Implementation of a lattice Boltzmann method on GPU based HPC Systems

M. Waldmann[1], G. Brito-Gadeschi[2], M. Gondrum[1], M. Meinke[1], W. Schröder[1]

[1]Institute of Aerodynamics and Chair of Fluid Mechanics, RWTH Aachen University, Aachen, Germany

[2]NVIDIA GmbH, München, Germany

Motivation

Advantages of porting scientific codes to GPUs:

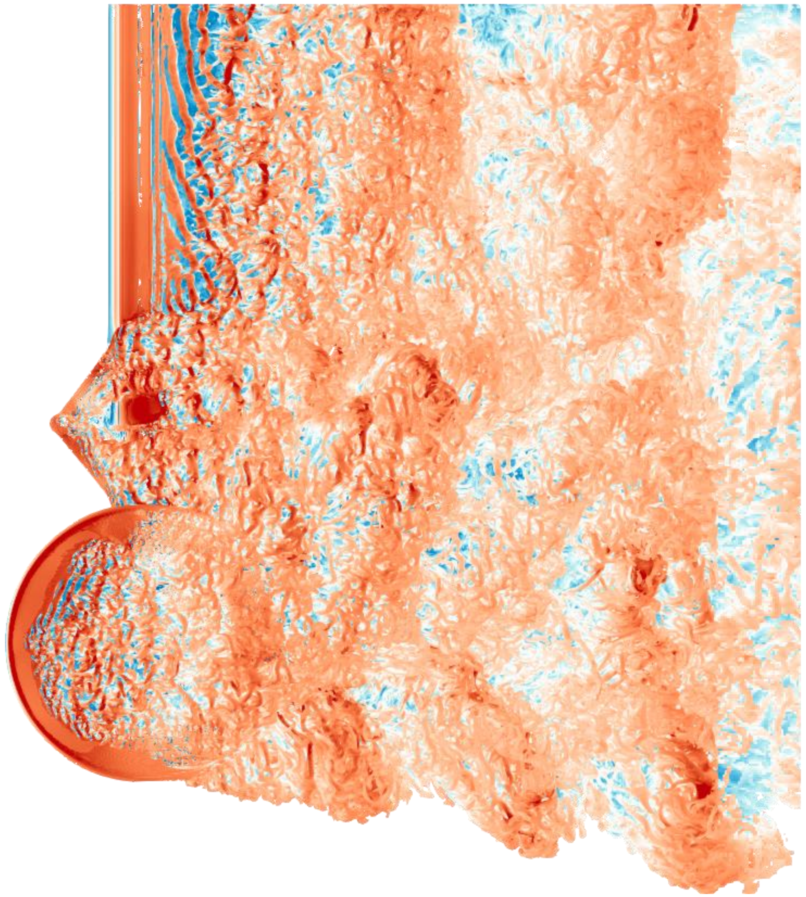
- Graphic Processing Units (GPUs) ensure very high performance for vector and matrix calculations
  - Typical calculations performed in many physical simulations
- Cost-performance ratio is typically lower for GPUs as for CPUs
- Relevance of GPUs for High-Performance Computing (HPC) systems is continuously growing
  - In the last decade, the number of GPU accelerated systems in the top 500 list has been constantly increasing
  - 6 of the 10^{th} fastest supercomputers (June 2021) are accelerated by GPUs

Fig. 1: Supercomputer „Juwels“
(Photo: dpa/Marius Becker)

Fig. 2: Supercomputer „HAWK“
(Photo: HLRS)

Outline

1. Multiphysics - Aerodynamisches Institut Aachen (m-AIA)
   - The m-AIA simulation framework
   - Grid generation using m-AIA
   - The lattice Boltzmann method in m-AIA
2. GPU porting of the lattice Boltzmann method
   - GPU-Models requirements
   - Validation and Performance
3. Exemplary simulation setups
   - Flow around a landing gear configuration
   - Flow through a human nasal cavity
4. Conclusion and outlook



m-AIA simulation framework:

(formerly known as ZFS)

- Several solver types are implemented for different physical issues:
- *Finite Volume method (FV):* Fluid mechanics
- *Discontinuous Galerkin method (DG):* Aeroacousics
- *Level-set method (LS):* Tracking contours of flames
- *Lattice Boltzmann method (LB):* Fluid mechanics
- *Lagrange solver:* Particle distributions

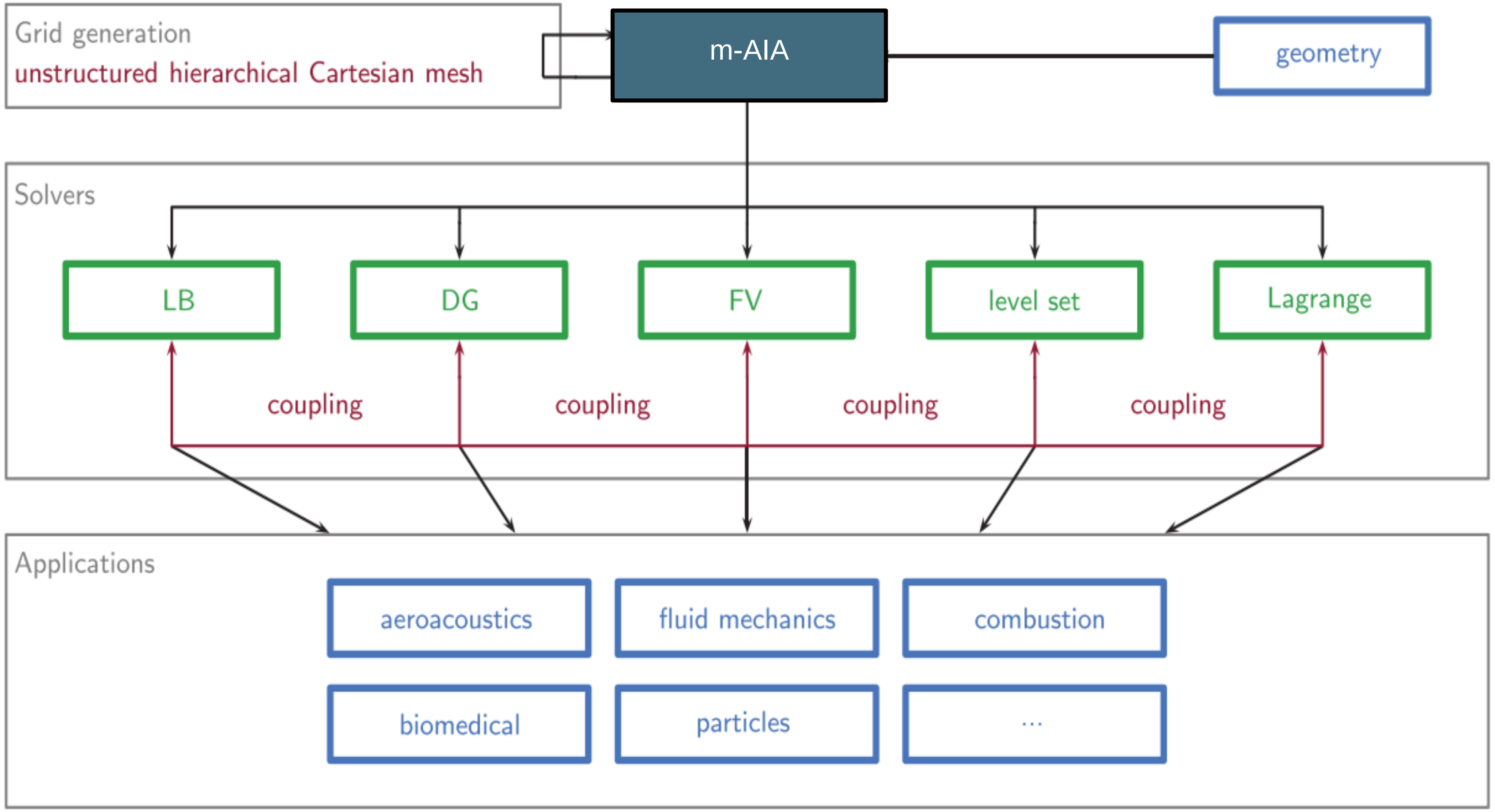


Fig. 3: Structure of the m-AIA simulation framework [1]

m-AIA simulation framework:

