CERFACS

EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

FAU

Friedrich-Alexander-Universität
Technische Fakultät

Tackling Performance Challenges of Large Scale Lattice Boltzmann Applications using Metaprogramming Techniques within the Multiphysics Framework WaLBerla

Airbus Scientific Computing Conference 2023

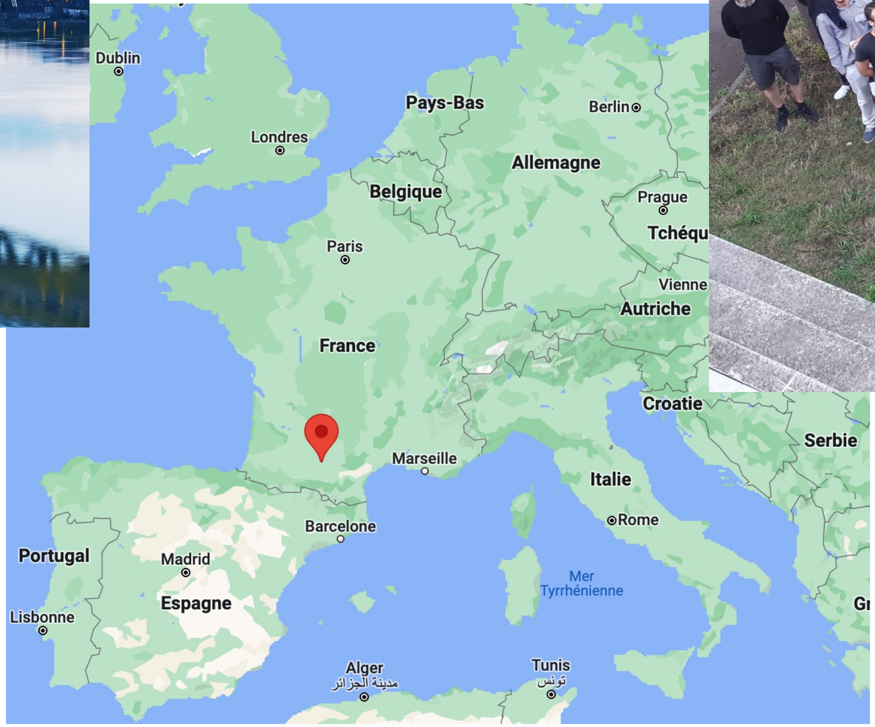
Markus Holzer

Supervisors: Gabriel Staffelbach, Ulrich Rüde and Catherine Lambert

October 25th, 2023

Cerfacs

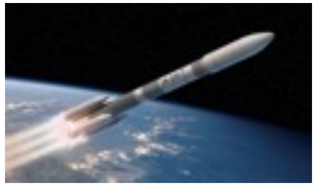
Pont Saint-Pierre of Toulouse
in the south west of France

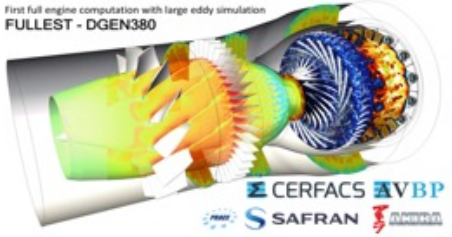


Ph.D. Students' Day

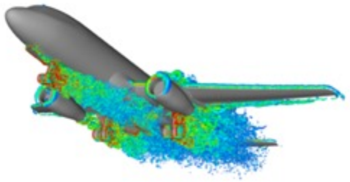
Cerfacs Strategic Research Plan 2023-2027

HSPS: HPC Simulation of Propulsions Systems

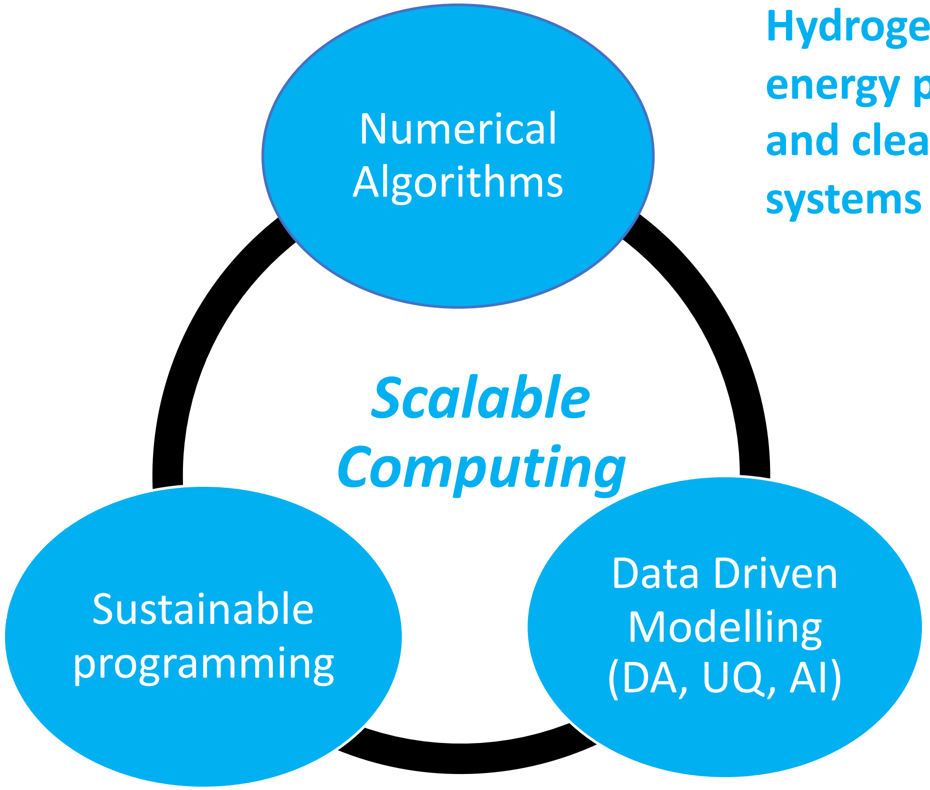




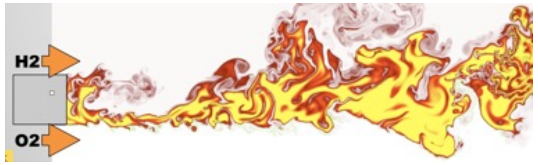
HSAA: HPC Simulation of Aerodynamics and Aerocoustics of Fixed/Mobile Surface



