30th NENE 2021
September 6 - 9
BLED-SLOVENIA
International Conference Nuclear Energy for New Europe

GPU-based Accelerated Computation of Coalescence and Breakup Frequencies for Polydisperse Bubbly Flows

Gašper Petelin[1,2,3,4], **Ronald Lehnigk**[1]
gasper.petelin@ijs.si, r.lehnigk@hzdr.de

Jeffrey Kelling[2], **Gregor Papa**[3,4], **Fabian Schlegel**[1]
j.kelling@hzdr.de, gregor.papa@ijs.si, f.schlegel@hzdr.de

ABSTRACT

Polydisperse bubbly flows appear in many industrial applications and are particularly relevant for nuclear and process engineering. A popular approach for their simulation is the two-fluid model, which requires information about the bubble size. An associated challenge is to incorporate the effects of coalescence and breakup on the size distribution, which can be tracked by a population balance equation. It can be solved by the method of classes, which splits the bubble population into a finite number of size groups. However, this technique is expensive because the source term assembly involves the computation of coalescence and breakup frequencies between all bubble size pairs. This work focuses on improving the performance of the class method implementation within the OpenFOAM Foundation release (www.openfoam.org), an open source computational fluid dynamics software that allows for domain decomposition and parallel execution on multiple central processing units. Here, the parallel computation of the coalescence and breakup frequencies is shifted to graphics processing units using the Nvidia CUDA framework. Calculations are done asynchronously with overlapping data transfers, treating size pairs as independent units of work that are computed in parallel. The coalescence and breakup frequency computation on graphics processing units leads to a significant speedup compared to the pre-existing implementation. The improvement is demonstrated for a co-current two-phase flow in a vertical pipe. Both the source code and the case setup are made publicly available.

1 INTRODUCTION

Bubbly flows occurring in nuclear applications are frequently simulated using the two-fluid model of Ishii [1], whereby the interface separating the phases is not resolved. Instead, they are treated as interpenetrating continua, allowing for a much coarser spatial discretization. In turn, all interfacial transfers of mass, momentum and energy need to be modeled. These processes strongly depend on the bubble size. In a polydisperse multiphase flow, the bubble size is a distributed variable that changes due to coalescence and breakup as well as density variations

[1] Institute of Fluid Dynamics, Helmholtz-Zentrum Dresden - Rossendorf, Dresden, Germany
[2] Computational Science, Helmholtz-Zentrum Dresden - Rossendorf, Dresden, Germany
[3] Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
[4] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia

and phase transfers. Its evolution can be tracked through a population balance model, i.e. by solving a transport equation for the size distribution function. Herein, coalescence and breakup events between different bubble sizes are considered through integral source terms, through which the need for a special solution procedure arises. Besides the quadrature-based method of moments, which solves for integral properties of the distribution only [2], a popular approach is to split the distribution function into classes or groups, each representing a range of bubble volumes from v_i to v_{i+1} that are assigned a common representative volume x_i. A transport equation for the number concentration N_i is set up for each group. Using the formulations of Kumar and Ramkrishna [3] as well as Liao et al. [4] and focusing on coalescence and breakup only, it states

$$\begin{aligned}\frac{\partial N_i}{\partial t} + \nabla \cdot (\vec{u} N_i) = &\sum_{j=0}^{i} N_j \sum_{k=j}^{i} \left(1 - \frac{1}{2}\delta_{jk}\right) \eta_{x_j+x_k} N_k \tilde{C}_{jk} - N_i \sum_{j=0}^{M} N_j \tilde{C}_{ij} \\ &+ \frac{1}{2} \sum_{j=i}^{M} N_j \left(\tilde{B}_{ij} \Delta v_i + \sum_{k=0}^{j} \tilde{B}_{kj} \eta_{x_j - x_k} \Delta v_k \right) - \frac{1}{2} N_i \sum_{j=0}^{i} \tilde{B}_{ji} \Delta v_j \,, \end{aligned} \tag{1}$$

wherein $\vec{u}$ is the velocity vector, δ_{jk} is Kronecker's delta, $\tilde{C}_{ij}$ and $\tilde{B}_{ji}$ are the binary coalescence and breakup frequencies between groups i and j and $\Delta v_i = v_{i+1} - v_i$ is the section spacing. In case the bubble size resulting from a coalescence or breakup event doesn't fall onto a representative size, i.e. if Δv_i is not constant, the corresponding number gain is distributed by means of