(formerly known as ZFS)

- m-AIA is built up modularly
- Direct-hybrid coupling of different solvers with each other
- A wide range of physical and numerical models
- Adaptive mesh refinement
- Dynamic load balancing

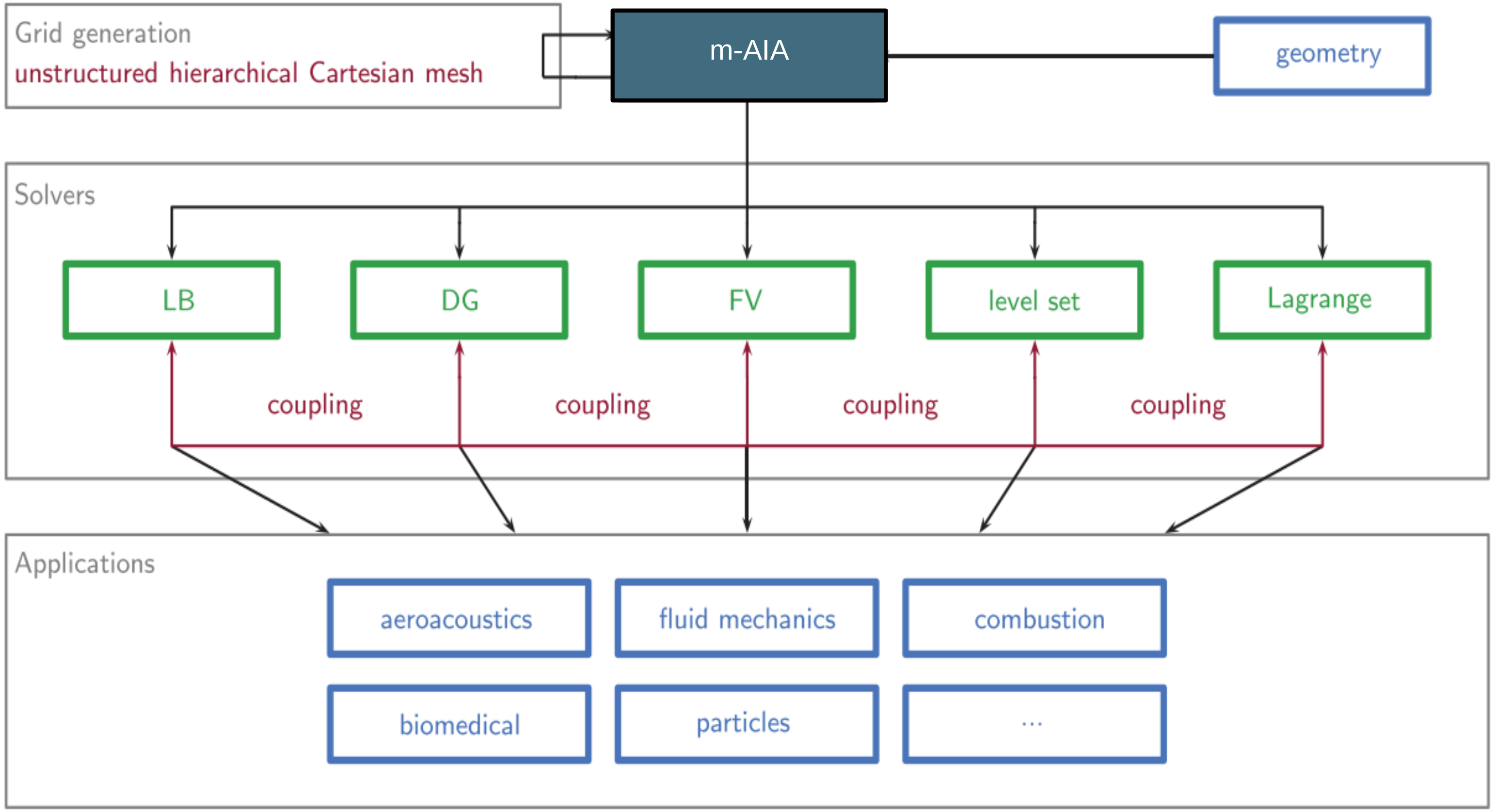
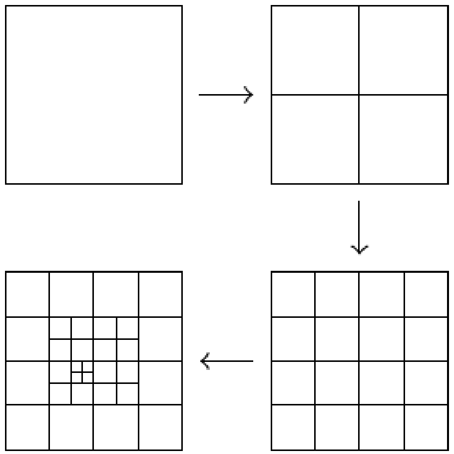


Fig. 3: Structure of the m-AIA simulation framework [1]

Grid generation using m-AIA:

- Unstructured joint-hierarchical Cartesian grid
- Massively parallel grid generator [2]
- Partitioning based on Space-filling Hilbert curve
    - The length of the segments are determined by weights



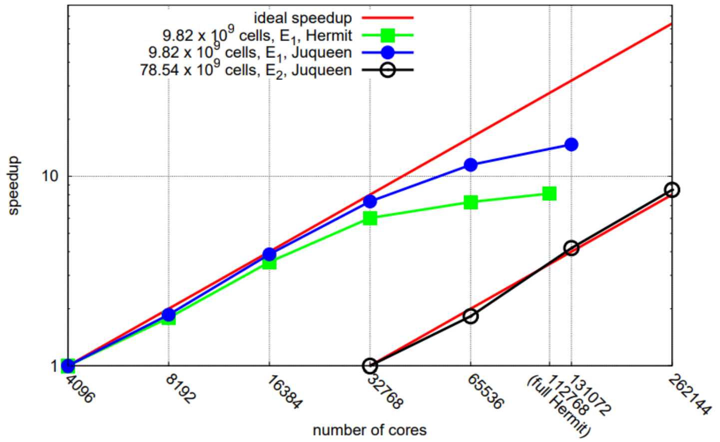


Fig. 4: Strong scaling of the grid generation on Hermit and Juqueen [2]

