

# Scalable Workflows for OpenFOAM Evaluation

Ilya Evdokimov<sup>1</sup>, Susann Haensch<sup>1</sup>, Fabian Schlegel<sup>1</sup>

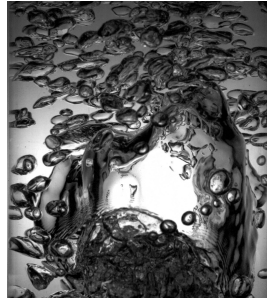
<sup>1</sup>*Helmholtz-Zentrum Dresden-Rossendorf, Institute for Fluid Dynamics*

10th December 2020



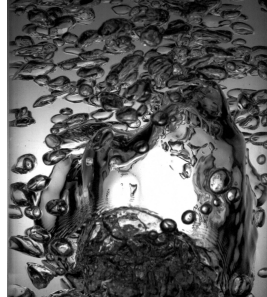
What is multiphase CFD?

- momentum exchange,



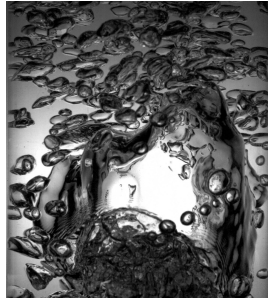
What is multiphase CFD?

- momentum exchange,
- heat transfer,



What is multiphase CFD?

- momentum exchange,
- heat transfer,
- phase change.



What is multiphase CFD?

- momentum exchange,
- heat transfer,
- phase change.



## Momentum transfer models in the Euler-Euler methodology


Our estimation gives 96 possible combinations of non-drag force models which all may be tested assuming that they are equally important for an arbitrary CFD case.

## └ Introduction

Introduction

What is multiphase CFD?

- momentum exchange,
- heat transfer,
- phase change.



Momentum transfer models in the Euler-Euler methodology

Our estimation gives 96 possible combinations of non-drag force models which all may be tested assuming that they are equally important for an arbitrary CFD case.

Multiphase CFD involves the simulation of two or more fluids, which includes momentum exchange, heat transfer, phase change. The amount of equations is doubled compared to single phase cases. Multiphase CFD for applications at the component-scale is usually done via Eulerian-Eulerian framework. It requires us to introduce momentum transfer models aiming to describe various microscopic phenomena between the phases (such as drag or lift forces acting on bubbles). These sub-models actually interact with each other. For a robust combination of sub-models we may need to try every potential model combination. In the particular example of a bubbly flow for a single case setup we could easily get 96 different model combinations to analyze. What if we need to run our simulations across 10 different setups representing different industrial applications? It gives us about 1000 of different cases. Obviously the task is beyond the human limit.

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python



What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,
  - GnuPlot

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,
  - GnuPlot
- Automated testing

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,
  - GnuPlot
- Automated testing
  - How to evaluate (sub-) model if it is not finished yet?

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,
  - GnuPlot
- Automated testing
  - How to evaluate (sub-) model if it is not finished yet?
- Design of Experiments and Optimization

What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases,
  - bash
  - Python
- Hundreds (Thousands?) data samples,
  - GnuPlot
- Automated testing
  - How to evaluate (sub-) model if it is not finished yet?
- Design of Experiments and Optimization

## The Idea

Represent case dependency and result analysis in a single meta-algorithm.

## └ Introduction

**Introduction**

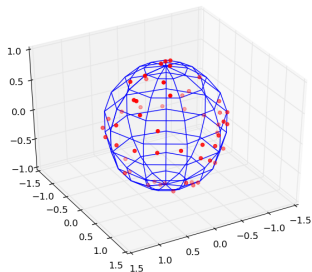
What is industry-oriented CFD?

- Tens (Hundreds?) small dedicated OpenFOAM test cases.
  - bash
  - Python
- Hundreds (Thousands?) data samples.
  - GnuPlot
- Automated testing
  - How to evaluate (sub-) model if it is not finished yet?
- Design of Experiments and Optimization

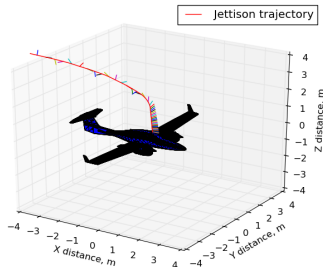
**The Idea**  
Represent case dependency and result analysis in a single meta-algorithm.

