Module 4

Running in parallel

Roadmap

1. **Running in parallel**

Running in parallel

- First of 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:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                24
On-line CPU(s) list:   0-23
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 44
Model name:            Intel(R) Xeon(R) CPU       X5670  @ 2.93GHz
Stepping:              2
CPU MHz:               1600.000
CPU max MHz:           2934.0000
CPU min MHz:           1600.0000
BogoMIPS:              5851.91
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              12288K
NUMA node0 CPU(s):     0-5,12-17
NUMA node1 CPU(s):     6-11,18-23
```

Total number of cores available after hyper threading (virtual cores)

Number of threads per core (hyper threading)

Number of cores per socket or physical processor

Number of sockets (physical processors)

This is what makes a processor expensive

Total number of physical cores
=
Number of cores per socket X Number of sockets

Total number of physical cores = 6 X 2 = 12 cores

Running in parallel

- OpenFOAM® does not take advantage of hyper threading technology (HT).
- HT is basically used by the OS to improve multitasking performance.
- This is what we have in the workstation of the previous example:
    - 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 clusters/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 computers interconnected have access to more memory (distributed 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 (speed-up):**

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

- **Save money:**

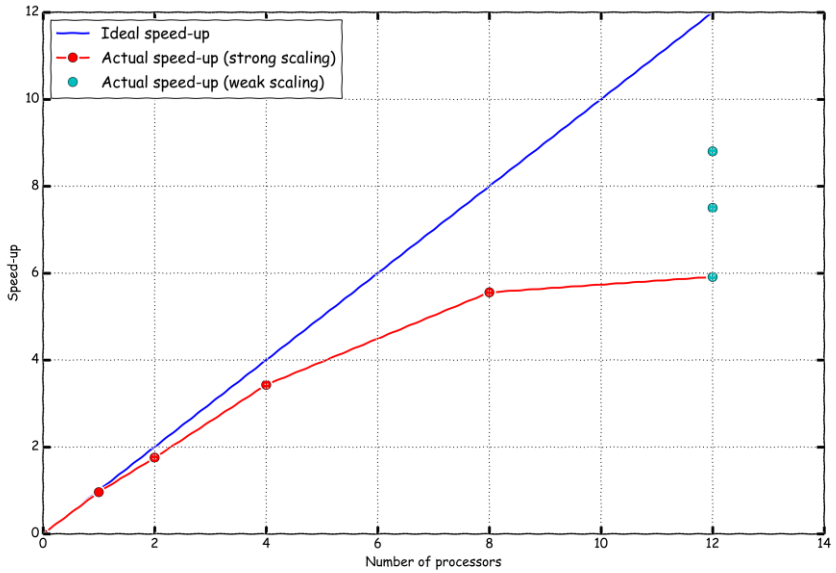
  In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel computers can be built from cheap, commodity components.

- **Limits to serial computing:**

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

Running in parallel

Speed-up and scalability example

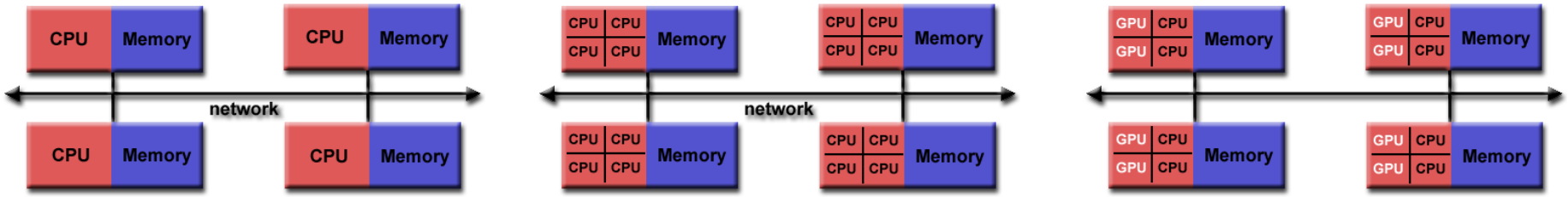


- In the context of high-performance computing (HPC), there are two common metrics that measure the scalability of the application:
  - **Strong scaling (Amdahl's law)**: which is defined as how the solution time varies with the number of processors for a fixed problem size (number of cells in CFD)
  - **Weak scaling (Gustafson's law)**: which is defined as how the solution time varies with the number of processors for a fixed problem size per processor (or increasing the problem size with a fix number of processors).
- In this example, when we reach 12 cores inter-processor communication slow-downs the computation. But if we increase the problem size for a fix number of processors, we will increase the speed-up.
- 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 – Clusters 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.
- 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 15000 processors.
- We have been able to run OpenFOAM® using single GPU and multiple GPUs.
- Do not ask 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 hardware.

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 need the dictionary *`decomposeParDict`* which is located in the **`system`** directory.

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

  To do so, OpenFOAM® uses 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.

- **Additionally, you might want to reconstruct (put back together) the decomposed domain.**

  This is done by using the `reconstrucPar` utility. You do not need a dictionary to use this utility. This step is optional.

Running in parallel

Domain Decomposition in OpenFOAM®

- The mesh and fields are decomposed using the `decomposePar` utility.
- They are broken up according to a set of parameters specified in a dictionary named *`decomposeParDict`* that is located in the **`system`** directory of the case.
- In the *`decomposeParDict`* dictionary the user must set the number of domains in which the case should be decomposed (using the keyword **numberOfSubdomains**). The value used should correspond to the number of physical cores available.

