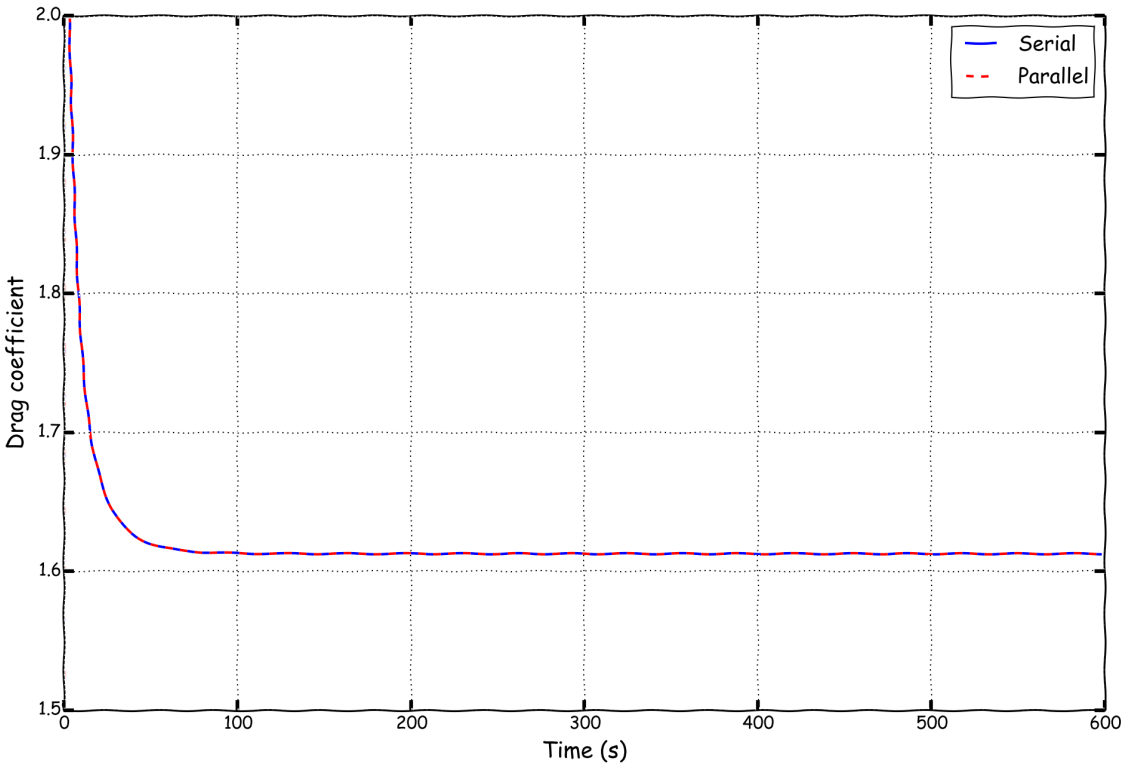
```
$PTOFC/parallel/cylinder_par/c0
```



- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, `$PTOFC` is pointing to the path where you unpack the tutorials.

Running in parallel

- As a part of the never ending task of verification and validation, it is a good idea to check from time to time that the serial solution and parallel solution are equal.
- It might happen that you get different results. If this happens, there is a problem in the parallel implementation or processors inter-communication.
- In this case, the serial and parallel solutions are equal.



Running in parallel

- In the directory **`$PTOFC/parallel/cylinder_par/c0`** you will find this tutorial. In the terminal window type:

```
1. $> cd $PTOFC/parallel/cylinder_par/c0
2. $> blockMesh
3. $> decomposePar
4. $> mpirun -np 4 checkMesh -parallel
5. $> mpirun -np 4 renumberMesh -overwrite -parallel
6. $> mpirun -np 4 icoFoam -parallel
7. $> paraFoam -builtin
```

In this case we are using 4 processors. In your case, use the maximum number of cores available in your computer. Remember, you will need to modify the `decomposeParDict` dictionary located in the **`system`** folder. Specifically, you will need to modify the entry **numberOfSubdomains**, and set its value to the number of processors you want to use.

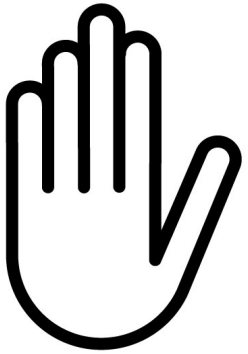
Running in parallel

- In step 4, we run the utility `checkMesh` in parallel.
- In step 5, we run the utility `renumberMesh` in parallel. This utility reduce the bandwidth of the linear system to solve. In other words, the solver will run faster (at least at the beginning of the simulation). Remember to use this utility after you decompose the domain.
- In step 6, we run the solver `interFoam` in parallel. Remember to always use the option `-parallel`. If you forget to use it, the solver will run but it will run eight copies of the same job.
- In step 7, we use `paraFoam` with the option `-builtin`. This option will let's visualize the solution without the need of reconstructing the domain.

