Project Title	SCAlable LAttice Boltzmann Leaps to Exascale
Project Acronym	SCALABLE
Grant Agreement No.	956000
Start Date of Project	01.01.2021
Duration of Project	36 Months
Project Website	www.scalable-hpc.eu

D5.2

Generated kernels for boundary handling and communication routines

Work Package	WP 5: Improving LBM solver versatility and advanced Extreme Scale SW technologies
Lead Author	Philipp Suffa (FAU)
Contributing Authors	Philipp Suffa, Markus Holzer, Prof. Ulrich Rüde, Prof. Harald Köstler
Reviewed By	
Due Date	December 31, 2022
Date	
Version	0.1

Dissemination Level

☒ PU: Public

☐ PP: Restricted to other programme participants (including the Commission)

☐ RE: Restricted to a group specified by the consortium (including the Commission)

☐ CO: Confidential, only for members of the consortium (including the Commission)

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Deliverable Information

Deliverable	D5.2
Deliverable Type	Report
Deliverable Title	Generated kernels for boundary handling and communication routines
Keywords	
Dissemination Level	Public
Work Package	WP 5: Improving LBM solver versatility and advanced Extreme Scale SW technologies
Lead Partner	FAU
Lead Author	Philipp Suffa
Contributing Authors	Philipp Suffa, Markus Holzer, Prof. Ulrich Rüde, Prof. Harald Köstler
Reviewed By	
Due Date	December 31, 2022
Planned Date	
Version	0.1
Final Version Date	

Generated kernels for boundary handling and communication routines

Philipp Suffa, Markus Holzer, Prof. Ulrich Rüde, Prof. Harald Köstler

January 18, 2023

Contents

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Chapter 1 Introduction

Task 5.1 [6] of the SCALABLE project was to extend the code generation pipeline of *lbmpy* [2] to support the generation of sparse data storage LBM kernels. This goal was successfully reached and demonstrated in deliverable 5.1 [6]. To support a full CFD application such as the simulation of a landing-gear of an air plane (LAGOON project [8]), the code generation pipeline has to be further enhanced to also support the code generation of boundary and communication kernels as shown in Figure 1.1. This is the content of the work package and deliverable 5.2, here.

In chapter 2 the generation of the boundary handling kernels is described, while in chapter 3 the focus is on the generation of the communication of sparse data. In chapter 4 the LAGOON test case running with generated sparse data storage kernels is evaluated.

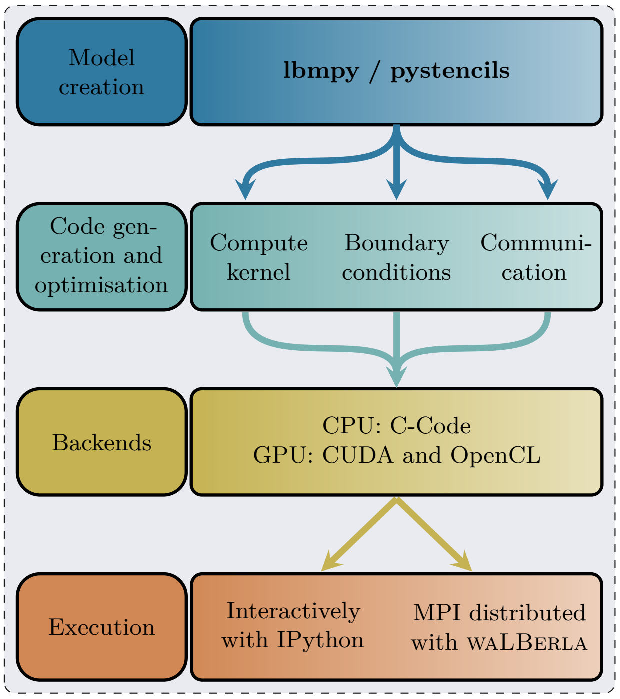


Figure 1.1: Complete workflow of combining *lbmpy* and WALBERLA for MPI parallel execution. It can be seen, that for a full CFD application a compute kernel as well as a boundary and communication kernel needs to be generated.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Chapter 2 Code generation for boundary handling of sparse data storage