$$\eta_v = \begin{cases} \dfrac{v - x_{i-1}}{x_i - x_{i-1}} & \text{if} \quad x_{i-1} \leq v \leq x_i \,, \\ \dfrac{x_{i+1} - v}{x_{i+1} - x_i} & \text{if} \quad x_i < v \leq x_{i+1} \,, \\ 0 & \text{else} \,, \end{cases} \tag{2}$$

which guarantees conservation of the total mass and number of bubbles. Details about the method of classes and its implementation in OpenFOAM[1] are presented by Lehnigk et al. [5].

While the method of classes has the benefit of providing direct information about the size distribution, which is useful for a comparison against experiments, the associated computational cost strongly depends on the number of bubble size group pairs, which scales non-linearly with the total number of size groups. Lehnigk et al. [5] have shown that the costliest step during the simulation is the computation of the pairwise coalescence and breakup frequencies. Depending on the selected models and the size of the computational mesh, a significant amount of field algebra is involved. One way to improve performance is parallelization. While OpenFOAM supports the use of multiple central processing units (CPUs) with a parallel computing architecture named Message Passing Interface (MPI), this architecture may not always parallelize the problem in the most optimal way and is not designed to use different types of hardware resources besides CPUs. CPUs are generally designed to handle a wide range of sequential tasks and are therefore limited when it comes to concurrency and parallelization.

In recent years graphics processing units (GPUs) have emerged as platforms for general-purpose parallel computing that can significantly accelerate certain computational tasks. Compared to CPUs, GPUs are built with a large number of cores and support fast memory transfer to achieve floating-point throughputs that can be orders of magnitude larger compared to CPUs,

[1] `https://openfoam.org`

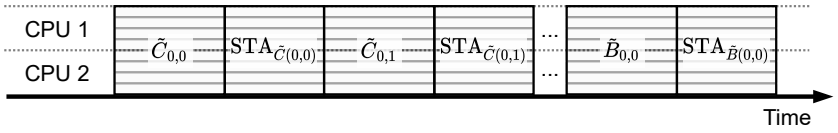

Figure 1: Resource use over time with MPI distributing work over two CPU cores with four control volumes each. Frequencies of coalescence $\tilde{C}_{ij}$ and breakup $\tilde{B}_{ij}$ pairs are computed sequentially and passed to the source term assembly (STA). Gray subdivisions show the mesh, decomposed over multiple CPUs with each gray rectangle representing an individual control volume.

traditionally used for CFD simulations. Due to their highly parallel design, it is becoming more common to shift some of the workload to the GPUs if possible. Unfortunately, due to their design, programming GPUs is usually more challenging compared to CPU programming and requires more detailed knowledge of the problem and hardware. To simplify interactions between code and hardware, the parallel computing platform Compute Unified Device Architecture (CUDA) is used. CUDA [6] is Nvidia's platform based on C/C++ that simplifies the management of GPUs and resources associated with them.

In this work, we explore the strategy of redefining the computation of coalescence and breakup frequencies in such a way to be more suitable for GPU execution. Due to GPUs being designed with concurrency in mind, they can significantly accelerate CFD simulations that include costly source term assemblies as in the present case.

2 PARALLELIZATION STRATEGY

As noted above, the computational cost for obtaining the coalescence and breakup frequencies can be very large for simulations with fine meshes and many bubble size pairs. Fortunately, the coalescence and breakup frequency computation only involves vectorized floating-point operations on existing scalar or tensor fields. Furthermore, there is no need for communication between cells or mesh regions and the frequencies for all pairs are independent of each other, making the problem easy to parallelize when additional computational resources are available. Modern CPUs are usually built with multiple cores, where each core supports one or more threads allowing for some degree of parallel processing. OpenFOAM already natively supports MPI, thus allowing simulations to use multiple CPU cores which is required for large meshes. The drawback of the OpenFOAM parallelization is its inability to use other hardware resources like GPUs and the communication inefficiency due to the mesh being split over multiple CPU cores. A specific drawback of the population balance model implementation is the sequential computation of the frequencies for all size group pairs. Figure 1 shows the CPU-based frequency computation and the resource usage over time on a CPU with 2 cores for a mesh with 8 control volumes. The mesh is first decomposed into separate regions, each of which is assigned to a separate CPU core that computes the corresponding frequencies. Once the frequency for the first pair is computed, the result is used to assemble the corresponding source term contribution in Eq. (1). This procedure is sequentially repeated for all of the subsequent coalescence and breakup pairs until all source terms are assembled.

