Journal of
open research software

de Buyl, P et al. 2017 RMPCDMD: Simulations of Colloids with Coarse-grained Hydrodynamics, Chemical Reactions and External Fields. *Journal of Open Research Software,* 5: 3, DOI: https://doi.org/10.5334/jors.142

SOFTWARE METAPAPER

RMPCDMD: Simulations of Colloids with Coarse-grained Hydrodynamics, Chemical Reactions and External Fields

Pierre de Buyl,[1] Mu-Jie Huang,[2] and Laurens Deprez[1]

[1] Instituut voor Theoretische Fysica, KU Leuven B-3001, Belgium

[2] Chemical Physics Theory Group, Department of Chemistry, University of Toronto, Toronto, Ontario M5S 3H6, Canada

Corresponding author: Pierre de Buyl
(pierre.debuyl@kuleuven.be)

The RMPCDMD software package performs hybrid Molecular Dynamics simulations, coupling Multiparticle Collision Dynamics to model the solvent and Molecular Dynamics to model suspended colloids, including hydrodynamics, thermal fluctuations, and chemically active solvent particles and catalytic colloids. The main usage of RMPCDMD is the simulation of chemically powered nanomotors, but other setups are considered: colloids in the presence of a thermal gradients or forced flows. RMPCDMD is developed in Fortran 2008 with OpenMP for multithreaded operation and uses the HDF5-based H5MD file format for storing data. RMPCDMD comes with documentation and a tutorial for the simulation of chemically powered nanomotors.

Keywords: Molecular Dynamics; Colloids; Hydrodynamics; Nanomotors; Fortran; Python

(1) Overview

Introduction

Molecular Dynamics (MD) is a computational technique to study condensed matter systems by numerically solving Newton's equations of motion for a large number of particles [1]. Often, due to the number of constituents in a system, so-called *coarse-grained* or *multiscale* strategies are used to reduce the computational cost of a simulation.

One simulation strategy for modelling colloids embedded in a solvent is to couple the Multiparticle Collision Dynamics (MPCD) algorithm [2] for the solvent with MD for the colloids [3]. It differs from *traditional* MD by the use of a coarse-grained representation of the solvent's structure and dynamics and by the possible occurrence of chemical species transformations in the course of the simulation. In the latter case, the extension Reactive MPCD (RMPCD) provides bulk chemical kinetics that reduce to reaction–diffusion equations [4]. The name RMPCDMD results from the concatenation of the different algorithms' acronyms: Reactive Multiparticle Collision Dynamics and Molecular Dynamics.

The focus of RMPCDMD lies on the modeling of chemically active colloids (e.g. nanomotors [5]) or of colloids subjected to external gradients (e.g. flows [6] or temperature gradients [7]). A RMPCDMD simulation consists in N_{solvent} solvent particles and N_{colloids} colloidal particles with a mass, position, and velocity, placed in a cuboid domain. Colloid–solvent and colloid–colloid interactions follow Lennard–Jones type potentials, but there is no interaction potential acting among the solvent particles. As $N_{\text{solvent}} \gg N_{\text{colloids}}$, this considerably speeds up a simulation in comparison to a full MD simulation. The time evolution of colloidal and solvent particles follows Newton's equations of motion, which are solved iteratively using the velocity-Verlet algorithm [1, 8]. The output from a simulation is a trajectory representing the coordinates of all particles in the course of time. Typically, solvent data is preaveraged during the simulation as its storage requires significant amounts of space.

Implementation and architecture

Development

RMPCDMD is implemented in Fortran, following the 2008 version of the standard, and uses OpenMP for multi-core execution. Trajectories are stored in the H5MD file format [9], and Python programs are provided to illustrate data analysis. The module `fortran_h5md` [10] provides helper routines to write H5MD files from Fortran. H5MD files are HDF5 files (https://www.hdfgroup.org/HDF5/) with an internal organization tailored for molecular simulations [9].