The problem of analyzing hundreds of cases is already solved in OpenFOAM to some extent via bash scripting. There are widely-used tutorials and test cases. In some particular cases tutorials and tests may involve GnuPlot as the major tool to plot graphs for all kinds of engineering needs. Everything runs automatically and checks established models and solvers. But what if we need to test a new model on dozens of cases? What if we need to run a Design-of-experiments kind of study? For that purpose we could employ various accessible optimization packages or even general-purpose programming languages. At the previous ISP RAS conference I had already presented an idea to represent OpenFOAM cases as graphs.

Example from "OpenFOAM Case Simulation Control Using Graphs":



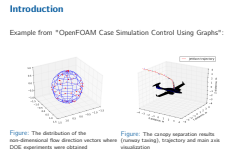
**Figure:** The distribution of the non-dimensional flow direction vectors where DOE experiments were obtained



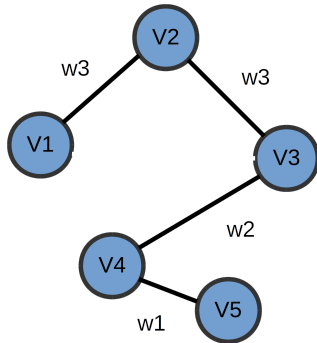
**Figure:** The canopy separation results (runway taxing), trajectory and main axis visualization



## └ Introduction



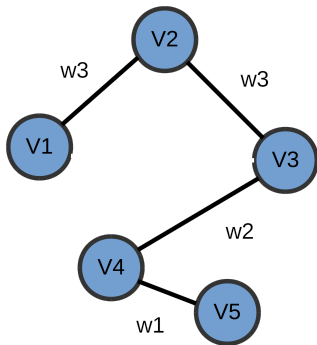
It was done in connection to the Design of Experiments study where aerodynamical characteristics of the canopy or windshield of the small airplane were calculated at different flow velocity angles for the purpose of building a surrogate model. This surrogate model was being included into 3-dimensional mechanic model for simulating separation from the airplane. Despite the success of the simulation study the project of automated OpenFOAM case management was doomed to fail.



When testing and evaluating CFD models we are also interested in

- modularity

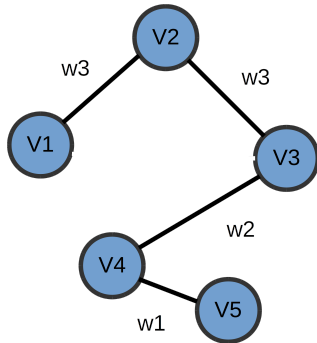
Figure: Directed Acyclic Graph



When testing and evaluating CFD models we are also interested in

- modularity
- extensibility

Figure: Directed Acyclic Graph



When testing and evaluating CFD models we are also interested in

- modularity
- extensibility
- flexibility

Figure: Directed Acyclic Graph

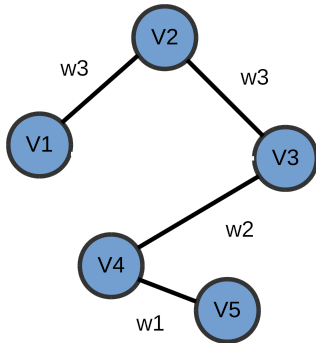


Figure: Directed Acyclic Graph

When testing and evaluating CFD models we are also interested in

- modularity
- extensibility
- flexibility
- user-friendliness

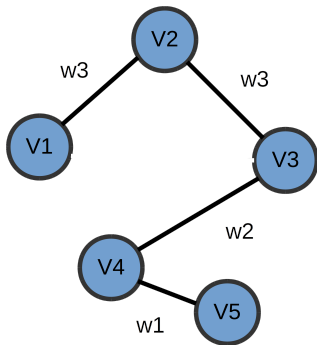


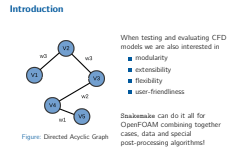
Figure: Directed Acyclic Graph

When testing and evaluating CFD models we are also interested in

- modularity
- extensibility
- flexibility
- user-friendliness

Snakemake can do it all for OpenFOAM combining together cases, data and special post-processing algorithms!