```
numberOfSubdomains 128;
method   scotch;
```

Number of subdomains

Decomposition method

- In this example, we are subdividing the domain in 128 subdomains, therefore we should have 128 physical cores available.
- The main goal of domain decomposition is to minimize the inter-processors communication and the processor workload.

Running in parallel

Domain Decomposition Methods

- These are the decomposition methods available in OpenFOAM® 9. To name a few:
  - 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

The information inside the directories **`polyMesh/`** and **`0/`** is decomposed using the utility `decomposePar`

`decomposePar`

`processor0`

`processor1`

`processor2`

`processor3`

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

Running in parallel

Running in parallel – Gathering all together

- After decomposing the mesh, we can run in parallel using MPI.

- The interface between each region, is know as halo zone.
- The inter-processor communication in the halo zone is managed by the MPI library.
- Remember, the main goal is to minimize the halo zone, therefore, the inter-processor communication.

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

- 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

Running in parallel – Gathering all together

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

`reconstructPar`

- When you reconstruct the case, you glue together all the information contained in the decomposed case.
- All the information (mesh, boundary conditions, initial conditions, and the solution), is transfer to the original case folder (**`polyMesh`** and time solution directories).
- This step is optional.

Running in parallel

Running in parallel – Gathering all together

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

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

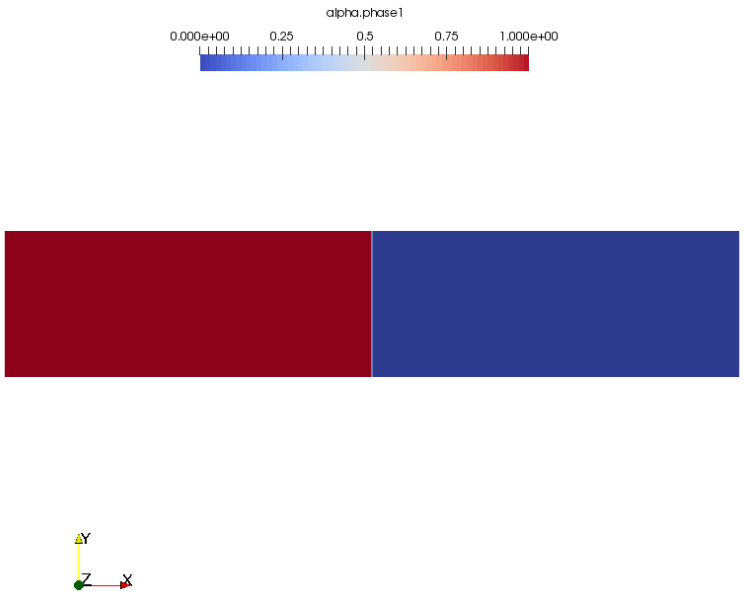
This step is optional

- When running in parallel, do not forget to add this flag.
- If you do not add this flag, the application will run, but it will execute <NPROCS> duplicates of the same job (it will be slower).

