TNL-SPH: Open-source modular SPH solver for modern distributed computing platforms based on GPU accelerators

T. Halada, L. Beneš
Faculty of Mechanical Engineering
Czech Technical University in Prague
Prague, Czech Republic
tomas.halada@fs.cvut.cz

J. Klinkovský, T. Oberhuber
Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague
Prague, Czech Republic

In this work, we present a novel open-source Smoothed Particle Hydrodynamics implementation targeting modern distributed computing platforms based on GPU accelerators. TNL-SPH focuses on hydrodynamic problems using primarily weakly compressible formulations but uses a modular and general design that allows the simple implementation of different SPH schemes and even different particle methods.

The code is being developed as a submodule of Template Numerical Library (TNL) [1], [2] which is an open source C++ library developed at FNSPE CTU in Prague, designed with the intention of simplifying the development of high performance numerical codes on modern parallel architectures, especially on GPUs. Similarly to TNL, TNL-SPH is implemented in C++ (currently C++17) using modern programming paradigms. The code is available at https://gitlab.com/tnl-project/tnl-sph.

I. STRUCTURE AND DESING OF TNL-SPH

In contrast to most of the other SPH codes, TNL-SPH is based on multilayer design. The code is a header-only library, and the only part that may be built are the attached examples.

A. Architecture dependent structures

The architecture dependent primitives (such as memory management and parallel algorithms) are implemented by the TNL library, independently of the actual numerical code. TNL currently supports multicore CPUs, GPUs both by NVIDIA and AMD and distributed systems, all managed via unified interface.

B. Particles data structure

On top of that, general particle data structure is implemented, providing neighbor search algorithms and general tools to work with a set of arbitrarily distributed points. The particle structure provides functions to loop over the neighbors which as an argument take a lambda function specifying the inter-particle interactions. The design based on lambda functions is inherent to the whole library, and thus the particle structure corresponds to other parallel algorithms.

Features such as domain decomposition or selection of particles in particular spatial subdomains are implemented, which allows distributed and multi-GPUs computations and which are useful for some of the SPH algorithms such as multiresolution.

The particle structure can be viewed as an analogy to structure for unstrucuted meshes in mesh based methods on top of which particular numerical methods are implemented.

C. Particles solver: Particle automaton

The third layer is based on the shared properties of particle methods and the idea of a general particle automaton. Various particle-based algorithms for solving PDEs using methods such as SPH or GFDM through item-based models such as MD or DEM to multi-agent modeling work on similar principle. There is a set of particles without any topological connections in between which carry certain properties and which interact with each other, the interaction being limited to a certain distance (usually the nearest neighborhood of the particles). This concept was properly mathematically analyzed and defined by Pahlke [3] as the *Unifying particle method*, which encompasses all common particle simulation tools.

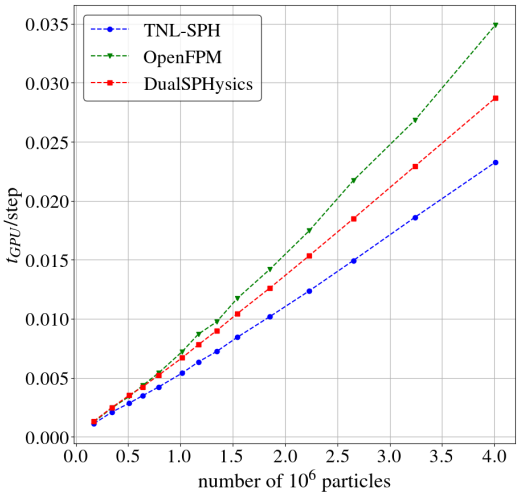

Fig. 1. Comparison of average computation time per single simulation time step of 3-dimensional dam break problem using NVIDIA A40.

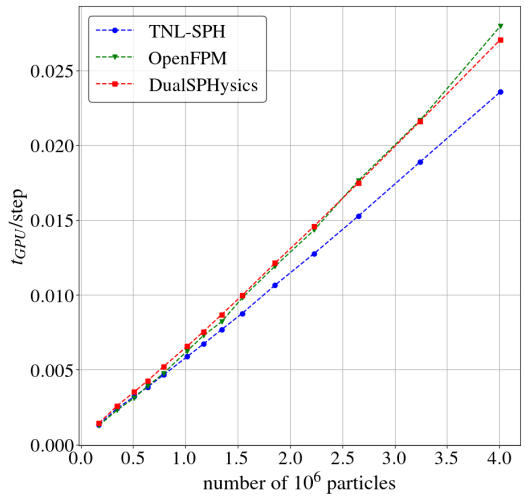

Fig. 2. Comparison of average computation time per single simulation time step of 3-dimensional dam break problem using NVIDIA A100.

For the *Unifying particle method*, several properties have been rigorously mathematically proven, such as Turing completeness, theorem about multi-core parallelizability [3] and also distributed memory parallelization [4].

To specify particle methods, we thus need to specify the properties (variables) carried by the particles, interaction between the particles (in this case the SPH schemes), and how the particles' positions and variables evolve (in this case the time integration scheme). We do so using the C++ lambda functions, and the entire SPH solver is implemented in this way based on the definition of *Unifying particle method*.

