TNL: Numerical library for modern parallel architectures

T. Oberhuber[1], J. Klinkovský[1], T. Halada[2], R. Fučík[1], P. Eichler[2], R. Straka[3]

[1]Faculty of Nuclear Science and Physical Engineerings, Czech Technical University in Prague,
[2]Faculty of Mechanical Engineering, Czech Technical University in Prague,
[3]AGH University of Krakow, Department of Heat Engineering and Environment Protection.

HPCSE 2026

Overview

TNL: Template Numerical Library

TNL = Template Numerical Library

- numerical library for modern parallel architectures
- offers unified interface for both multi-core CPUs a GPUs
- written in C++ profiting from features C++17
- $\approx$ 300k lines of code and documentation
- **tnl-project.org**
- MIT license
- affiliated project of Numfocus (numfocus.org)

NUMFOCUS [AFFILIATED PROJECT]

TEMPLATE NUMERICAL LIBRARY

Core features

Backends for various hardware platforms:

- `TNL::Devices::Host` - CPU
- `TNL::Devices::Cuda` - CUDA GPUs
- `TNL::Devices::Hip` - HIP GPUs

Parallel primitives:

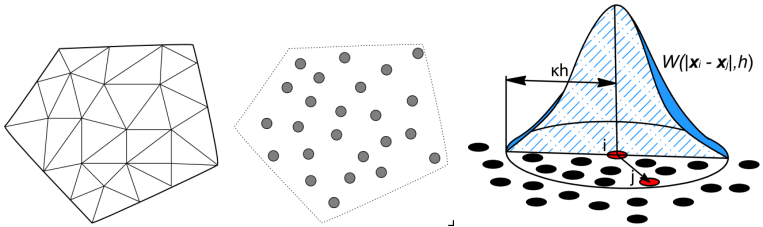
- `parallelFor`, `reduction`, `scan`, `sort`

Higher-level containers:

- Vectors supporting **expression templates**
- N-dimensional arrays (inspired by NumPy)
- matrices, graphs, meshes, grids, etc.

Smoothed Particle Hydrodynamics

SPH is a meshfree particle method based on Lagrangian continuum description



- material particles without mutual topological connections
- value of function at some point is determined as the weighted contribution of the surrounding particles

SPH: Smoothed Particle Hydrodynamics

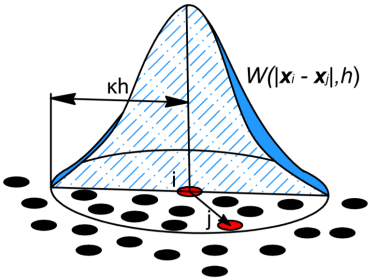
The computational domain Ω is covered by finite number of points (particles), each of them carries value f_i on elementary volume V_i.

$$\langle f(\mathbf{x})\rangle = \int_\Omega f(\mathbf{x}')W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}'$$

$$\rightarrow \quad \langle f(\mathbf{x})\rangle_i = \sum_{i=1}^{N} f_i\, W(\mathbf{x}-\mathbf{x}_i,h)V_i$$

$$\langle \nabla f(\mathbf{x})\rangle = -\int_\Omega f(\mathbf{x}')\cdot\nabla W(\mathbf{x}-\mathbf{x}',h)d\mathbf{x}'$$

$$\rightarrow \quad \langle \nabla f(\mathbf{x})\rangle_i = -\sum_{i=1}^{N} f_i\, \nabla W(\mathbf{x}-\mathbf{x}_i,h)V$$