The use of GPUs for computing the frequencies allows for a different approach. In general, GPU programming requires manual data transfers [7] between CPU and GPU as well as decomposing the problem in a different way that is more suitable for the GPU architecture. With GPU parallelization, tasks can be divided into four distinct steps. The corresponding resource usage is shown in figure 2. At first, GPU memory is allocated and fields like the turbulent dis-

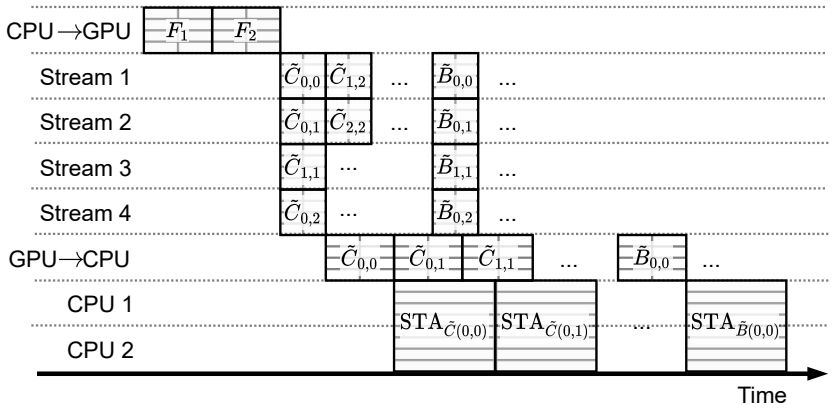

Figure 2: Resource usage over time with two CPU cores and an additional GPU with four streams for a mesh with eight control volumes. Fields (F_1 and F_2) that are necessary for computation are first transferred to the GPU memory (CPU→GPU). Once fields are transferred, the parallel computation of the individual kernels ($\tilde{C}_{0,0}$, $\tilde{C}_{0,1}$, $\tilde{C}_{1,1}$, $\tilde{C}_{0,2}$) starts, organized into several streams. Computed coalescence and breakup frequencies are then transferred back to the CPU (GPU→CPU) and passed to the source term assembly ($\text{STA}_{\tilde{C}}$, $\text{STA}_{\tilde{B}}$).Gray subdivisions show the mesh, decomposed over multiple CPUs with each gray rectangle representing an individual control volume.

sipation or shear strain rate, which are necessary for the computation, are transferred from the host memory to the GPU memory. In the next step, the pairwise frequency computations are organized into individual computational units of work called CUDA kernels. Once created, kernels are sent to the GPU and scheduled for computation in CUDA streams. Each stream acts as a queue containing multiple kernels and memory transfers waiting to be executed in sequence. CUDA streams execute kernels scheduled on it one by one, where the first kernel put into the stream is also executed first. Note that in reality GPUs can have hundreds of streams, all of them concurrently calculating frequencies and thus achieving larger throughput. When a kernel finishes execution, the obtained frequency for the pair is written to the GPU memory. The last steps are the transfer of the values back to the CPU memory and their consumption during the source term assembly. With this kind of execution, transferring data between GPU and CPU memory can become the bottleneck. For that reason the kernel execution and the data transfer back to the CPU is interleaved to reduce the waiting time and to more efficiently utilize the limited transfer speed. This allows frequencies of individual pairs to be transferred back to the CPU immediately after they are computed, with no need to wait for the results of other pairs.

