Advanced Automatic Code Generation For Multiple Relaxation-Time Lattice Boltzmann Methods

Frederick Hennig, Markus Holzer, and Ulrich Rüde

Presented by Russell Bentley

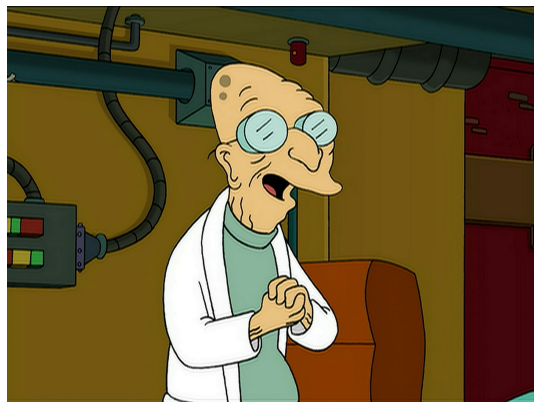
Stony Brook

2024

What are we even talking about?

ADVANCED AUTOMATIC CODE GENERATION FOR MULTIPLE RELAXATION-TIME LATTICE BOLTZMANN METHODS

- Present my graphics final project
  - Computational Fluid Dynamics (CFD)
  - MRT Lattice Boltzmann Methods (LBM)
  - Theory & Implementation
  - We'll understand the title (hopefully)
- Present paper
  - waLBerla, `pystencils`, and `lbmpy`
  - Implementation & Symbolic Simplification,
- Future Work



Graphics Final Introduction

- Final Project for Computer Graphic
  - Rough Goal to implement [11, 12]
- Lattice Boltzmann method (LBM)
  - Numerically model fluid dynamics
  - Known to struggle with turbulence
- CM-MRT: A new collision operator
  - Based on non-orthogonal moment space [5]
  - Adoption in graphics research [9–12]

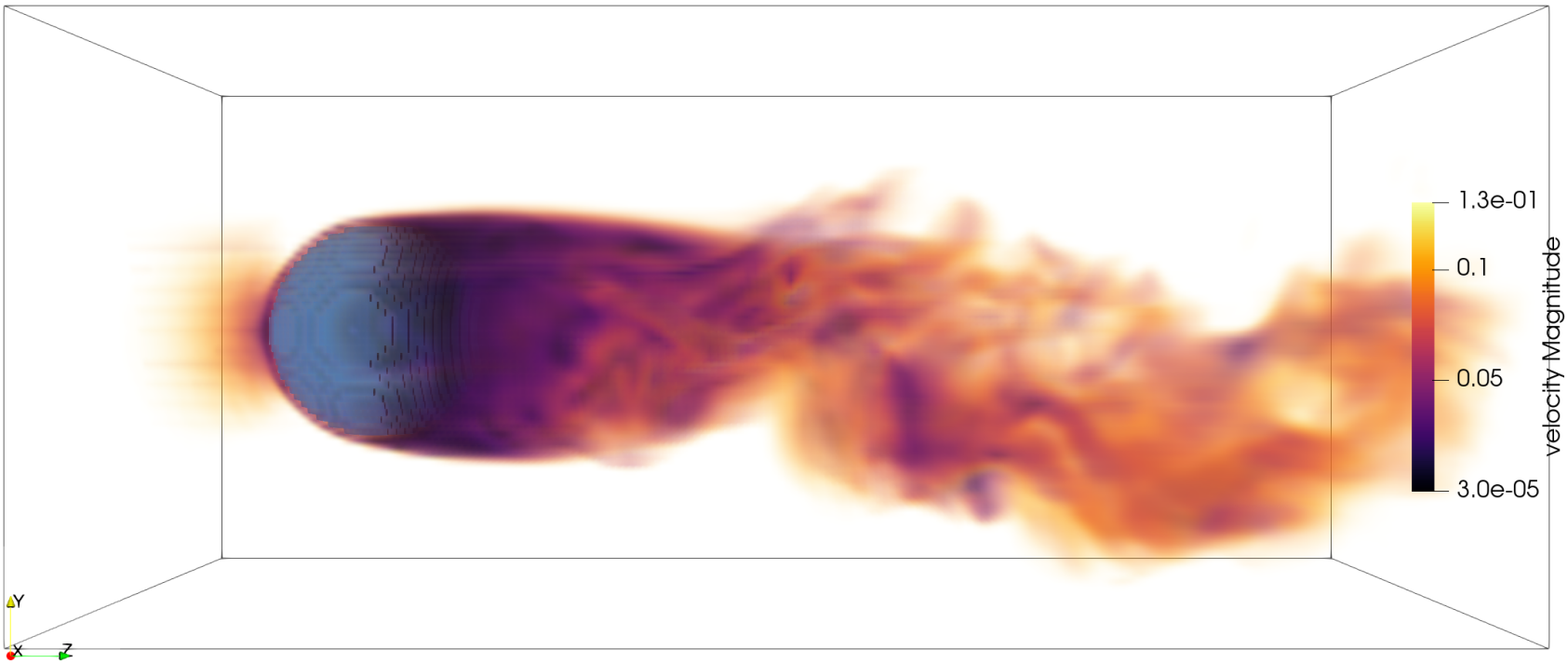
Fast and Scalable Turbulent Flow Simulation with Two-Way Coupling

WEI LI, ShanghaiTech University/SIMIT/UCAS
YIXIN CHEN, ShanghaiTech University/DGene
MATHIEU DESBRUN, ShanghaiTech/Caltech
CHANGXI ZHENG, Columbia University
XIAOPEI LIU, ShanghaiTech University