2.1 List adaptations for boundary handling

To implement sparse data storage in waLBerla [3], two data structures are needed: One array storing all PDF values in a linear fashion. These PDFs are usually sorted per direction. Thus one can think of a linear array containing all PDFs in the center followed by a linear array containing all PDFs pointing north and so on. This is a so called Structure of Array format and was previously shown to be more usually efficient than other storage formats [9]. The other data structure needed is a list, called the index list, to store the neighbouring information of the cell. More precisely, the index list stores a pointer for every PDF of a cell to the propagation destination of the propagation direction with respect to the streaming pattern. So as it is shown in Figure 2.1 (assuming pull streaming pattern) cell 6 stores cell 1 as pull pointer in the northern direction and cell 11 as pull pointer in the direction south. For more information about the list generation see [9].

To extend the generated sparse data LBM kernels with boundary handling, the list has to be modified in different ways for different boundary conditions.

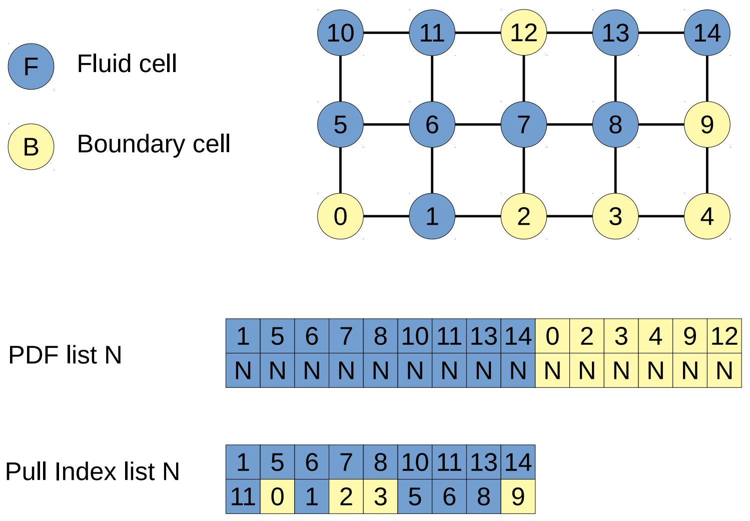
Interestingly, no-slip and periodic boundary conditions can directly be encoded in the index array. No-slip conditions, which consist of a simple bounce-back scheme, can be resolved by storing the indices such that the direction flip of the PDF values happens directly inside the cell. The same holds for periodic boundary conditions. There, the index list entry of the cell at the boundary just has to be set to the cell at the other side of the domain. Thus, both cases do not introduce any additional calculations or an extension to the index or PDF list and come for free from a computational point of view.

For other boundary conditions such as velocity bounce back (UBB) or pressure boundaries, the PDF list as well as the index list has to be extended. To fill the PDF and index list, a flag field containing the whole domain is used. To implement a velocity bounce-back boundary for sparse data storage, all cells, which are marked as UBB in the flag field, are appended to the PDF list. Further, the index list is modified, such that the indices of a cell, which is in the neighborhood of the UBB boundary, are adjusted to point to the boundary cell.

So let's assume, a pull propagation scheme and a D2Q9 stencil is used as shown in Figure 2.1: If the cell with index 0 is set to a velocity bounce back boundary, the cell is first registered to the PDF list. Next, the cells in the direct neighborhood of the UBB boundary cell 0 are collected, and the pull indices of the directions pulling from the boundary cell are adjusted. In Figure 2.1 the pull indices of direction N of cell 5 have to be adjusted to point to the northern PDF of cell 0. The same procedure is then also done for other directions.

Other boundaries such as the pressure boundary work according to the UBB boundary.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.



PDF list N

1	5	6	7	8	10	11	13	14	0	2	3	4	9	12
N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

Pull Index list N

1	5	6	7	8	10	11	13	14
11	0	1	2	3	5	6	8	9

Figure 2.1: Illustration of the PDF list with fluid cell (blue) and the appended boundary cells (yellow). Here only one direction of the PDF is displayed (north), to simplify the illustration. The index list here is generated for the pull pattern. Again, only the northern pull indices are illustrated, so for example cell 5 has to pull its northern PDF from boundary cell 0.