CLIMAIR HPC for modelling climate-air transport links

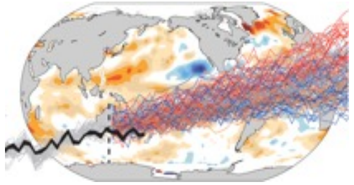


SHEPCS: HPC Simulation of Hydrogen-based energy production and clean energy systems



MODEST: HPC Modelling for Environment and Safety

CLIPROC: Climate Variability and Predictability: From Ocean to Continental Impacts



Cerfacs Strategic Research Plan 2023-2027

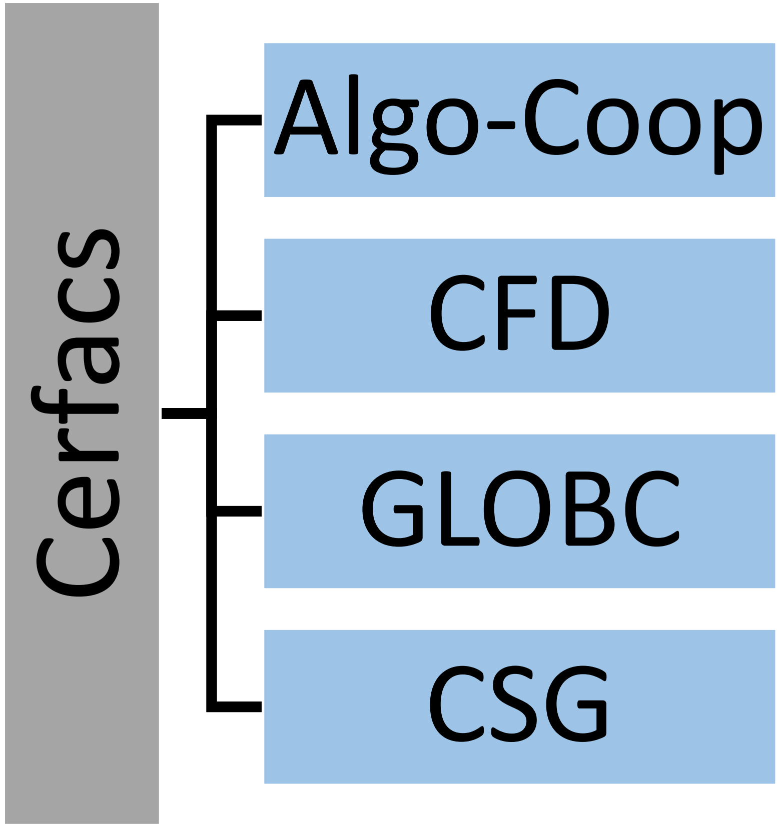
- **NUMERICAL ALGORITHMS**
  - Sparse Linear Algebra – Discretization and Finite Elements – optimization
  - Novel numerical approaches applied to CFD -> lattice Boltzmann methods

- **SUSTAINABLE PROGRAMMING**
  - Sustaining, Improving, optimizing, and refactoring legacy codes and Quantum, advanced programming Methods (DSL, PU, New Langages) & Technology watch Coupling
  - HPC Workflow (including Data Management) & User Interface

- **DATA DRIVEN MODELLING**
  - Uncertainty Quantification
  - Data Assimilation
  - Physics-based AI

Algo-Coop

1. Parallel Algorithms Team
2. Scientific Software Operational Performance Team
   - Software engineering
   - Codemetrics and software sustainability
   - Heterogenous computing in exascale simulations
   - Machine learning and AI
   - Quantum computing
   - Code Generation

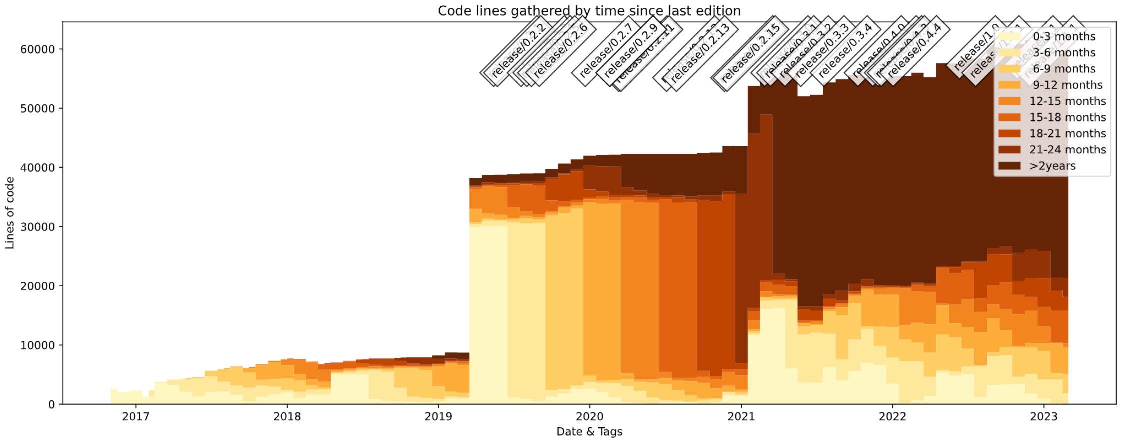


Codemetrics

- Analysis of the technical dept of software

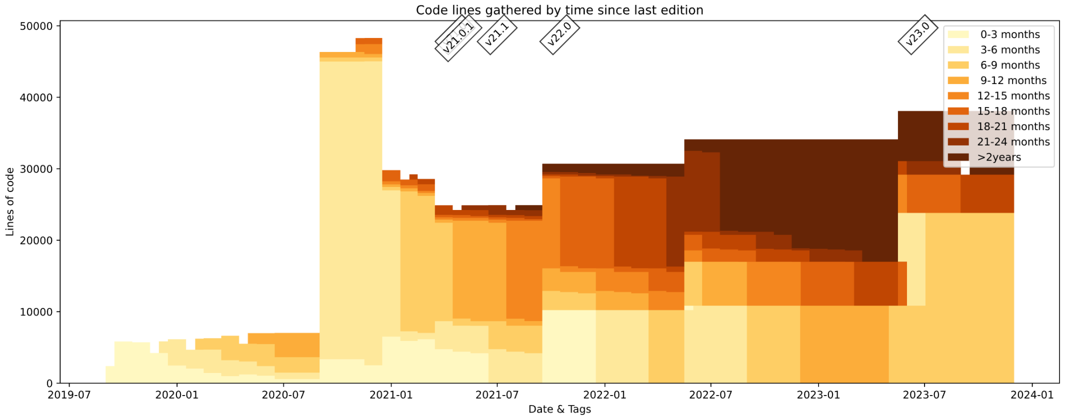
  technical dept is the implied cost of additional rework caused by choosing an easy (limited) solution now instead of using a better approach that would take longer

lbmpy (code generator for LBM)