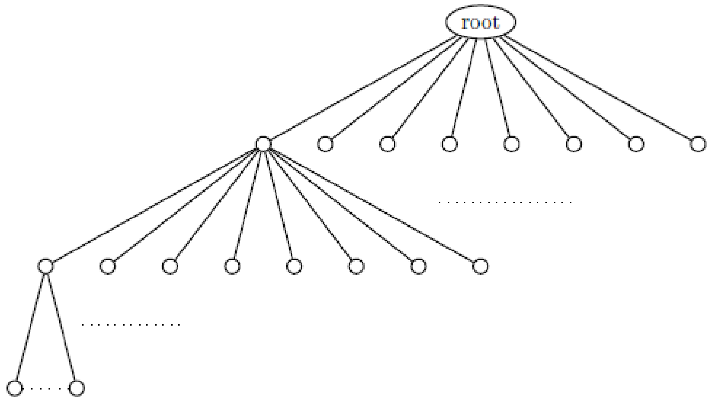


Fig. 5: Octree structure of m-AIA's grid [2]

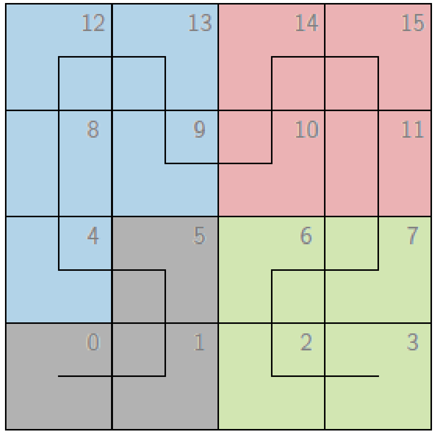


Fig. 6: Space-filling Hilbert curve

Memory layout of m-AIA:

- Cells are numbered according to their Hilbert id and their level of refinement
- Cell data is allocated in continuous arrays pursuant to the Hilbert id
- Conservative variables as well as *Particle Probability Distribution Functions* (PPDFs) are stored with an Array-of-Structure layout (AoS)
    - Object-oriented memory layout
- Switching to a Structure-of-Arrays (SoA) layout allows for better vectorization
    - More important for GPU implementation

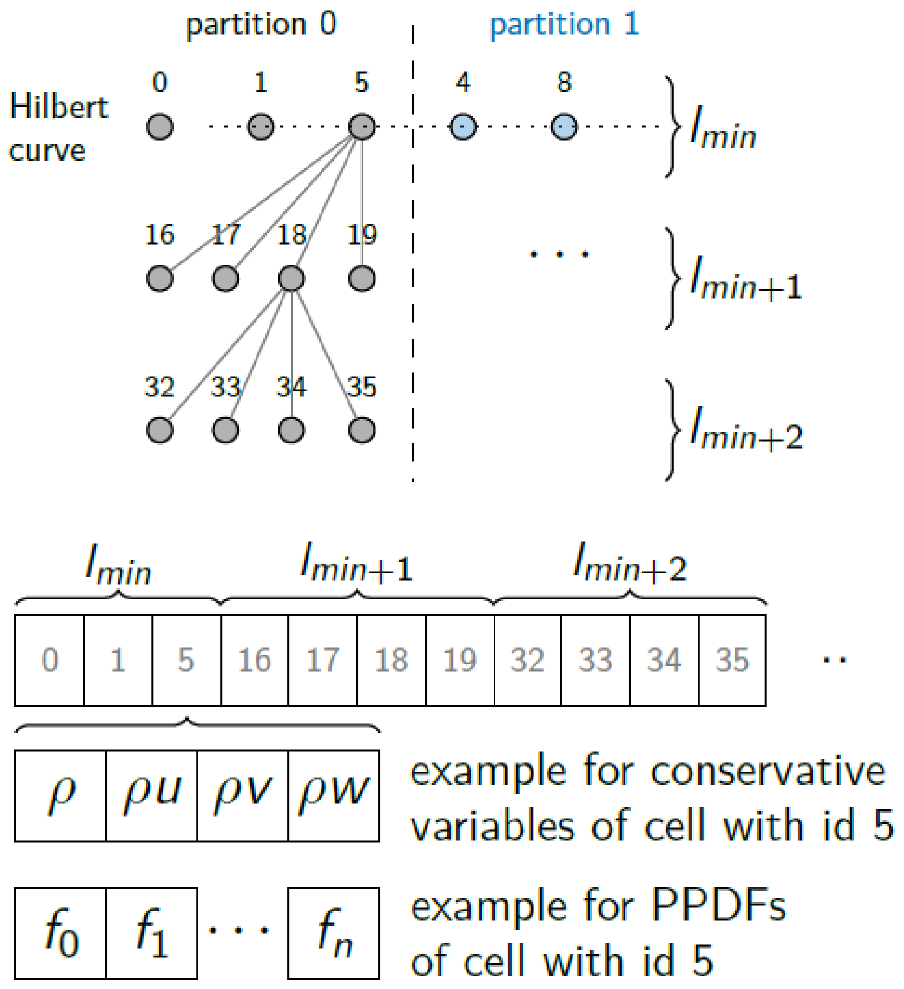


Fig. 7: Memory layout of the lattice Boltzmann solver of m-AIA

The lattice Boltzmann method in m-AIA:

- Several formulations of the discrete Boltzmann equation are implemented
- 2D and 3D discretization models are implemented (for example the D3Q27)
- f is the *Particle Probability Distribution Function* (PPDF)
- f_i^{eq} is the Maxwell distribution function [3]

Propagation step **Collision step**

$$\frac{\partial f}{\partial t} + \xi \frac{\partial f}{\partial x} + \frac{F}{m}\frac{\partial f}{\partial \xi} = \Omega(f)$$

$$f_i(r + \xi_i\, \delta t, t + \delta t) - f_i(r,t) = \frac{1}{\tau}\left(f_i^{eq}(r,t) - f_i(r,t)\right)$$

$$f_i^{eq} = \rho\, t_p \left(1 + \frac{v_\alpha \xi_\alpha}{c_s^2} + \frac{v_\alpha v_\beta}{c_s^2}\left(\frac{\xi_\alpha \xi_\beta}{c_s^2} - \delta_{\alpha\beta}\right)\right)$$

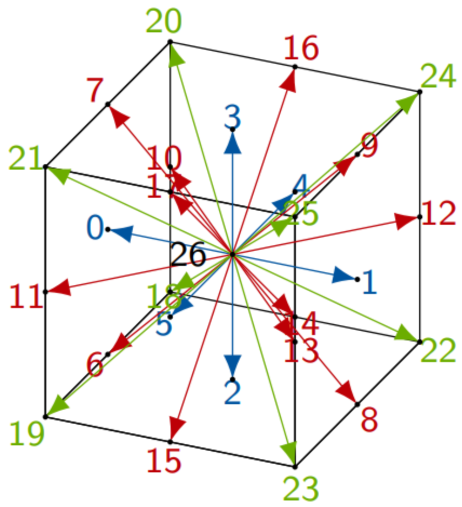


Fig. 8: D3Q27 discretization model

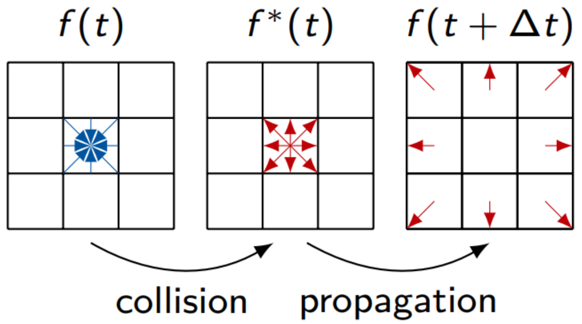


Fig. 9: Collision and propagation step

GPU-Model requirements:

- Portability: Code has to run on different HPC systems (e.g. JURECA, JUWELS, HAWK)
- Maintainability: Code duplication should be avoided
- Compatibility: The GPU-Model must be compatible with our hybrid multi-threading / multi-processing approach

GPU-Model	Portability	Maintainability	Compatibility
Cuda	-	-	-
OpenACC	+	+	+
OpenCL	+	-	+
OpenMP 4.0	+	+	+
Parallel STL	+	+	+

Tab. 1: Comparison of different GPU-Models

GPU porting of the lattice Boltzmann method

Advantages of the parallel *Standard Template Library* (STL) of *C++17:*

- The parallel STL is implemented by all Compilers supporting *C++17*
- Multi-threading and multi-processing is supported
- No code duplication is needed
- No additional external libraries are needed

GPU-Model	Portability	Maintainability	Compatibility
Cuda	-	-	-
OpenACC	+	+	+
OpenCL	+	-	+
OpenMP 4.0	+	+	+
Parallel STL	+	+	+

Tab. 1: Comparison of different GPU-Models

GPU porting of the lattice Boltzmann method

<u>Advantages of the parallel *Standard Template Library* (STL) of *C++17*:</u>

- The parallel STL is available for all Compilers supporting *C++17*
- Multi-threading and multi-processing is supported
- No code duplication is needed
- Execution policy used: par_unseq

```
// Open-MP example
#pragma omp parallel for
for(Mint i = 0; i < a_noCells(); i++) {
    a_variable(i, PV->U) = 1.0;
    a_variable(i, PV->RHO) = a_coordinate(i, 1) * m_densityGrad;
}

// Equivalent pstl
auto myRange = ranges::iota_view{0, noCells};
auto begin_ = ranges::begin(myRange);
std::for_each_n(std::execution::par_unseq, begin_, a_noCells(), [=](Mint i) {
    a_variable(i, PV->U) = 1.0;
    a_variable(i, PV->RHO) = a_coordinate(i, 1) * m_densityGrad;
}
```

GPU-Model	Portability	Maintainability	Compatibility
Cuda	-	-	-
OpenACC	+	+	+
OpenCL	+	-	+
OpenMP 4.0	+	+	+
Parallel STL	+	+	+

Tab. 1: Comparison of different GPU-Models

Algorithm 1: Collision step Lattice Boltzmann

Result: Post collision PPDFs f_i

foreach *cell* $\leftarrow 0$ **to** *noCells* **do** // exec. par_unseq
  for $i \leftarrow 0$ **to** *noPPDFs* **by** 1 **do**
    $\rho \mathrel{+}= f(i)$; // calculate density
    $\mathbf{u} \mathrel{+}= \xi \cdot f(i)$; // calculate velocity
  end
  for $i \leftarrow 0$ **to** *noPPDFs* **by** 1 **do**
    $f^{eq}(i) = \ldots$; // Maxwell distribution
    $f(i) = \ldots$; // collision equation
  end
end

Validation and Performance:

- A 3D lid-driven cavity flow is simulated on a uniform mesh
- Different mesh resolutions are used
  - Consisting of up to 2.3 billion cells
- Reynolds number is set to Re = 500
- Mach number is set to Ma = 0.1
- BGK-Collision operator with the D3Q27 model
- Interpolated Bounce-Back Scheme applied at all walls
- Simulations are conducted on CPU and GPU based systems
  - on up to 128 nodes

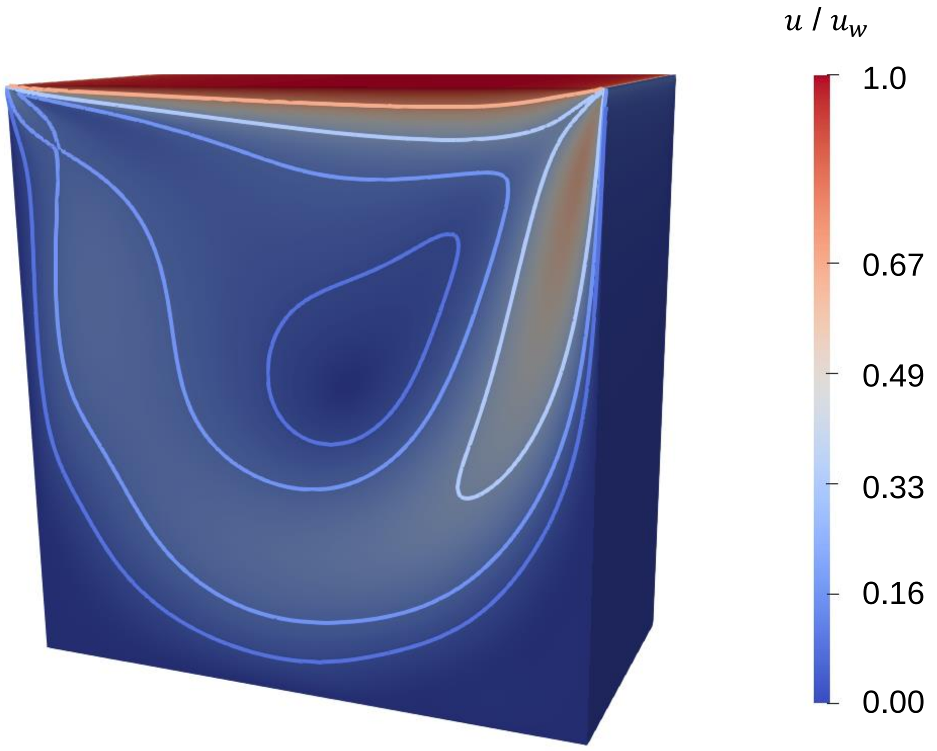


Fig. 10: Velocity magnitude for a 3D lid-driven cavity

GPU porting of the lattice Boltzmann method

PSTL on CPUs:

- The simulation using the multi-threading option of the PSTL implementation compiled with the NVHPC compiler is as fast as an OpenMP implementation compiled with GCC

