MDProcessing.jl: Julia Language Application to the Molecular Simulation Trajectory Analysis

Vasily Pisarev[1,2], Mikhail Panov[1]

[1]*HSE University*

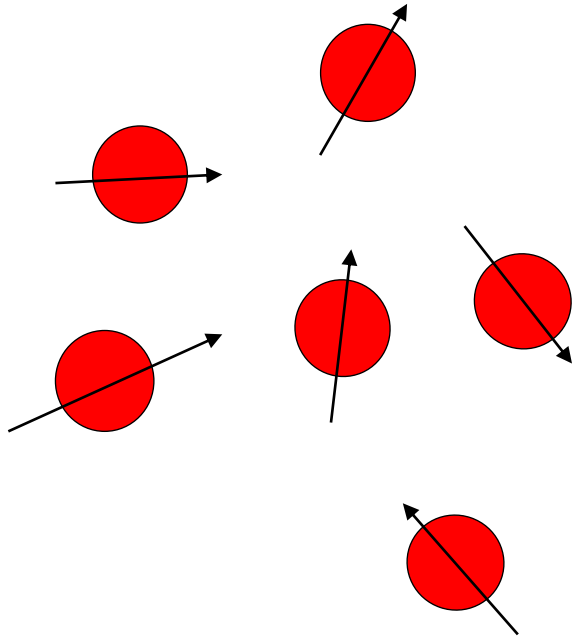
[2]*Joint Institute for High Temperatures of RAS*

Molecular Dynamics Simulations

$$\boldsymbol{F}_i = m_i \boldsymbol{a}_i$$

Forces are computed from an analytical potential (force field)

$$\boldsymbol{F} = -\nabla V(r) \qquad \boldsymbol{F}_i = \sum_{j \neq i} \boldsymbol{F}_{ij}$$



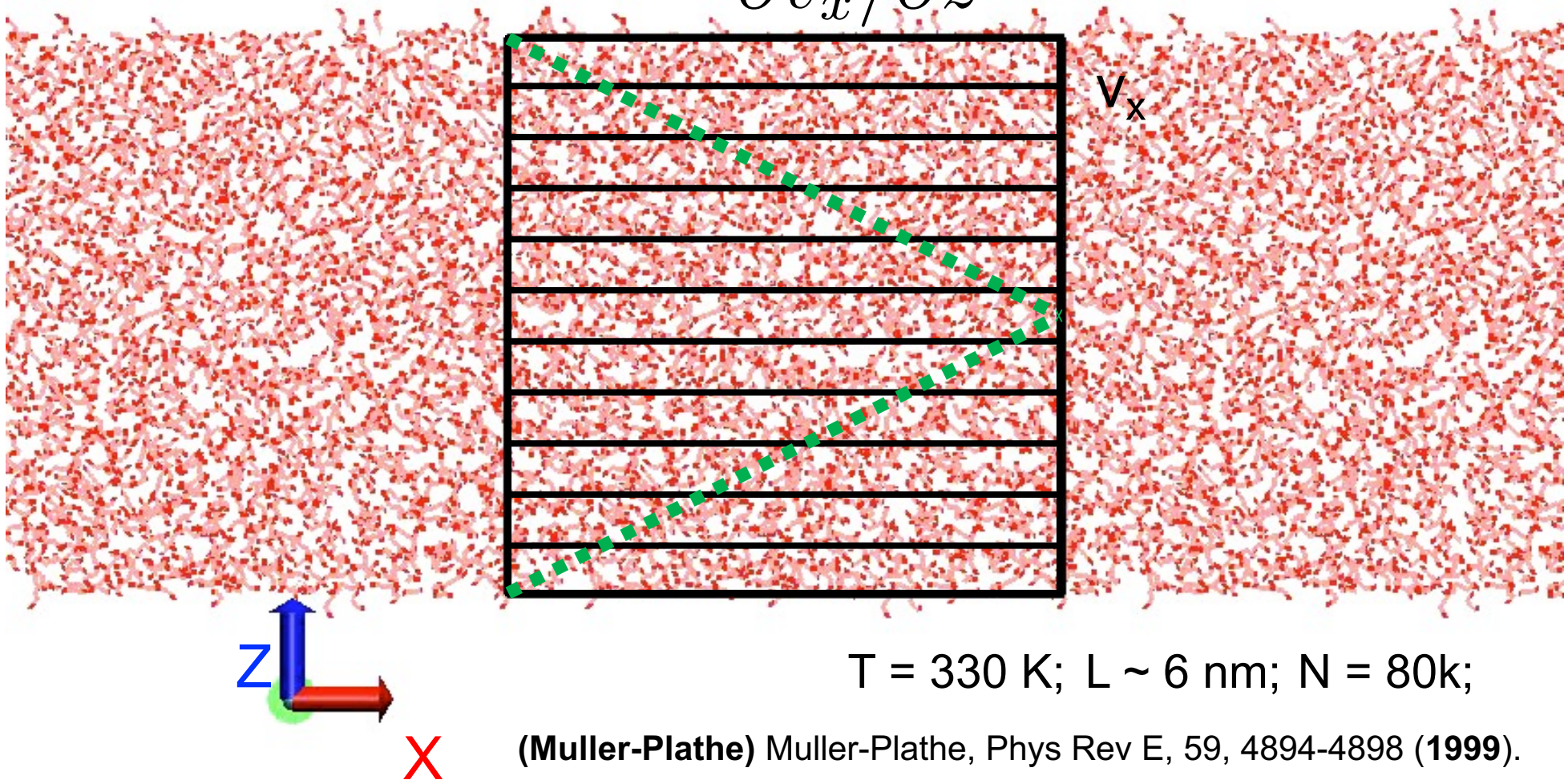
Equations of motion are solved using a finite-difference algorithm

$$\boldsymbol{r}_i(t + \Delta t) = \boldsymbol{r}_i(t) + \boldsymbol{v}_i(t)\Delta t + \boldsymbol{a}_i(t)\frac{\Delta t^2}{2}$$

$$\boldsymbol{v}_i(t + \Delta t) = \boldsymbol{v}_i(t) + \frac{\boldsymbol{a}_i(t) + \boldsymbol{a}_i(t + \Delta t)}{2}\Delta t$$