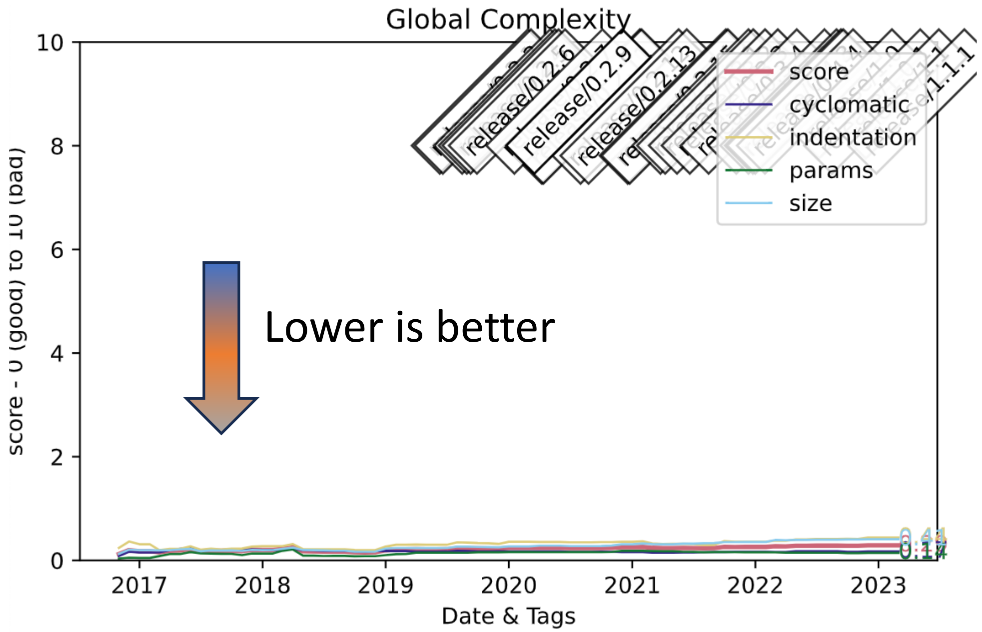
Old code ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ New code

NEK RS Navier-Stokes solver

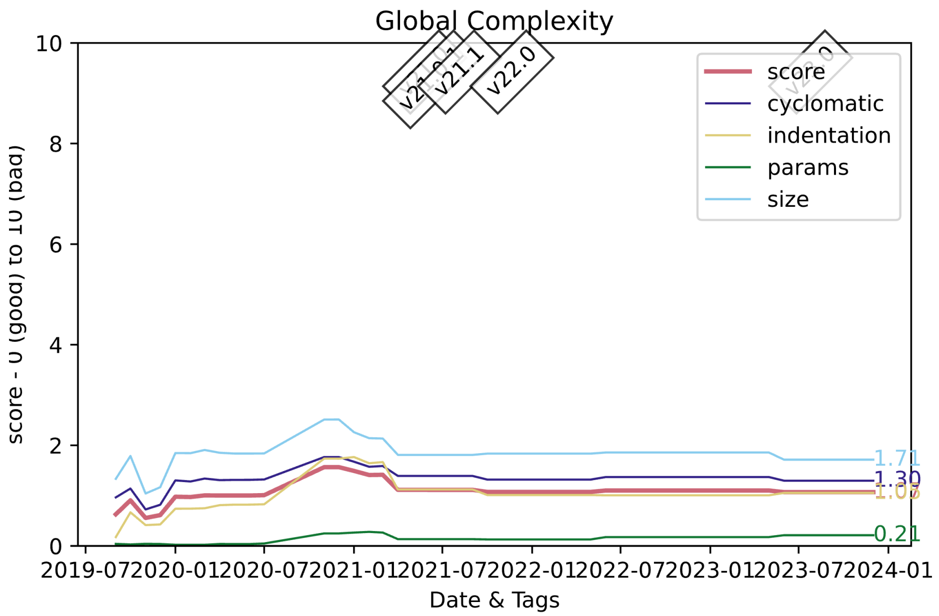


Codemetrics

lbmpy (code generator for LBM)



NEK RS Navier-Stokes solver



Code generation allows to work in a lower complexity context

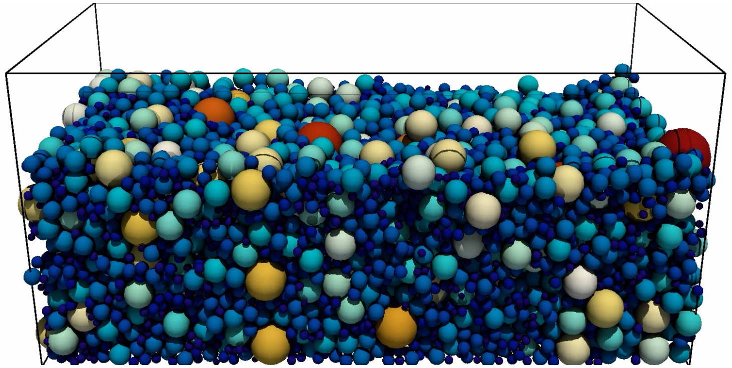
Introduction to waLBerla

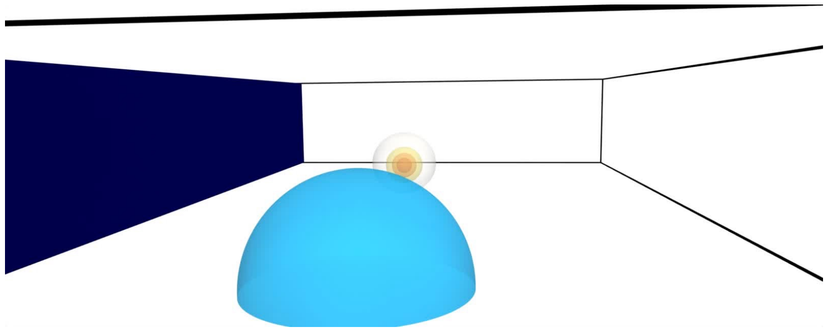
CERFACS
EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