Running in parallel

- Let's post-process each decomposed domain individually.
- For this we are going to use the yf17 tutorial, that you will find in the directory:

```
$PTOFC/parallel/yf17
```



- From this point on, please follow me.
- We are all going to work at the same pace.
- Remember, `$PTOFC` is pointing to the path where you unpack the tutorials.

Running in parallel

- In this directory you will find three sub-directories, namely **`hierarchical`**, **`scotch`** and **`scotch_celldist.`** Let's decompose these cases and visualize the partitioning.

- First we will start with the scotch partitioning method. In the terminal type:

1. `$> cd $PTOFC/parallel/yf17/scotch`
2. `$> fluent3DMeshToFoam ../../../meshes_and_geometries/fluent_yf17_yf17.cas`
3. `$> decomposePar`

- By the way, you do not need to run this case. We just need to decompose it to visualize the partitions.

Running in parallel

- Let's post-process each decomposed domain individually. Type in the terminal:

  - `$> touch processor0.OpenFOAM`
  - `$> touch processor1.OpenFOAM`
  - `$> touch processor2.OpenFOAM`
  - `$> touch processor3.OpenFOAM`

- Notice that we are using 4 processors/partitions.

- If you want to change the number of partitions, feel free to modify the *`decomposeParDict`* dictionary.

Running in parallel

- Let's post-process each decomposed domain individually. From now on follow me.

  - `$> cp processor0.OpenFOAM processor0`
  - `$> cp processor1.OpenFOAM processor1`
  - `$> cp processor2.OpenFOAM processor2`
  - `$> cp processor3.OpenFOAM processor3`
  - `$> paraFoam`

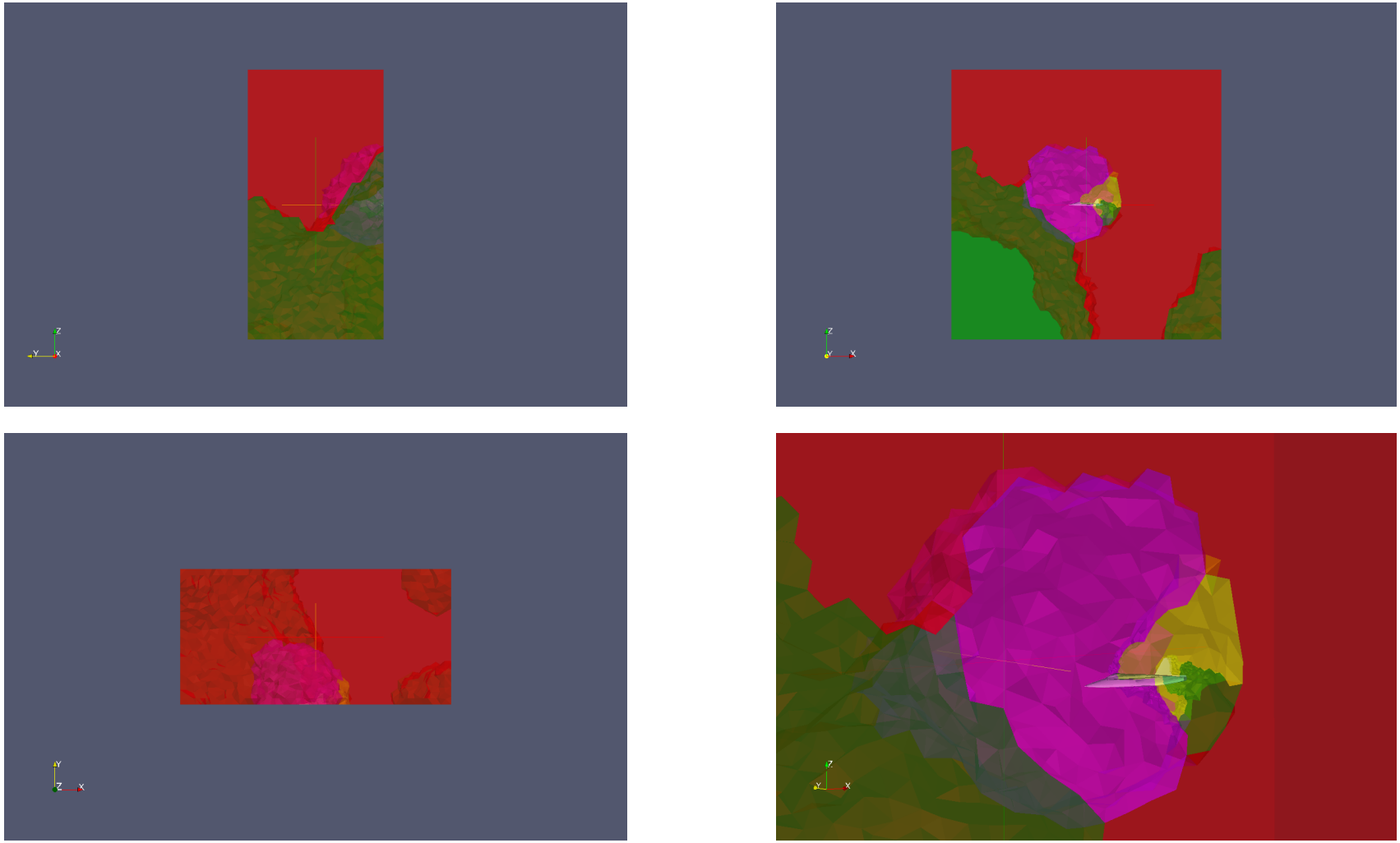
- To visualize the partitions, you will need to open in `paraFoam` each *`processorN.OpenFOAM`* file.