Example: shear flow

$$\eta = \frac{j_z(p_x)}{\partial v_x / \partial z}$$



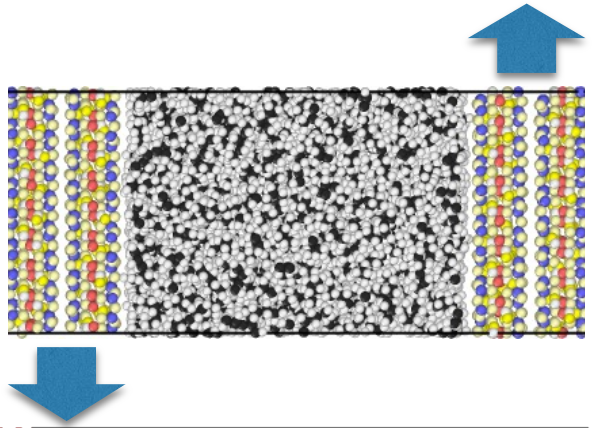
T = 330 K; L ~ 6 nm; N = 80k;

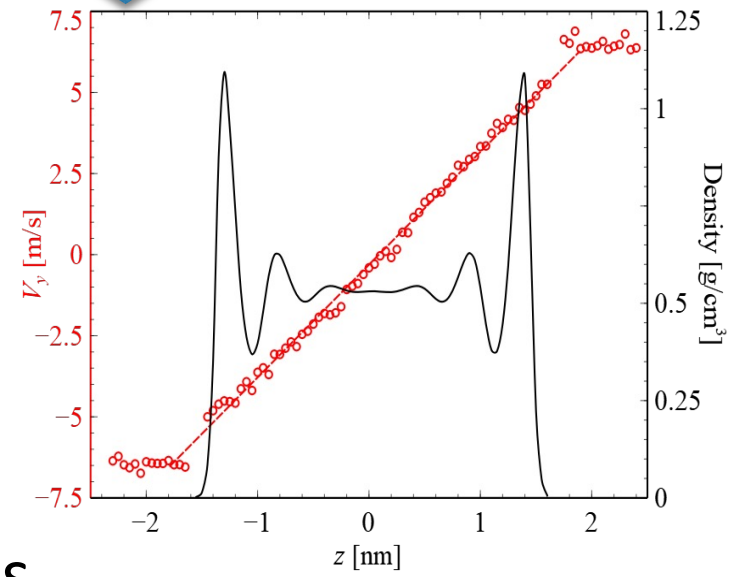
(Muller-Plathe) Muller-Plathe, Phys Rev E, 59, 4894-4898 (**1999**).

Подробности: Кондратюк Н Д, Писарев В В
"Теоретические и вычислительные подходы к предсказанию вязкости жидкостей"
УФН, 2023

Molecular Dynamics Simulations

- Computation of thermophysical properties involves
  - Structural analysis of instantaneous states
  - Averaging properties over a trajectory
- Typical system: 1k – 10M particles
- Typical simulation time: 10 ps – 100 ns (10k – 100M time steps)
- Large amount of data per trajectory, analysis becomes computationally challenging task





Tools for Simulation

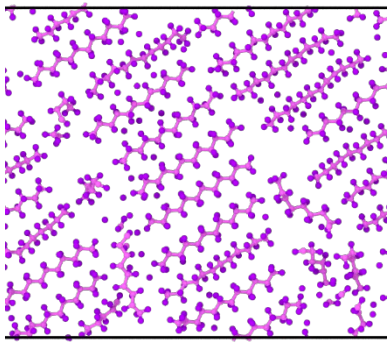
- Primary requirements:
  - Fast implementation of standard integration methods, thermostats, force fields
  - Correctness and robustness
- Simulation packages: Gromacs, LAMMPS, DL_POLY, HOOMD
- Highly optimized for CPU, GPU or both
- Almost necessary to use a programming language with a low-level control: C++ or Fortran (maybe Rust, we may see new packages released in the next few years)

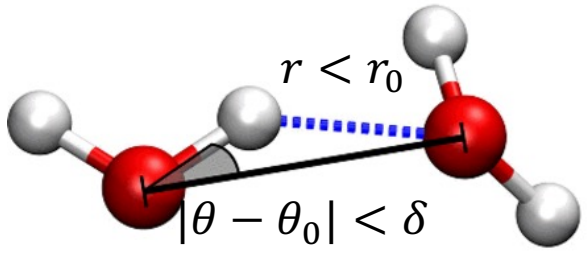
Trajectory analysis

- The most time-consuming part is already done
  - Less pressure on performance
- Two kinds of needs
  - Standardized analysis
  - Exploratory analysis, development of new structural and dynamic properties

Non-standard properties

- Correlation between interatomic separation and velocity directions
- Characterize the degree of molecular ordering
- Identification of H-bonds