FAU
Friedrich-Alexander-Universität
Technische Fakultät

Introduction: waLBerla

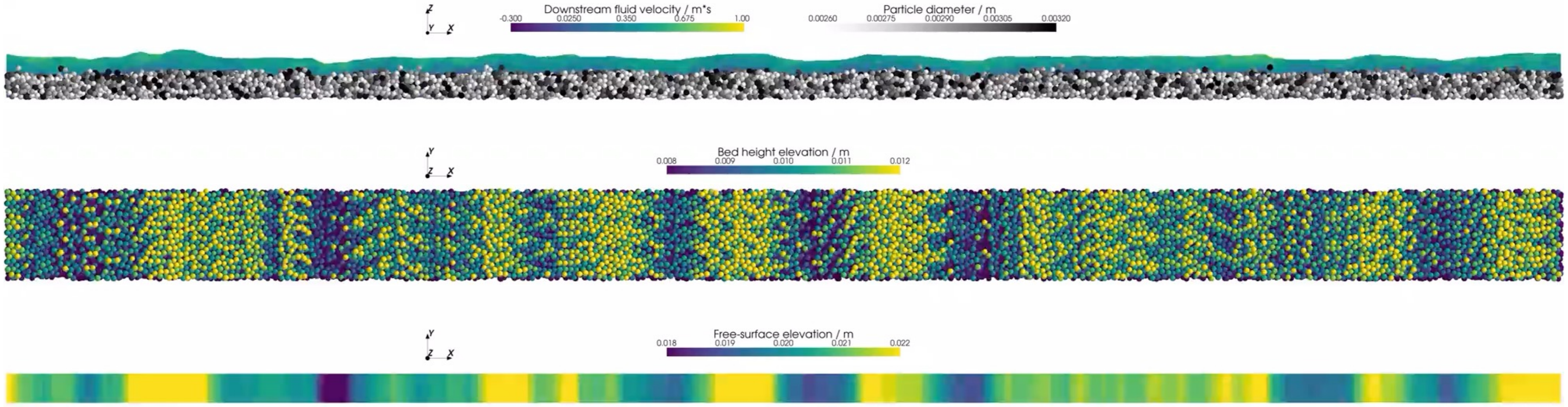
- Written in C++ with a python-based code generator
- Main applications: CFD with the lattice Boltzmann method (LBM), rigid body dynamics using the Discrete Element Method (DEM), particulate flows, free-surface and phase-field flows
- Open source: www.walberla.net





Introduction: waLBerla

- Designed for extreme-scale problems (largest simulation: 1 835 008 processes on IBM Blue Gene/Q @ Jülich)
- Applied on various different architecture:
  - CPU: Intel and AMD architectures as well as ARM chips (e.g. A64FX in Fugaku)
  - GPU: Latest NVIDIA and AMD GPUs



CoEC Center of Excellence in Combustion

EoCoE

HiDALGO Centre of Excellence

EU exascale lighthouse code

SCALABLE

CEEC

EXCELLERAT

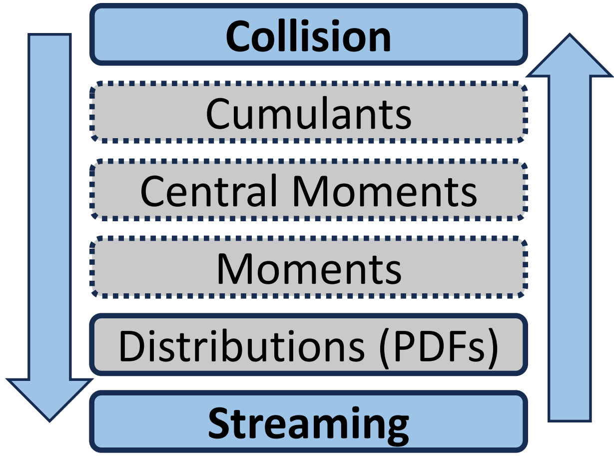
CERFACS

EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

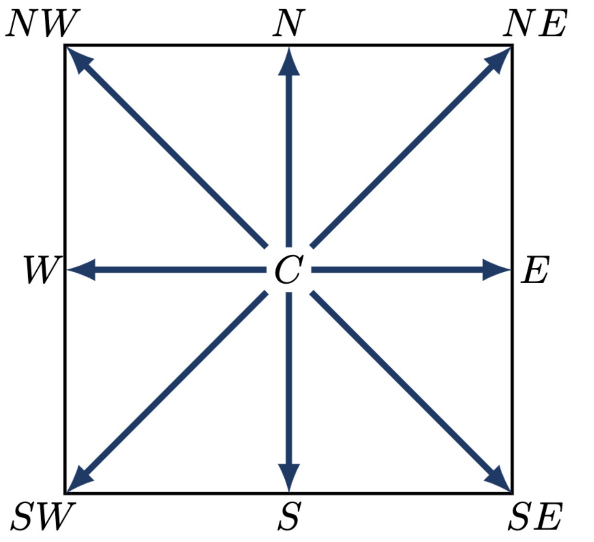
FAU
Friedrich-Alexander-Universität
Technische Fakultät

Stencil Code Generation with pystencils

Excursion: LBM



- Mesoscopic discretisation method used to solve PDEs
- Linear Advection (easy to parallelise) and non-linear Diffusion (local collision operator)
- Many different “Versions” with different complexity levels
- Explicit 2nd order scheme



Collision $$f_i^*(\boldsymbol{x}, t) = f_i^{\mathrm{eq}}(\boldsymbol{x}, t) + (1 - \Omega)\, f_i^{\mathrm{neq}}(\boldsymbol{x}, t)$$

Streaming $$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) = f_i^*(\boldsymbol{x}, t)$$

Code Generation Basic Idea

Models / Features

- Different Stencils (2D and 3D)
- Moment-based methods (MRT)
  - Efficient SRT and TRT implementations
  - Moment basis construction
  - Various equilibria
  - Forcing approaches
- Different collision space: cumulant method
- Entropic stabilization
- Locally varying relaxation rates e.g. to include turbulence models
- Coupling of multiple kernels

Hardware / Optimization

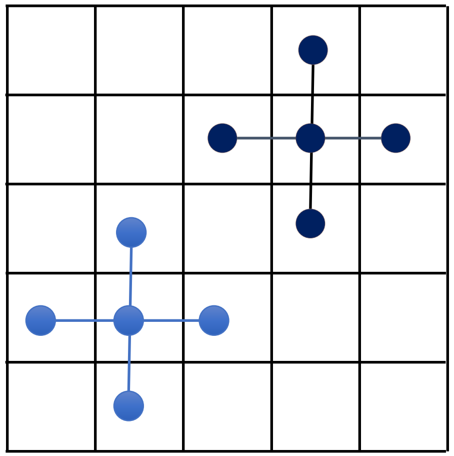
- GPU support
- Vectorization (AVX2, AVX512, QPX, SVE)
- Inner loop splitting to improve prefetching due to lower number of load/store streams
- Sparse (list-based) kernels for domains with many boundary cells
- Data layout: simple two grid stream-collide, AA pattern, EsoTwist