Running in parallel

- In `paraFoam` open each `*.OpenFOAM` file within the `processorN` directory.
- By doing this, we are opening the mesh partition for each processor.
- Now, choose a different color for each set you just opened, and visualize the partition for each processor. You will need to play with transparency and volume rendering to see the cell distribution among the processors.
- If you partitioned the mesh with many processors, creating the files `*.OpenFOAM` manually can be extremely time consuming, for doing this in an automatic way you can create a small shell or Python script.
- Also, changing the color for each set in `paraFoam` can be extremely time consuming, for automate this you can write a Python script.
- After you finish with the case directory **`scotch`**, do the same with the case directory **`hierarchical`**, which contains the hierarchical partitioning and compare both partitioning methods.

Running in parallel

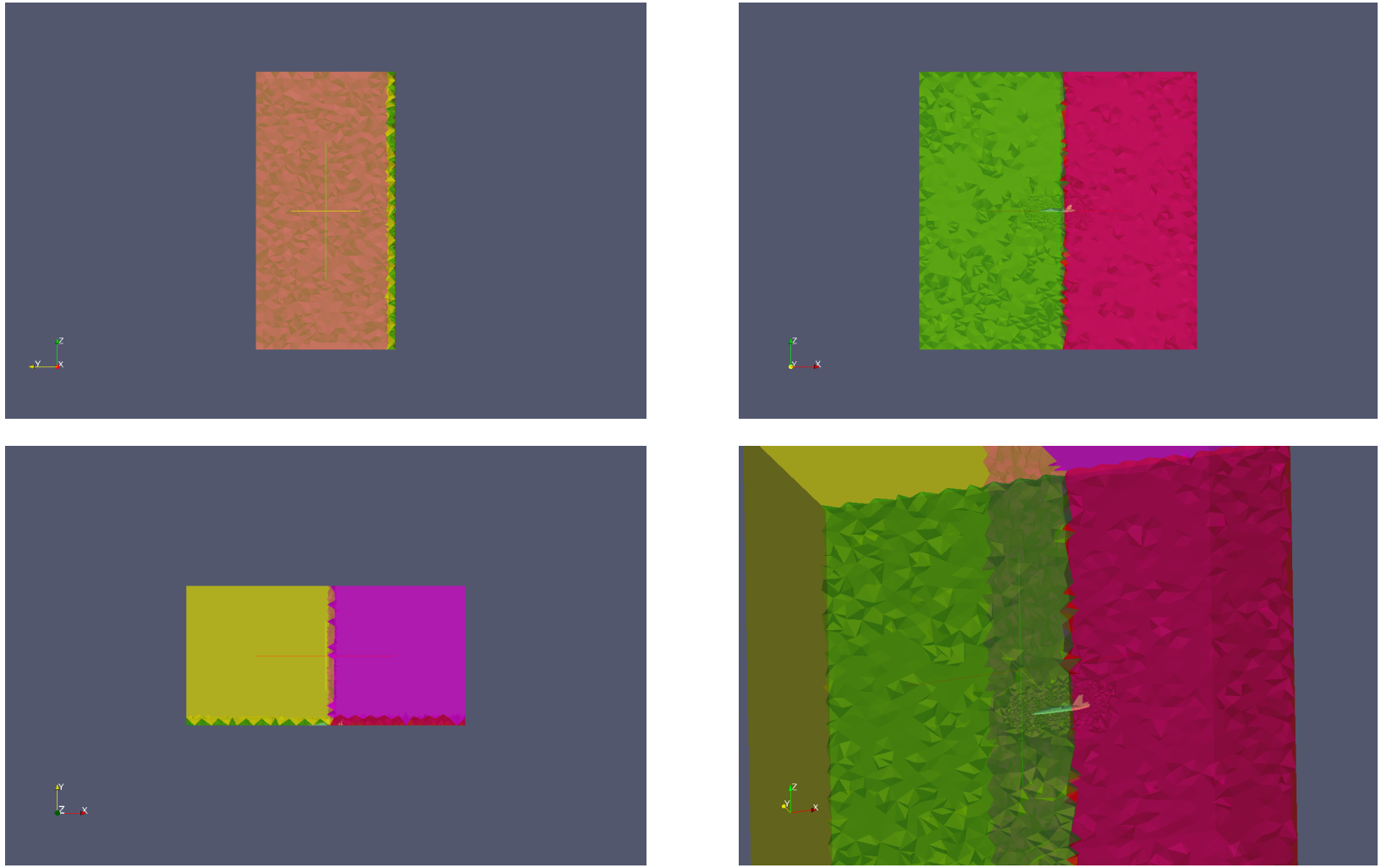
By following the instructions, you should get something like this