## └ Introduction

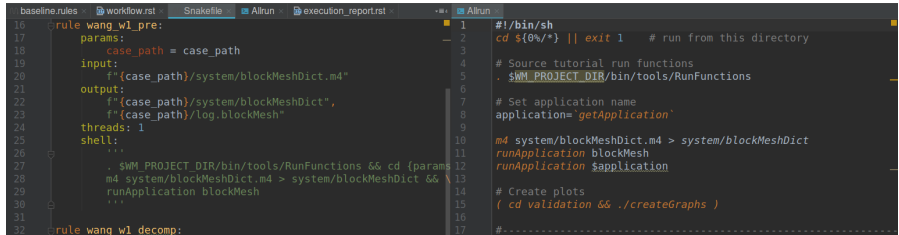


In this system the OpenFOAM cases should be represented as Directed Acyclic Graphs. It should provide at least several properties like

- modularity in terms that it is easy to develop additional functionality that can be plugged into the current system
- extensibility meaning it should be easy to add new cases and migrate the system to different environments
- flexibility as a property allowing us to build unlimited variants of the workflow and complex OpenFOAM simulations
- user-friendliness which means that at some point case-management system should be less complex than the regular Allrun script

The original project also involved Graphical User Interface. But there is a better solution. This is a meta-programming library called **Snakemake** which operates on DAG which helps to fulfill all bespoke requirements.

# An Advanced Allrun-script



```
baseline.rules x workflow.rst x Snakefile x Allrun x execution_report.rst x Allrun
16 rule wang_wl_pre:
17     params:
18         case_path = case_path
19     input:
20         f"{case_path}/system/blockMeshDict.m4"
21     output:
22         f"{case_path}/system/blockMeshDict",
23         f"{case_path}/log.blockMesh"
24     threads: 1
25     shell:
26         ...
27         . $WM_PROJECT_DIR/bin/tools/RunFunctions && cd {params
28         m4 system/blockMeshDict.m4 > system/blockMeshDict && \
29         runApplication blockMesh
30         ...
31 rule wang_wl_decomp:
32
1  #!/bin/sh
2  cd ${0%/*} || exit 1 # run from this directory
3
4  # Source tutorial run functions
5  . $WM_PROJECT_DIR/bin/tools/RunFunctions
6
7  # Set application name
8  application='getApplication'
9
10 m4 system/blockMeshDict.m4 > system/blockMeshDict
11 runApplication blockMesh
12 runApplication $application
13
14 # Create plots
15 ( cd validation && ./createGraphs )
16
17 #-----
```

Figure: Snakefile vs Allrun script

Snakefiles are building blocks of workflows.



## └─ An Advanced Allrun-script



Figure: Snakefile vs Allrun script

Snakefiles are building blocks of workflows.

From system engineering point of view we are building main workflow using bottom-to-top level approach. At some point the extension of the workflow happens only due to newly added case files with zero changes in the top aggregation level. Almost the same way like we add new libraries into OpenFOAM. Here is an example for the case-level workflow file on the left. It is quite verbose if you compare to regular Allrun-type script on the right. It happens at least because of additional specification of input and output files for specific preprocessing job. This job is actual node in the DAG and Snakemake checks if each previous job provides its outputs as inputs to the dependent job.

# Workflow Building Blocks

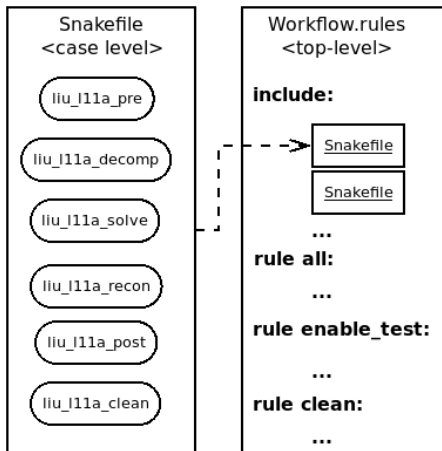


Figure: "Building blocks" form Main workflow

## Workflow Building Blocks

These case-level snakefiles are automatically combined into big workflow. Everything resembles a big C++ library building process.