In designing RMPCDMD, we tried to adopt practices for scientific computing that help to deliver reproducible results and facilitate the extension of the code and its maintenance. We also made special efforts on the

usability of the software. The design choices are the results of the authors' experience on scientific programming, with further motivation by the article on "Best Practices for Scientific Computing" by Wilson *et al* [11]:

- The algorithms are implemented in Fortran modules in the `src/` directory and shared between the simulations found in `programs/`. **Table 1** summarizes their content.
- Subroutines and functions are limited in size, so that understanding any single code unit remains reasonable.
- Data is encapsulated in *derived types*.
- No global variable is used, to clarify to the reader what data is passed to subroutines.
- The programming style strives for legibility, conciseness, and efficiency.
- All simulation parameters are given from a text configuration file, there is no hard-coded setting in the code that would require re-compilation between runs.
- The code is tracked with the git revision control software [12] and published on GitHub (https://github.com/pdebuyl-lab/RMPCDMD).
- The build procedure and partial testing are automatically executed on the Travis CI platform (https://travis-ci.org/pdebuyl-lab/RMPCDMD) for continuous integration.

Instead of building a generic program that could handle the logic of different geometries, boundary conditions, and colloid types, specific simulation programs have been implemented. Each program calls the algorithms from the modules and resembles a statically compiled "simulation script". The amount of administrative code (reading parameters, defining storage, etc.) is however larger than what could be achieved in a higher-level language such as Python.

Quality control

CMake is used to build RMPCDMD and run automated test programs in `test/` that cover some of the base algorithms such as particle sorting or the histogram routine. For spatial sorting, for instance, the compact Hilbert index of the particles is re-computed after sorting to check the consistency of the sorted list. The Fortran module `fortran_tester` [13] provide convenience routines for testing (e.g. equality assertions). All programs in `test/` whose filename starts with `test_` are executed automatically by Travis CI when the code repository is updated. Users can also execute the `make test` command on their machine after having compiled RMPCDMD.

Automated testing using actual simulations is difficult and very much time intensive, and we rely on it only for a small part of the code. In simulations without a thermostat or wall interactions, the total momentum and energy of the system must be conserved. This is verified for a short run of the `single_dimer_pbc` simulation as part of the continuous integration testing: A Python program `test/dimer_test.py` reads the simulation output and validates, to a set accuracy, these conservation laws. This also insures that the common foundation of RMPCDMD, such as the force computation, neighbor listing, particle sorting, etc., fulfil their role.

Further validation of the code is done via comparison of physical results. For pure solvent simulations in equilibrium, one can verify that the velocity distribution of the solvent particles is a Boltzmann distribution at the set temperature. The Poiseuille flow simulations allows one to verify the viscosity, via the velocity profile. For colloids, one can check the diffusion coefficient against the literature results [14]. These validations must be performed by the user. The presence of analysis programs in Python in the `experiments/` directory gives a good starting point for this work.

User perspective

For the end-user, the interface comes as a single command-line tool, `rmpcdmd`, that drives the simulations and provides access to utility programs. A typical command is

```
rmpcdmd run single_dimer_pbc dimer.parameters
dimer.h5 auto
```

where `single_dimer_pbc` is the simulation program, `dimer.parameters` is the configuration file, `dimer.h5` the output datafile, and `auto` is the setting for the

Module name	Functionality
`common`	Routines of general use: histogramming, parsing of command-line arguments, timing and minimum-image convention function.
`hilbert`	Computation of the compact Hilbert index of a lattice cell and vice-versa.
`cell_system`	Cell data structure.
`particle_system`	Particles data structure and sorting routine.
`particle_system_io`	Data structures to facilitate the storage of thermodynamic and particle data.
`interaction`	Computation of Lennard-Jones force and energy.
`neighbor_list`	Data structure and routines to update the neighbor list.
`mpcd`	Algorithms for MPCD streaming and collision, and for bulk reactions.
`md`	Algorithms for velocity-Verlet and rigid body MD.