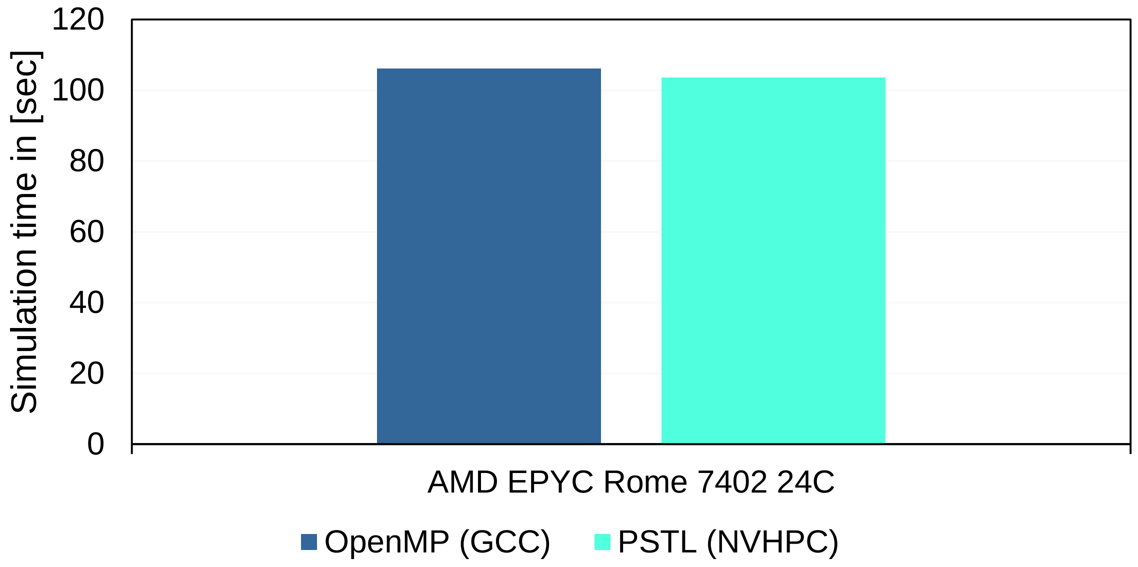


Fig. 13: Comparison of OpenMP and parallel STL on a single node (3D lid-driven cavity testcase)

GPU porting of the lattice Boltzmann method

JUWELS Booster (Nvidia A100):

- 4 CPUs per node with 1 GPU per CPU
- Grid size: $1.6 \cdot 10^8$ cells (blue)

  and $2.3 \cdot 10^9$ cells (orange)

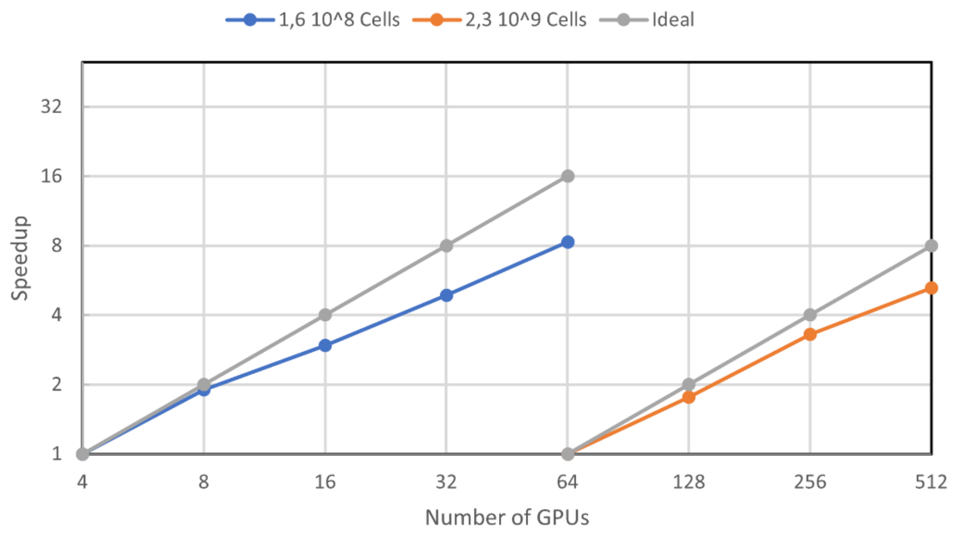


Fig. 11: Strong scaling on JUWELS Booster

Selene (Nvidia A100-SXM4-80):

- 8 CPUs per node with 1 GPU per CPU
- Grid size: $6.3 \cdot 10^8$ cells

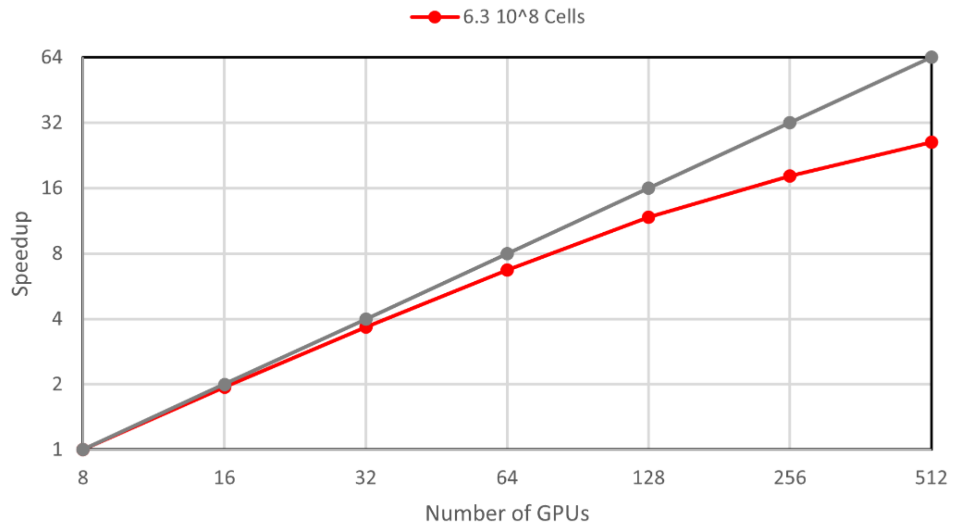


Fig. 12: Strong scaling on Selene

GPU porting of the lattice Boltzmann method

Comparison between AoS and SoA:

- AoS: Speed-up 1.71 between 2 Nvidia A-100 GPUs and 2 AMD EPYC 7742 64C
- SoA: Speed-up 5.30 between 2 Nvidia A-100 GPUs and 2 AMD EPYC 7742 64C
- 2 Nvidia A-100: Speed-up 2.81 between SoA and Aos layout