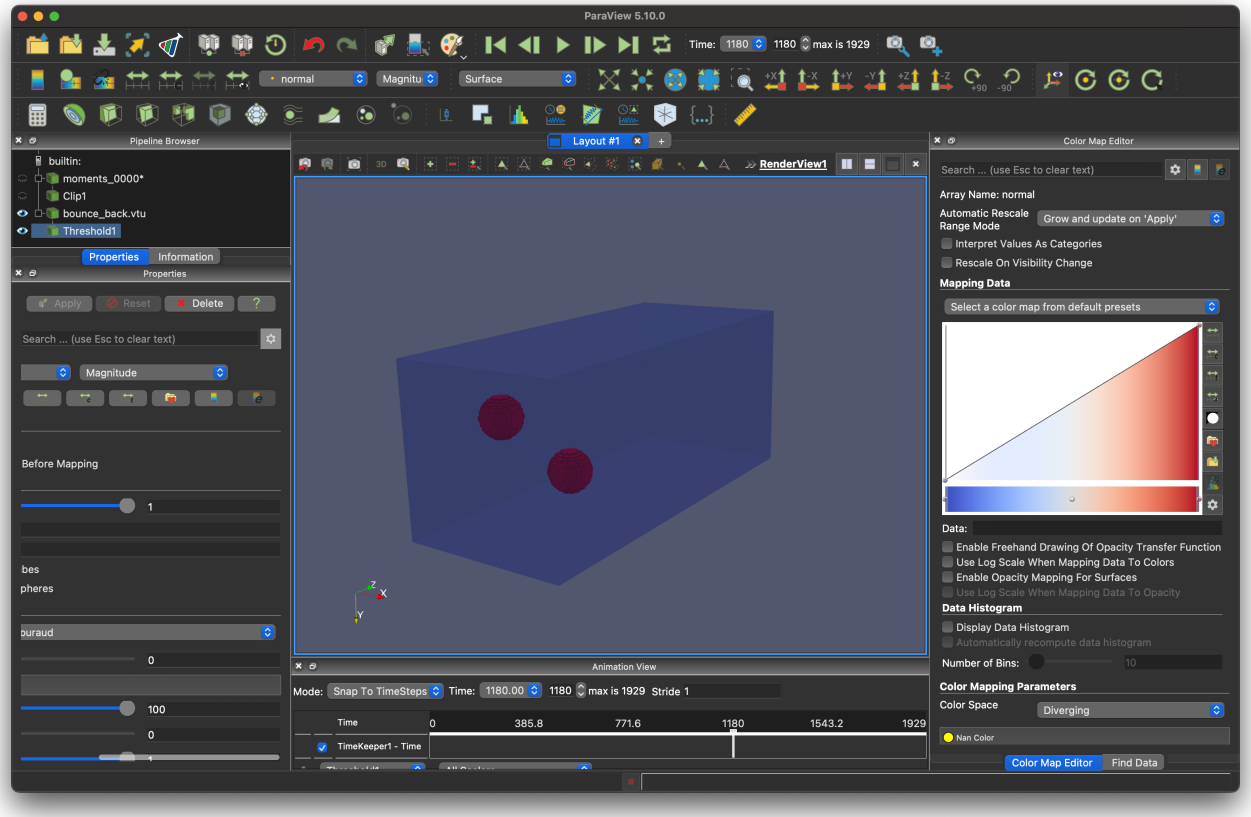
Fast and Versatile Fluid-Solid Coupling for Turbulent Flow Simulation

CHAOYANG LYU, ShanghaiTech University / SIMIT / UCAS
WEI LI, ShanghaiTech University / Inria
MATHIEU DESBRUN, Inria / Ecole Polytechnique / Caltech
XIAOPEI LIU, ShanghaiTech University