3 DEMONSTRATION CASE

For demonstration purposes, an isothermal co-current flow of air and water in a vertical pipe is simulated. The case originates from the experiment of Lucas et al. [8], who used wire-mesh sensor technology to measure the void distribution as well as bubble velocities and size distributions. The relevant test section is a pipe with a length of $3\,\mathrm{m}$ and an inner diameter of $51.2\,\mathrm{mm}$. Air is injected through a set of needles distributed uniformly over the pipe cross-section. For the case considered here, the air and water superficial velocities under normal temperature and pressure conditions are $0.219\,\mathrm{m\,s^{-1}}$ and $1.611\,\mathrm{m\,s^{-1}}$. This combination leads to a transition from bubbly to near-slug flow conditions. Resulting from shear-induced lift, the bubbles initially migrate towards the wall. As coalescence and isothermal growth outweigh breakup of the bubbles, they gradually move towards the pipe center. In the simulation,

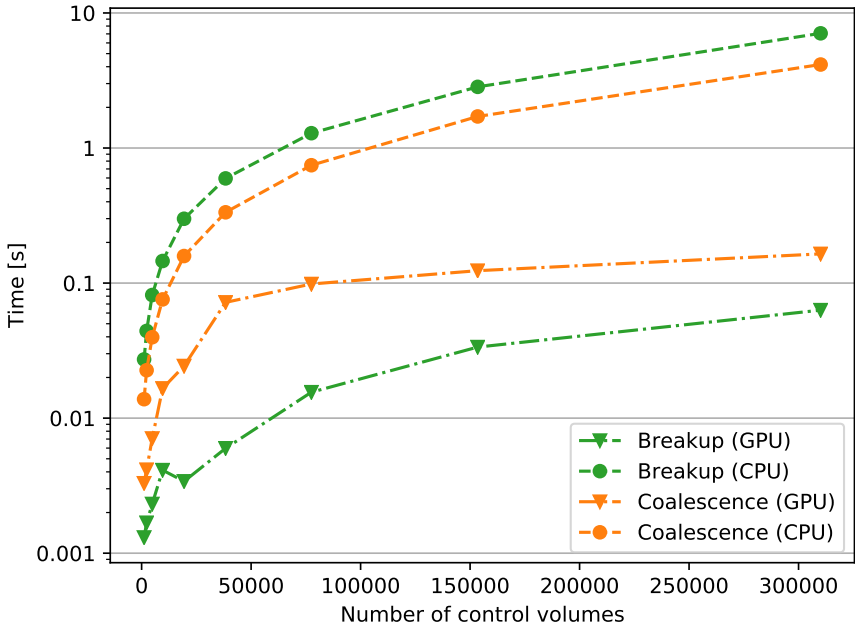

Figure 3: Performance of the coalescence and breakup frequency computation on a single GPU versus using a CPU with four cores as a function of the number of control volumes. Note the time axis being in logarithmic scale.

the interfacial transfer of momentum due to drag and non-drag forces as well as shear- and bubble-induced turbulence are modeled according to Liao et al. [9]. Coalescence and breakup frequencies are computed using the models of Lehr et al. [10]. In the base case, the size distribution is approximated using 18 size groups with a spacing of $1\,\mathrm{mm}$ in terms of the bubble diameter. The computational mesh consists of a total of 19380 control volumes. Further information about the case and a discussion of the simulation results is available in the article of Lehnigk et al. [5]. The actual case setup can be retrieved from Haensch et al. [11].

4 RESULTS

Simulations were performed on a machine with an Intel i5-8400 (2.80GHz), 16 GB of RAM and a GeForce RTX 2060 GPU (1920 CUDA cores, 1680 MHz processor clock, 6 GB memory). To mitigate the impact of object construction and memory allocation happening at the beginning, the simulation was run for a total of 50 time steps. All figures below show the average computational time for a single time step. Figures 3 and 4 compare the performance of the frequency computation on a single GPU versus a CPU with four cores as a function of the number of control volumes and size groups. Note that these figures compare just the GPU-accelerated parts. The benefit of computing the frequencies on GPUs is highly dependent on the complexity of the coalescence and breakup models, size of the mesh, number of size groups and hardware. In general, GPU computations have some overhead compared to CPU execution. This overhead is primarily due to limited data transfer speed and some delay when starting kernels. However, even with these limitations, using GPUs can result in a significant performance improvement compared to just utilizing CPU parallelization. With the demonstration case, we achieve a 10 times speedup in the computational time needed for calculating the coalescence and breakup frequencies. In general, the approach benefits from larger meshes and more size groups where a CPU-based computation might be too computationally expensive to evaluate.