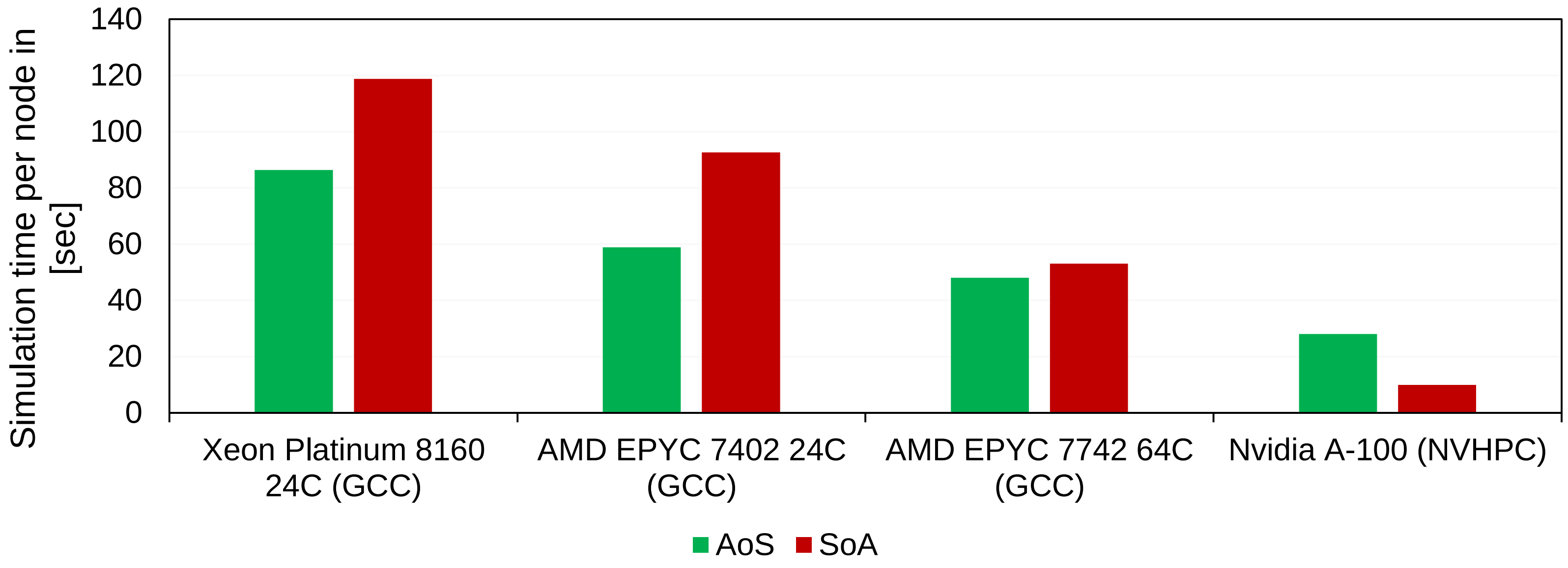


Fig. 14: Comparison of AoS and SoA based systems on multiple nodes
(3D lid-driven cavity testcase $80 \cdot 10^6$)

Comparison between AoS and SoA:

- When using the AoS, the propagation step is with 73.1% the most time-consuming step of the solution step
- Using the SoA reduces the percentage share of the propagation step to 50.8%
- Since the absolute time required for the communication is equal for both simulations, its percentage share increases from 5.2% to 15.2%

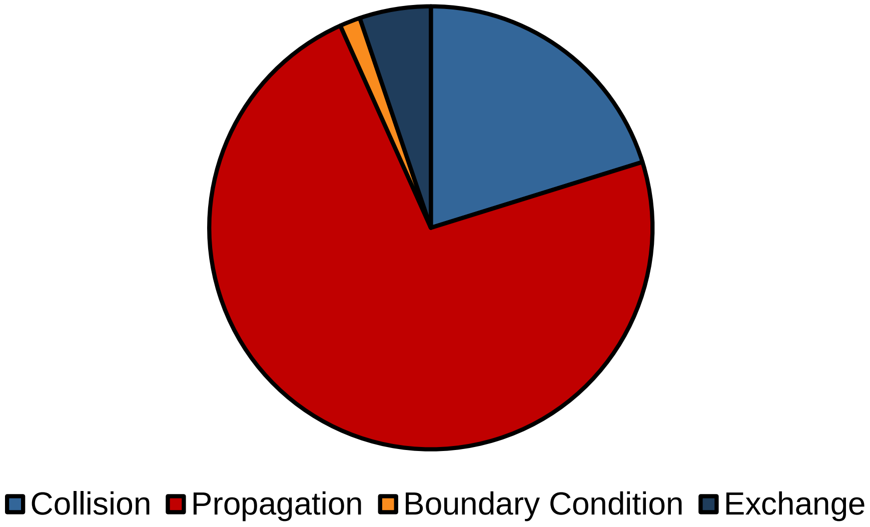


Fig. 15: Percentage share of the individual steps in the total solution step for the AoS

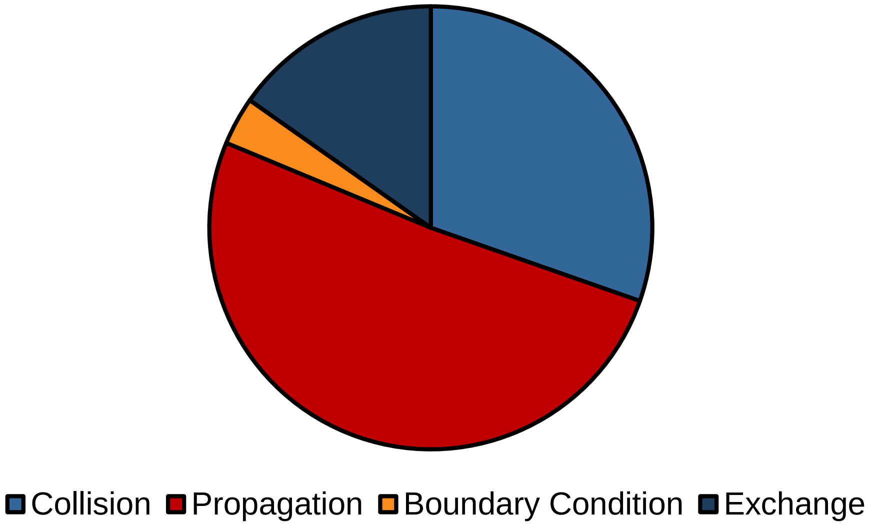


Fig. 16: Percentage share of the individual steps in the total solution step for the SoA

Exemplary setup: Nose landing gear

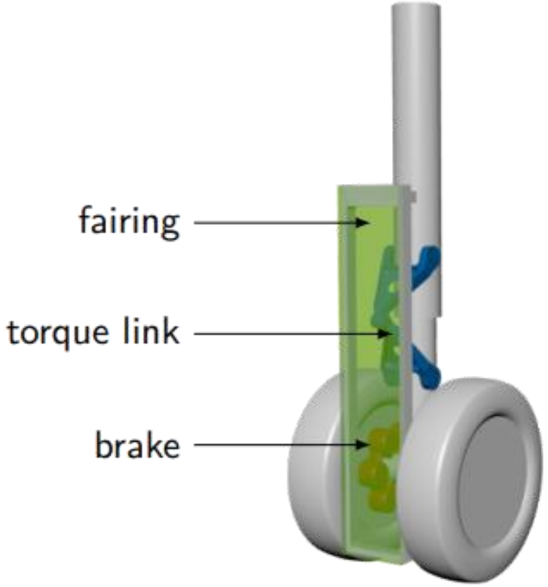
Flow parameters

- D = 150 mm (Wheel diameter)
- M = 0.1 (~35 m/s)
- $Re_D = 350{,}000$

A-Tunnel

- Open-jet closed-circuit vertical aeroacoustic wind tunnel at Delft University of Technology
- Conducting PIV and acoustic measurements