Solution Code Generation:

Write a program that writes programs (or performance hotspots)

Code Generation Basic Idea

- Stencil code: apply the same operation on every element of a structured array
- Easy to parallelize
- Well suited for accelerators
- Many important methods can be formulated in a stencil form (e.g. LBM, FDM, FVM, Multigrid)



Represent problem in a symbolic form to allow for optimisations from a very high level and separation of concerns

Code Generation Toolchain

Method

- Method description
- Definition of fixed and free parameters

↓

Collision Equations

- Collision operator as a list of equations
- mapping Q symbolic pdfs to Q output pdfs

↓

Update Assignments

- Encodes pdf storage
- Memory access optimizations (Streaming pattern)
- Full or sparse storage

↓

C / CUDA Code

- Clear interface (C-pointers)
- Low level optimizations

Mathematical optimisation

Architectural optimisation

Method Description

Model definition

Cumulant-Based Method | Stencil: D3Q19 | Zero-Centered Storage: ✗ | Force Model: None

Continuous Hydrodynamic Maxwellian Equilibrium

Compressible: ✓ | Deviation Only: ✗ | Order: 2

$$f(\rho, (u_0, u_1, u_2), (v_0, v_1, v_2)) = \frac{3\sqrt{6}\rho e^{-\frac{3(-u_0+v_0)^2}{2} - \frac{3(-u_1+v_1)^2}{2} - \frac{3(-u_2+v_2)^2}{2}}}{4\pi^{\frac{3}{2}}}$$

Derivation details

Relaxation Info

Cumulant	Eq. Value	Relaxation Rate
1	$\rho \log(\rho)$	0.0
x	ρu_0	0.0
y	ρu_1	0.0
z	ρu_2	0.0
xy	0	ω
xz	0	ω
yz	0	ω
$x^2 - y^2$	0	ω
$x^2 - z^2$	0	ω
$x^2 + y^2 + z^2$	ρ	1.0
$xy^2 + xz^2$	0	1.0
$x^2y + yz^2$	0	1.0
$x^2z + y^2z$	0	1.0
$xy^2 - xz^2$	0	1.0
$x^2y - yz^2$	0	1.0
$x^2z - y^2z$	0	1.0
$x^2y^2 - 2x^2z^2 + y^2z^2$	0	1.0
$x^2y^2 + x^2z^2 - 2y^2z^2$	0	1.0
$x^2y^2 + x^2z^2 + y^2z^2$	0	1.0

Equilibrium

Moments/Cumulants that span the collision space

Symbolic relaxation rates

Fixed/numeric relaxation rates

Derivation: update rule

$$partial_{mm10e0} \leftarrow pdfs^{13}_{(1,0,-1)} + pdfs^{3}_{(1,0,0)} +$$

$$partial_{mm1e00} \leftarrow pdfs^{9}_{(1,1,0)} + pdfs^{7}_{(1,-1,0)} +$$

Symbolic representation in index notation. This representation contains the field access relative to the center cell.

Makes it possible to extract information for MPI routines.

```
1  update
```

Subexpressions:

$\xi_2 \leftarrow 2$

$\xi_3 \leftarrow \frac{1}{3}$

$\xi_4 \leftarrow 0.333333333333333$

$\xi_5 \leftarrow \frac{2}{3}$

$\xi_6 \leftarrow \frac{1}{2}$

$\xi_7 \leftarrow \frac{1}{4}$

$partial_{mm10e0} \leftarrow pdfs^{13}_{(1,0,-1)} + pdfs^{3}_{(1,0,0)} + pdfs^{17}_{(1,0,1)}$

$partial_{mm1e00} \leftarrow pdfs^{9}_{(1,1,0)} + pdfs^{7}_{(1,-1,0)} + partial_{mm10e0}$

$partial_{m0m1e0} \leftarrow pdfs^{12}_{(0,1,-1)} + pdfs^{2}_{(0,1,0)} + pdfs^{16}_{(0,1,1)}$

$partial_{m00e0} \leftarrow pdfs^{5}_{(0,0,-1)} + pdfs^{0}_{(0,0,0)} + pdfs^{6}_{(0,0,1)}$

$partial_{m01e0} \leftarrow pdfs^{11}_{(0,-1,-1)} + pdfs^{1}_{(0,-1,0)} + pdfs^{15}_{(0,-1,1)}$

$partial_{m0e00} \leftarrow partial_{m00e0} + partial_{m01e0} + partial_{m0m1e0}$

$partial_{m10e0} \leftarrow pdfs^{14}_{(-1,0,-1)} + pdfs^{18}_{(-1,0,1)} + pdfs^{4}_{(-1,0,0)}$

$partial_{m1e00} \leftarrow pdfs^{10}_{(-1,1,0)} + pdfs^{8}_{(-1,-1,0)} + partial_{m10e0}$

$partial_{mm1e10} \leftarrow -pdfs^{9}_{(1,1,0)} + pdfs^{7}_{(1,-1,0)}$

$partial_{m0e10} \leftarrow partial_{m01e0} - partial_{m0m1e0}$

$partial_{m1e10} \leftarrow -pdfs^{10}_{(-1,1,0)} + pdfs^{8}_{(-1,-1,0)}$

$partial_{mm10e1} \leftarrow pdfs^{13}_{(1,0,-1)} - pdfs^{17}_{(1,0,1)}$

$partial_{m0m1e1} \leftarrow pdfs^{12}_{(0,1,-1)} - pdfs^{16}_{(0,1,1)}$

$partial_{m00e1} \leftarrow pdfs^{5}_{(0,0,-1)} - pdfs^{6}_{(0,0,1)}$

$partial_{m01e1} \leftarrow pdfs^{11}_{(0,-1,-1)} - pdfs^{15}_{(0,-1,1)}$

$partial_{m0e01} \leftarrow partial_{m00e1} + partial_{m01e1} + partial_{m0m1e1}$