Technical realisation of SPH method

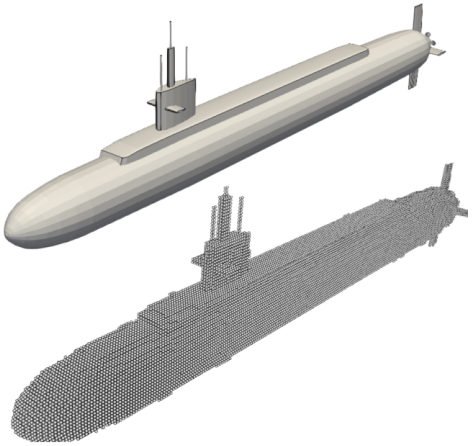
Algorithm Explicit SPH program

- **discretize domain into particles**

while $t < t_{\text{end}}$ **do**

- search for particle neighbors
- perform particle interactions
- integration

end while



Technical realisation of SPH method

Algorithm Explicit SPH program

- discretize domain into particles

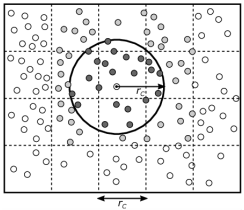
while $t < t_{\text{end}}$ **do**

- **search for particle neighbors**
- perform particle interactions
- integration

end while

Cell list algorithms

- used in most technical applications



- with explicitly stored neighbors - Verlet list
- without explicitly stored neighbors

Technical realisation of SPH method

Algorithm Explicit SPH program

- discretize domain into particles

while $t < t_{\text{end}}$ **do**

- search for particle neighbors
- **perform particle interactions**
- integration

end while

$$\frac{\mathrm{D}\rho_i}{\mathrm{D}t} = \frac{1}{\rho_i}\sum_{j=1}^{N}(\mathbf{u}_i - \mathbf{u}_j)\cdot\nabla_i W_{ij} V_j$$

$$\frac{\mathrm{D}\mathbf{u}_i}{\mathrm{D}t} = -\frac{1}{\rho_i}\sum_{j=1}^{N}(p_i + p_j)\nabla_i W_{ij} V_j + \alpha h c_0 \mathscr{V} + \mathbf{f}_i$$

$$p = b\left[\left(\frac{\rho}{\rho_0}\right)^{\gamma} - 1\right] + p_0$$

Technical realisation of SPH method

Algorithm Explicit SPH program

- discretize domain into particles

while $t < t_{\text{end}}$ **do**

- search for particle neighbors
- perform particle interactions
- **integration**

end while

Integration scheme

- symplectic integration schemes
  - Verlet scheme
  - Leimkuhler Matthews scheme (modified Verlet scheme)
- RK23, RK45

Technical realisation of SPH method

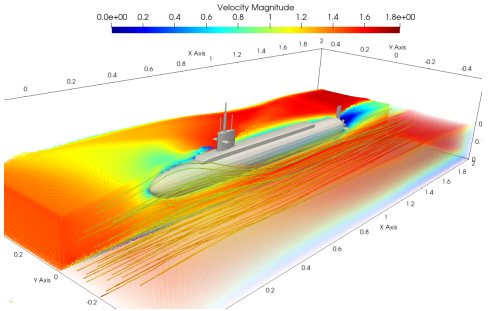
Algorithm Explicit SPH program

- discretize domain into particles

while $t < t_{\text{end}}$ **do**

- search for particle neighbors
- perform particle interactions
- integration

end while



TNL-SPH module for the Smoothed particle hydrodynamics

TNL-SPH

Implementation of SPH based on TNL:

- Weakly Compressible SPH - WCSPH,
- Riemann SPH - RSPH,
- WCSPH with Boundary Integrals - WCSPH-BI,

Modular architecture with container `Particles` offering:

- neighbor search based on cell lists or Verlet lists,
- neighbor traversal with lambda-based interaction kernels,
- CUDA/HIP backends,
- distributed multi-GPU execution,