- You can do the post-processing and visualization on the decomposed case or reconstructed case. We are going to address this later.
- If you are dealing with moving bodies where the mesh topology is changed or if you are using AMR, you will need to use `reconstructParMesh` before `reconstrucPar`.

Running in parallel

Kelvin Helmholtz instability in a coarse mesh



Volume fraction

www.wolfdynamics.com/wiki/kelvin_helmholtz/ani1.gif

Processors	Clock time (seconds)	Mesh size in x, y, and z directions
1	955	800 X 160 X 1
2	564	800 X 160 X 1
4	333	800 X 160 X 1
8	234	800 X 160 X 1
12	244	800 X 160 X 1

You will find this case in the directory: **`$PTOFC/parallel/kelvin_helmholtz`**

Running in parallel

Visualization of a parallel case

- The traditional way is to first reconstruct the case and then do the post-processing and visualization on the reconstructed case.
- To do so, we type in the terminal:

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

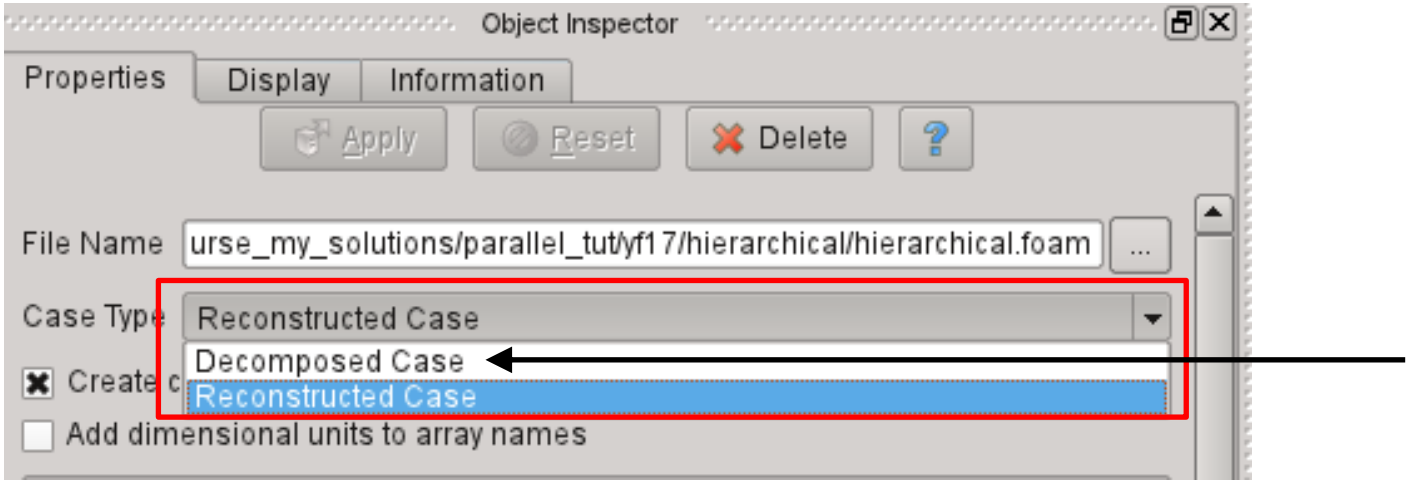
- Step 1 reconstruct the case. Remember, you can choose to reconstruct all the time steps, the last time step or a range of time steps.
- In step 2, we use `paraFoam` to visualize the reconstructed case.

Running in parallel

Visualization of a parallel case

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

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



Running in parallel

Visualization of a parallel case

- Both of the previous methods are valid.
- When we use the option `-builtin` with `paraFoam`, we have the option to work on the decomposed case directly. In other words, we do not need to reconstruct the case.
- This option is also faster than running `paraFoam` with no flags.
- 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 input from the user.

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

Running in parallel

Exercises

- Choose any tutorial or design your own case and do a 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 you get any speed-up by using `renumberMesh`?
- What applications do not run in parallel?