2.2 Generated boundary kernels

As already mentioned in section 2.1, the no-slip, as well as the periodic boundaries, are a special case for sparse data storage, because the boundaries can be realised by adjusting the index list in the right way. So to realise no-slip boundaries, if there is a no-slip boundary in direction d, the pull index of the PDF in direction d is just set to the inverse direction of d of the same cell. For periodic boundaries, the cell on the other side of the domain is just used for the pull index. This means, that there is no need for an explicit boundary kernel for boundaries no-slip and periodic. It is automatically integrated into the normal LBM kernel through the modified index list.

This is not the case for other boundary conditions such as velocity bounce back or pressure boundaries. Here an explicit kernel has to be generated. For this kernel, the third list in addition to the PDF and the index list is needed, which is called the ”boundary index list”. This list is also generated from the flag field and it consists of structures with three elements, namely the boundary cell index, the neighbouring fluid cell index, and the direction of the neighbouring. So in this boundary index list, all relevant directions of all boundary cells (excluding no-slip and periodic) and the neighbouring fluid cells in these directions are registered. This means, that only directions are stored, which are streaming in a neighbouring fluid cell.

The boundary kernel then iterates over this boundary index list, and thus updates the PDFs of all boundary cells with respect to the neighbouring fluid cells according to the boundary rule such as velocity bounce back or pressure.

As already shown in deliverable 5.1, these kernels can be generated for CPU as well as for GPU accelerators.

The Scalable project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 956000.

Chapter 3 Code generation for communication of sparse data storage

3.1 List adaptations for communication handling

To run in parallel, the framework waLBerla is based on a domain partitioning consisting of blocks. The data structures for the sparse kernels such as the PDF list or the pull index list are stored locally per block. To run a sparse data storage application on more then one block, data has to be transferred between blocks. Therefore, communication between blocks has to be integrated in the code generation pipeline for sparse kernels.

To achieve communication between blocks, the PDF list has to be modified again. This time, the cells behind an MPI interface, which are called ghost layers, are appended to the PDF list, which already contains the fluid cells of the current block and maybe also some boundary cells. Again, the index list of fluid cells next to the MPI interface is adjusted, so that the relevant directions pull from the ghost layer cells, as it is shown in Figure 3.1.

Furthermore, for every communication direction of a block, a pair of pointers pointing to the start of the ghost layer cells of this direction and the number of PDFs, which are sent to the neighbour, is stored (see Figure 3.1).

3.2 Packing / unpacking kernels

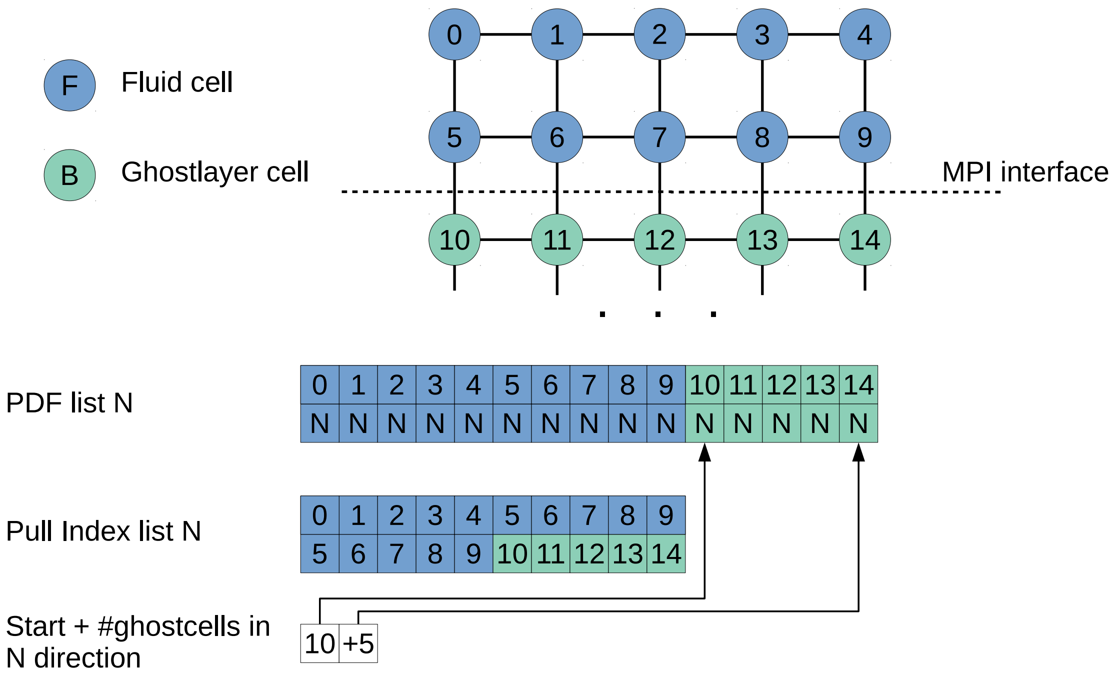
In the actual packing kernel, which is called with a specific direction d, all PDFs that are sent to the neighbour block in d are collected, with the help of the pair consisting of the starting pointer to the PDF list for d and the number of PDFs to be sent. The collected PDFs are packed in a buffer, which is then handed over to the standard waLBerla communication routines. These communication routines are again available for a CPU backend as well as for a GPU backend.