Geometry



Experimental setup

Fig. 15: Setup at TUD's A-Tunnel with nozzle *Delft 40x70.*

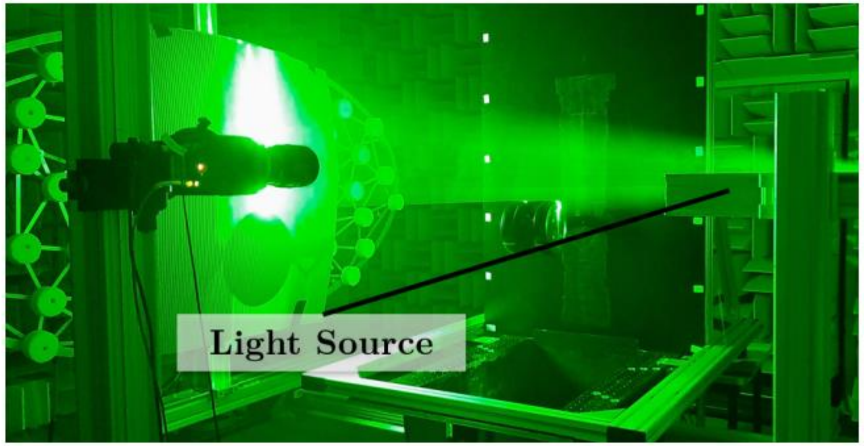
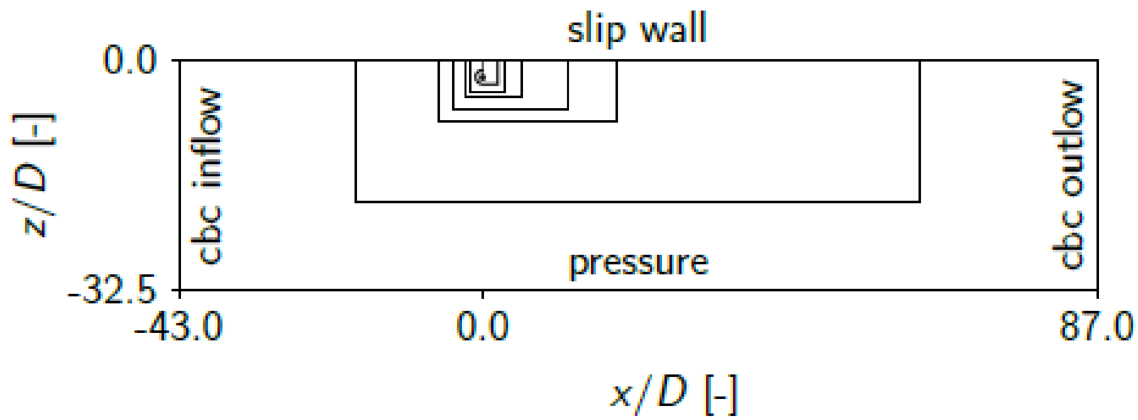


Fig. 16: PIV setup during image acquisition.

Exemplary setup: Nose landing gear

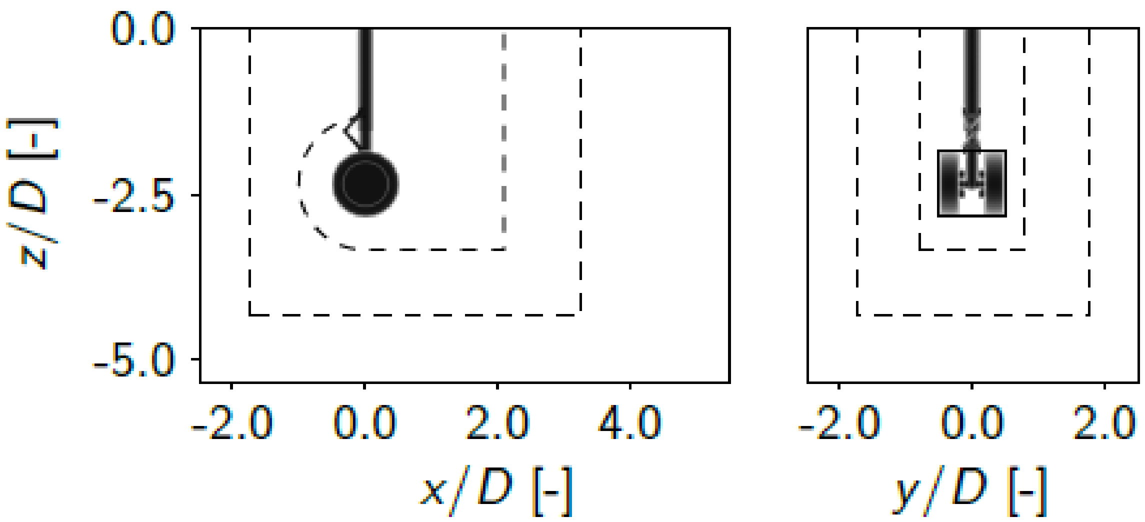
Computational domain CFD

- Domain size: (130 x 65 x 32.5) D
- Physical domain size: (80 x 40 x 20) D
- Sponge region on coarsest refinement level

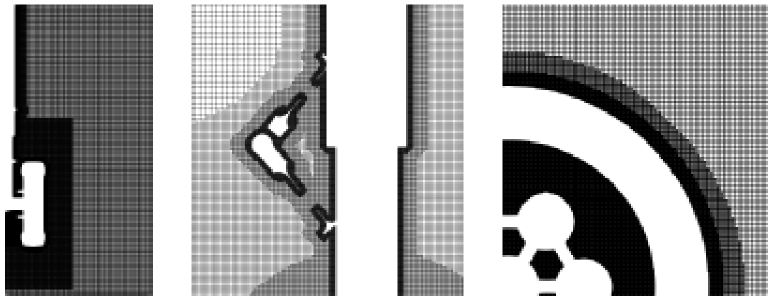


Grid resolution study

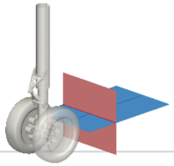
Grid	noCells/D	dt [s]	noCells
coarse	252	1e-06	150 million
medium	504	5e-07	200 million
fine	1008	2.5e-07	705 million