Mesh partitioning – Scotch Method

Running in parallel

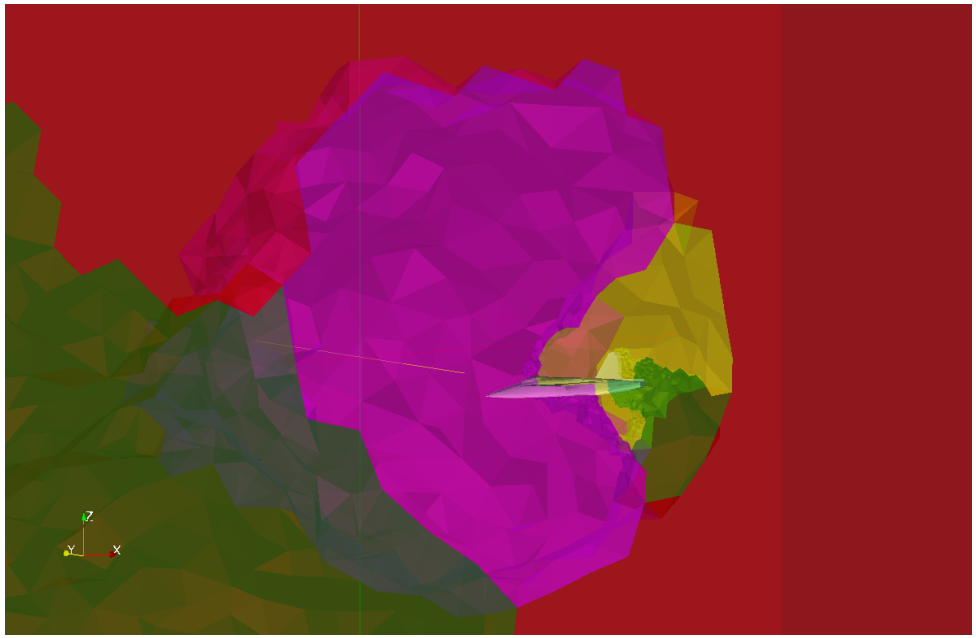
By following the instructions, you should get something like this