Figure: "Building blocks" form Main workflow

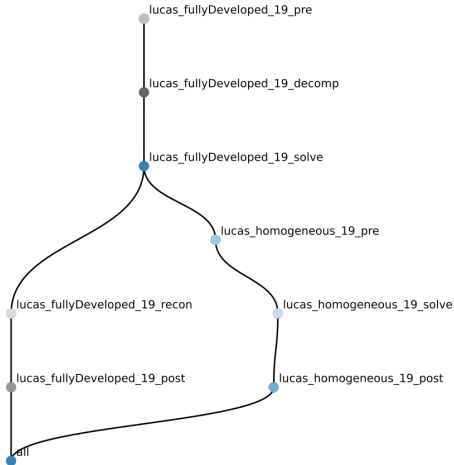


Figure: Multistage OpenFOAM case simulation as DAG

## └ OpenFOAM DAG



Figure: Multistage OpenFOAM case simulation as DAG

This slide illustrates:

- Fixed evaluation order
- Case dependency

The homogeneous cases use results of fullyDeveloped cases AND only start if previous stage is completed and it produced explicitly defined files. The figure illustrates only a small part of the main workflow. The main workflow could contain dozens of cases.

# Main Workflow

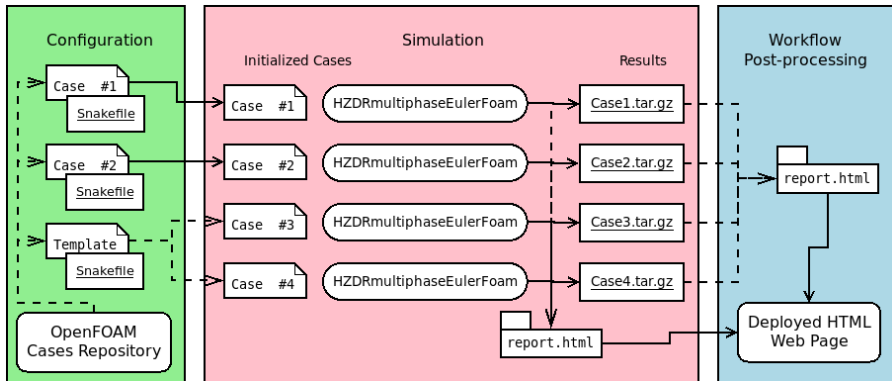


Figure: Main workflow stages

## └ Main Workflow

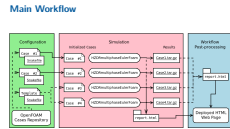


Figure: Main workflow stages

Because Snakemake operates on DAGs and DAGs are sort of predefined data structures, so the library must know the starting node or nodes and the all graph must have fixed end points without any loops because loops make execution time generally undetermined. For this reason just like with C++ library build example, we need a configuration step. On this step special algorithm looks into dedicated directory and finds recursively all case-level files. Today we also have at this step some templating functionality for Design Of Experiments kind of studies. On the next, simulation step, the workflow knows all Snakefiles it needs to include into itself. It includes them and runs the workflow. Reports also may be produced on this step since all report instructions are already included into case-level snakefiles. The separate post-processing workflow stage has specific procedures helping to analyze workflow at the scale

# Workflow vs Alltest

At first glance no difference to "Alltest-Allrun" approach.  
However:

- The workflow is native to Python not bash



At first glance no difference to "Alltest-Allrun" approach.  
However:

- The workflow is native to Python not bash
- The user defines explicit dependencies between execution stages

At first glance no difference to "Alltest-Allrun" approach.  
However:

- The workflow is native to Python not bash
- The user defines explicit dependencies between execution stages
- Case-level Snakefile may be standardized in the same degree as Allrun script