Table 1: A summary of the functionality provided by the Fortran modules of RMPCDMD found in the `src/` directory.

random-number seed. While the individual programs can be run directly, the command-line tool `rmpcdmd` aims to provide a single file to place in the program "PATH" on the user's computer and to automate some actions. The commands available are:

- `run`: execute one of the simulation programs. The setting of the environment variable `OMP_NUM_THREADS`, and the start and end time of the run are shown. Bash's `time` provides the "user time" actually used by the program.
- `seeder`: print out a signed 64-bit integer value generated by the computer's `/dev/urandom` device.
- `plot`: display an observable from a simulation file.
- `timers`: print out (or display) the timing information from a simulation file.

A benefit of using separate simulation programs is that the configuration file for a simulation only contains directly relevant parameters. All simulation programs share the same command-line syntax, read parameters from a text file using the ParseText Fortran library [15], read the seed for the RNG from the command line and output the trajectory to a H5MD file.

A generic plotting utility is provided and can be invoked as

```
rmpcdmd plot dimer.h5 --obs temperature
```

to plot the temperature, for instance. This plotting program and the other analysis programs that come with RMPCDMD are written in Python and rely on NumPy for storing and processing numerical data, SciPy [16] for numerical algorithms, Matplotlib [17] to produce 2D figures, h5py [18] to read HDF5 files and Mayavi [19] to visualize 3D data.

Documentation

A documentation is found in the directory `doc/`. It is published on the web, http://lab.pdebuyl.be/rmpcdmd/ and also serves as the homepage of RMPCDMD. The documentation is built with the Sphinx documentation generator [20] and is written in reStructuredText. An extension to Sphinx is bundled with RMPCDMD to provide (i) automatic links from the documentation to the source code documentation that is annotated with Doxygen (http://www.stack.nl/~dimitri/doxygen/) and (ii) automatic inclusion of the Doxygen header of Fortran programs. Specific instructions for the installation and execution of RMPCDMD are provided, as well as general informations on the algorithms are presented. The description of the simulation programs is generated automatically from the beginning of the corresponding `.f90` files and includes all parameters for the simulations.

A tutorial on the simulation of chemically powered nanomotors, based on RMPCDMD, was given at the recent workshop "Modeling of chemically powered nanomotors" held at the KU Leuven in April 2016 (http://lab.pdebuyl.be/2016-nanomotor/). This tutorial is reproduced in the documentation of RMPCDMD and represents a unique resource in the field of nanomotor simulation.

Algorithms

For completeness, we state here exactly what algorithms are implemented in RMPCDMD, with the corresponding reference to the literature.

RMPCDMD implements the original MPCD collision rule [2] and the Anderson thermostat [21]. Random shifting of the spatial grid is always performed [22] to ensure Galilean invariance. Flows are generated by applying a constant acceleration to solvent particles [23, 24]. Stick boundary conditions at the wall use the bounce-back rule and virtual particles during the collision step [25].

Molecular Dynamics is performed with the velocity-Verlet algorithm [1, 3, 8]. The interaction potentials, both for solvent–colloid and colloid–colloid interactions, are purely repulsive cut-off 12–6 Lennard-Jones potentials of the form

$$V(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + \frac{1}{4}\right) \text{ for } r < 2^{1/6}\sigma$$

where the energy parameter ε and the length scale parameter σ depend on the species considered. Verlet and cell lists are used to compute the solvent–colloid forces and energies [1]. The sorting of solvent particles in cells is the same as for the MPCD collision step. The rigid body dynamics uses RATTLE [26] in its iterative implementation for the Janus particle and exactly for the dimer motor.

Spatial sorting of the solvent particles is based on compact Hilbert indices [27]. Random numbers are generated with the Threefry 2x64 RNG [28], allowing each OpenMP thread to generate random numbers independently. The RNG is provided by the `random_module` library [29], where it is implemented in C99. The choices are inspired by the `nano-dimer` [30] simulation code, which implements these algorithms in OpenCL C for large-scale parallel processors.