Demo



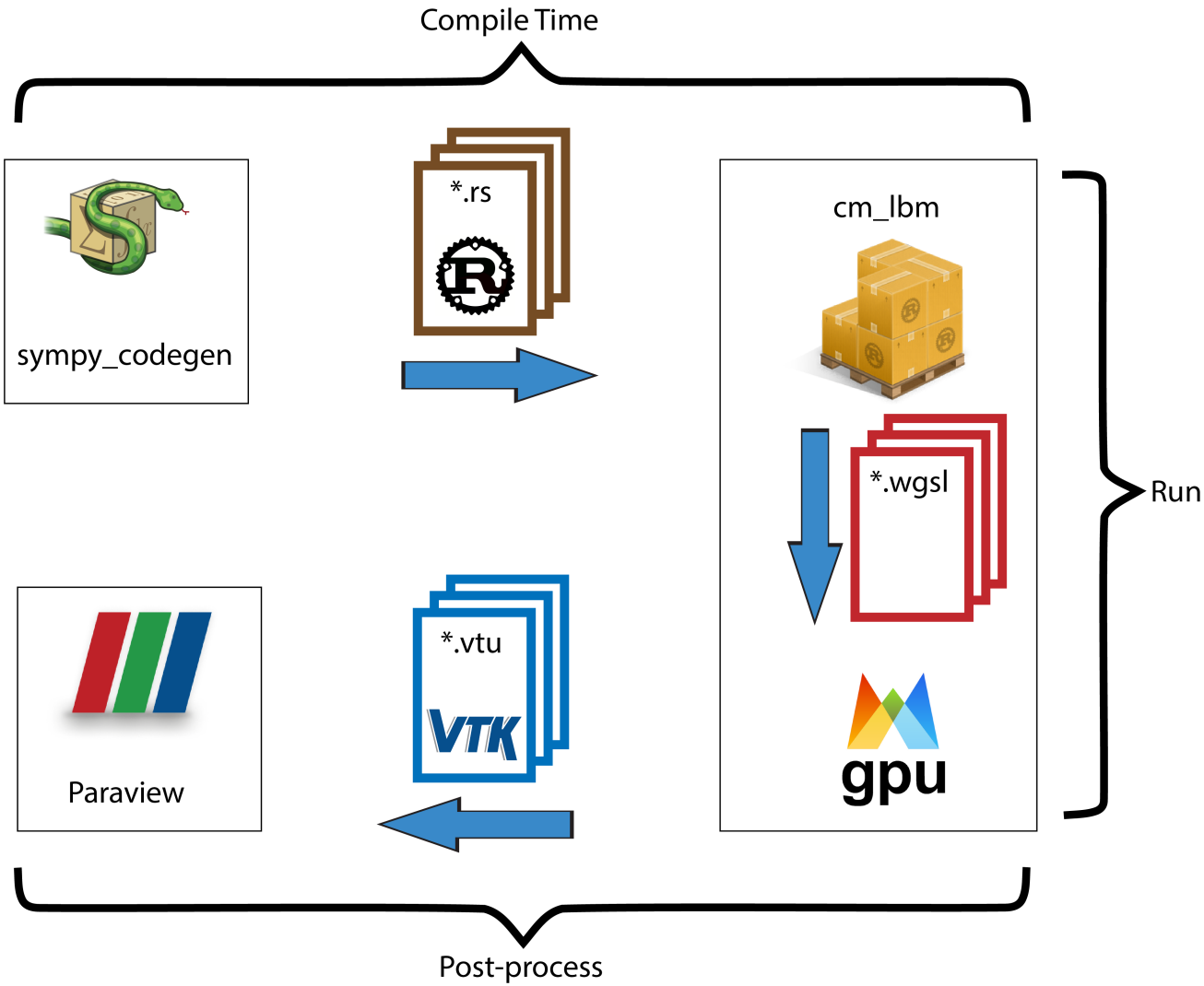
Paraview + VTK



https://www.paraview.org
https://vtk.org

Stony Brook University

Architecture



https://github.com/SallySoul/cm_lbm

Stony Brook University

Classic CFD

- Discretize space into control Volumes
- Solve for macroscopic fluid values
- Common to require global solve for pressure term

Conservation of Mass

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0$$

Conservation of Momentum

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u}) + \nabla \cdot (\rho \boldsymbol{u} \times \boldsymbol{u}) = -\nabla \boldsymbol{p} + \nabla \cdot \sigma$$

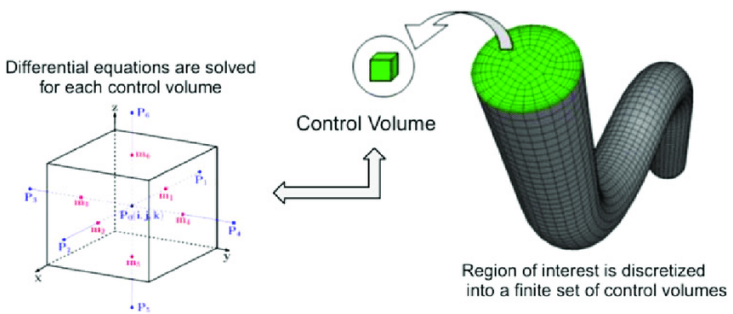
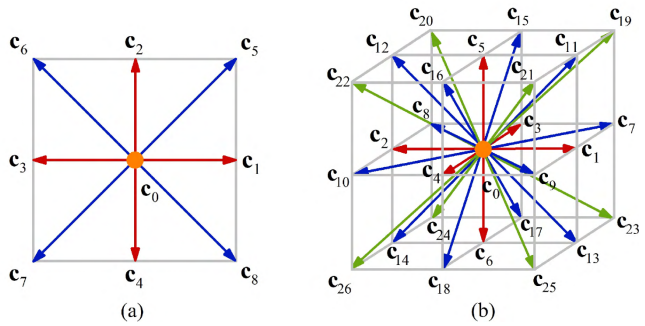


Figure from ResearchGate

Lattice Boltzmann method (LBM): Discretization

- Model microscopic particles
- Recover macroscopic properties
- D dimensional grid of points x
  - Particle position distribution
- Q dimensional "lattice" at each point
  - Particle velocity distribution $f(x, \xi, t)$
  - Directions c_i *streaming* and *collision*



(a) D2Q9 (b) D3Q27

$$c_i = \begin{cases} (0,0,0) & i = 0 \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm) & i = 1,\ldots,6 \\ (\pm 1,\pm 1,0),(\pm 1,0,\pm 1),(0,\pm 1,\pm 1) & i = 7,\ldots,18 \\ (\pm 1,\pm 1,\pm 1) & i = 19,\ldots,26 \end{cases}$$

Figure from [11]