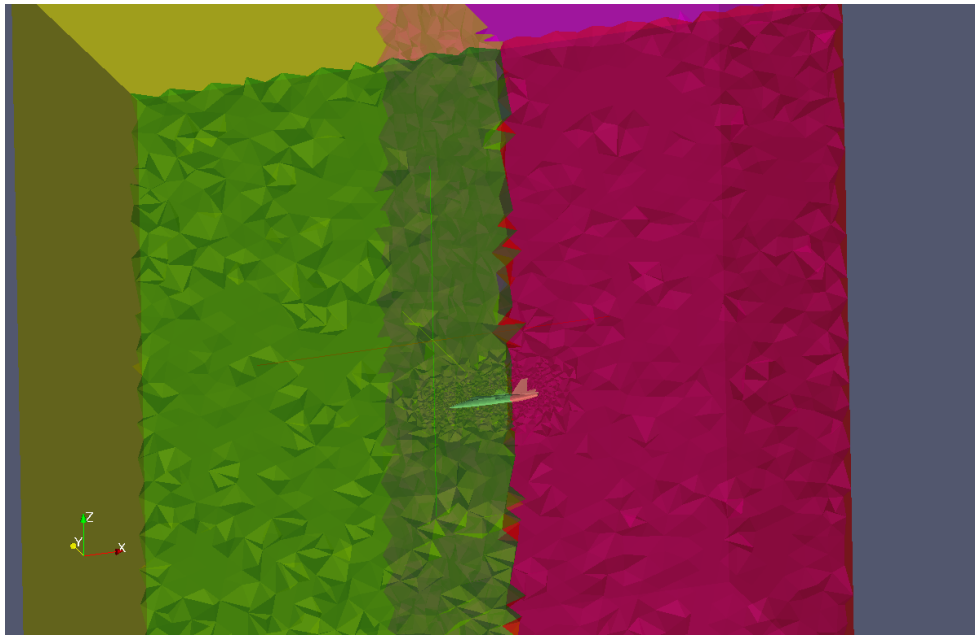
Mesh partitioning – Hierarchical Method

Running in parallel

By following the instructions, you should get something like this



Mesh partitioning – Scotch Method



Mesh partitioning – Hierarchical Method

Running in parallel

- By the way, if you do not want to post-process each decomposed domain individually, you can use the option `-cellDist` when decomposing the domain. This option applies for all decomposition methods.
- In the terminal type:

1. `$> cd $PTOFC/parallel/yf17/scotch_celldist`
2. `$> fluent3DMeshToFoam ../../../meshes_and_geometries/fluent_yf17_yf17.cas`
3. `$> decomposePar`

- The option `-cellDist` will write the cell distribution as a **`volScalarField`** field that can be used for post-processing. The new field variable will be saved in the file *`cellDist`* located in the time directory **`0`**.

- At this point, you can launch `paraFoam` and post process the case as you usually do. You will need to play with transparency and volume rendering to see the cell distribution among the processors.

Running in parallel

Decomposing big meshes

- One final word, the utility `decomposePar` does not run in parallel. So, it is not possible to distribute the mesh among different computing nodes to do the partitioning in parallel.

- If you need to partition big meshes, you will need a computing node with enough memory to handle the mesh. We have been able to decompose meshes with up to 500.000.000 elements, but we used a computing node with 512 gigs of memory.

- For example, in a computing node with 16 gigs of memory, it is not possible to decompose a mesh with 30.000.000. You will need to use a computing node with at least 32 gigs of memory.

- Same applies for the utility `reconstructPar`.

Running in parallel

Do all utilities run in parallel?

- At this point, you might be wondering if all solvers/utilities run in parallel.
- To know what solvers/utilities do not run in parallel, in the terminal type:
  - `$> find $WM_PROJECT_DIR -type f | xargs grep -sl 'noParallel'`
- Paradoxically, the utilities used to decompose the domain and reconstruct the domain do not run in parallel.
- Another important utility that does not run in parallel is `blockMesh`. So to generate big meshes with `blockMesh` you need to use a big fat computing node.
- Another important utility that does not run in parallel by default is `paraFoam`.
- To compile `paraFoam` with MPI support in the file *`makeParaView4`* (located in the directory `$WM_THIRD_PARTY_DIR`), set the option **`withMPI`** to true,
  - **`withMPI = true`**
- While you are working with the file *`makeParaView4`*, you might consider enabling Python support,
  - **`withPYTHON = true`**

Running in parallel

Right syntax to run in parallel

- Do not forget the right syntax used to run OpenFOAM® solvers in parallel:

  - `$> mpirun -np <NPROCS> <application/utility> -parallel`

  where `mpirun` is a shell script to use the mpi library, `<NPROCS>` is the number of processors you want to use (same as the number of partitions), `<application/utility>` is the OpenFOAM® solver or utility you want to use, and `-parallel` is a flag you shall always use if you want to run in parallel.

- When running in parallel, do not forget to use the flag `-parallel`. If you forget to use it, the solver will run but it will run `<NPROCS>` copies of the same job.

Running in parallel

Exercises

- This final section is optional, self-paced and do it at anytime.
- The proposed exercises are designed to test your knowledge and to reinforce the concepts addressed during the presentations.
- All the concepts to be addressed in the following exercises have been treated in the previous slides, so the reader should not have problems answering the questions.
- If you have doubts, do not hesitate in asking.
- To help you answering the exercises, we will give you a few tips.
- And whenever it is possible, the solution will be given.