Due to OpenFOAM natively supporting distributed computing, an arbitrary number of

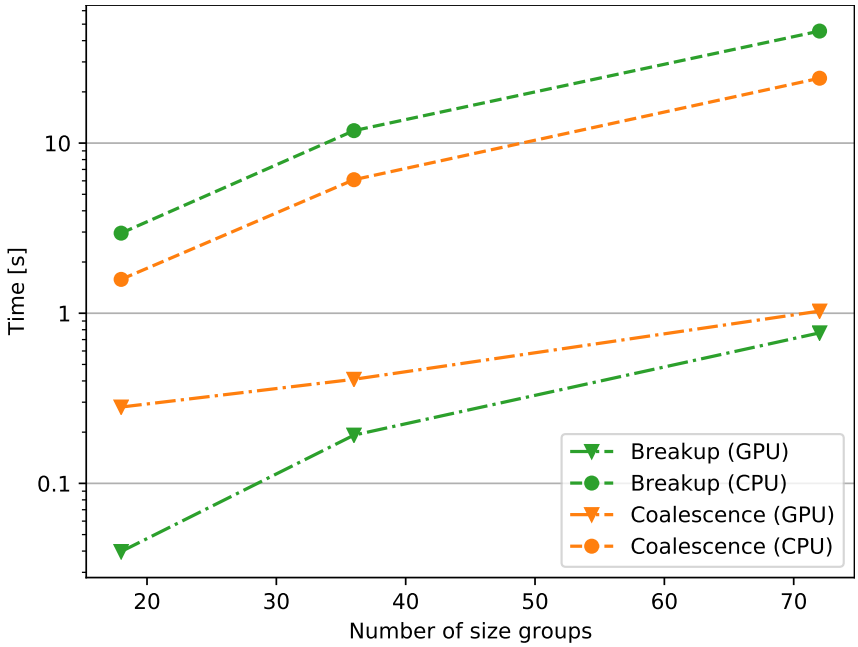

Figure 4: Performance of the coalescence and breakup frequency computation on a single GPU versus using a CPU with four cores as a function of the number of size groups. Note the time axis being in logarithmic scale.

CPUs can be used for running a simulation. Figure 5 shows the execution time of one time step for a different number of CPU cores, with and without an additional GPU. As expected there is some benefit with adding additional cores, but it often happens that we experience diminishing returns where increasing the number of CPU cores does not improve the performance or may even degrade it. Doubling the number of cores from one to two almost halves the run-time of the case making the parallelization close to ideal. In contrast to this, adding the 5th and 6th core when 4 are already in use does not make the simulation any faster. A limit exists where additional computational CPU resources will not improve the execution time. The main benefit of the GPU-based computation is that it additionally lowers this limit and reduces the simulation run time beyond the limit of what is possible with just adding additional CPU resources. In such cases GPU-based computation can offer a great benefit with more efficient use of available resources and thus decreasing the total execution time of a simulation.

5 CONCLUSIONS

This work demonstrates one possible strategy of improving computational time of simulations using OpenFOAM's population balance framework by changing parallelization strategy and shifting coalescence and breakup frequency computation to the GPU. With this approach, a substantial increase in performance can be obtained. Shifting work to the GPU is especially useful with larger meshes and a large number of size groups, where inefficiencies stemming from managing CUDA kernels and transferring memory between CPU and GPU become almost insignificant. The source code developed in this work is publicly available from Schlegel et al. [12].