The unpacking kernel then receives a buffer from the waLBerla communication routine. Again the position in the PDF list is determined using the pointer for a specific direction d and the number of ghost layer cells in direction d. Then the received buffer is just written to the determined position on the PDF list, and the communication is done.

Because the packing and unpacking kernel look exactly the same for various stencils as well as for various collision methods, these kernels do not need to be generated at all. The whole logic is included in the PDF and index list as well as in the created pointers to determine the right positions of the buffers in the PDF list.

It has to be said, that mesh refinement for sparse data storage is not implemented yet, which would make the communication between blocks much more complex.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
N	N	N	N	N	N	N	N	N	N	N	N	N	N	N

0	1	2	3	4	5	6	7	8	9
5	6	7	8	9	10	11	12	13	14

Figure 3.1: Illustration of the PDF list with fluid cells (blue) and ghost layer cells (green), which are cells from another MPI process, separated by a MPI interface. Ghost layer cells are appended to the PDF list and the index list is modified, so that for example the northern pull index of cell 6 points to the northern PDF of ghost layer cell 11. Also the pointer to the start of the ghost layer cells and the number of cells is stored, which is used by the packing / unpacking kernels to find the right position in the PDF list. To simplify the illustration, only PDFs in direction north are shown in the lists, of course other directions have to be stored for the PDF list, the index list and the start pointer as well.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Chapter

4 Results

To test the implemented generation pipeline for a full CFD simulation, the LAGOON (LAnding-Gear nOise database for CAA validatiON) test case [7] is simulated. The test case consists of a velocity-bounce-back boundary in the west and a pressure boundary at the outlet in the east.

To damp spurious wave generation at the outlet a sponge region is established:

$$\tilde{\phi}\,(t+\Delta t) = \phi\,(t+\Delta t) - \sigma_{\text{sponge}}(z)\left[\phi\,(t+\Delta t) - \phi_{\text{target}}\right]. \qquad (4.1)$$

Here ϕ is the order parameter to which the sponge region is applied to [4]. For the sake of simplicity, it is applied to the relaxation rate ω in the scope of this deliverable. The length of the sponge region is chosen to be $L_{\text{sponge}} = 640$ in lattice units and $\phi_{\text{target}} = 1.9$.

The rest of the domain boundaries are set to periodic boundaries, except the landing gear geometry, which is set to no-slip conditions. Therefore, four different generated boundary conditions for sparse data storage are tested for their functionality. To verify the correctness of the generated communication kernels, the simulation is run with 30 blocks is x-direction, 6 blocks in y-direction, and 4 blocks in z-direction, where each block consists of 64^3 cells. This results in a total cell count of 188,743,680 and a cell size of $9.0mm$.

The velocity of the inflow boundary is set to 78.99 m/s and the reference length for the Reynolds number is the wheel diameter of $0.3m$. Therefore, the Reynolds number is $1.59 \cdot 10^6$. The goal was to simulate 0.3 seconds of the flow around the landing gear with a time step size of $5.6969 \cdot 10^{-6}$, which results in 52,659 time steps to run. The relaxation rate for the Lattice Boltzmann method is then $\omega = 1.9999874$. As collision model the regularised cumulant operator is used [5].