LBM Theory Primer

- Lattice Boltzmann equation
- $f(x,t)$ approximates ξ
- Hermite Polynomial Basis [5]
  - Quadrature Rule with c_i.
- Collision operator, Ω
  - Relax towards equilibrium f_{eq}
- Moments of f are macroscopic quantities
- Advance time with two operators, *streaming* and *collision*

$$\partial_t f(x,\xi,t) + \xi \cdot \nabla f(x,\xi,t) = \Omega(f)$$

$$\rho(x,t) = \sum_{i=0}^{Q-1} f_i(x,t)$$

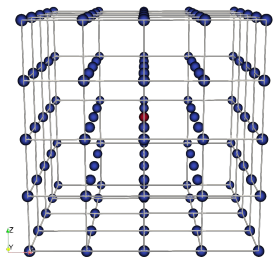
$$\rho(x,t)u(x,t) = \sum_{i=0}^{Q-1} c_i f_i(x,t)$$

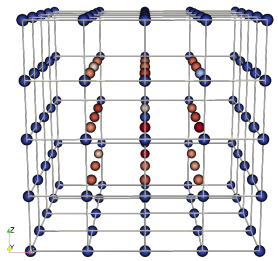
Figure from [11]

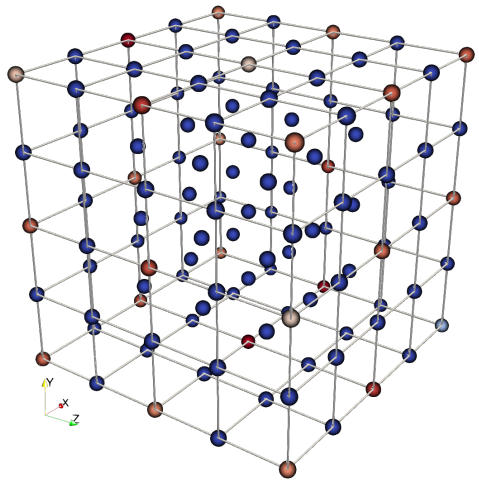
Streaming

- Non-local operator (Transport)
- Particles stream to next point in grid
- Information travels at speed of sound!

$$f_i^*(x,t) = f_i(x - c_i, t)$$







Collision - Single Relaxation Time (SRT)

- Collision is local
- Relax distributions towards equilibrium
- SRT is standard collision operator
  - Relaxes all distributions at constant rate
  - Also known as Bhatnagar-Gross-Krook (BGK)
- First order approximation
- Equilibrium from continuous Maxwell-Boltzmann distribution

$$\mathrm{SRT}(f) = \frac{1}{\tau}(f_{\mathrm{eq}} - f)$$

$$f_{\mathrm{eq}} = w_i \rho \left(1 + 3c_i \cdot u + \frac{9(c_i \cdot u)^2}{2} - \frac{3(u \cdot u)^2}{2}\right)$$

Multiple Relaxation Time (MRT)

- M transforms f into moment space
- 26 of them!
- Relax moments separately
- M^{-1} transforms result back to distribution

$$
\begin{aligned}
m_{0,j} &= \{1\}, \\
m_{1,j} &= \{c_{j,x}\}, \\
m_{18,j} &= \{c_{j,x}^2 c_{j,y}^2 + c_{j,x}^2 c_{j,z}^2 - c_{j,y}^2 c_{j,z}^2\}, \\
m_{19,j} &= \{c_{j,x}^2 c_{j,y}^2 - c_{j,x}^2 c_{j,z}^2\}, \\
m_{26,j} &= \{c_{j,x}^2 c_{j,y}^2 c_{j,z}^2\},
\end{aligned}
$$

$$
\mathrm{MRT}(f) = M^{-1} \cdot R \cdot M(f_{\mathrm{eq}} - f)
$$

Relaxation Rates, pt 1

- Independent relaxation rates for each moment
- Zeroth moment, density, should be conserved.
- Next three are momentum, $r_i = 2$ is common
- Next six moments are related to momentum flux
  - Relaxation rate is related to viscosity, v

$$R = \begin{bmatrix} r_0 & 0 & \cdots & 0 \\ 0 & r_1 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & & \cdots & r_{26} \end{bmatrix}$$

$$r_0 = 0,$$

$$r_i = 2, \text{ for } i = 1, 2, 3$$

$$r_i = (3v + 1/2)^{-1}, \text{ for } i = 4, \cdots, 9$$

Relaxation Rates, pt 2

- Higher-order relaxation rates are an area of active research
- [11] Describes an adaptive approach
- I implemented fixed rates as described in [9].

$$R = \begin{bmatrix} r_0 & 0 & \cdots & 0 \\ 0 & r_1 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & & \cdots & r_{26} \end{bmatrix}$$

$$r_i = (3v_i + 1/2)^{-1}$$

$$v_i = 0.005, \text{ for } i = 9, \ldots, 16$$

$$v_i = 0.007, \text{ for } i = 17, \ldots, 22$$

$$v_i = 0.009, \text{ for } i = 26$$

Central Moment MRT (CM-MRT)

- MRT violates Galilean invariance
  - u changes collision outcome
- Center Moments on u [4]
- $\bar{m}_\rho = M_u f_{eq}(u, \rho)$
  - Equilibrium in CM space
  - Only depends on ρ!

$$
\begin{aligned}
\bar{c}_j &= c_j - u \\
m_{0,j} &= \{1\}, \\
m_{1,j} &= \{\bar{c}_{j,x}\}, \\
m_{18,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 + \bar{c}_{j,x}^2 \bar{c}_{j,z}^2 - \bar{c}_{j,y}^2 \bar{c}_{j,z}^2\}, \\
m_{19,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 - \bar{c}_{j,x}^2 \bar{c}_{j,z}^2\}, \\
m_{26,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 \bar{c}_{j,z}^2\},
\end{aligned}
$$

$$
\text{CM-MRT}_{u,\rho}(f) = M_u^{-1} \cdot R \cdot (\bar{m}_\rho - M_u f)
$$

Central Moment MRT (CM-MRT)