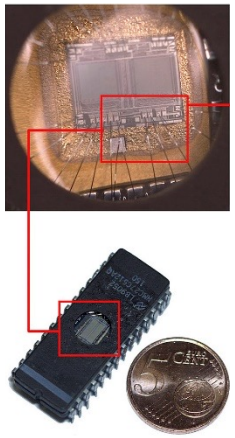


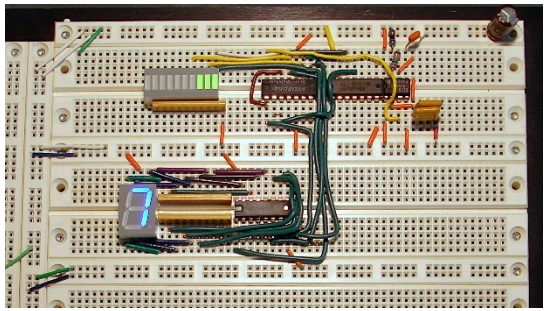
Existing approaches

Static implementation as an executable program (TraVIS, LAMMPS, Gromacs)	Scripting language binding (VMD, OVITO)
✓ Fast	✓ Fast “standard” methods implemented in C++
✓ Can be done on the fly within MD simulation software (LAMMPS, Gromacs)	✓ Easy user extension
	✓ Interop with programs written in the scripting language – processing is easily incorporated into a higher-level pipeline
• Not extensible without rebuilding the executable	
• Interop with other programs only via file output	• User-defined extensions will be less performant than “built-in” operations

Existing approaches

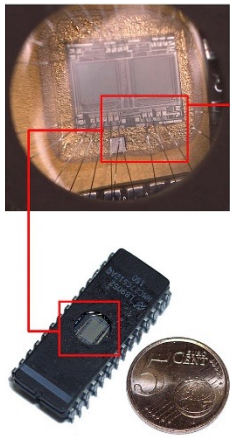
Static implementation as an executable program (TraVIS, LAMMPS, Gromacs)	Scripting language binding (VMD, OVITO)
✓ Fast	✓ Fast “standard” methods implemented in C++
✓ Can be done on the fly within MD simulation software (LAMMPS, Gromacs)	✓ Easy user extension
	✓ Interop with programs written in the scripting language – processing is easily incorporated into a higher-level pipeline
• Not extensible without rebuilding the executable	
• Interop with other programs only via file output	• User-defined extensions will be less performant than “built-in” operations



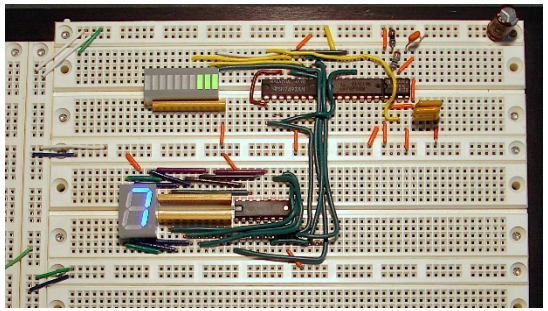


Existing approaches

Static implementation as an executable program (TraVIS, LAMMPS, Gromacs)	Scripting language binding (VMD, OVITO)
✓ Fast	✓ Fast “standard” methods implemented in C++
✓ Can be done on the fly within MD simulation software (LAMMPS, Gromacs)	✓ Easy user extension
	✓ Interop with programs written in the scripting language – processing is easily incorporated into a higher-level pipeline
• Not extensible without rebuilding the executable	
• Interop with other programs only via file output	• User-defined extensions will be less performant than “built-in” operations



Middle ground?



Why use Julia programming language?

- Dynamic typing and duck typing – good for exploratory programming and data analysis
- Type inference: in a carefully-written program, types can be statically inferred
- JIT compilation – carefully written code can be statically type-inferred and compiled to fast native code
  - Core computations can be implemented in pure Julia
  - User-defined data types are as performant as built-in types
  - Raw loops are fast - no need to jump through hoops to offload heavy computations to a specialized numerical library
- Convenient out-of-the-box data containers such as arrays, tuples, dictionaries, sets, multiple useful data structures provided by third-party packages

Why use Julia programming language?

- Parallel features:
  - Native task-based multithreading
  - Distributed-memory computing via built-in Distributed.jl or MPI bindings
  - CUDA programming via CUDA.jl

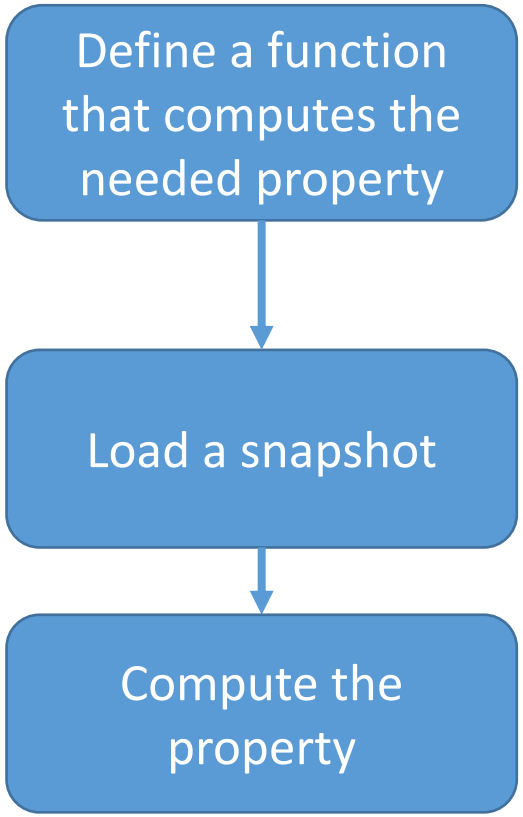
MDProcessing.jl package

- A package for computing properties from MD trajectories
- Written in Julia
- Data format: LAMMPS dump files (XYZ and LAMMPS data files are planned)
- Built-in analysis functions are defined, user extensions in Julia are possible