The simulation run on JUWELS Cluster [1] on 15 nodes for 2 hours and 16 minutes. The performance of the generated code is 1.80 mega lattice cell updates per second (MLUPS) per core, so the total performance on 15 nodes is 1297.15 MLUPS. A full list of all simulation parameter is shown in Table 4.1. The outcome of the simulation can be seen in Figure 4.1, where the Q-criterion of the velocity is displayed. Further, the domain partitioning is indicated by the grey lines, where one block is one cube.

Number of blocks	30 x 6 x 4
Number of cells per block	64 x 64 x 64
Cell size	9.0 mm
Total number of cells	188,743,680
Number of cores	720
Performance per core	1.80 MLUPS
Total performance	1297.15 MLUPS
Simulated time	3 s
Time steps	52,659
Runtime	136 min
Reynolds number	$1.59 \cdot 10^6$
Relaxation rate ω	1.9999874
Inflow velocity	78.99 m/s

Table 4.1: Summary of all relevant simulation parameters for LAGOON test case on JUWELS Cluster.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

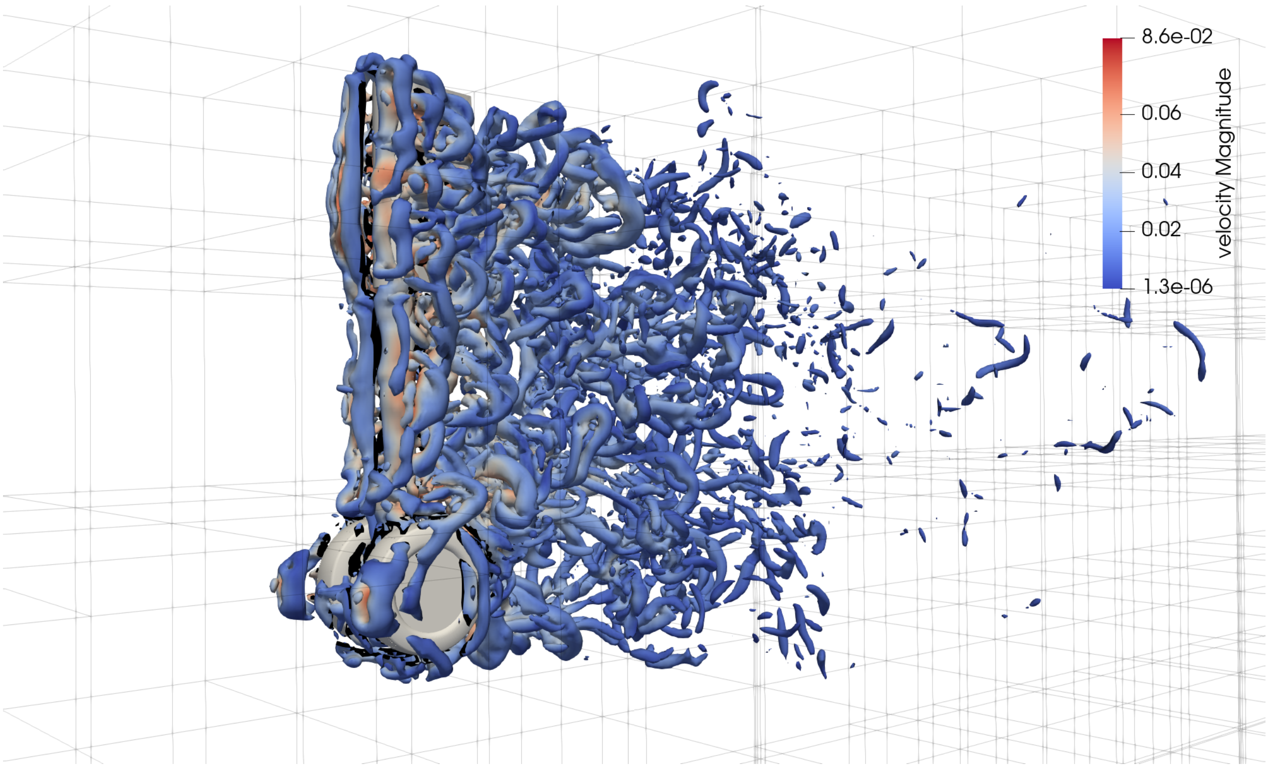


Figure 4.1: LAGOON test case visualised by the q-criterion. The grey lines are indicating the domain partitioning into multiple blocks.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Chapter 5 Conclusion