- MRT violates Galilean invariance
- Center Moments on u [4]
- f_{eq} only depends on ρ in CM space

$$
\begin{aligned}
\bar{c}_j &= c_j - u \\
m_{0,j} &= \{1\}, \\
m_{1,j} &= \{\bar{c}_{j,x}\}, \\
m_{18,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 + \bar{c}_{j,x}^2 \bar{c}_{j,z}^2 - \bar{c}_{j,y}^2 \bar{c}_{j,z}^2\}, \\
m_{19,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 - \bar{c}_{j,x}^2 \bar{c}_{j,z}^2\}, \\
m_{26,j} &= \{\bar{c}_{j,x}^2 \bar{c}_{j,y}^2 \bar{c}_{j,z}^2\},
\end{aligned}
$$

$$
\text{CM-MRT}_{u,\rho}(f) = M_u^{-1} \cdot R \cdot (\bar{m}_\rho - M_u f)
$$

Code Example: SymPy

$$f_i^* = f_i + \Omega_i(f, \rho, u).$$

```
u = cm_mrt.u()
f = cm_mrt.f()
density = Symbol("density")
omega = Symbol("omega")
eq_op = cm_mrt.f_eq(density, u)
bgk_op = f + omega * (eq_op - f)

(source_body, debug_raw) = util.rust_generate_op(bgk_op.evalf())
```

```
u = cm_mrt.u()
f = cm_mrt.f()
density = Symbol("density")
riv = Symbol("riv")
m = cm_mrt.M(u)
m1 = m.inv()
r = cm_mrt.R(riv)
f = cm_mrt.f()
mbar = cm_mrt.MBar(density)
cm_mrt_op = f + m1 * r * (mbar - (m * f))

(source_body, debug_raw) = util.rust_generate_op(cm_mrt_op.evalf())
```

- Using SymPy was inspired by [7]
- Looks like a compiler

Code Example: Shader

```
result[0] = -0.0698812019566736*density + 1.88679245283019*q0*ux * ux*
* uy*uz * uz*(1.0 - ux) * (1.0 - ux) + 1.88679245283019*q10*uz * uz*(-
.0) + 1.88679245283019*q11*uy * uy*(1.0 - ux) * (1.0 - ux)*(1.0 - uz)
.0 - ux) * (1.0 - ux)*(-uz - 1.0) * (-uz - 1.0) + 1.88679245283019*q13
 (-ux - 1.0) + 1.88679245283019*q14*uy * uy*(-ux - 1.0) * (-ux - 1.0)*
15*ux * ux*(1.0 - uy) * (1.0 - uy)*(1.0 - uz) * (1.0 - uz) + 1.8867924
uz - 1.0) * (-uz - 1.0) + 1.88679245283019*q17*ux * ux*(1.0 - uz) * (1
5283019*q18*ux * ux*(-uy - 1.0) * (-uy - 1.0)*(-uz - 1.0) * (-uz - 1.0
x)*(1.0 - uy) * (1.0 - uy)*(1.0 - uz) * (1.0 - uz) + 1.88679245283019*
 1.88679245283019*q20*(1.0 - ux) * (1.0 - ux)*(1.0 - uy) * (1.0 - uy)*
21*(1.0 - ux) * (1.0 - ux)*(1.0 - uz) * (1.0 - uz)*(-uy - 1.0) * (-uy
0 - uy)*(1.0 - uz) * (1.0 - uz)*(-ux - 1.0) * (-ux - 1.0) + 1.8867924
 * (-uy - 1.0)*(-uz - 1.0) * (-uz - 1.0) + 1.88679245283019*q24*(1.0 -
uy - 1.0) * (-uy - 1.0) + 1.88679245283019*q25*(1.0 - uy) * (1.0 - uy)
- 1.0) + 1.88679245283019*q26*(-ux - 1.0) * (-ux - 1.0)*(-uy - 1.0) *
9245283019*q3*ux * ux*uz * uz*(1.0 - uy) * (1.0 - uy) + 1.886792452830
) + 1.88679245283019*q5*ux * ux*uy * uy*(1.0 - uz) * (1.0 - uz) + 1.88
(-uz - 1.0) + 1.88679245283019*q7*uz * uz*(1.0 - ux) * (1.0 - ux)*(1.0
```

- $\approx$ 5kb for moments shader
- 1.1mb for CM-MRT shader
- Cheap to compute
- Challenging to debug

Code Example: Rust

```
let q25 = f[25]; => f32
let q26 = f[26]; => f32
result[0] = -0.0698812019566736 * density
    + 1.88679245283019 * q0 * ux * ux * uy * uy * uz * uz
    + q0
    + 1.88679245283019 * q1 * uy * uy * uz * uz * (1.0 - ux) * (1.0 - ux)
    + 1.88679245283019
        * q10
        * uz
        * uz
        * (-ux - 1.0)
        * (-ux - 1.0)
        * (-uy - 1.0)
        * (-uy - 1.0)
    + 1.88679245283019
        * q11
        * uy
        * uy
        * (1.0 - ux)
        * (1.0 - ux)
        * (1.0 - uz)
        * (1.0 - uz)
```

- With formatting, $\approx$ 45k lines
- Easier to test and debug
- Tricky parts match shader exactly