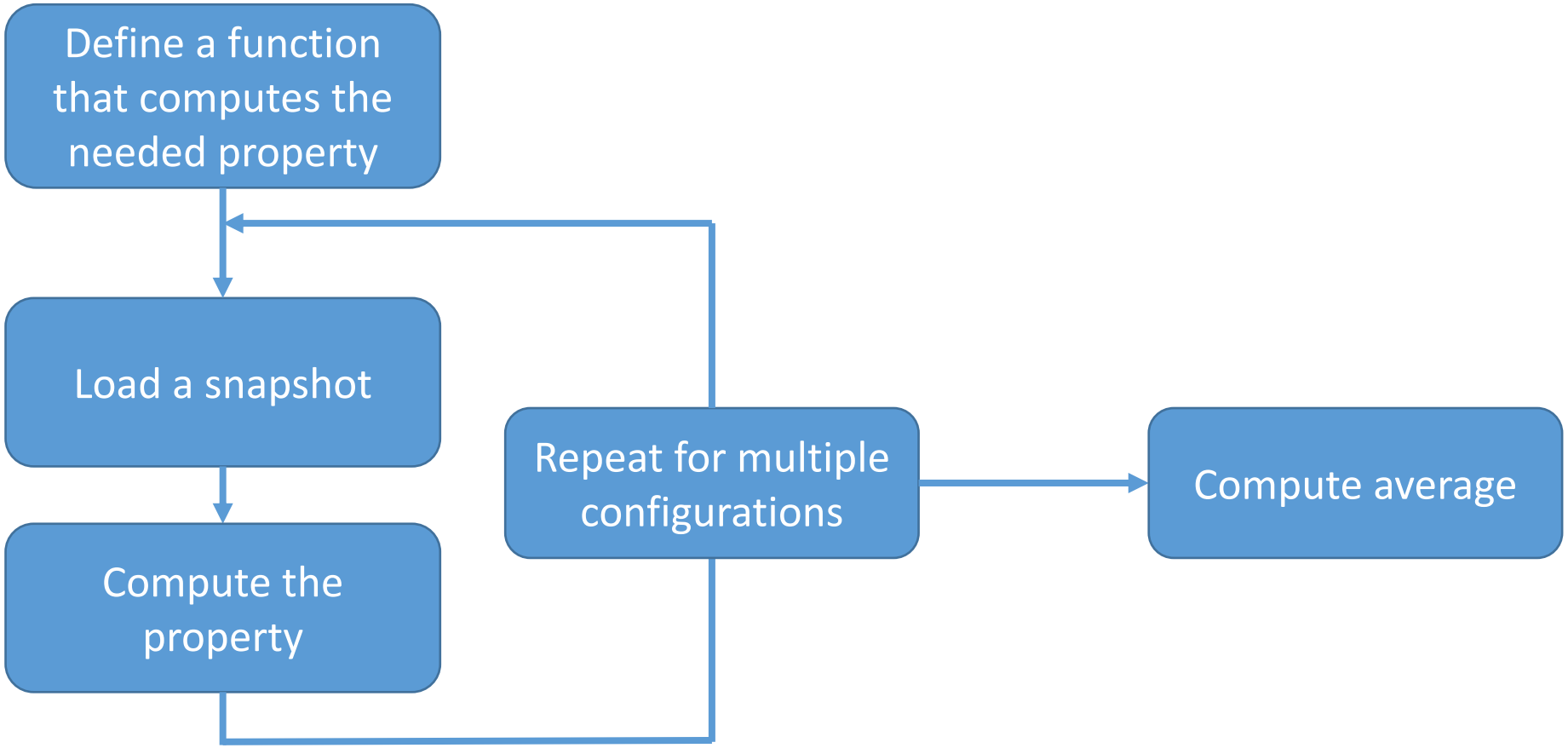
MDProcessing.jl intended workflow

Define a function that computes the needed property

MDProcessing.jl intended workflow

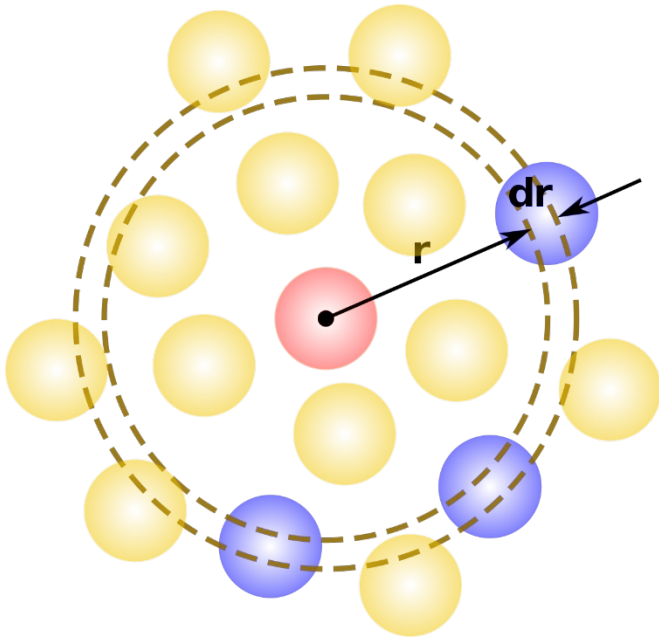


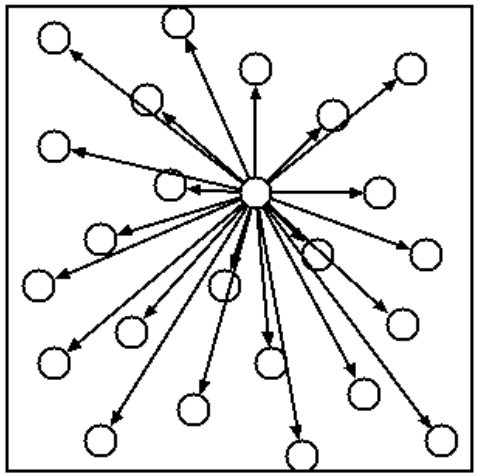
MDProcessing.jl intended workflow

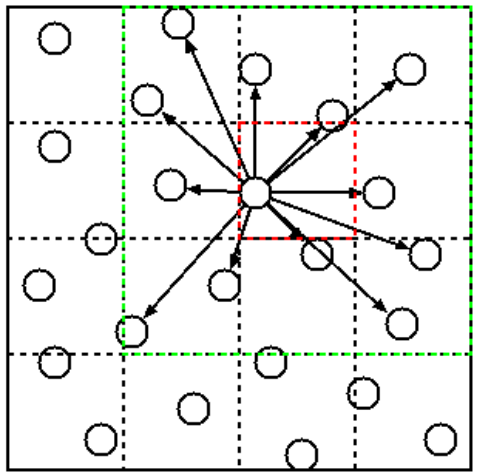


Performance assessment

- Test problem: computing the radial distribution function