To obtain a consistent measure of the temperature T even in the presence of flows, we compute T by averaging the center-of-mass reference frame temperature over all MPCD cells as

$$T = \frac{1}{3N_c}\sum_{\xi}\frac{1}{N_\xi - 1}\sum_{i} m_i(\upsilon_i - \upsilon_\xi)^2$$

where N_c is the number of cells, the variable ξ represents a cell, N_ξ the number of particles in a cell, υ_ξ the center-of-mass velocity of the cell, and m_i and υ_i the mass and velocity of particle.

Applications of RMPCDMD

The prototypical use of the MPCD algorithm to study chemically active particles is the dimer nanomotor, whose model was introduced by Rückner and Kapral [5]. It consists of two rigidly linked spheres evolving in a solvent. Chemical reactions are catalyzed by one of the spheres and create a local concentration gradient, effectively propelling the dimer along its symmetry axis. This simulation is implemented in the program `single_dimer_pbc`.

To facilitate usage of RMPCDMD by newcomers, ready-made simulation "experiments" are provided in `experiments/`. There, makefiles allow the user to simply type

```
make simulation
```

to start a simulation. The terminology that we have chosen reflects our opinion that computer simulations can be viewed as numerical experiments in which one prepares a setup and conducts a study to understand a certain physical phenomenon. The ability to modify a given experimental choice (i.e. thermostating or not, surface properties, etc.) to assess its effect is therefore critical. By providing several chemical models and collision rules, RMPCDMD provides such a framework.

Four programs are provided to analyze the data of the first experiment `01-single-dimer`. `plot_velocity.py` displays the laboratory reference frame velocity or the directed velocity of the dimer (with the option `--directed`). `plot_histogram.py` displays the cylindrical shell histogram of the reaction product concentration. `plot_msd.py` displays the mean-squared displacement of the dimer's center-of-mass position. This quantity is a practical interest in comparison to experimental studies [31]. `view_last_simulation_frame.py` displays a 3D representation of the dimer motor using the Mayavi visualization library [19], either with its full trajectory as a line (with option `--unwrap`) or with the product solvent particles B (with option `--show-B`). The two first analysis programs are modeled after the results by Rückner and Kapral [5]. Their output is shown in **Figures 1** and **2** for a simulation of the "experiment" `01-single-dimer`.[1] The tree-dimensional visualization of `view_last_simulation_frame.py` is shown in **Figure 3**. The assessment of the correctness for a simulation, discussed in section *quality control*, can be started by using the provided plotting tool to observe the conservation of the center of mass velocity (`rmpcdmd plot dimer.h5 --obs center_of_mass_velocity`) or of the internal energy (`rmpcdmd plot dimer.h5 --obs internal_energy`).

While the aims of `single_dimer_pbc` are to reproduce Ref. [5] and to provide a starting point for other developments, the other simulation programs in RMPCDMD are related to forward-looking research projects that will benefit from a robust codebase. These other setups are: `poiseuille_flow` generates a Poiseuille flow between parallel plates, with no colloid. `chemotactic_cell` generates a Poiseuille flow with a two-species chemical gradient perpendicular to the flow and an embedded spherical colloid or a dimer nanomotor. `single_janus_pbc` implements a composite Janus nanomotor [32]. The rigid-body RATTLE or elastic network version can be chosen via an option in the configuration file. `single_sphere_thermo_trap` generates a thermal gradient between two plates, with an embedded spherical colloid.

(2) Availability

Operating System

RMPCDMD has been tested on Linux using the gcc/gfortran and icc/ifort compilers and on OS X (El Capitan) using gcc/gfortran (installed via MacPorts).