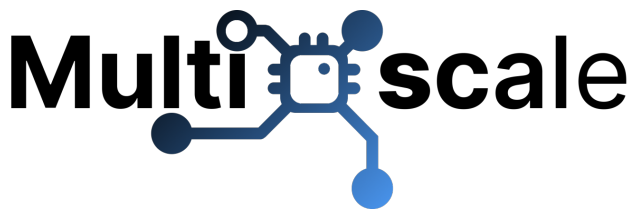
Graphics Final Conclusion

- How are we feeling about the paper title?
  - MRT Lattice Boltzmann methods?
  - Code generation using `sympy`?
- LBM + `sympy`
  - "A Literate Lattice Boltzmann Code" [8]

ADVANCED AUTOMATIC CODE GENERATION FOR MULTIPLE RELAXATION-TIME LATTICE BOLTZMANN METHODS

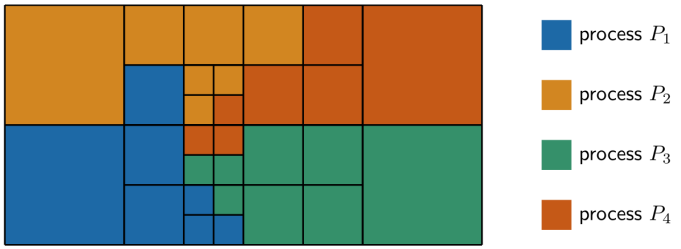
Paper Context

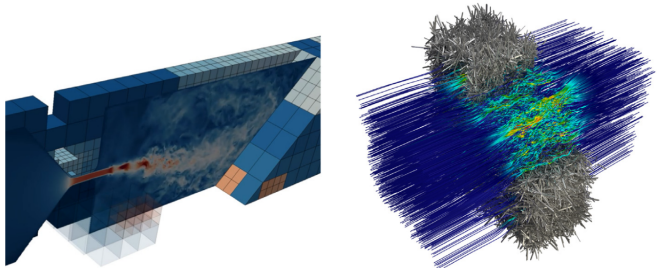
- Today's paper is part of a series
- Collaborative HPC work at German University
  - Fridrich-Alexander-Universität
- Also part of MultiXScale
  - European joint effort on exascale simulation



waLBerla

- waLBerla framework
  - Introduced in [3] (2021)
  - HPC-scale simulations
  - Stencil problems specifically
  - Bring your Compute Kernels
- Provides:
  - Multi-node domain partitioning
  - Communication patterns
  - Checkpoints



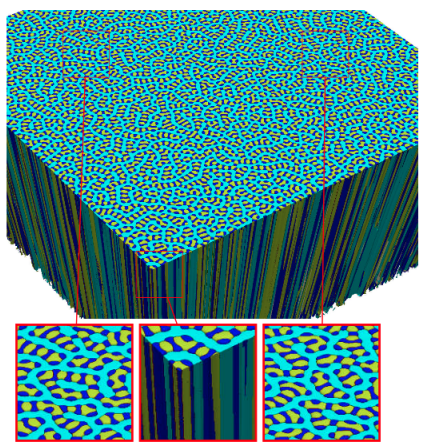


(a) Adaptive grid refinement for turbulent flow inside a vocal fold [40].

(b) Fluid flow through a porous crystal geometry [75].

pystencils

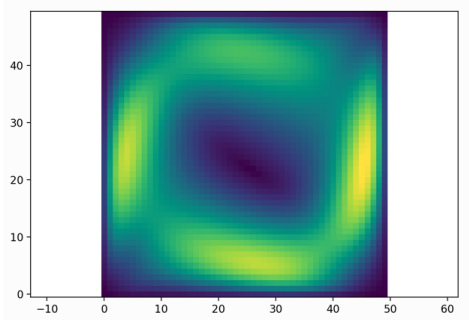
- Introduced in [2] (2019)
- Stencil Compiler
  - Generate C or CUDA source
  - `sympy` based IR
  - Assignments + Instructions
  - Static Single Assignment (SSA)
- Kernels can be
  - Compiled and used with `numpy`
  - Integrated with other code (e.g. waLBerla)



From [2]

lbmpy

- Compiler for LBM Kernels
  - Introduced in [1] (2021)
  - Built on `pystencils`
- Supports
  - Many grid configurations ($D3Q27$)
  - Arbitrary governing equations (f_{eq})
- Originally only supported SRT, MRT
- Deployable to exa-scale machines with waLBerla



Introduction

- Presenting [7] (2023)
- Re-architecture of lbmpy
  - Zero-centered Storage
  - Central Moment (CM)
  - Cumulants (K)
  - Support more generic governing eqns
- Minimizes arithmetic operations for CM and K methods
- Demonstrate small relative cost compared to SRT

ADVANCED AUTOMATIC CODE GENERATION FOR MULTIPLE RELAXATION-TIME LATTICE BOLTZMANN METHODS

FREDERIK HENNIG*, MARKUS HOLZER*†, AND ULRICH RÜDE*†

Abstract. The scientific code generation package *lbmpy* supports the automated design and the efficient implementation of lattice Boltzmann methods (LBMs) through metaprogramming. It is based on a new, concise calculus for describing multiple relaxation-time (MRT) lattice Boltzmann methods that enable the numerically advantageous subtraction of the constant background component from the populations. These techniques are generalized to a wide range of collision spaces and equilibrium distributions. The article contains an overview of *lbmpy*'s front-end and its code generation pipeline, which implements the new LBM calculus by means of symbolic formula manipulation tools and object-oriented programming. The generated codes have only a minimal number of arithmetic operations. Their automatic derivation rests on two novel Chinoera transforms that have been specifically developed for efficiently computing raw and central moments. Information contained in the symbolic representation of the methods is further exploited in a customized sequence of algebraic simplifications, further reducing computational cost. When combined, these algebraic transformations lead to concise and compact numerical kernels. Specifically, with these optimizations, the advanced central moment- and cumulant-based methods can be realized with only little additional cost as when compared with the simple BGK method. The effectiveness and flexibility of the new *lbmpy* code generation is demonstrated by simulating Taylor-Green vortex decay and the automatic derivation of an LBM algorithm to solve the shallow water equations.