Running in parallel

Exercises

- Choose any tutorial or design your own case and do an scalability test. Scale your case with two different meshes (a coarse and a fine mesh).
- Run the same case using different partitioning methods. Which method scales better? Do you get the same results?
- Do you think that the best partitioning method is problem dependent?
- Compare the wall time of a test case using the maximum number of cores and the maximum number of virtual cores. Which scenario is faster and why?
- Run a parallel case without using the `-parallel` option. Does it run? Is it faster of slower? How many outputs do you see on the screen?
- Do we get any speed-up by using `renumberMesh`?
- What applications do not run in parallel?

Roadmap

1. ~~Running in parallel~~
2. **Running in a cluster using a job scheduler**
3. Running on the cloud

Running in a cluster using a job scheduler

- Running OpenFOAM® in a cluster is similar to running in a normal workstation with shared memory.
- The only difference is that you will need to launch your job using a job scheduler.
- Common job schedulers are:
    - Terascale Open-Source Resource and Queue Manager (TORQUE).
    - Simple Linux Utility for Resource Management (SLURM).
    - Portable Batch System (PBS).
    - Sun Grid Engine (SGE).
    - Maui Cluster Scheduler.
    - BlueGene LoadLeveler (LL).
- Ask your system administrator the job scheduler installed in your system. Hereafter we will assume that you are using PBS.

Running in a cluster using a job scheduler

- To launch a job in a cluster with PBS, you will need to write a small shell script where you tell to the job scheduler the resources you want to use and what you want to do.

```
#!/bin/bash
#
# Simple PBS batch script that reserves 16 nodes and runs a
# MPI program on 128 processors (8 processor on each node)
# The walltime is 24 hours !
#
#PBS -N openfoam_simulation                    //name of the job
#PBS -l nodes=16,walltime=24:00:00             //max resources and execution time
#PBS -m abe -M joel.guerrero@unige.it          //send an email as soon as the job
                                               //is launch or terminated

cd PATH_TO_DIRECTORY                           //go to this directory

decomposePar                                   //decompose the case

mpirun -np 128 pimpleFoam -parallel > log      //run parallel solver
```

The green lines are not PBS comments, they are comments inserted in this slide. PBS comments use the number sign (#).

Running in a cluster using a job scheduler

- To launch your job you need to use the `qsub` command (part of the PBS job scheduler). The command `qsub` will send your job to queue.
  - `$> qsub script_name`
- Remember, running in a cluster is no different from running in your workstation or portable computer. The only difference is that you need to schedule your jobs.
- Depending on the system current demand of resources, the resources you request and your job priority, sometimes you can be in queue for hours, even days, so be patient and wait for your turn.
- Remember to always double check your scripts.

Running in a cluster using a job scheduler

- Finally, remember to always plan how you will use the resources available.
- For example, if each computing node has 8 gigs of memory available and 8 cores. You will need to distribute the work load in order not to exceed the maximum resources available per computing node.
- So if you are running a simulation that requires 32 gigs of memory, the following options are valid:
  - Use 4 computing nodes and ask for 32 cores. Each node will use 8 gigs of memory and 8 cores.
  - Use 8 computing nodes and ask for 32 cores. Each node will use 4 gigs of memory and 4 cores.
  - Use 8 computing nodes and ask for 64 cores. Each node will use 4 gigs of memory and 8 cores.
- But the following options are not valid:
  - Use 2 computing nodes. Each node will need 16 gigs of memory.
  - Use 16 computing nodes and ask for 256 cores. The maximum number of cores for this job is 128.

