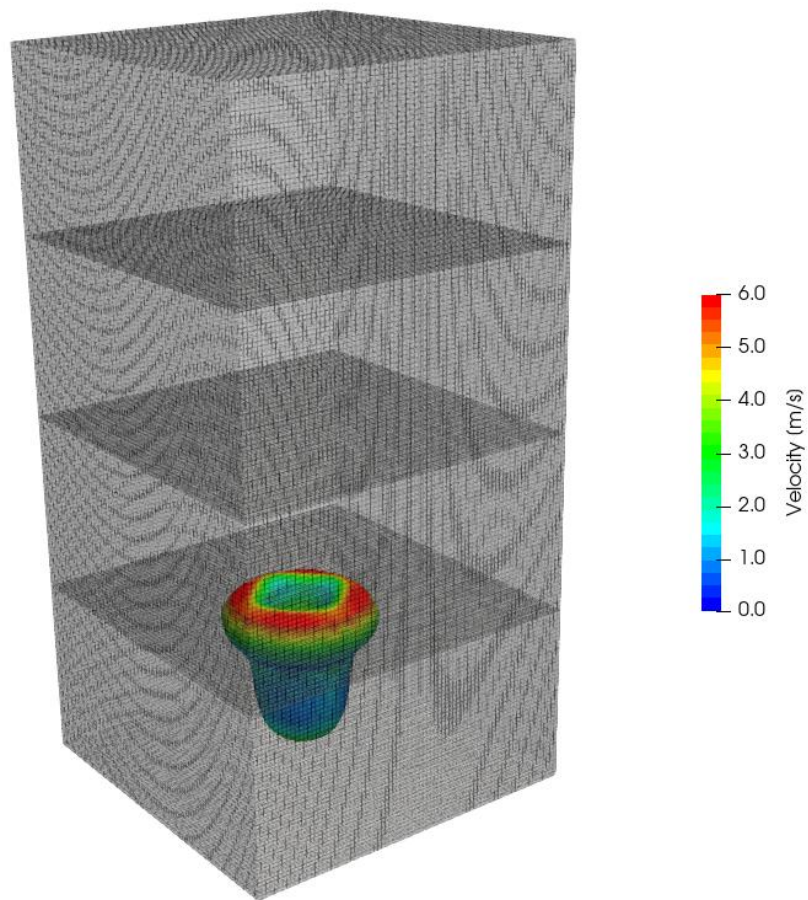


Tutorial Nine

Parallel Processing



Bahram Haddadi



7th edition, March 2025

 Except where otherwise noted, this work is licensed under <http://creativecommons.org/licenses/by-nc-sa/3.0/>

Contributors:

- Bahram Haddadi
- Christian Jordan
- Michael Harasek
- Clemens Gößnitzer
- Sylvia Zibuschka
- Yitong Chen
- Vikram Natarajan
- Jozsef Nagy



Technische Universität Wien
Institute of Chemical, Environmental
& Bioscience Engineering



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-nc-sa/3.0/>

Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

You are free:

- to Share — to copy, distribute and transmit the work
- to Remix — to adapt the work

Under the following conditions:

- Attribution — you must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that, they endorse you or your use of the work).
- Noncommercial — you may not use this work for commercial purposes.
- Share Alike — if you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

With the understanding that:

- Waiver — any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain — where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights — In no way are any of the following rights affected by the license:
 - Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
 - The author's moral rights;
 - Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.
- Notice — for any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

This offering is not approved or endorsed by ESI® Group, ESI-OpenCFD® or the OpenFOAM® Foundation, the producer of the OpenFOAM® software and owner of the OpenFOAM® trademark.

Available from: www.fluiddynamics.at

Background

1. Parallel computing

Modern Computational Fluid Dynamics (CFD) problems often involve complex geometries, turbulence, multiphase flows, and chemical reactions, making simulations computationally expensive. Parallel computing reduces computation time by distributing the workload across multiple processors, enabling simulations that would otherwise be infeasible on a single machine.