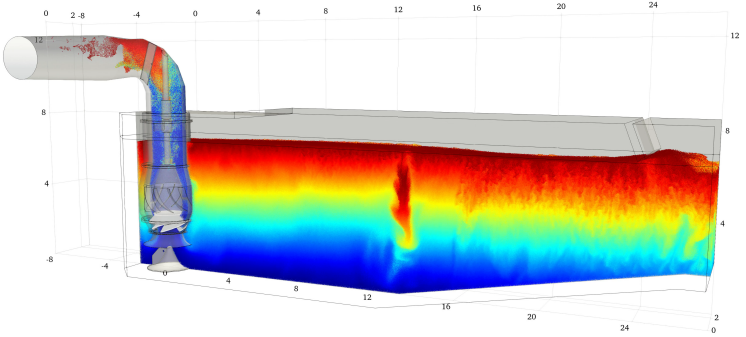
T. Halada, L. Beneš, J. Klinkovský, T. Oberhuber, *TNL-SPH: Open-source modular SPH solver for modern computing platforms based on GPU accelerators*, to appear in Computer Physics Communications, 2026.

TNL-SPH: Benchmarking

GPU	TNL-SPH		DualSPHysics			OpenFPM		
	t_{step} [s]	MPUPS	t_{step} [s]	MPUPS	Speed-up	t_{step} [s]	MPUPS	Speed-up
A40	0.023	172.3	0.029	139.6	**1.23**×	0.035	114.9	**1.49**×
A100	0.024	170.2	0.027	148.2	**1.15**×	0.028	143.4	**1.18**×
H100 SXM	0.013	320.5	0.015	276.2	**1.16**×	—	—	—
H100 NVL	0.014	280.4	0.017	241.6	**1.16**×	0.019	213.6	**1.31**×
GTX 1080Ti	0.082	49.0	0.080	50.3	**0.97**×	0.100	39.9	**1.22**×
RTX 5000	0.042	95.1	0.054	73.7	**1.29**×	0.060	66.5	**1.43**×
L40s	0.011	358.7	0.015	270.7	**1.32**×	0.016	250.5	**1.43**×
RTX A4000	0.046	87.5	0.060	66.5	**1.31**×	0.067	59.9	**1.46**×

Table: SPHERIC Test 02 – 3D dam break benchmark with $\Delta x = 0.006\,\text{m}$ corresponding approximately to 4×10^6 particles. The table shows average computation time per simulation step and millions of particle updates per second (MPUPS).

TNL-SPH: Pump simulation



Lattice Boltzmann Method

- LBM describes fluid flow by particle distribution functions

$$f_i(\mathbf{x}, t),$$

  where f_i represents the density of particles located at position $\mathbf{x}$, time t, and moving with microscopic velocity $\boldsymbol{\xi}_i, i = 1, \ldots, Q$.
- The discrete evolution equation is

$$\underbrace{f_i(\mathbf{x} + \boldsymbol{\xi}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t)}_{\text{streaming}} = \underbrace{\Omega_i(\mathbf{x}, t)}_{\text{collision term}}, i = 1, \ldots, Q.$$

- Macroscopic density and momentum are recovered as moments of the distribution functions:

$$\rho(\mathbf{x}, t) = \sum_{i=1}^{Q} f_i(\mathbf{x}, t), \qquad \rho\mathbf{u}(\mathbf{x}, t) = \sum_{i=1}^{Q} f_i(\mathbf{x}, t)\boldsymbol{\xi}_i.$$

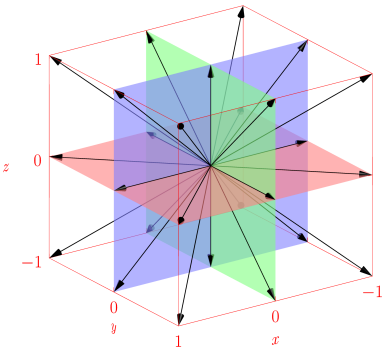
LBM: Discrete velocity models

Different discrete models $DdQq$ are used

- d = spatial dimension,
- q = number of discrete velocities,

like for example

- **D1Q3** – standard 1D model,
- **D2Q9** – standard 2D model,
- **D3Q27** – standard 3D model.



LBM: Streaming strategies

The streaming step propagates the particle distribution functions to neighboring lattice nodes along the discrete velocity directions $\boldsymbol{\xi}_i$.