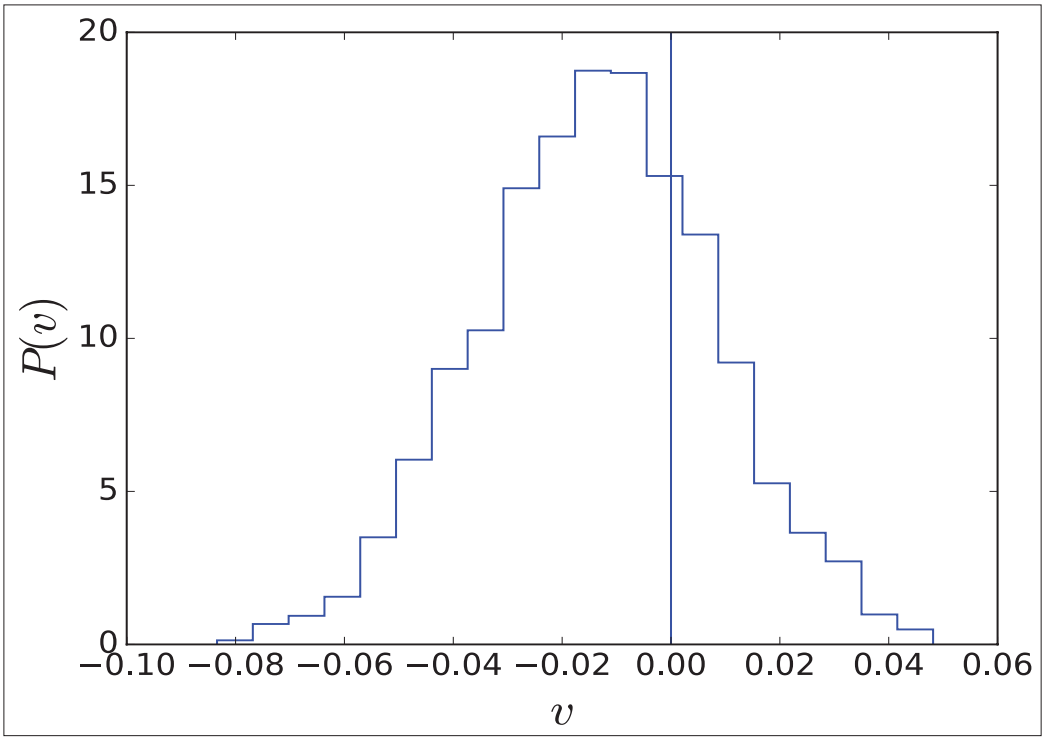

Figure 1: Output of the program `plot_velocity.py` with options `--directed` and `--histogram` for the single dimer simulation.