- Benchmarks one of the costly operations in particle simulation analysis – neighbor search
- Algorithmic complexity $O(N^2)$ with naïve search, $O(N \times R_c^3)$ with cell list structure

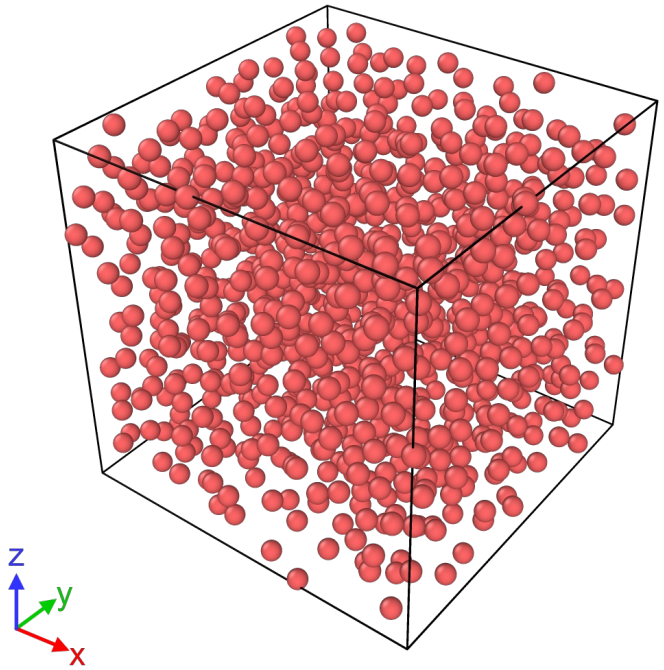




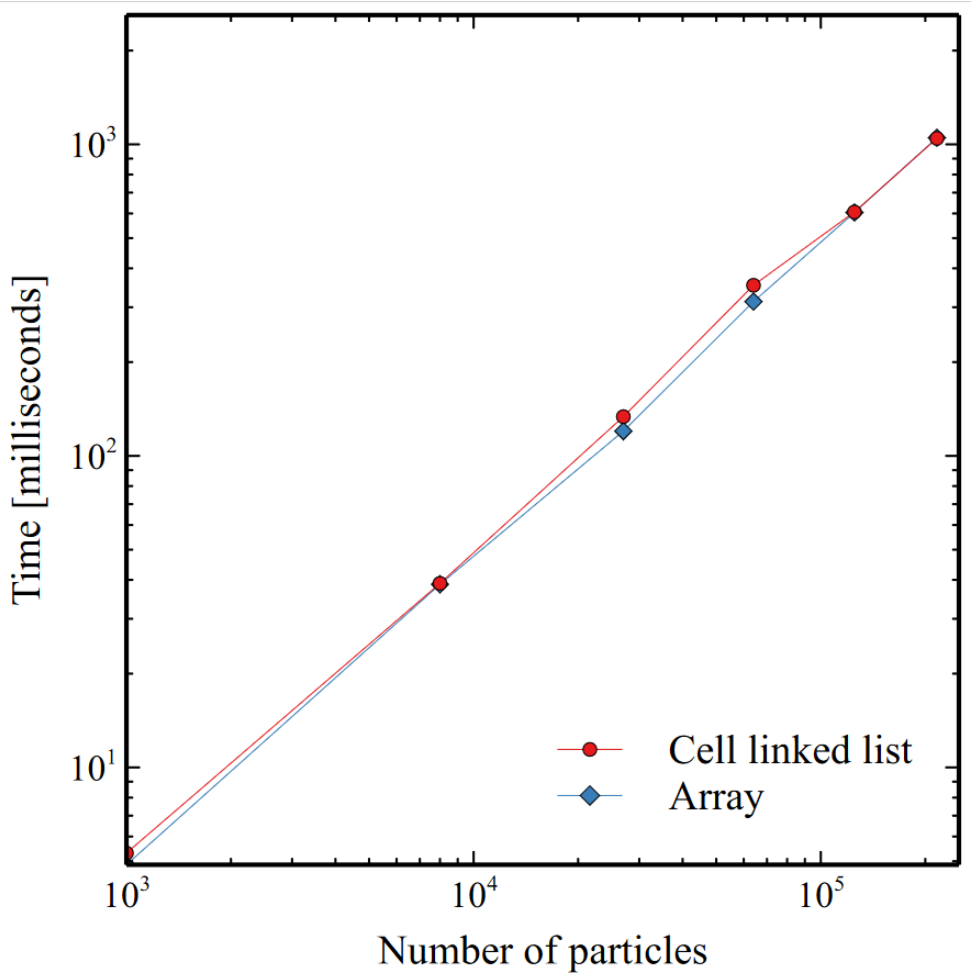


Performance assessment

- Test system: Lennard-Jones fluid
- Constant density $\rho = 0.75$
- Varying system size and cut-off radius