Streaming strategies

Strategy	In-place	GPU-friendly	Memory traffic	Complexity
Push streaming	No	Medium	High	Low
Pull streaming	No	High	High	Low
Fused stream-collide kernels	Usually no	High	Medium	Medium
AA-pattern	Yes	Very high	Low	High
Swap algorithms	Yes	Medium	Medium	High

LBM: Collision operators

The collision operator relaxes the distribution functions toward equilibrium determined by local macroscopic quantities.

The more advanced collision operators offer:

- better stability
- better accuracy
- lower numerical viscosity
- allows to simulate turbulent flows at high Reynolds numbers

Collision is local for each lattice node.

Single Relaxation Time, SRT
↓
Two Relaxation Time, TRT
↓
Multiple Relaxation Time, MRT
↓
Central moments
↓
Cumulants
↓
Entropic LBM

LBM: Parallel implementation

LBM exhibits the following properties:

- Highly regular memory access patterns - nD-array
- Streaming is linear advection
- Collision step is fully local
- Explicit time integration - no need to solve linear systems
- Naturally suited for massive GPU parallelism

TNL-LBM module for the lattice Boltzmann method

TNL-LBM

Implementation of LBM based on TNL:

- lattice stored in `NDArray`,
- CUDA/HIP backends,
- MPI-based multi-GPU execution,

Modular architecture with pluggable components:

- collision operators - SRT, MRT, KBC, CLBM, CuLBM,
- streaming patterns - pull, AA-pattern,
- boundary conditions,
- macroscopic quantities,
- discrete equilibrium functions.

J. Klinkovský, P. Eichler, R. Straka, T. Oberhuber, R. Fučík, *TNL-LBM: Scalable lattice Boltzmann method implementation based on Template Numerical Library*, The Journal of Supercomputing, vol. 82, article no. 167, 2026.

TNL-LBM module for the lattice Boltzmann method

Benchmark problem: channel flow with D3Q27 model, cumulant collision operator, lattice size 512^3, in single-precision.

		Karolina			LUMI			
N_{nodes}	N_{GPUs}	GLUPs	Sp	Eff	N_{GPUs}	GLUPs	Sp	Eff
1	1	**5.40**	1.00	1.00	1/2	**0.45**	1.00	1.00
	2	**10.83**	2.01	1.00	1	**0.93**	2.08	1.04
	4	**21.79**	4.04	1.01	2	**1.72**	3.85	0.96
	8	**43.40**	8.04	1.00	4	**3.61**	8.12	1.01
2	16	**85.90**	15.91	0.99	8	**8.06**	18.11	1.13
4	32	**166.90**	30.92	0.97	16	**59.59**	133.90	4.18
8	64	**256.84**	47.58	0.74	32	**107.04**	240.53	3.76
16	128	**263.62**	48.84	0.38	64	**132.95**	298.77	2.33

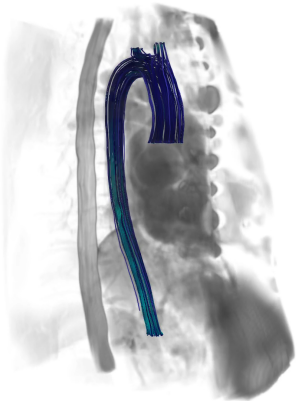
Karolina: Nvidia A100 (19.5 TFLOPS, 1.5 TB/s),
LUMI: Dual chip AMD Instinct MI250x (47.9 TFLOPS, 3.2 TB/s).

TNL-LBM module for the lattice Boltzmann method

Project	Case	Collision	Single precision	Double precision
waLBerla	Lid-driven cavity	Cumulant	3.120 GLUPS	0.656 GLUPS
TNL-LBM	Channel flow	Cumulant	2.892 GLUPS	0.354 GLUPS
VirtualFluids	Lid-driven cavity	Cumulant	2.708 GLUPS	0.167 GLUPS
OpenLB	Lid-driven cavity	BGK	2.509 GLUPS	0.411 GLUPS
Palabos	Lid-driven cavity	Cumulant	0.197 GLUPS	0.191 GLUPS
TCLB	Channel flow	Cumulant	0.680 GLUPS	0.283 GLUPS

Performance comparison of open-source LBM projects on NVIDIA RTX A5000 (lattice size N 2563, velocity set D3Q27)

TNL-LBM: Restoration of blood flow from MRI



FEM: The Finite Element Method

GPU challenges

Finite-element method requires:

- unstructured meshes,
- sparse linear algebra.