Parallel computing involves dividing the computational domain into smaller sub-domains, which are assigned to different processors. This step is known as **domain decomposition**. These processors simultaneously perform calculations and communicate with each other to synchronize data exchange. This approach significantly reduces computational time, making it essential for large-scale simulations in engineering and scientific research.

Parallel computing can be carried out in two ways. One is done on a single computer with multiple internal processors, known as a **Shared Memory Multiprocessor**. The other way is achieved through a series of computers interconnected by a network, known as a **Distributed Memory Multicomputer**.

1.1. Shared versus distributed memory

	<i>Shared Memory Multiprocessor</i>	<i>Distributed Memory Multicomputer</i>
<i>Memory</i>	Data is saved in a global memory that can be accessed by all processors	Each computer has a local memory and a processor can only access its local memory
<i>Data transfer between processors</i>	The sender processor simply needs to write the data in a global variable and the receiver can read it	Message is sent explicitly from one computer to another using a message passing library, e.g. Message Passing Interface (MPI)

In OpenFOAM® the application of parallel computing can be executed using the *decomposePar* command. This allows the solver to be run on multiple processors. The workflow of parallel computation in OpenFOAM® is summarized below:

- Division of the mesh into sub-domains
- Running of the solver in parallel
- Reconstruction of the meshes and connecting the results.

2. Introduction to compressible flow

Until now, we have primarily considered incompressible fluid flows, where the density remains constant. However, in many real-world scenarios, fluid density varies significantly due to changes in pressure and temperature. This variation

in density makes the flow compressible, requiring specialized solvers and numerical methods.

A classic example of compressible flow is the flow through a converging-diverging nozzle, where fluid accelerates to supersonic speeds. Compressibility becomes dominant in flows when the Mach number is greater than about 0.3. The Mach number is defined as follows:

$$Ma = \frac{u}{c} = \frac{\text{local velocity}}{\text{speed of sound}}$$

Flow Classification Based on Mach Number:

- **Subsonic Flow ($Ma < 0.3$):** Flow is nearly incompressible, and density changes are negligible.
- **Transonic Flow ($0.3 < Ma < 1.0$):** Density variations begin to affect flow behavior.
- **Supersonic Flow ($Ma > 1.0$):** Shock waves and expansion waves appear, requiring advanced solvers.
- **Hypersonic Flow ($Ma > 5.0$):** Extremely high-speed flow, where thermal and chemical effects become significant.

When a fluid flow is compressible, temperature and pressure are affected strongly by variations in density. It is therefore important to consider the linkage between pressure, temperature and density in compressible flow, usually by applying an equation of state from thermodynamics, e.g. the ideal gas equation. More complex real-gas equations of state (e.g., Peng-Robinson, Van der Waals) may be required for high-pressure or reactive flows.

3. Compressible flow solvers

There are two general types of solution schemes for compressible flow: pressure-based and density-based.

3.1. Pressure-based solvers

This type of solver was historically derived from the solution approach used on incompressible flows. They solve for the primitive variables. The discretized momentum and energy equations are used to update velocities and energy. The pressure is obtained by applying a pressure-correction algorithm on the continuity and momentum equations. Density is then calculated from the equation of state.

3.2. Density-based solvers

Density-based solvers are suitable for solving the conserved variables. Similar to pressure-based solvers, the conserved velocity and energy terms are updated from the discretized momentum and energy equations. We can then solve for density from the continuity equation, afterwards we use the equation of state to update the pressure.

In general, density based solvers are more suitable for high-speed compressible flows with shocks. This is because density based solvers solve for conserved quantities across the shock, so the discontinuities will not affect the results.

compressibleVoF – depthCharge3D

Tutorial outline

Use the compressibleVoF solver, simulate the example case for 0.5 s.

Objectives

- Understanding the difference between incompressible and compressible solvers
- Understanding parallel processing and different discretization methods

Data processing

Investigate the results in ParaView.