D. Particular numerical schemes

The last layer contains the actual numerical schemes of the SPH method. We use the multi-set approach where one set represents the fluid, and another sets represent the boundary particles and additional particle groups such as open boundary buffers. Thanks to the TNL data structures, the equations are written in vector form, which is a huge simplification of the notation. Some parts, such as time integration schemes, are written using expression templates, which simplifies the implementation even more to standard mathematical expression notation.

All parts of the numerical schemes (such as diffusive terms, equations of state, viscous terms, etc.) are implemented in separate parts, which makes them easy to modify them or add new ones without changing the source code at several places.

II. IMPLEMENTED NUMERICAL SCHEMES AND FEATURES

TNL-SPH currently provides several SPH schemes and several formulations of boundary conditions and algorithms using the weakly compressible fluid model:

- δ-WCSPH formulation with various formulations of diffusive terms
- Riemann based WCSPH formulation
- DBC, mDBC, BI and recent BI-Conservative formulation of solid walls [5]
- Inlet and outlet boundary conditions allowing to prescribe arbitrary Neumann or Dirichlet BC
- Periodic boundary conditions
- Unconditionally stable SPH scheme proposed by Cercos-Pita et. al [5]
- Verlet, Symplectic Verlet, RK45 and implicit midpoint time integration schemes

Together with the sole numerical schemes, additional tools are provided to analyze the results and control the simulation. The code includes sensors to evaluate desired values of variables in given points, interpolation planes or grids to interpolate the particle data back to the continuum and evaluate the results at particular sections, tools to measure water level, or tools to create surface representation of the particles. The energy analysis functions are also part of the solver.

To generate particles, an external preprocessing tool *stl2particles* [7] based on *gmsh* is provided which is able to discretize the external geometry (in .stl, .step, .igs or any other arbitrary format which gmsh can process) into a single layer of boundary elements. Multilayer discretization of external geometries is not yet available, and for these purposes, we recommend using *GenCase* by DualSPHysics [8].

III. PERFORMANCE, VALIDATION, EXAMPLES AND USAGE

To evaluate the performance of TNL-SPH we compare the code with DualSPHysics [8], the most popular SPH code running on GPU and OpenFPM [9], which is a general framework for distributed particle simulations implementing the particle data structure similarly as we do. For comparisons, 3-dimensional dambrek (SPHERIC test case 02) was used, solved with the δ-WCSPH scheme using DBC boundary conditions, artificial viscosity, diffusive term proposed by Molteni and Verlet time integration scheme. In both TNL-SPH and OpenFPM, exactly the same numerical scheme and settings used by DualSPHysics were reproduced and the case was run with the same parameters.

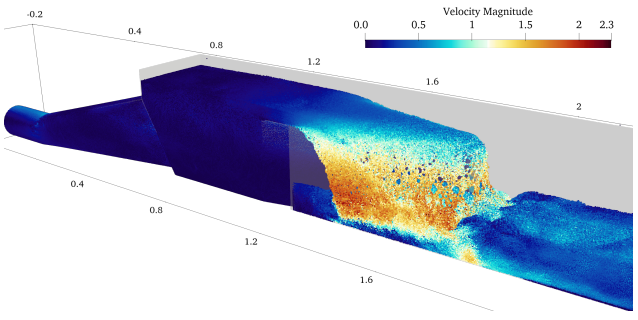

Fig. 3. Example simulation of flow in discharge object of pumping station using unconditionally stable SPH scheme [5].

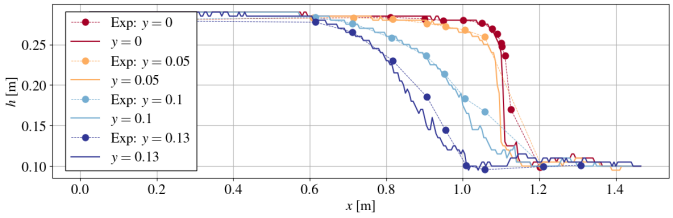

Fig. 4. Averaged water levels in the central plane of the channel $y = 0$ m and planes at $y = 0.05$ m, $y = 0.1$ and water level at the wall $y = 0.13$. Simulation data *(solid lines)* are compared experimental data *(filled circles with thin dashed line)*.

All configuration files for the three solvers, together with post-processing tools and a detailed description of the setup, are publicly available at [10]. The benchmarks were performed using NVIDIA-A40 (results are in figure 1) and NVIDIA-A100 (results are in figure 2). We can see that TNL-SPH is about 15 % -20% faster than DualSPHysics and OpenFPM on NVIDIA A100 and about 30 % faster than OpenFPM on NVIDIA A100. Moreover, in contrast to the current release of DualSPHysics (version 5.4), TNL-SPH is able to utilize multiple GPUs.

The conventional SPH benchmarks such as two- and three-dimensional dam break test case, Poiseuille flow or flow in open channels which are provided including the available experimental data and post-processing to validate the schemes. Automated performance and accuracy testing based on these benchmarks is also currently in progress.