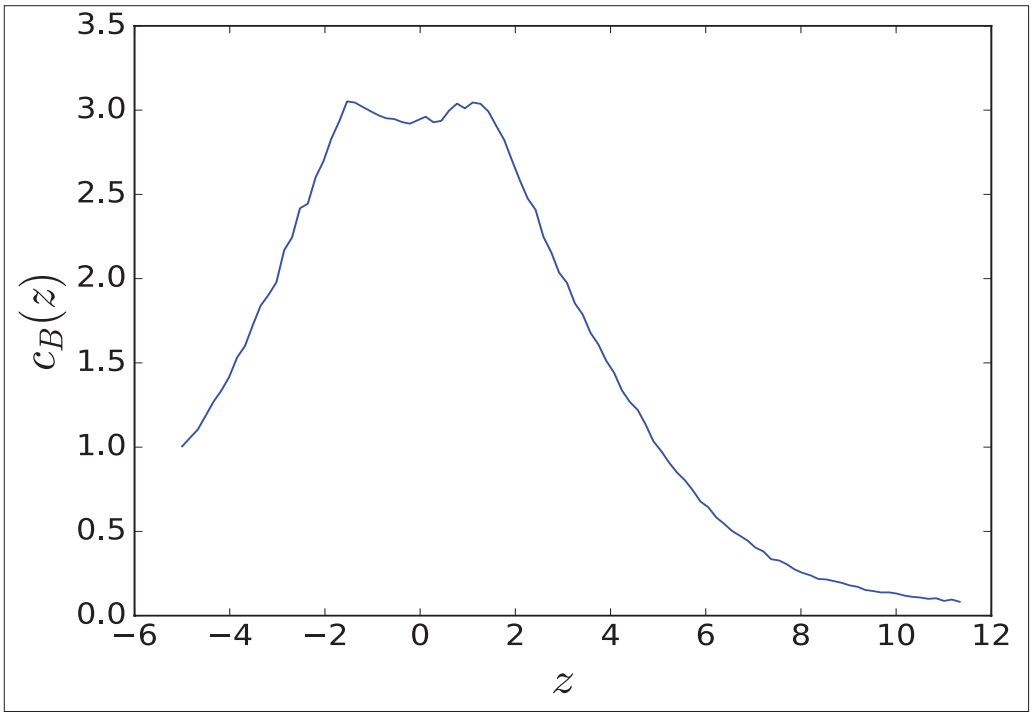

Figure 2: Output of the program `plot_histogram.py` for the single dimer simulation. The region close to $z = 0$ is the catalytic sphere and displays the highest values for the concentration of B product.

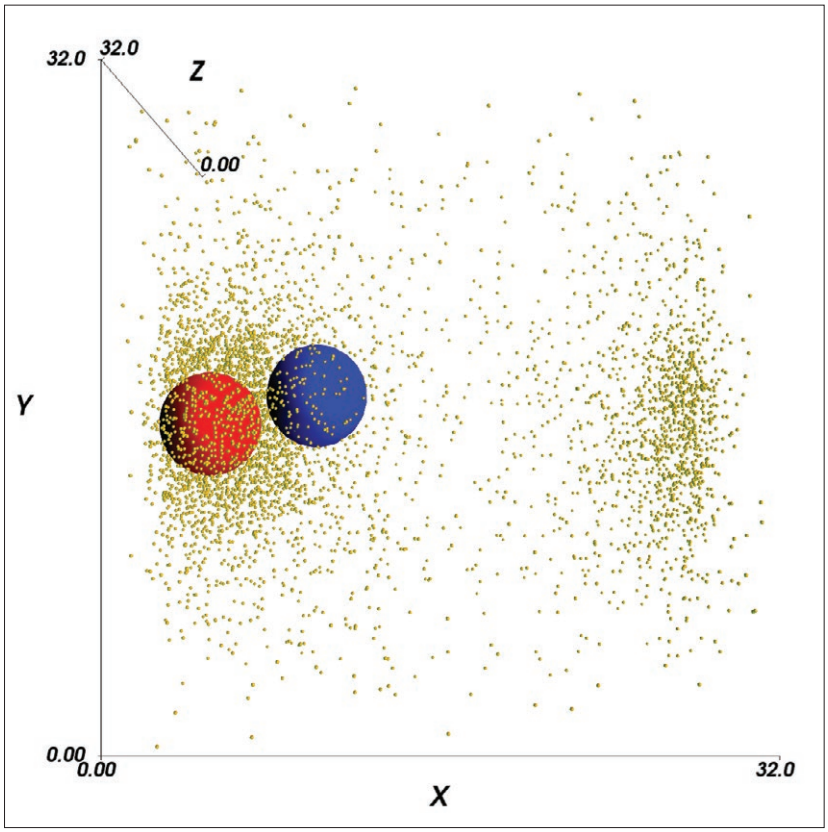

Figure 3: Output of the program `view_last_simulation_frame.py` for the single dimer simulation with option `--show-B`. The product of reaction B is mostly found around the catalytic sphere (red) and the periodic character of the simulation box is visible.

Programming Language
Fortran 2008 for the simulation code, C99 for the Random Number Generator, bash and make for the execution and Python (with NumPy, SciPy, matplotlib and h5py) for analysis.

Additional system requirements
The largest RAM usage in RMPCDMD comes from the solvent coordinates and requires about $N_{\text{solvent}} \times 250$ bytes of storage. The example simulations in the `experiments/` directory require between 100MB and 200MB of RAM.