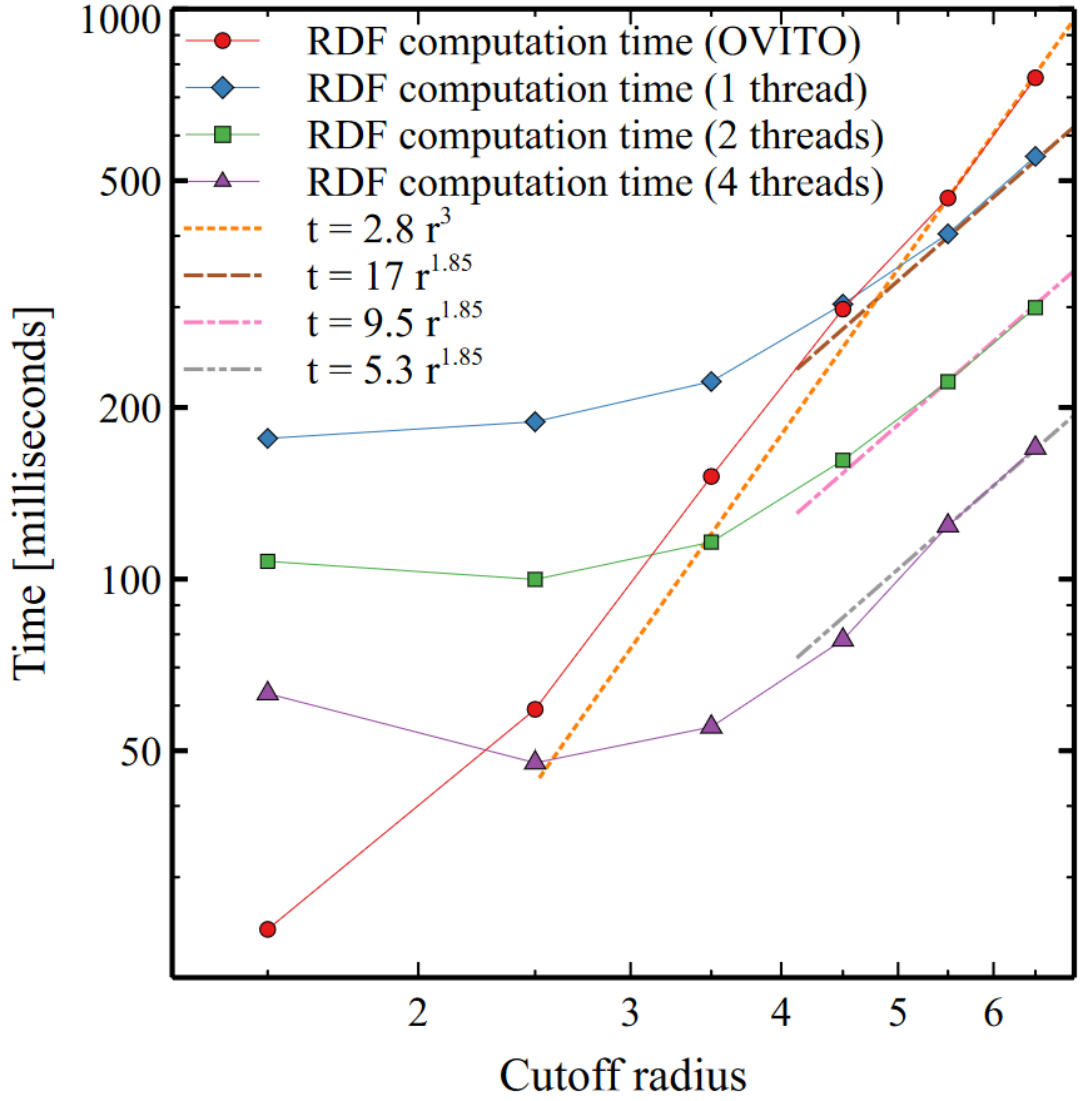


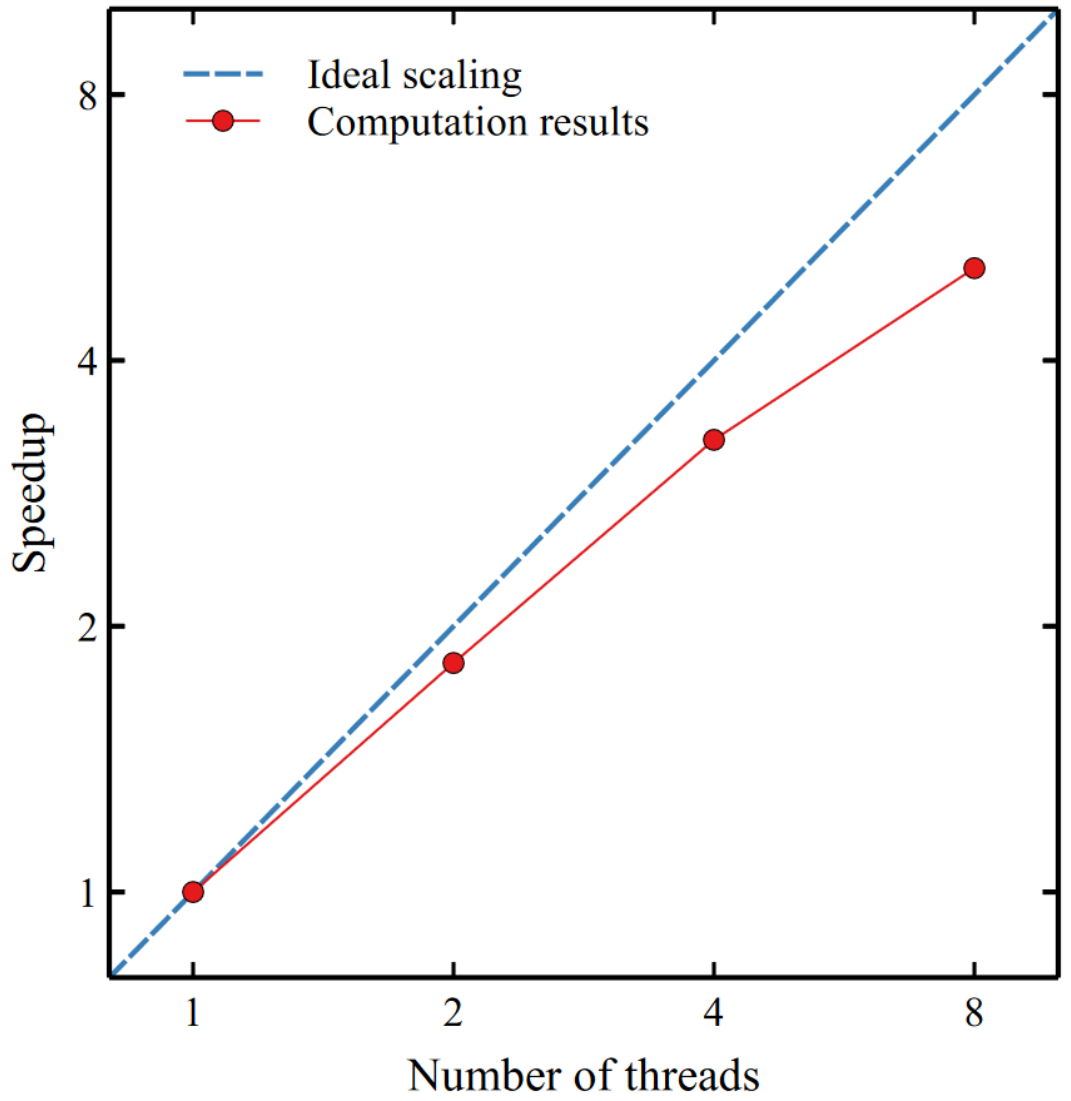
RDF compute time for different system sizes (R_c = 2.5)