Roadmap

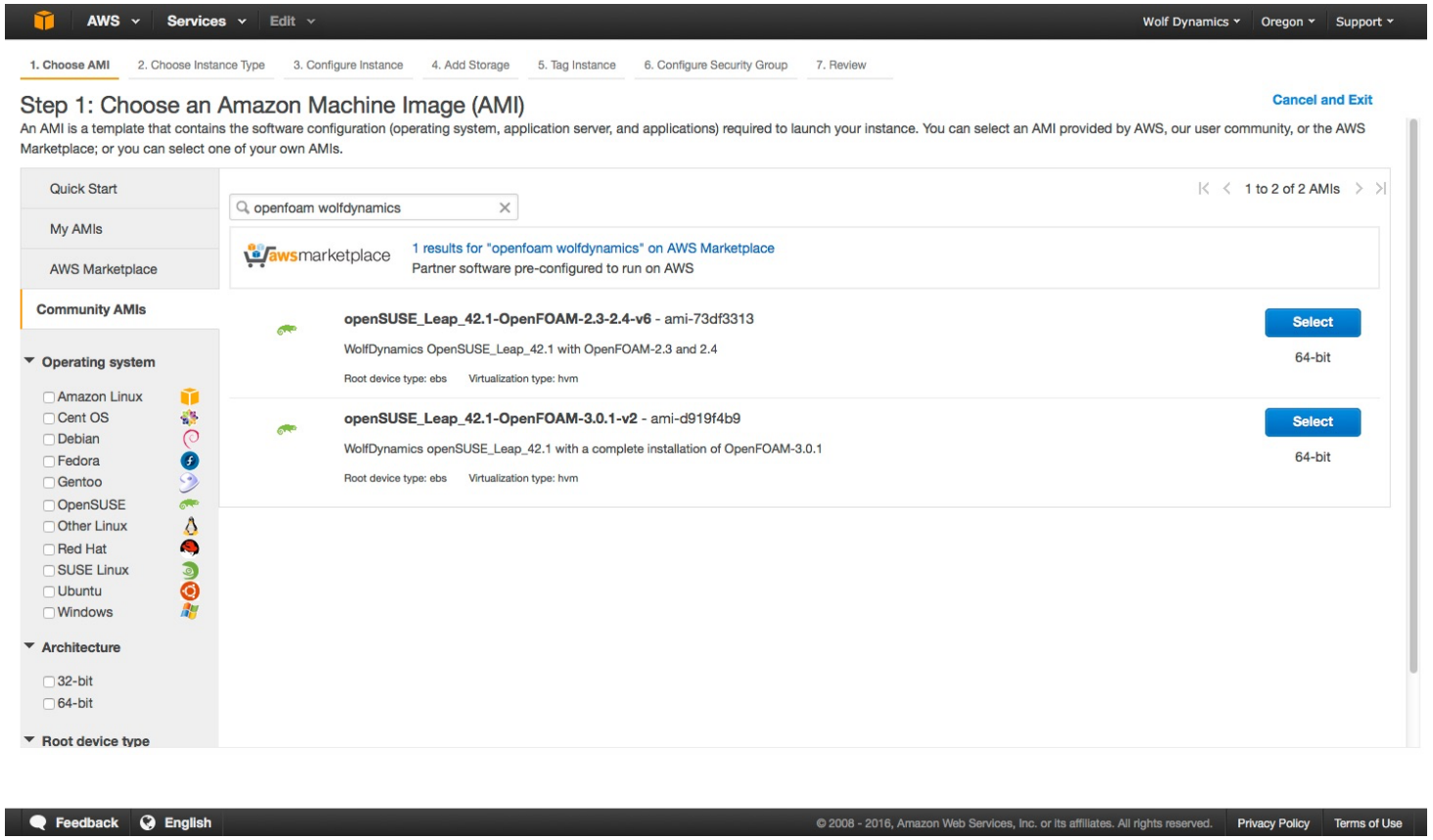
1. ~~Running in parallel~~
2. ~~Running in a cluster using a job scheduler~~
3. **Running on the cloud**

Running on the cloud

- And finally, running OpenFOAM® on the cloud is similar to running in your workstation or in a cluster.
- Moving to the cloud is the most cost effective option for users that do not require a dedicated cluster or server for running their simulations and are looking for computing power at very affordable prices.
- When working on the cloud, you can scale up or scale out.
- You only need to choose a cloud computing service provider and deploy an image of your OS with the desired applications installed.
- If you are interested in trying cloud computing, we recommend Amazon Elastic Compute Cloud platform (Amazon EC2).
- You can find more information in these links:
    - https://aws.amazon.com/ec2/
    - http://www.wolfdynamics.com/hpc.html

Running on the cloud

- If you are interested in trying OpenFOAM in Amazon EC2, you can use our Amazon Machine Image (AMI). Just sign in or create an Amazon Web Service (AWS) account, and then look for the Amazon Machine Image **openfoam wolfdynamics** in the community AMIs.