1. Pre-processing

1.1. Copying tutorial

Copy the tutorial from following directory to your working directory:

```
$FOAM_TUTORIALS/compressibleVoF/depthCharge3D
```

1.2. 0 directory

In the 0 directory copy the *alpha.water.orig*, *p.orig*, *p_rgh.orig* and *T.orig* files to *alpha.water*, *p*, *p_rgh* and *T* respectively.

Note: You can also skip copying the *.orig files, since running the *setFields* will do it for you.

1.3. constant directory

Phases and common physical properties of the two phases are set in the *phaseProperties* file.

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

phases (water air);

sigma          0.07;

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

Individual phase properties are set in *physicalProperties.phase* files, e.g. *physicalProperties.air*.

1.4. system directory

The *decomposeParDict* file includes the parallel settings, such as the number of domains (partitions) and also how the domain is going to be divided into these subdomains for parallel processing.

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

numberOfSubdomains 4;

method          hierarchical;

simpleCoeffs
{
    n             ( 1 4 1 );
}

hierarchicalCoeffs
{
    n             ( 1 4 1 );
    order         xyz;
}

manualCoeffs
{
    dataFile      "";
}
```

```
distributed      no;

roots            ( );
// * * * * *
* * * * * //
```

`numberOfSubdomains` should be equal to the number of cores used. `method` should show the method to be used. In the above example, the case is simulated with the `hierarchical` method and 4 processors.

If the `simple` method is being used, the parameter `n` must be changed accordingly. The three numbers `(1 4 1)` indicate the number of pieces the mesh is split into in the x, y and z directions, respectively. Their multiplication result should be equal to `numberOfSubdomains`.

If the `hierarchical` method is being used, these parameters and the order in which the mesh should be split up in each direction should be provided.

If the `scotch` method is being used, then no user-supplied parameters are necessary except for the number of subdomains.

Note: In order to check the quality of the mesh, the checkMesh tool can be used (run it from main case directory). If the message “Mesh OK” is displayed – the mesh is fine and no corrections need to be done. If the mesh fails in one or more tests, try to recreate or refine the mesh for a better mesh quality (less non-orthogonally and skewness).

If non-orthogonal cells exist in a mesh, another option is using non-orthogonal corrections in the fvSolution file in the algorithm sub-dictionary (e.g. PIMPLE or PISO). Usually using 1 or 2 as nNonOrthogonalCorrectors is enough.

2. Running simulation

```
>blockMesh
```

```
>setFields
```

For running the simulation in parallel mode the computing domain needs to be divided into subdomains and a core should be assigned to each subdomain. This is done by following command:

```
>decomposePar
```

This decomposes the mesh according to the supplied instructions. One possible source of error is the product of the parameters in `n` does not match up to the number of the subdomains. This appears for the simple and hierarchical methods.

After executing this command four new directories will be made in the simulation directory (processor0, processor1, processor2 processor3), and each subdomain calculation will be saved in the respective processor directory.

Note: When the domain is divided to subdomains in parallel processing new boundaries are defined. The data should be exchanged with the neighbor boundary, which it is connected to in the main domain.


```
>mpirun -np <No of cores> solver -parallel > log
```

<No of cores> is the number of cores being used. `solver` is the solver for this simulation. For example, if 4 cores are desired, and the solver is `compressibleInterFoam` following command is used:

```
>mpirun -np 4 foamRun -solver compressibleVoF -parallel > log
```

> `log` is the filename for saving the simulation status data, instead of printing them to the screen. For checking the last information which is written to this file the following command can be used during the simulation running:

```
>tail -f log
```

Note: Before running any simulation, it is important to run the `top` command (type the 'top' command in the terminal), to check the number of cores currently used on the machine. Check the load average. This is on the first line and shows the average number of cores being used. There are three numbers displayed, showing the load averages across three different time scales (one, five and 15 minute respectively).

Add the number of cores you plan to use to this number – and you will get the expected load average during your simulation. This number should be less than the total number of cores in the machine – or the simulation will be slowed or the machine will crash (if you run out of memory). If you are running on a multi user server it is recommended to leave at least a few cores free, to allow for any fluctuations in the machine load.

Note: top command execution can be interrupted by typing `q` (or `ctrl+c`)

The simulation can take several hours, depending on the size of the mesh and time step size.

3. Post-processing

For exporting data for post processing, at first all the processors data should be put together and a single combined directory for each time step was created. By executing the following command all the cores data will be combined and new directories for each time step will be created in the simulation main directory:

```
>reconstructPar
```

Convert the data to ParaView format:

```
>foamToVTK
```

Note: To do the reconstruction or `foamToVTK` conversion from a start time until an end time the following flags can be used:

```
>reconstructPar -time [start time name, e.g. 016]:[end time name, e.g. 020]
```

```
>foamToVTK -time [start time name, e.g. 016]:[end time  
name, e.g. 020]
```

Using above commands without entering end time will do the reconstruction or conversion from start time to the end of available data:

```
>reconstructPar -time [start time name, e.g. 016]:
```

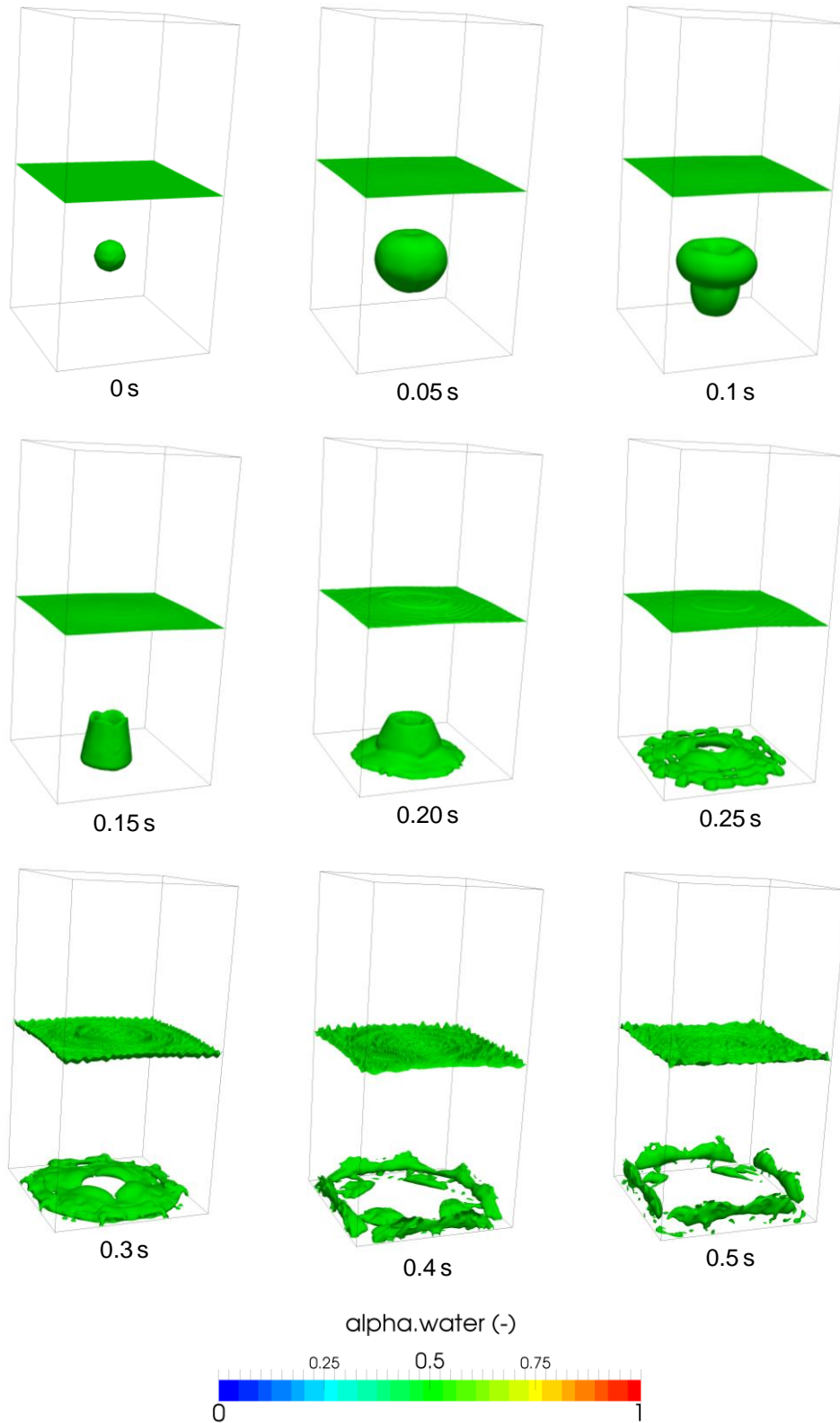
```
>foamToVTK -time [start time name, e.g. 016]:
```

For reconstructing or converting only one time step the commands should be used without end time and “.”:

```
>reconstructPar -time [time name, e.g. 016]
```

```
>foamToVTK -time [time name, e.g. 016]
```

The simulation results are as follows:



3D depth charge, alpha = 0.5 iso-surfaces, parallel simulation

The first number n after the header, but before the opening brackets, 1024000 in this example, refers to the number of points in the mesh. Within the brackets, n lines follow. Each line contains one number between 0 and $n-1$, where n is the number of cores to be used for the computation. This number refers to the core being used to compute the corresponding cell in the points file in the constant directory. For example, if the second line in the points file brackets

reads (0.125 0 0) and the second line in the `cellDecomposition` directory reads 0, this means that the cell (0.125 0 0) will be processed by processor 0.

This `cellDecomposition` file can now be edited. Although this can be done manually, it is probably not feasible for any sufficiently large mesh. The process must thus be automated by writing a script to populate the `cellDecomposition` file according to the desired processor breakdown.

When the new file is ready, save it under a different name:

```
>cp cellDecomposition manFile
```

Now, edit the `decomposeParDict` file. Select decomposition method `manual`, and for the `dataFile` field in the `manual` coeffs range, specify the path to the file which contains the manual decomposition. Note that OpenFOAM® searches in the `constant` directory by default, in case relative paths are being used:

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

numberOfSubdomains 4;

method            manual;

simpleCoeffs
{
    n              ( 1 4 1 );
    delta          0.001;
}

hierarchicalCoeffs
{
    n              ( 1 4 1 );
    delta          0.001;
    order          xyz;
}

manualCoeffs
{
    dataFile       "manFile";
}

distributed       no;

roots             ( );

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

Delete the old processor directories, decompose the case with the new decomposition settings and run the simulation.

4.2. Visualizing the processor breakdown

It may be interesting to visualize how exactly OpenFOAM® breaks down the mesh. This can be easily visualized using ParaView. After running the simulation, but before running the `reconstructPar` command, repeat the following for each of the processor directories:

```
>cd processor<n>
```

where n is the processor number

```
>foamToVTK
```

convert the individual processor files to VTK, next, open ParaView:

```
>paraview &
```

For each of the processor directories, perform the following steps:

- Open the VTK files in the relevant processor directory
- Double click them to open them and click on “Apply”
- The part of the mesh decomposed by that core will appear, in grey.
- Change the color in the drop-down menus in the toolbar. This is to ensure that each individual part can be easily seen.

Once this is done for all processors, the entire mesh will appear. However, the processor regions can now easily be seen in a different color.

In order to save this, there are two options. The first option is to take a screenshot:

File > Save a screenshot

The second option is to save the settings and modifications as a ParaView state file.

File > Save State

The current settings and modifications can then be easily recovered by:

File > Load State

Saving the state allows changes to be made afterwards. Saving a screenshot keeps only a picture, while losing the ability to make changes after exiting ParaView. Doing both is recommended.