The goal of task 5.2 was to support a full CFD simulation. Therefore, the effort of task 5.1 [6] was continued by extending the *lbmpy* generation pipeline with sparse boundary kernels as well as sparse communication handling. As it can be seen by the simulation of the LAGOON landing gear configuration in chapter 4, this goal was well achieved. Because of the code generation, the generated kernels can run on CPUs as well as on GPUs. Furthermore, the flexibility of the approach is demonstrated by the different boundary conditions and by the inherent possibility to employ the advanced cumulant collision operator. Due to the functionality of the code generator, it is not necessary to design and implement each of the necessary kernels individually and with time-consuming and error-prone manual effort. Instead implementing one kind of boundary condition results in immediate support of all boundary conditions with similar data access patterns. Thus, for example, by supporting velocity boundary conditions, pressure boundaries are available immediately without further effort.

We further emphasise that it is now possible to use waLBerla to compute an example configuration with an industrially-relevant number of cells. Though we have not yet put emphasis on optimising the generated compute kernels in this first demonstration, this already meshes with almost 200 million cells, running on more than 700 cores. Thus this simulation size is already larger than the references data of [8].

Thus the results reported here in D5.2 can set a reference point for the code optimizations in WP3. We further point out that our compute kernels will support the usage of heterogeneous computing in terms of NVIDIA and AMD GPUs. Performance analysis and improvements of these are relevant for and are closely connected to work in WP4 and WP2. Additionally, we expect that a systematic analysis of the energy consumption between classical directly addressed compute kernels and indirect addressing kernels as presented in D5.2 can lead to interesting findings for WP6.

The next steps, planned for work package 5.3 will be optimizations of the generated kernels. This includes in-place streaming and SIMD vectorization on the CPUs. These will be implemented and benchmarked next.

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

List of Figures

The Scalable project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 956000.

Bibliography

[1] Damian Alvarez. “JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre.” In: *Journal of large-scale research facilities JLSRF* 7 (Oct. 2021). DOI: 10.17815/jlsrf-7-183.

[2] Martin Bauer, Harald Köstler, and Ulrich Rüde. “lbmpy: Automatic code generation for efficient parallel lattice Boltzmann methods.” In: *Journal of Computational Science* 49 (2021), p. 101269. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2020.101269.

[3] Martin Bauer et al. “waLBerla: A block-structured high-performance framework for multiphysics simulations.” In: 2020. DOI: 10.1016/j.camwa.2020.01.007.

[4] Yongliang Feng et al. “ProLB: A Lattice Boltzmann Solver of Large-Eddy Simulation for Atmospheric Boundary Layer Flows.” In: *Journal of Advances in Modeling Earth Systems* 13.3 (2021), e2020MS002107. DOI: https://doi.org/10.1029/2020MS002107.

[5] Martin Geier et al. “The cumulant lattice Boltzmann equation in three dimensions: Theory and validation.” In: *Computers & Mathematics with Applications* (2015). DOI: 10.1016/j.camwa.2015.05.001.

[6] Markus Holzer et al. *Deliverable 5.1: Code generation for sparse data storage LBM kernels*. https://scalable-hpc.eu/wp-content/uploads/2022/08/SCALABLE_D5.1_Basic-generated-sparse-data-storage-LBM-kernels.pdf. 2021 (accessed December 7, 2014).

[7] Eric Manoha, Jean Bulté, and Bastien Caruelle. “Lagoon : An Experimental Database for the Validation of CFD/CAA Methods for Landing Gear Noise Prediction.” In: May 2008. ISBN: 978-1-60086-983-9. DOI: 10.2514/6.2008-2816.

[8] Alois Sengissen et al. “Simulations of LAGOON landing-gear noise using Lattice Boltzmann Solver.” In: (). DOI: 10.2514/6.2015-2993.

[9] Markus Wittmann. “Hardware-effiziente, hochparallele Implementierungen von Lattice-Boltzmann-Verfahren für komplexe Geometrien.” doctoralthesis. Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2016.

The Scalable project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement number 956000.