Running on the cloud

- Remember these Linux commands when working in an Amazon Linux instance:

1. `$> lsblk`
   Lists information about all available or the specified block devices
2. `$> file -s /dev/device_name`
   To know if there is a file system on the device (volume)
3. `$> mkfs -t ext4 device_name`
   To create a file system on the device
4. `$> mount device_name mount_point`
   To mount a device with a file system
5. `$> umount mount_point`
   To unmount a device
6. `$> useradd -m username`
   To create a new user
7. `$> userdel -r username`
   To erase a user
8. `$> su username`
   To change user
9. `$> chown user:group dirname`
   To change directory owner and group

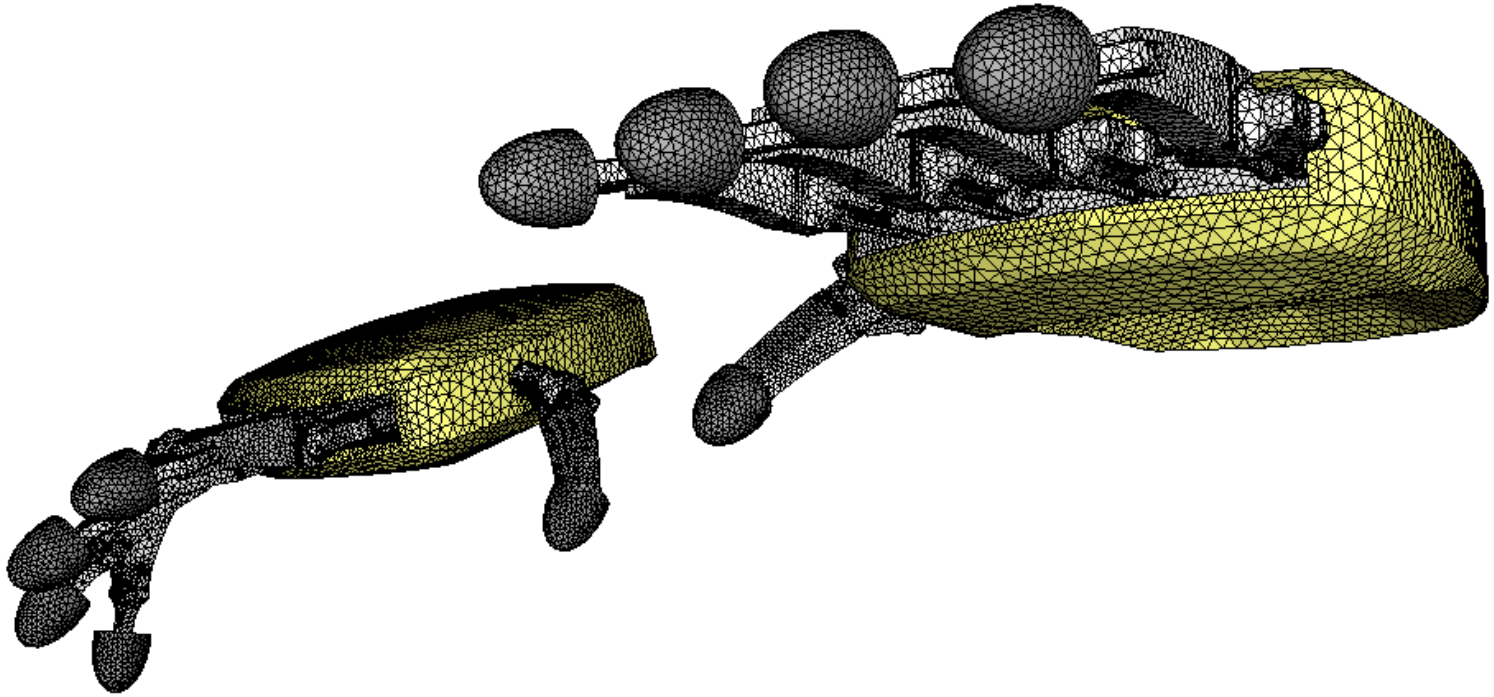
Module 5

Wrap up

Main takeaways

- After finishing this module, you should be able to do the following:
  - Run solvers in parallel.
  - Use the postprocessing utilities in parallel.
  - Identify the number of physical cores in your workstation/notebook.
  - Do scalability tests.
  - Visualize with paraFoam/paraview the decomposed case.
  - Visualize with paraFoam/paraview the partitions.
  - Explore the source code.
  - Find information in the source code and the doxygen documentation.

Additional tutorials



- In the course's directory (**`$PTOFC`**) you will find many tutorials (which are different from those that come with the OpenFOAM® installation). We highly encourage you to go through each one to understand and get functional using OpenFOAM®.
- Remember, in each case directory you will find a *`README.FIRST`* file with general instructions of how to run the case.
- If you have a case of your own, let us know and we will try to do our best to help you to setup your case. But remember, the physics is yours.