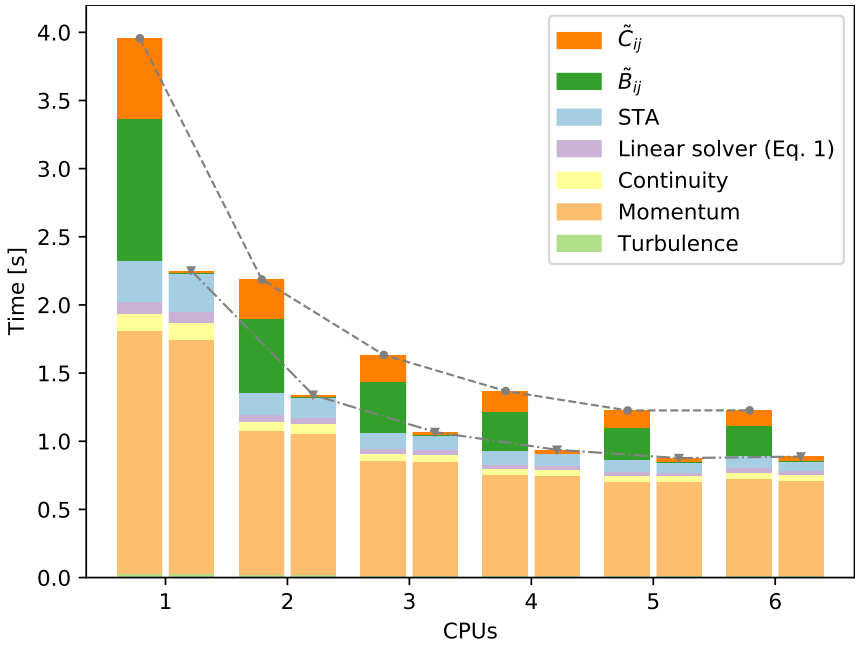

Figure 5: Overall performance with CPU-based and GPU-based coalescence and breakup frequency computation as a function of CPU cores, including contributions from solving the turbulence, momentum and continuity equations as well as assembling the source terms for Eq. (1) for every size group and solving the resulting equations with a linear solver. Left columns show performance without GPU while right columns show performance with additional GPU.

ACKNOWLEDGMENTS

This work was supported by the Helmholtz European Partnering Program in the project Crossing borders and scales (Crossing) and Slovenian Research Agency (research core funding No. P2-0098).

REFERENCES

[1] M. Ishii. Thermo-fluid dynamic theory of two-phase flow. *NASA Sti/Recon Technical Report A*, 75:29657, 1975.

[2] D. L. Marchisio, R. D. Vigil, and R. O. Fox. Implementation of the quadrature method of moments in CFD codes for aggregation–breakage problems. *Chemical Engineering Science*, 58(15):3337–3351, 2003.

[3] S. Kumar and D. Ramkrishna. On the solution of population balance equations by discretization-I. A fixed pivot technique. *Chemical Engineering Science*, 51(8):1311–1332, 1996.

[4] Y. Liao, R. Oertel, S. Kriebitzsch, F. Schlegel, and D. Lucas. A discrete population balance equation for binary breakage. *International Journal for Numerical Methods in Fluids*, 87(4):202–215, 2018.

[5] R. Lehnigk, W. Bainbridge, Y. Liao, D. Lucas, T. Niemi, J. Peltola, and F. Schlegel. An open-source population balance modeling framework for the simulation of polydisperse multiphase flows. *AIChE Journal*, 2021. Accepted for publication.

[6] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008.

[7] Raphael Landaverde, Tiansheng Zhang, Ayse K Coskun, and Martin Herbordt. An investigation of unified memory access performance in cuda. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2014.

[8] D. Lucas, E. Krepper, and H-M Prasser. Development of co-current air-water flow in a vertical pipe. *International Journal of Multiphase Flow*, 31(12):1304–1328, 2005.

[9] Y. Liao, T. Ma, E. Krepper, D. Lucas, and J. Fröhlich. Application of a novel model for bubble-induced turbulence to bubbly flows in containers and vertical pipes. *Chemical Engineering Science*, 202:55–69, 2019.

[10] F. Lehr, M. Millies, and D. Mewes. Bubble-size distributions and flow fields in bubble-columns. *AIChE Journal*, 48(11):2426–2443, 2002.

[11] Susann Hänsch, Ilya Evdokimov, Ronald Lehnigk, Yixiang Liao, Richard Meller, and Fabian Schlegel. HZDR Multiphase Case Collection for OpenFOAM (Version 2.0.0). Rodare, 2021. https://doi.org/10.14278/rodare.1049.

[12] Fabian Schlegel, Mazen Draw, Ilya Evdokimov, Susann Hänsch, Harris Khan, Ronald Lehnigk, Jiadong Li, Hongmei Lyu, Richard Meller, Gašper Petelin, and Matej Tekavčič. HZDR Multiphase Addon for OpenFOAM (Version 2.1.1). Rodare, 2021. http://doi.org/10.14278/rodare.1133.