Key words. Lattice Boltzmann Method, Metaprogramming, Code Generation

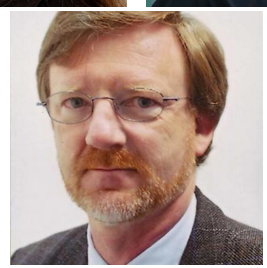
MSC codes. 65Y20, 82C80

1. Introduction. Modern scientific computing is characterized by growing complexity on several fronts. On the one side, the mathematical models become more advanced and powerful so that the corresponding numerical methods and algorithms grow increasingly intricate and involved. At the same time, contemporary computer architectures become more difficult to program. This is especially true in high end parallel computing that is characterized by rapidly evolving new hardware architectures, such as general-purpose graphics processing units (GPUs).

Thus the central step of scientific computing, i.e., the step from a model to executable software, faces increasing challenges. The developers of algorithms and scientific software are squeezed in from two sides: Increasingly complex simulation models must be realized in software for increasingly complex advanced supercomputer systems. We note here also that legacy software often lacks performance portability. For example it may present an almost prohibitive effort to port a legacy flow solver to a modern multi-GPU system. It may be speculative, but likely enormous computational resources are underused world wide, since outdated software is underperforming on the given computer systems on which it runs.

At the same time, we observe that modern mathematical methods have evolved to a state where their derivation and analysis depends increasingly on researchers making use of symbolic formula manipulation packages. For the present article, we will consider the class of lattice Boltzmann methods (LBMs), whose most advanced variants could hardly be derived without using symbolic mathematical software. Given this situation, it is a natural step to integrate the algebraic manipulations routinely into the workflow of designing and implementing these methods. A first step of this

*Chair for System Simulation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 91058 Germany (frederik.hennig@fau.de).
†CERFACS, 42 Avenue Gaspard Coriolis, 31100 Toulouse, France (holzer@cerfacs.fr).



Zero-centered Storage

- For incompressible hydrodynamics:
- f is generally close to f^0.
- So lets only store $\partial f = f - f^0$
- Improve numerical accuracy
- Can also use smaller float format!
- Adds code-gen complexity

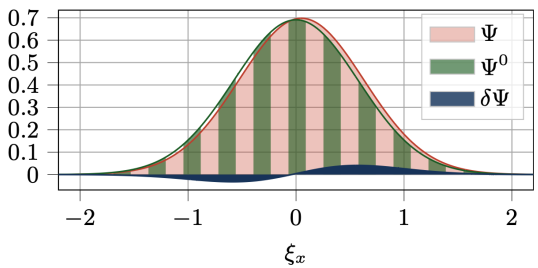


FIGURE 2.1. *Plot of* Ψ, Ψ^0 *and* $\delta\Psi$ *in one dimension at* $\rho = 1.01$, $u_x = 0.05$.

From [7]

Moment Transform, pt. 1

- Transform populations to Raw Moments (Mf)
- Use Chimera transform to compute moments
  - Introduced in [6]
- More efficient than Matrix transform
- Inverse uses trick to split populations
  $f_i^* = f_i^+ + f_i^-,\ f_{\bar{i}}^* = f_i^+ - f_i^-$

$$m_{xy|\gamma} := \sum_{z\in\{-1,0,1\}} f_{xyz} \cdot z^{\gamma}$$

$$m_{x|\beta\gamma} := \sum_{y\in\{-1,0,1\}} m_{xy|\gamma} \cdot y^{\beta}$$

$$m_{\alpha\beta\gamma} := \sum_{x\in\{-1,0,1\}} m_{x|\beta\gamma} \cdot x^{\alpha}$$

Moment Transform, pt. 2

- To compute central moments
- First compute raw moments $m_{\alpha\beta\gamma}$
- Then transform them to central moments $\kappa_{\alpha\beta\gamma}$

$$\kappa_{ab|\gamma} := \sum_{c=0}^{\gamma} \binom{\gamma}{c} (-u_z)^{\gamma-c} m_{abc}$$

$$\kappa_{a|\beta\gamma} := \sum_{c=0}^{\gamma} \binom{\gamma}{c} (-u_z)^{\gamma-c} m_{abc}$$

$$\kappa_{\alpha\beta\gamma} = \sum_{a,b,c=0}^{\alpha\beta\gamma} \binom{\alpha}{a} \binom{\beta}{b} \binom{\gamma}{c} (-u_x)^{\alpha-a} (-u_y)^{\beta-b} (-u_z)^{\gamma-c} m_{abc}$$