$partial_{m10e1} \leftarrow pdfs^{14}_{(-1,0,-1)} - pdfs^{18}_{(-1,0,1)}$

$partial_{mm1e20} \leftarrow pdfs^{9}_{(1,1,0)} + pdfs^{7}_{(1,-1,0)}$

$partial_{m0e20} \leftarrow partial_{m01e0} + partial_{m0m1e0}$

$partial_{m1e20} \leftarrow pdfs^{10}_{(-1,1,0)} + pdfs^{8}_{(-1,-1,0)}$

$partial_{mm10e2} \leftarrow pdfs^{13}_{(1,0,-1)} + pdfs^{17}_{(1,0,1)}$

$partial_{m0m1e2} \leftarrow pdfs^{12}_{(0,1,-1)} + pdfs^{16}_{(0,1,1)}$

$partial_{m00e2} \leftarrow pdfs^{5}_{(0,0,-1)} + pdfs^{6}_{(0,0,1)}$

$partial_{m01e2} \leftarrow pdfs^{11} ... + pdfs^{15} ...$

Generation: compute kernel

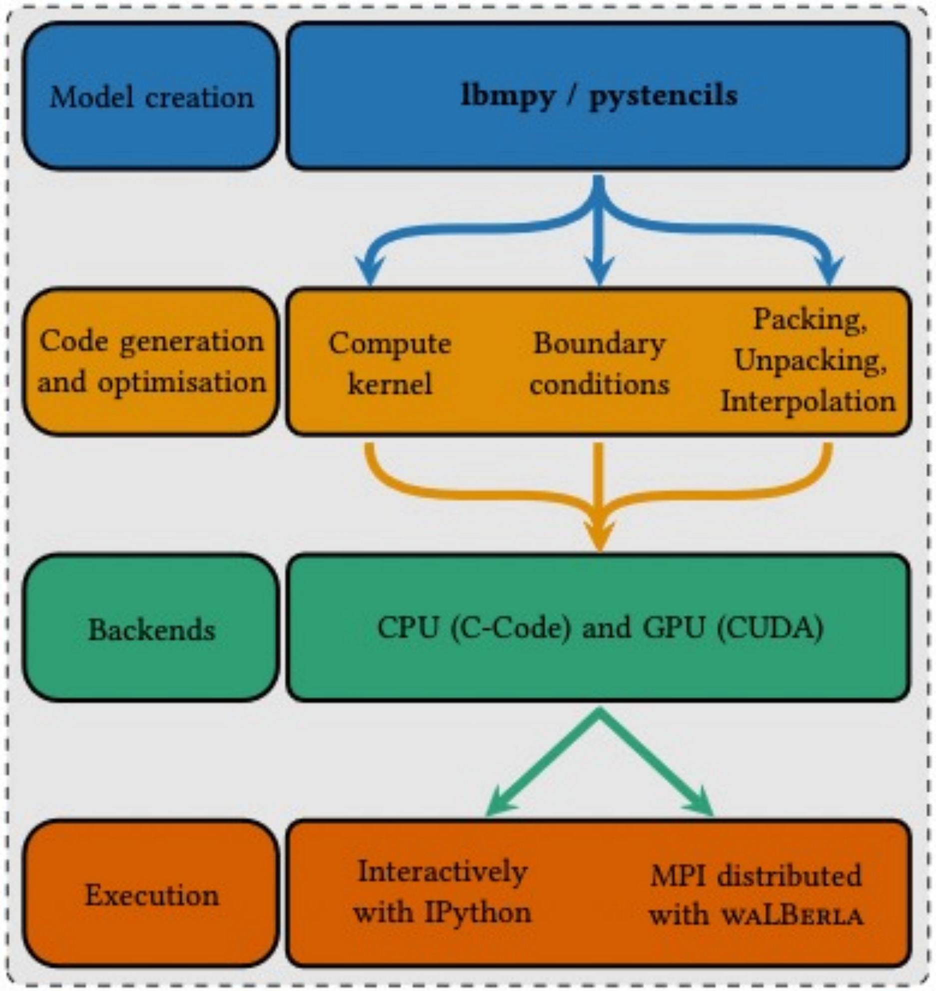
Simple API based on raw pointer notation.

Makes it very general and easily to combine with existing code or even to call the low level code directly from high level languages like Python

```
ps.show_code(ast_kernel)
```

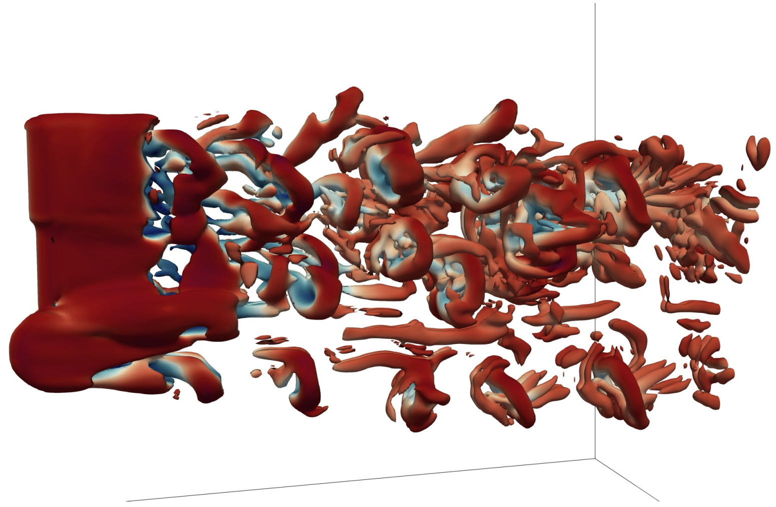
```
FUNC_PREFIX void kernel(double * RESTRICT const _data_pdfs, double * RESTRICT  _data_pdfs_tmp, double omega)
{
   const int64_t xi_2 = 2;
   const double xi_3 = 0.33333333333333331;
   const double xi_4 = 0.33333333333333331;
   const double xi_5 = 0.66666666666666663;
   const double xi_6 = 0.50000000000000000;
   const double xi_7 = 0.25000000000000000;
   for (int64_t ctr_2 = 1; ctr_2 < 33; ctr_2 += 1)
   {
      for (int64_t ctr_1 = 1; ctr_1 < 33; ctr_1 += 1)
      {
         for (int64_t ctr_0 = 1; ctr_0 < 33; ctr_0 += 1)
         {
            const double partial_m_m1_0_e_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 117913] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 509797] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 669325];
            const double partial_m_m1_e_0_0 = partial_m_m1_0_e_0 + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 275095] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 353771];
            const double partial_m_0_m1_e_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 470526] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 630054] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 78642];
            const double partial_m_0_0_e_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 195364] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 236980] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2];
            const double partial_m_0_1_e_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 39270] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 431154] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 590682];
            const double partial_m_0_e_0_0 = partial_m_0_0_e_0 + partial_m_0_1_e_0 + partial_m_0_m1_e_0;
            const double partial_m_1_0_e_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 157215] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 549099] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 708627];
            const double partial_m_1_e_0_0 = partial_m_1_0_e_0 + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 314397] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 393073];
            const double partial_m_m1_e_1_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 275095] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 353771];
            const double partial_m_0_e_1_0 = partial_m_0_1_e_0 - partial_m_0_m1_e_0;
            const double partial_m_1_e_1_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 314397] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 393073];
            const double partial_m_m1_0_e_1 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 509797] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 669325];
            const double partial_m_0_m1_e_1 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 470526] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 630054];
            const double partial_m_0_0_e_1 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 195364] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 236980];
            const double partial_m_0_1_e_1 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 431154] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 590682];
            const double partial_m_0_e_0_1 = partial_m_0_0_e_1 + partial_m_0_1_e_1 + partial_m_0_m1_e_1;
            const double partial_m_1_0_e_1 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 549099] - _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 708627];
            const double partial_m_m1_e_2_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 275095] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 353771];
            const double partial_m_0_e_2_0 = partial_m_0_1_e_0 + partial_m_0_m1_e_0;
            const double partial_m_1_e_2_0 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 314397] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 393073];
            const double partial_m_m1_0_e_2 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 509797] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 669325];
            const double partial_m_0_m1_e_2 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 470526] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 630054];
            const double partial_m_0_0_e_2 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 195364] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 236980];
            const double partial_m_0_1_e_2 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 431154] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 590682];
            const double partial_m_0_e_0_2 = partial_m_0_0_e_2 + partial_m_0_1_e_2 + partial_m_0_m1_e_2;
            const double partial_m_1_0_e_2 = _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 549099] + _data_pdfs[ctr_0 + 34*ctr_1 + 1156*ctr_2 + 708627];
```