Both exhibit:

- irregular memory access,
- indirect indexing,
- irregular workload distribution,
- difficult mapping of GPU threads.

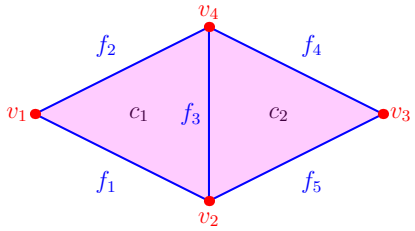
FEM: Unstructured numerical meshes

Unstructured numerical mesh is defined by:

- sets mesh entities,
  - vertexes - 0D entities,
  - edges - 1D entities,
  - faces - 2D entities,
  - cells - 3D entities,
- coordinates of the vertexes,
- adjacency and incidence of mesh entites.

Both **adjacency and incidence** can be represented in the form of **sparse boolean matrices**.

FEM: Unstructured numerical meshes



$$I_{0,1} = \left(\begin{array}{c|ccccc} & f_1 & f_2 & f_3 & f_4 & f_5 \\ \hline v_1 & 1 & 1 & & & \\ v_2 & 1 & & 1 & & 1 \\ v_3 & & & & 1 & 1 \\ v_4 & & 1 & 1 & 1 & \end{array}\right) \qquad I_{0,2} = \left(\begin{array}{c|cc} & c_1 & c_2 \\ \hline v_1 & 1 & \\ v_2 & 1 & 1 \\ v_3 & & 1 \\ v_4 & 1 & 1 \end{array}\right)$$

FEM: Segments

Efficient data structures and algorithms for sparse matrices

- Performance on GPU is strongly affected by the sparsity pattern of the matrix.
- TNL provides a data abstraction for sparse-matrix formats called **segments**.

FEM: Segments

Segments: data abstraction for sparse-matrix formats

- The following formats (segments) are supported:
  - CSR, Adaptive CSR, Ellpack, Sliced Ellpack, Chunked Ellpack, Bisection Ellpack.
- All formats can be combined with **row-length-based sorting**.
- Ellpack-based formats support **row-major** or **column-major ordering**.
- Various **threads mapping strategies** for
  - parallel traversing of matrix rows
  - parallel reduction within rows.
- Sparse matrices can be: general, symmetric and **boolean (binary)**.

FEM: Unstructured meshes

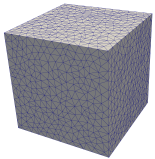
`Mesh` is a data structure for unstructured meshes in TNL:

- **Due to C++ templates, the mesh is fine-tuned for specific numerical method at compile-time.**
- supports CUDA/HIP backends for GPU execution,
- can have arbitrary dimension.

J. Klinkovský, T. Oberhuber, R. Fučík, V. Žabka, *Configurable open-source data structure for distributed conforming unstructured homogeneous meshes with GPU support*, ACM Transactions on Mathematical Software, vol. 48, no. 3., art. no. 3, 2022.

TNL-MHFEM: Module for the Mixed-hybrid Finite Element Method

TNL-MHFEM



Solver the following advection-diffusion problem:

$$\sum_{j=1}^{n} N_{i,j}\frac{\partial Z_j}{\partial t}+\sum_{j=1}^{n}\mathbf{u}_{i,j}\cdot\nabla Z_j+\nabla\cdot\left[m_i\left(-\sum_{j=1}^{n}D_{i,j}\nabla Z_j+\mathbf{w}_i\right)+\sum_{j=1}^{n}Z_j\mathbf{a}_{i,j}\right]+\sum_{j=1}^{n}r_{i,j}Z_j=f_i$$

for $i=1,...,n$, where the unknown vector function $\vec{Z}=(Z_1,...,Z_n)^T$ depends on position vector $\vec{x}\in\Omega\subset\mathbb{R}^d$ and time $t\in[0,T]$, $d=1,2,3$.