Simplification Passes

- Conserved Quantity Rewriting
  - Use moments instead of $\rho, \boldsymbol{u}$
- Propagation of Logarithms
  - Relevant for Cumulants
- Common Subexpression Elimination (CSE)
  - Make new assignments for them
- Expression Propagation
  - For assignments with constant RHS
  - Copy into use-sites
- Unused Subexpression Elimination
  - Remove unused assignments

$$\kappa_{000} = \rho$$
$$\kappa_{100} = -F_x/2$$
$$\kappa^*_{000} = \kappa_{000}$$
$$\kappa^*_{100} = \kappa_{100} + F_x$$
$$m^*_{10|0} = \kappa^*_{000}\boldsymbol{u}_x + \kappa^*_{100}$$

To

$$m^*_{10|0} = \rho\boldsymbol{u}_x + F_x/2$$

Symbolic Simplification Results

TABLE 4.1

Arithmetic operation counts in the symbolic representation of compute kernels generated by lbmpy *for several method definitions on the D2Q9, D3Q19 and D3Q27 stencils. Numbers for kernels derived without simplification (N), standard simplification (S) and simplification plus CSE (S+CSE) are listed.*

	D2Q9			D3Q19			D3Q27		
Method	N	S	S+CSE	N	S	S+CSE	N	S	S+CSE
SRT	432	113	**91**	1156	312	**204**	3842	428	**285**
TRT	437	191	**111**	1161	480	**239**	3847	668	**343**
O-MRT	196	142	**122**	554	397	**324**	928	596	**488**
R-O-MRT	196	105	**89**	554	290	**232**	928	407	**329**
WO-MRT	178	117	**102**	507	339	**283**	843	484	**415**
R-WO-MRT	178	87	**75**	507	244	**196**	843	316	**265**
CM	236	156	**132**	638	415	**344**	1140	747	**600**
R-CM	236	108	**95**	638	255	**216**	1140	417	**342**
K	676	167	**142**	1854	454	**376**	7623	1035	**820**
R-K	676	114	**100**	1854	272	**231**	7623	489	**397**

For reference, my had CM implementation (slight over estimate) $220,702$ arithmetic operations, 676 for SRT

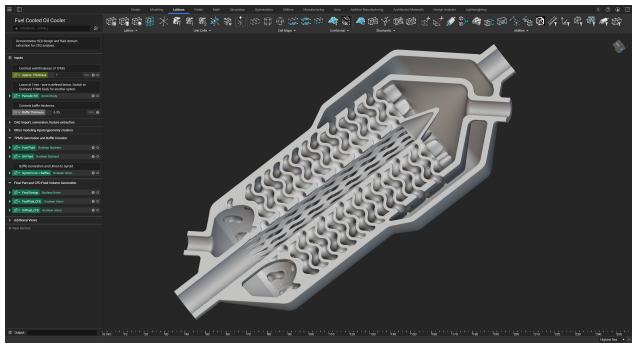
From [7]

Paper Conclusion

- `lbmpy` now has SOTA collision spaces
  - Far more efficient than other implementations
- High level symbolic representation
  - Easy to experiment with variant methods
  - Easy to target alternate architectures
- Easy to scale with waLBerla

Future Work (For Me)

- Tracer Particles
- Boundary Conditions
  - Generate code for [12]?
  - Implicit Geometry Support?
- Problem Setup
  - Unit conversions,
  - Adaptive relaxation from [11]
- Alternative Compute Targets (Metal)



From ntop, [12]

Conclusion

- Today we covered:
  - Lattice Boltzmann Methods
  - Central Moments (CM) collision Space
  - Code generation with `sympy`
- Questions?

References I

[1] 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.

[2] Martin Bauer et al. "Code generation for massively parallel phase-field simulations". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2019, pp. 1–32.

[3] Martin Bauer et al. "waLBerla: A block-structured high-performance framework for multiphysics simulations". In: *Computers & Mathematics with Applications* 81 (2021), pp. 478–501.

[4] Alessandro De Rosis. "Nonorthogonal central-moments-based lattice Boltzmann scheme in three dimensions". In: *Physical Review E* 95.1 (2017), p. 013310.

References II

[5] Alessandro De Rosis and Kai H Luo. "Role of higher-order Hermite polynomials in the central-moments-based lattice Boltzmann framework". In: *Physical Review E* 99.1 (2019), p. 013301.

[6] Martin Geier et al. "The cumulant lattice Boltzmann equation in three dimensions: Theory and validation". In: *Computers & Mathematics with Applications* 70.4 (2015), pp. 507–547.

[7] Frederik Hennig, Markus Holzer, and Ulrich Rüde. "Advanced automatic code generation for multiple relaxation-time lattice Boltzmann methods". In: *SIAM Journal on Scientific Computing* 45.4 (2023), pp. C233–C254.

[8] Adrian Kummerländer. *A literate Lattice Boltzmann Code*. http://literatelb.org. 2020.

References III

[9] Wei Li, Kai Bai, and Xiaopei Liu. "Continuous-scale kinetic fluid simulation". In: *IEEE transactions on visualization and computer graphics* 25.9 (2018), pp. 2694–2709.

[10] Wei Li, Kui Wu, and Mathieu Desbrun. "Kinetic Simulation of Turbulent Multifluid Flows". In: *ACM Transactions on Graphics (TOG)* 43.4 (2024), pp. 1–17.

[11] Wei Li et al. "Fast and scalable turbulent flow simulation with two-way coupling.". In: *ACM Trans. Graph.* 39.4 (2020), p. 47.

[12] Chaoyang Lyu et al. "Fast and versatile fluid-solid coupling for turbulent flow simulation". In: *ACM Transactions on Graphics* 40.6 (2021), p. 201.