At first glance no difference to "Alltest-Allrun" approach.  
However:

- The workflow is native to Python not bash
- The user defines explicit dependencies between execution stages
- Case-level Snakefile may be standardized in the same degree as Allrun script
- Post-processing information may be specified at the lowest level and elevated to the top level report

At first glance no difference to "Alltest-Allrun" approach.  
However:

- The workflow is native to Python not bash
- The user defines explicit dependencies between execution stages
- Case-level Snakefile may be standardized in the same degree as Allrun script
- Post-processing information may be specified at the lowest level and elevated to the top level report
- Snakemake reports are actual selling point!

## └ Workflow vs Alltest

At first glance no difference to "Alltest-Allrun" approach. However:

- The workflow is native to Python not bash.
- The user defines explicit dependencies between execution stages
- Case-level Snakefile may be standardized in the same degree as Allrun script
- Post-processing information may be specified at the lowest level and elevated to the top level report
- Snakefile reports are actual selling point!

The old Allrun scripts are good and fit the purpose when we are testing OpenFOAM. However, what if we would like to develop new post-processing methods?

What if we would like to add case-dependency. Still doable in bash but it is not so easy.

The last, very important thing. How to easily control results of dozens of cases if we have not just test dichotomy completed/failed but a spectrum of situations where results may be both valid and not valid



Figure: An example of automatic Snakemake reports in OpenFOAM case context

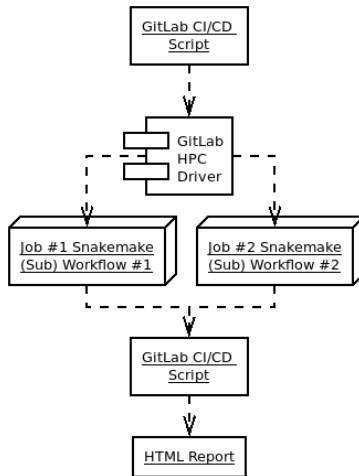


Figure: Workflow scaling through separating parts of the workflow

## └ Scalability



Figure: Workflow scaling through separating parts of the workflow

What happens when we need to simulate 56 cases? We run out of time. For that reason workflows that contain even small cases need to have a way to be scaled. As our experience shows, Snakemake library works just fine on clusters. There are various ways it could handle cluster jobs. Our approach may be not optimal but it works. Besides of other tasks we need to embed workflow into our Ci/Cd procedures running on the GitLab server. So in our case the main workflow is being split on four workflows. The code is identical but each part is controlled by its own configuration files. This is one more useful feature of Snakemake: workflow configuration via yaml files.



# An Example of Scaled Workflow



Figure: Main workflow stages but in the GitLab

# Scalable Workflows for OpenFOAM Evaluation

## └ An Example of Scaled Workflow



Figure: Main workflow stages but in the GitLab

At the end of the day the whole system has pretty appealing User Interface. Workflows can be started or interrupted, or restarted from GitLab web interfaces. Reports are also deployed automatically as gitlab pages so in some sense it also enables easy communication in teams when OpenFOAM results need to be discussed

- Workflows may work no worse than `bash` scripts from UI/UX point of view. Launching may be accomplished by a single command

- Workflows may work no worse than `bash` scripts from UI/UX point of view. Launching may be accomplished by a single command
- Workflows contain significant potential for changes of different kinds: various pre-processing strategies, advanced post-processing using Python scripts

- Workflows may work no worse than `bash` scripts from UI/UX point of view. Launching may be accomplished by a single command
- Workflows contain significant potential for changes of different kinds: various pre-processing strategies, advanced post-processing using Python scripts
- Automatic reports may be generated and aggregated for numerous cases in the single place

- Workflows may work no worse than `bash` scripts from UI/UX point of view. Launching may be accomplished by a single command
- Workflows contain significant potential for changes of different kinds: various pre-processing strategies, advanced post-processing using Python scripts
- Automatic reports may be generated and aggregated for numerous cases in the single place
- Workflows may be embedded into CI/CD pipelines