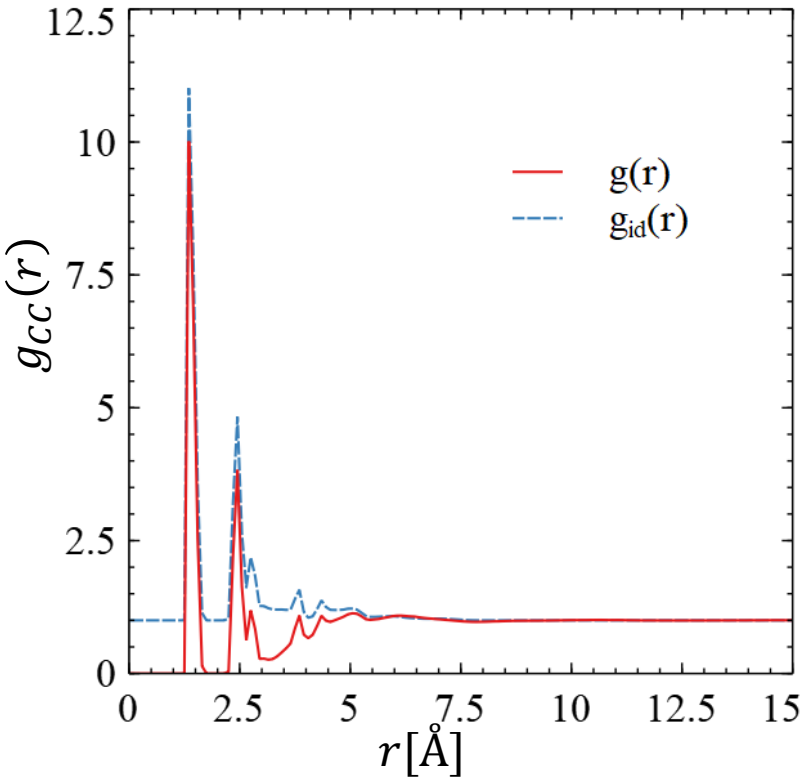
Performance assessment

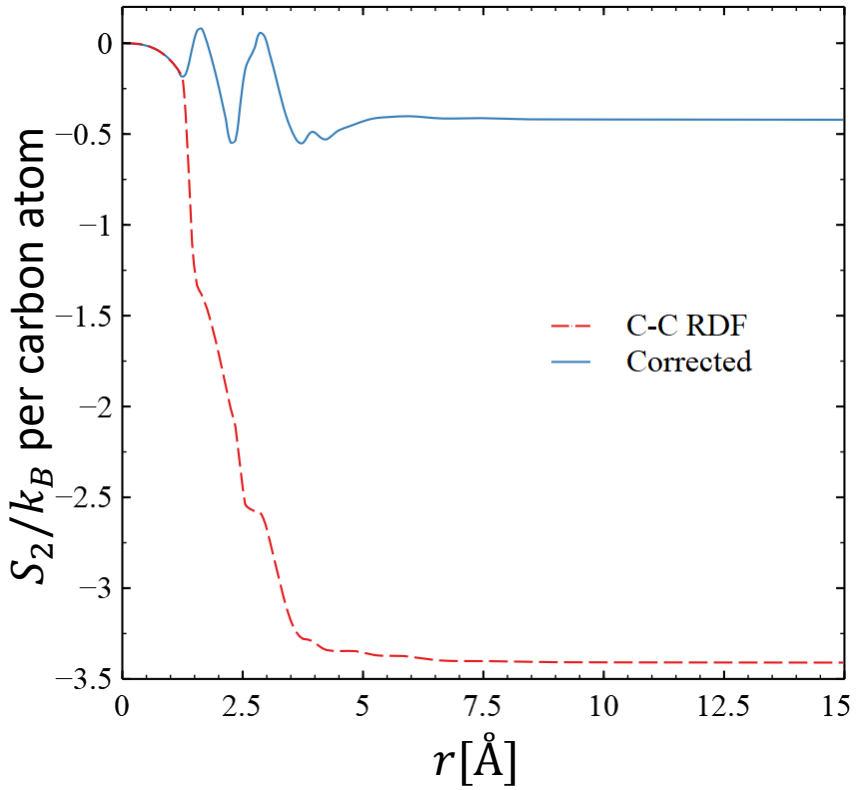
Effect of cut-off radius and multi-threading