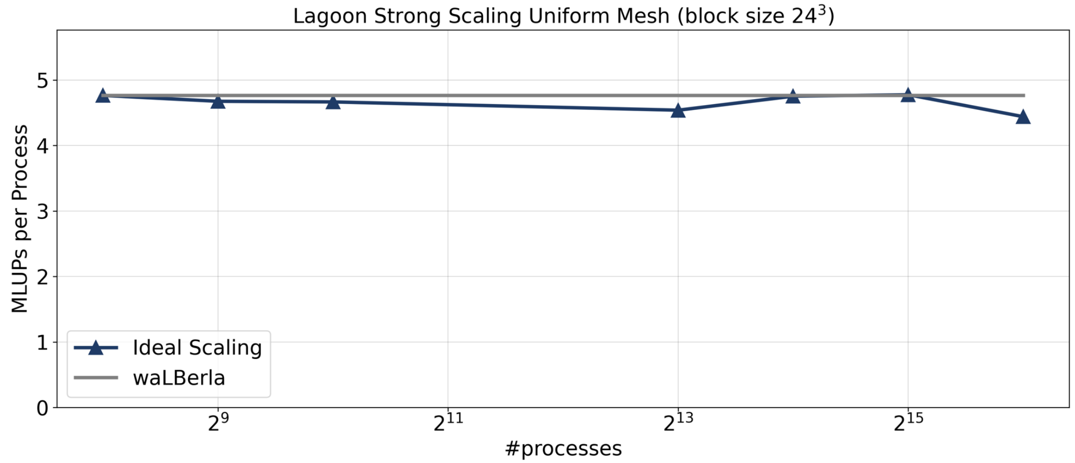
Combination with HPC frameworks like waLBerla



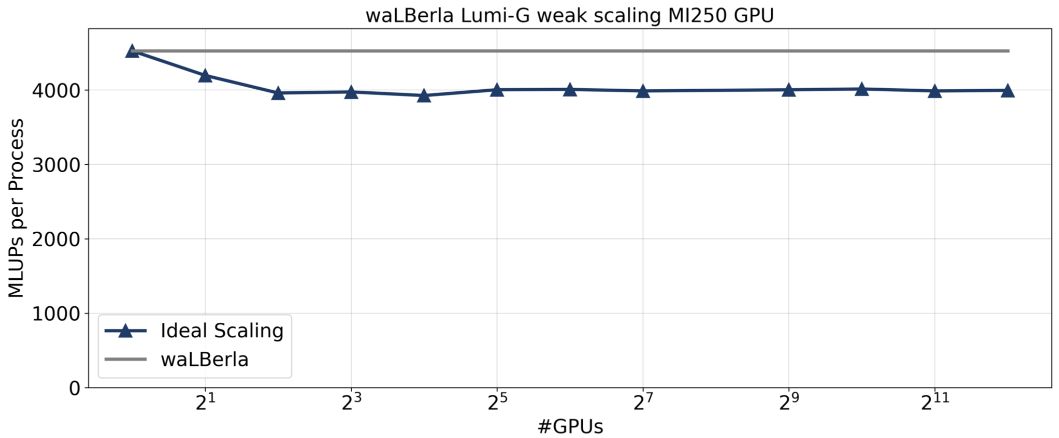
- Generation of:
  - Compute kernels for cell updates
  - Boundary conditions
  - Packing, Unpacking kernels to pack and unpack buffers for MPI communications
- Strictly defined API of the printed kernels provides additional advantages like simple embedding in boiler plate codes to combine the generated compute kernels with existing HPC frameworks
- Execution of the compute kernels in Python via C-API

Results: Lagoon Uniform mesh

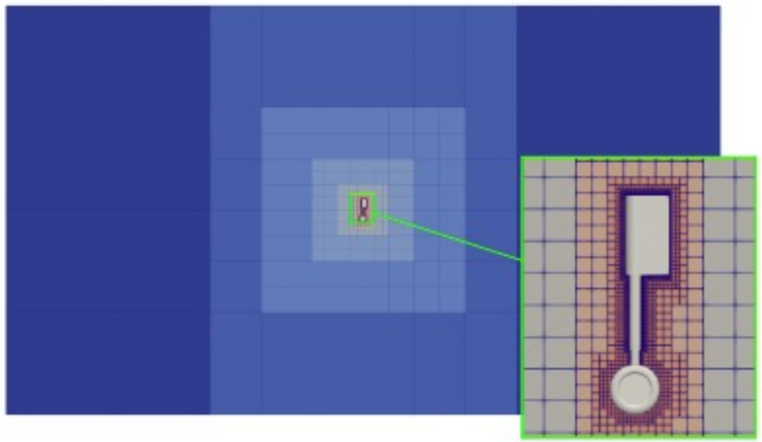




- Strong scaling experiments on up to 65 536 AMD EPYC 7742 (HAWK) shows almost perfect scaling efficiency
- Weak scaling experiments on up to 4096 AMD MI250 GPUs shows almost perfect scaling efficiency