Dependencies
The CMake and Make tools, and the HDF5 library. For data analysis, NumPy, SciPy, matplotlib and h5py.

List of contributors
Pierre de Buyl: Project creator, code, documentation, build system, tutorial.
Laurens Deprez: Code: RATTLE for the dimer, angle-dependent MPCD, boundary conditions and forcing for flows, Lennard-Jones 9-3 walls, chemotactic cell setup.
Mu-Jie Huang: Tutorial: Janus particle.
Peter Colberg: Initial OpenMP parallelization, continuous integration testing and build system.

Software location
Archive
Name: Zenodo
Persistent identifier: DOI: 10.5281/zenodo.198599
Licence: BSD 3-clause
Publisher: Pierre de Buyl
Version published: 1.0
Date published: 9 December 2016

Code repository
Name: GitHub
Persistent identifier: https://github.com/pdebuyl-lab/RMPCDMD
Licence: BSD 3-clause
Date published: 9 December 2016

(3) Reuse Potential
RMPCDMD's most valuable content consists in the collection of algorithms in `src/`, many of which are not available widely, and the simulation protocols implemented in `programs/`. All steps of a simulation appear in the body of the corresponding program, including calls to the force computation, RATTLE, neighbor listing, etc, so that following the full execution of the code is explicit.

There are two directions to modify RMPCDMD. The first one is to add an *application*, or a *virtual experiment*, via an additional program in `programs/`, using the closest existing program as a basis. The second one is to implement a new algorithm or feature in an existing or new module under `src/`, ideally with dedicated tests in `test/`. Ideas for further extensions include alternative chemical schemes and geometries, performance tuning for many-motor simulations, more efficient rigid-body implementation, etc.

Given the scarcity of open-source nanomotor simulation code, we briefly compare RMPCDMD to `nano-dimer` [30] (by Peter Colberg, available under the MIT license), these two being the only open-source softwares to provide support for chemically-powered nanomotor simulation. RMPCDMD has, in addition to `nano-dimer`, support for external fields or rigid-body dynamics for assemblies larger than the dimer. `nano-dimer`, on the other hand, is based on OpenCL C and on the Message Passing Interface for distributed parallel execution, and is able to handle many-motors simulations that are out of reach of RMPCDMD for performance reasons.

Given the research activity on the topic of active colloids, we believe that providing a reference implementation of a chemically active MPCD fluid can serve both as a tool to reproduce earlier results from the literature and as a starting point for further modeling, by us or by other groups. RMPCDMD has not been used for published work yet. It is in use for ongoing research projects at the *Instituut voor Theoretische Fysica* of the KU Leuven.

Acknowledgements
The authors thank Raymond Kapral for discussions and for motivating this line of research, and Peter Colberg for his contributions to the code, his comments on the manuscript and for interesting discussions. The authors thank all authors of the open-source projects that they rely on for their research. PdB is a postdoctoral fellow of the Research Foundation-Flanders (FWO).

Competing Interests
The authors have no competing interests to declare.

Note
[1] Here, `bulk_rmpcd = T`, `bulk rate = 0.1` and `N_loop = 1024`. All other parameters are as in the repository file `experiments/01-single-dimer/dimer.parameters`.

References
1. **Frenkel, D** and **Smit, B** 2002 *Understanding molecular simulation: From algorithms to applications.* 2nd edition (Academic Press, San Diego, CA).
2. **Malevanets, A** and **Kapral, R** 1999 Mesoscopic model for solvent dynamics. *J. Chem. Phys.,* 110: 8605. DOI: https://doi.org/10.1063/1.478857
3. **Malevanets, A** and **Kapral, R** 2000 Solute molecular dynamics in a mesoscale solvent. *J. Chem. Phys.,* 112: 7260. DOI: https://doi.org/10.1063/1.481289
4. **Rohlf, K, Fraser, S** and **Kapral, R** 2008 Reactive multiparticle collision dynamics. *Comput. Phys. Commun.,* 179: 132. DOI: https://doi.org/10.1016/j.cpc.2008.01.027
5. **Rückner, G** and **Kapral, R** 2007 Chemically powered nanodimers. *Phys. Rev. Lett.,* 98: 150603. DOI: https://doi.org/10.1103/PhysRevLett.98.150603
6. **Prohm, C, Gierlak, M** and **Stark, H** 2012 Inertial microfluidics with multi-particle collision dynamics. *Eur.*