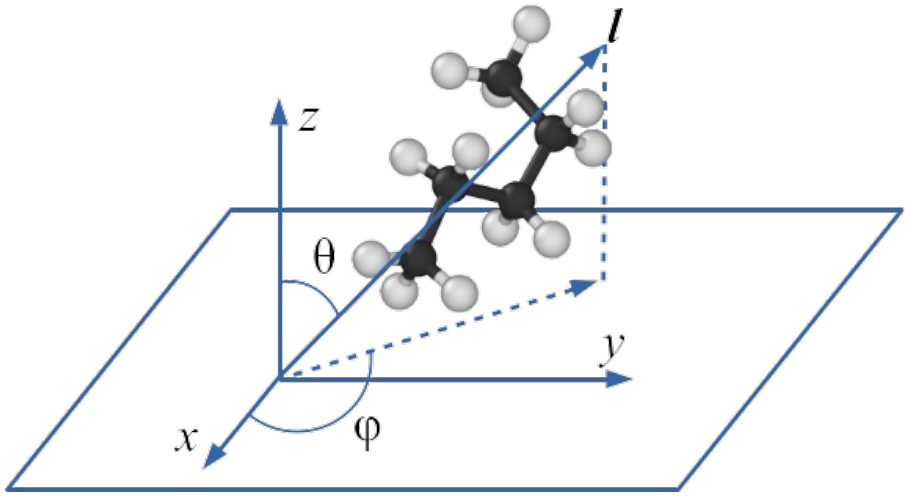
Some results

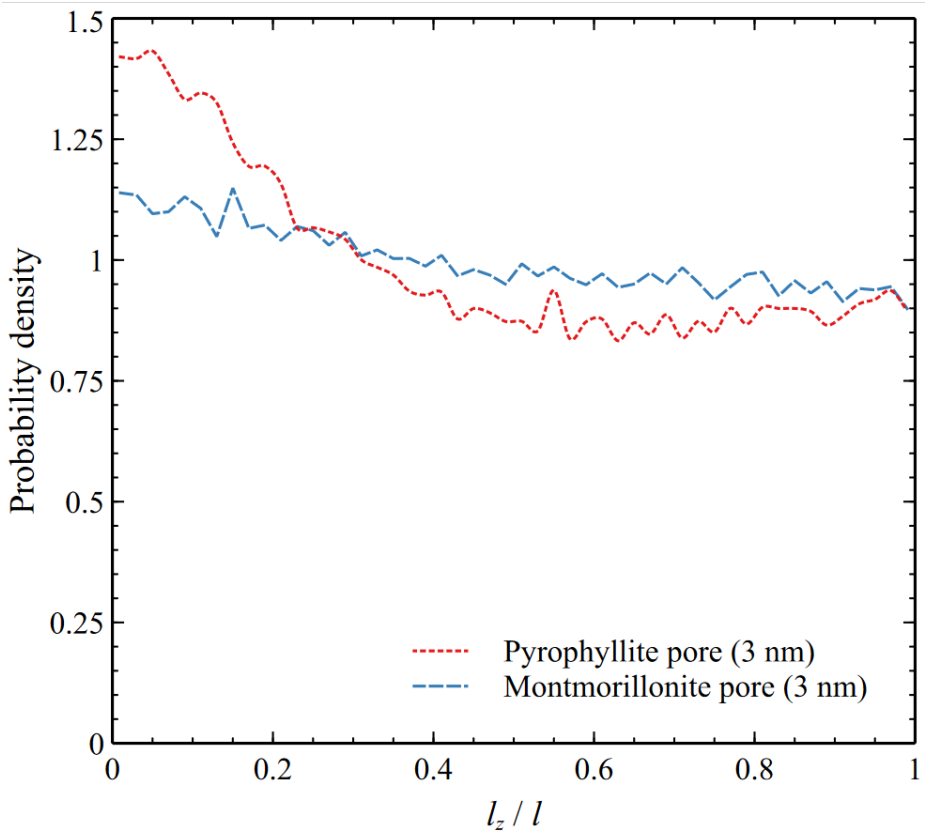




Computation of RDFs in molecular systems separated into intra- and intermolecular contributions
Nikitiuk, Salikova, Kondratyuk, Pisarev // J Mol Liq 2022

Some results





Straight-line fitting and orientational distributions of molecules
Pisarev & Kalinichev // J Mol Liq 2022

Conclusions

- Julia language is suitable for performance-demanding computational tasks
- Dynamic typing and scripting nature of the language make it convenient to do exploratory analysis and prototyping new ideas
- Performance-critical parts of the analysis can be optimized *in the same language*
- The core functions of MDProcessing.jl are shown to have the performance comparable to OVITO (written in C++)

- Project repository: https://gitlab.com/pisarevvv/mdprocessing.jl/