Grid with medium resolution



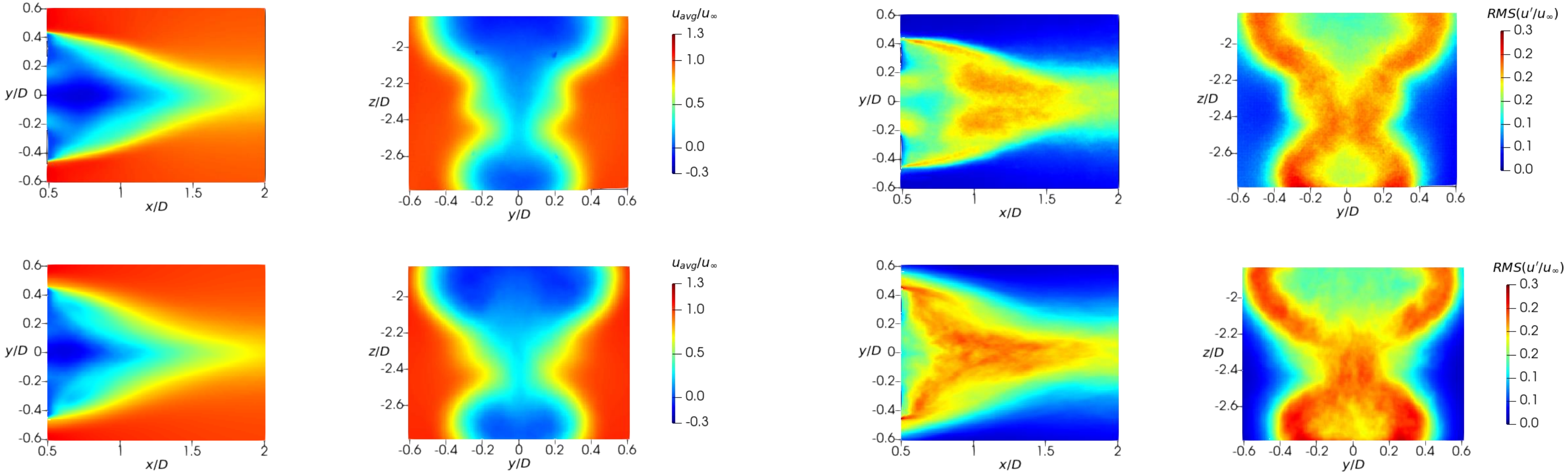
Exemplary setup: Nose landing gear



BlBkTl

experimental

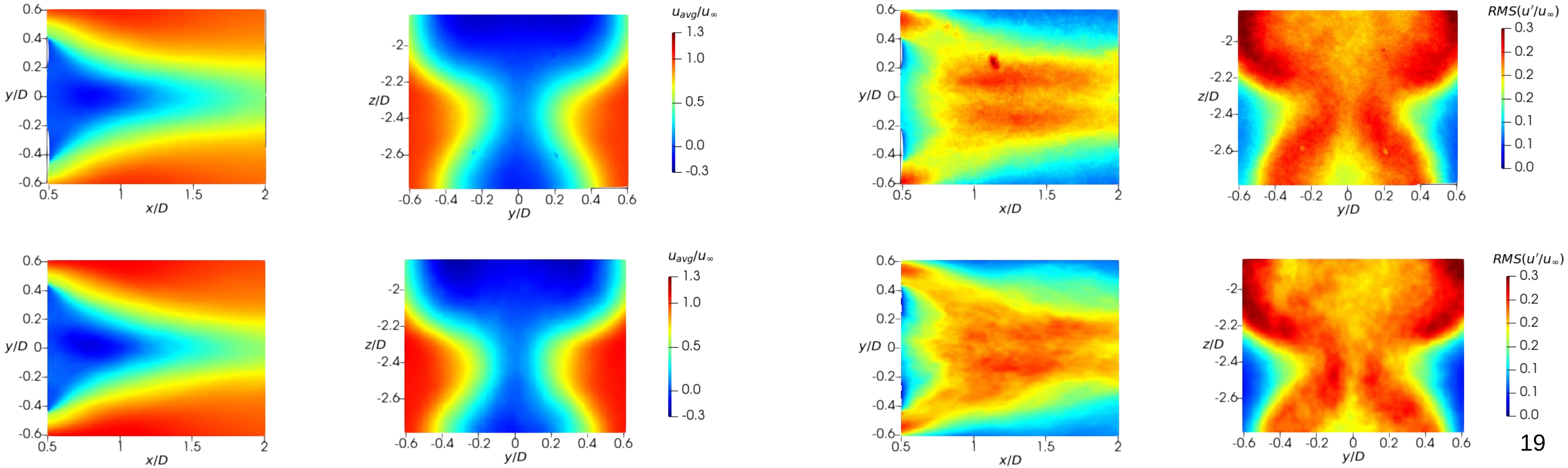
numerical



BlBkTl + solid fairing

experimental

numerical



Exemplary setup: Nasal cavity

Simulation setup for the simulation of respiration:

- Reynolds number based on the pharynx's diameter is in the range of Re = 500 – 2,000
- A locally refined mesh with up to $200 \cdot 10^6$ cells is used
- At the inner walls, an interpolated bounce-back scheme is set
- At the outlet, the volume flux is prescribed
- At the inlet, the equation of Saint Venant and Wantzel is used

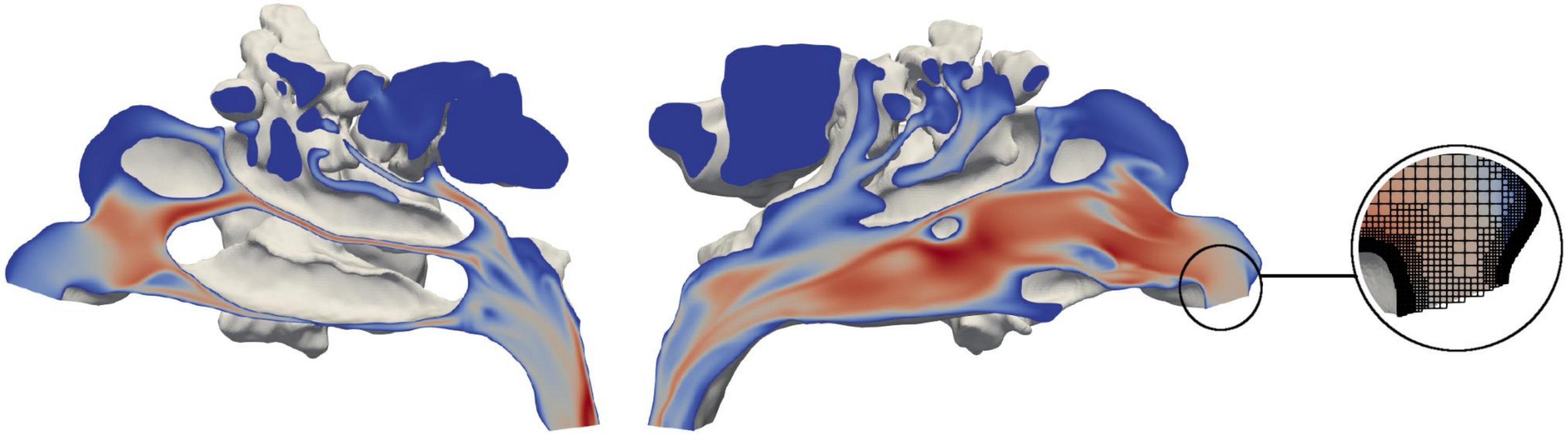


Fig. 17: Flow field in the nasal cavity

Conclusion and outlook

Conclusion:

- Lattice Boltzmann solver of m-AIA is ported to GPUs using the parallel STL of *C++17*
- Memory layout of m-AIA was changed from an AoS to a SoA
- The GPU porting increased the performance by a factor of 1.71 compared to a HAWK node (AoS)
- SoA: the simulation is 5.3 faster on two Nvidia A-100 than on a HAWK node
  - However, performance on CPUs decreased after the change

Outlook:

- Further functionalities of the LB solver will be ported
  - Initialization, further boundary conditions, I/O?, ..
- Further solvers of m-AIA will be ported to GPUs
  - Starting with the DG solver to run couple CFD/CAA simulation
- Improvement of the communication routine
  - p.e. hiding the communication behind the solution step of a coupled solver

References

[1] Lintermann, A. and Meinke, M. and Schröder, W.: Zonal Flow Solver (ZFS): „A highly efficient multi- physics simulation framework“, International Journal of Computational Fluid Dynamics 34 (2020), doi: 10.1080/10618562.2020.1742328

[2] Lintermann, A. and Schlimpert, S. and Grimmen, J.H. and Günther, C. and Meinke, M and Schröder, W.: „Massively parallel grid generation on HPC systems“, Computer Methods in Applied Mechanics and Engineering (2014), doi:10.1016/j.cma.2014.04.009

[3] Bhatnagar, P. L. and Gross, E.P. and Krook, M.: “A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems”, Physical Review 94 (3), doi:10.1103/PhysRev.94.511