TNL-MHFEM:McWhorter–Sunada problem from porous media flow

- Based on the mixed-hybrid finite element method (MHFEM)
- Semi-implicit time discretization
- General spatial dimension (1D, 2D, 3D)
- Structured and unstructured meshes

R. Fučík, J.Klinkovský, T. Oberhuber, J. Mikyška, *Multidimensional Mixed–Hybrid Finite Element Method for Compositional Two–Phase Flow in Heterogeneous Porous Media and its Parallel Implementation on GPU*, Computer Physics Communications, vol. 238, pp. 165-180, 2019.

TNL-MHFEM: McWhorter–Sunada problem 3D

	GPU	CPU										
		1 core		2 cores			6 cores			12 cores		
Id.	CT	CT	GSp_1	CT	Eff	GSp_2	CT	Eff	GSp_6	CT	Eff	GSp_{12}
$3D_1^{\triangle}$	0,3	0,4	1,09	0,2	0,80	0,68	0,1	0,41	0,44	0,2	0,18	**0.50**
$3D_2^{\triangle}$	0,6	1,6	2,78	0,9	0,87	1,59	0,5	0,60	0,78	0,4	0,36	**0.65**
$3D_3^{\triangle}$	3,5	67,4	19,35	35,8	0,94	10,28	15,7	0,71	4,52	7,8	0,72	**2.24**
$3D_4^{\triangle}$	54,2	2 918,9	53,87	1 551,1	0,94	28,63	786,7	0,62	14,52	396,1	0,61	**7.31**
$3D_5^{\triangle}$	2 654,8						46 014,7		17,33	23 949,5		**9.02**

Measured on Intel Xeon Gold 6146 and Nvidia Tesla V100.

Distributed McWhorter–Sunada problem 3D based on MPI

	GPU						
	1 rank	2 ranks		3 ranks		4 ranks	
Id.	CT	CT	Sp_2	CT	Sp_3	CT	Sp_4
$3D_1^{\triangle}$	0,3	0,5	0,6	0,7	0,5	0,8	**0.4**
$3D_2^{\triangle}$	0,6	0,9	0,7	1,1	0,5	1,3	**0.5**
$3D_3^{\triangle}$	3,5	4,2	0,8	4,8	0,7	5,7	**0.6**
$3D_4^{\triangle}$	54,2	36,7	1,5	33,4	1,6	30,6	**1.8**
$3D_5^{\triangle}$	2 654,8	1 415,4	1,9	996,7	2,7	793,3	**3.3**

Conclusion

TNL offers **abstractions for various numerical methods**, including:

- SPH – irregular particle interactions,
- LBM – regular stencil computations,
- FEM – irregular mesh traversal and sparse linear algebra.

We are currently working on:

- FVM solver for compressible flow (Google Summer of Code 2025).

Future plans include:

- Hybrid solvers combining different numerical methods (LBM+MHFEM [1], LBM+SPH, etc.).

[1] Klinkovský, J., Trautz, A. C., Fučík, R., Illangasekare, T. H., *Lattice Boltzmann method–based efficient GPU simulator for vapor transport in the boundary layer over a moist soil: Development and experimental validation*. Computers & Mathematics with Applications, 138, pp. 65-87, 2023.

Conclusion

Thank you for your attention!

	**TNL**	`https://gitlab.com/tnl-project/tnl`
	**TNL-SPH**	`https://gitlab.com/tnl-project/tnl-sph`
	TNL-LBM	`https://gitlab.com/tnl-project/tnl-lbm`
	**TNL-MHFEM**	`https://gitlab.com/tnl-project/tnl-mhfem`