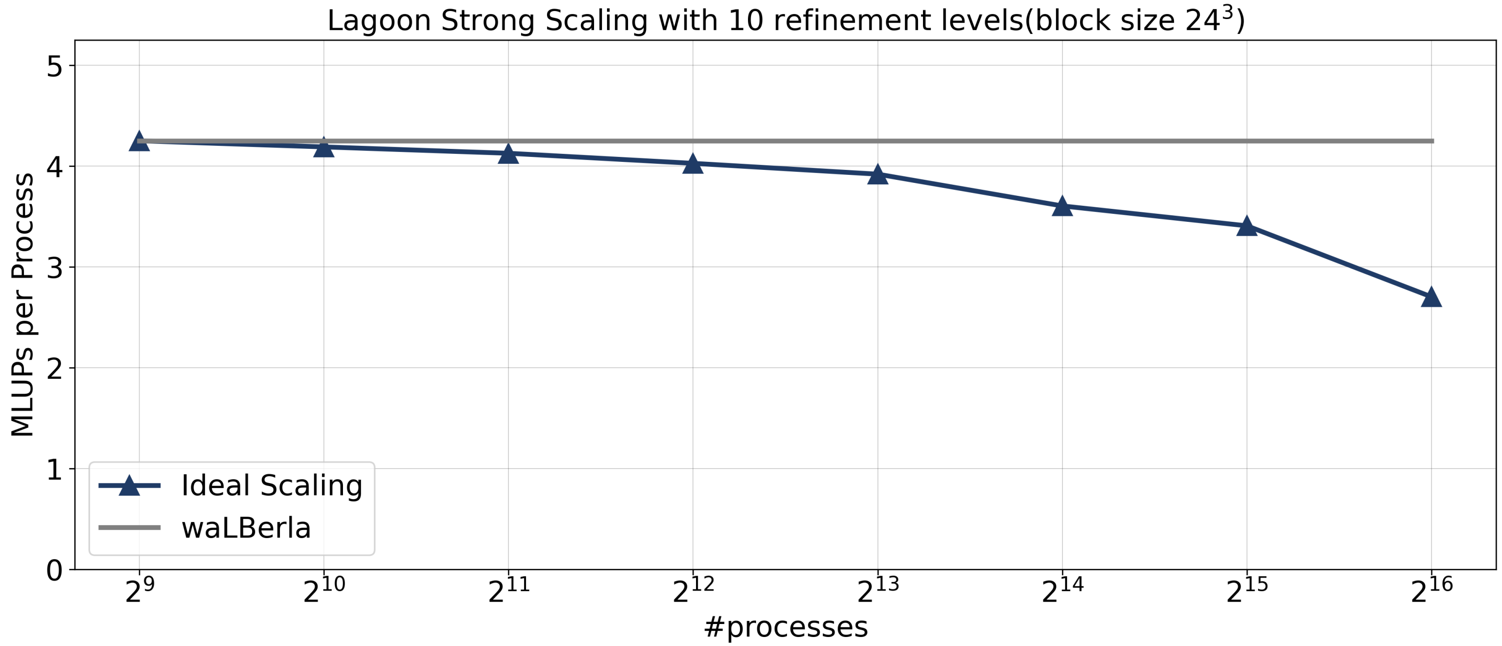


Results: Mesh Refinement for turbulent flows



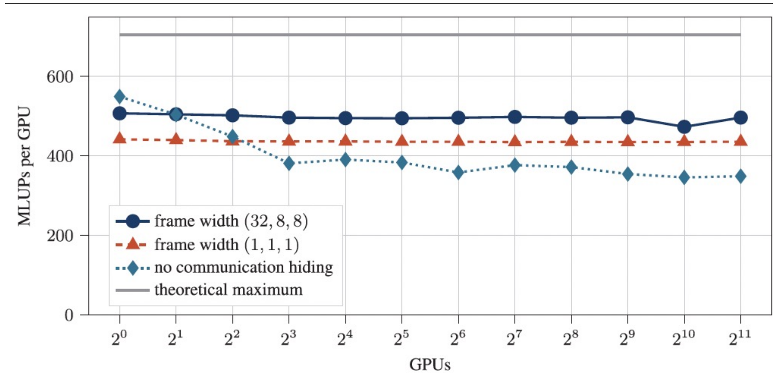
Simulation of the flow around a landing gear of an airplane to show an example for a setup with several mesh resolutions

- Domain size: 40 x 20 x 20 m resolved with 1 302 663 168 lattice cells
- Resolution around the object: 0.00025 m with 10 refinement levels
- Cores: 65 536 on the HAWK supercomputer
- About 64 % scaling efficiency



Results: Multiphase flows

Large scale bubble rise scenario simulated on the Piz Daint supercomputer with several hundred air bubbles.[1]



Weak scaling performance benchmark on the Piz Daint supercomputer.[1]

An example of the bubble propagation through the concentric annular pipe at different timesteps.[2]

- Usage of code generation for efficient compute kernels for LBM multiphase flows
- Analysis of physical results and performance
- Almost perfect scalability due to code generation for MPI-packing routines

1. M. Holzer, M. Bauer, H. Köstler, et al. "Highly Efficient Lattice Boltzmann Multiphase Simulations of Immiscible Fluids at High-Density Ratios on CPUs and GPUs through Code Generation". In: The International Journal of High Performance Computing Applications 35.4 (2021). DOI: 10 .1177/10943420211016525.
2. T. Mitchell, M. Holzer, C. Schwarzmeier, et al. "Stability assessment of the phase-field lattice Boltzmann model and its application to Taylor bubbles in annular piping geometries". In: Physics of Fluids (2021). DOI: 10.1063/5.0061694
3. C. Schwarzmeier, M. Holzer, T. Mitchell, et al. "Comparison of free-surface and conservative Allen-Cahn phase-field lattice Boltzmann method". In: Journal of Computational Physics (2022). DOI: 10.1016/j.jcp.2022.111753

EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

Friedrich-Alexander-Universität
Technische Fakultät

Conclusion

Conclusion

- Better separation of concerns due to Code Generation
- Complex Multiphysics problems can be tackled in large scales
- Sophisticated interplay between generated hotspot code and handwritten framework around
- High level of modularity increases maintainability and extensibility
- Convincing performance results on a large number of different architectures (AMD-, Intel and ARM CPUs and NVIDIA and AMD GPUs)
- waLBerla -> EU lighthouse code due to uncompromised performance decisions

CERFACS

EUROPEAN CENTRE FOR RESEARCH AND ADVANCED TRAINING IN SCIENTIFIC COMPUTING

FAU

Friedrich-Alexander-Universität
Technische Fakultät

Thank you very much for your attention!