In addition to test cases, large industrial applications are provided, particularly simulations of free surface flow with open boundaries in complex three-dimensional geometries of discharge objects of pumping stations. Problem description is provided in [11]. Figure 3 shows the results of one of the configurations obtained with δ-WCSPH using only physical viscosity, employing a conservative BI formulation for solid walls and implicit midpoint time integration. The resulting scheme corresponds to the unconditionally stable scheme proposed by Cercos-Pita et al. [5]. As far as the authors are aware, this is the first application of the SPH unconditionally stable scheme in three dimensions, for complex geometries and for problems with open boundaries. Figure 4 shows the comparison of the water levels obtained from the simulation with the experimental data [11].

Conclusion

In this work, novel open source modular SPH solver for modern distributed computing platforms based on GPU acceletaros is introduced. There is a question, why develop yet another SPH solver? The simple answer could be: Why not? Competition brings inspiration and progress, and based on results 1 and 2, we believe we are competitive.

We decided to build on multilayer approach, separating architecture-dependent part, part with core particle data structure (which can be used for multiple purposes and which represents an analogy to the structure for unstructured networks for mesh-based methods), part shared to all the particle methods and the actual numerical SPH schemes.

Compared to other open source SPH codes, emphasis is placed not only on optimization but also on the design, structure, standard, and tools of integrated development. The whole development is open and public. Although DualSPHysics [8] or GPUSPH [12] are pioneers in the field of SPH computations using GPUs, the source codes are quite rigid, and their modification or extension takes more work than it should. Exception is AQUAgpusph [13], which brings much more sophisticated and disciplined design, however, it is not as heavily geared towards optimization raw performance. The ideal scenario would be to take the best of both worlds. Here, we presented our progress so far towards this goal.

Acknowledgment

The work was supported by Grant Agency of the Czech Technical University in Prague, grant No. SGS22/148/OHK2/3T/12. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Bibliography

References

[1] Oberhuber, T., Klinkovský, J. & Fučík, R.: TNL: NUMERICAL LIBRARY FOR MODERN PARALLEL ARCHITECTURES. *Acta Polytechnica*. **61**, 122-134 (2021)

[2] Klinkovský, J., Oberhuber, T., Fučík, R. & Žabka, V. Configurable Open-source Data Structure for Distributed Conforming Unstructured Homogeneous Meshes with GPU Support. *ACM Transactions On Mathematical Software*. **48**, 1-30 (2022,9), http://dx.doi.org/10.1145/3536164

[3] Pahlke, J. & Sbalzarini, I. A Unifying Mathematical Definition of Particle Methods. *IEEE Open Journal Of The Computer Society*. **4** pp. 97-108 (2023), http://dx.doi.org/10.1109/OJCS.2023.3254466

[4] Pahlke, J. & Sbalzarini, I. Proven Distributed Memory Parallelization of Particle Methods. *ACM Transactions On Parallel Computing*. **11**, 1-45 (2024,11), http://dx.doi.org/10.1145/3696189

[5] Cercos-Pita, J., Duque, D., Merino-Alonso, P. & Calderon-Sanchez, J.: Boundary conditions for SPH through energy conservation. *Computers & Fluids*. **285** pp. 106454 (2024,12)

[6] Cercos-Pita, J., Merino-Alonso, P., Calderon-Sanchez, J. & Duque, D.: The role of time integration in energy conservation in Smoothed Particle Hydrodynamics fluid dynamics simulations. *European Journal Of Mechanics - B/Fluids*. **97** pp. 78-92 (2023,1)

[7] Kreuzova, T. CFD Geometry and Meshes: stl2particles. *GitLab Repository*. (2024), https://gitlab.com/cfdgeometryandmeshes/stl2particles

[8] Domínguez, J. et al.: DualSPHysics: from fluid dynamics to multiphysics problems. *Computational Particle Mechanics*. (2021)

[9] Incardona, P., Leo, A., Zaluzhnyi, Y., Ramaswamy, R. & Sbalzarini, I. OpenFPM: A scalable open framework for particle and particle-mesh codes on parallel computers. *Computer Physics Communications*. **241** pp. 155-177 (2019), https://www.sciencedirect.com/science/article/pii/S0010465519300852

[10] Halada, T. Benchmark of several GPU implementations of SPH method. *GitHub Repository*. (2024), https://github.com/tomashalada/sph-benchmarks

[11] Sedlář, M., Machalka, J. & Komárek, M.: Modeling and Optimization of Multiphase Flow in Pump Station. *Journal Of Physics: Conference Series*. **1584**, 012070 (2020,7)

[12] Hérault, A., Bilotta, G. & Dalrymple, R. SPH on GPU with CUDA. *Journal Of Hydraulic Research*. **48**, 74-79 (2010)

[13] Cercos-Pita, J. AQUAgpusph, a new free 3D SPH solver accelerated with OpenCL. *Comput Phys Commun*. (2015)

[14] Halada, T., Oberhuber, T. & Klinkovsky, J. Template Numerical Library - TNL-SPH. *GitLab Repository*. (2014), https://gitlab.com/tnl-project/tnl-sph