Phys. J. E, 35(8): 80. ISSN 1292–8941. DOI: http://doi.org/10.1140/epje/i2012-12080-3

7. **Lüsebrink, D, Yang, M** and **Ripoll, M** 2012 Thermophoresis of colloids by mesoscale simulations. *J. Phys. Cond. Matt.,* 24(28): 284132. DOI: https://doi.org/10.1088/0953-8984/24/28/284132
8. **Allen, MP** and **Tildesley, DJ** 1987 *Computer Simulation of Liquids,* (Clarendon Press, Oxford).
9. **de Buyl, P, Colberg, PH** and **Höfling, F** 2014 H5MD: A structured, efficient, and portable file format for molecular data. *Comp. Phys. Commun.,* 185: 1546–1553. DOI: https://doi.org/10.1016/j.cpc.2014.01.018
10. **de Buyl, P** 2015–2016 fortran_h5md. URL https://github.com/pdebuyl/fortran_h5md.
11. **Wilson, G, Aruliah, DA, Brown, CT, Hong, NPC, Davis, M, Guy, RT, Haddock, SHD, Huff, KD, Mitchell, IM, Plumbley, MD, White, BWEP** and **Wilson, P** 2014 Best practices for scientific computing. *PLoS Biol,* 12: e1001745. DOI: https://doi.org/10.1371/journal.pbio.1001745
12. **Git** URL https://git-scm.com/.
13. **de Buyl, P** 2015–2016 fortran_tester. URL https://github.com/pdebuyl/fortran_tester.
14. **Lee, SH** and **Kapral, R** 2004 Friction and diffusion of a brownian particle in a mesoscopic solvent. *J. Chem. Phys.* 121: 11163. DOI: https://doi.org/10.1063/1.1815291
15. **de Buyl, P** 2011–2016 Parsetext. URL https://github.com/pdebuyl/ParseText.
16. **Jones, E, Oliphant, T** and **Peterson, P** et al. 2001– SciPy: Open source scientific tools for Python. URL http://www.scipy.org/.
17. **Hunter, JD** 2007 Matplotlib: A 2D graphics environment. *Computing In Science & Engineering,* 9(3): 90–95. DOI: https://doi.org/10.1109/MCSE.2007.55
18. **Collette, A** 2013 *Python and HDF5* (O'Reilly).
19. **Ramachandran, P** and **Varoquaux, G** 2011 Mayavi: 3D visualization of scientific data. *Comput. Sci. Eng.,* 13(2): 40–51. DOI: https://doi.org/10.1109/MCSE.2011.35
20. **Brandl, G** 2007–2016 Sphinx. URL http://www.sphinx-doc.org/.
21. **Noguchi, H, Kikuchi, N** and **Gompper, G** 2007 Particle-based mesoscale hydrodynamic techniques. *EPL (Europhysics Letters),* 78(1): 10005. DOI: https://doi.org/10.1209/0295-5075/78/10005
22. **Ihle, T** and **Kroll, DM** 2001 Stochastic rotation dynamics: A galilean-invariant mesoscopic model for fluid flow. *Phys. Rev. E,* 63: 20201. DOI: https://doi.org/10.1103/PhysRevE.63.020201
23. **Allahyarov, E** and **Gompper, G** 2002 Mesoscopic solvent simulations: Multiparticle-collision dynamics of three-dimensional flows. *Phys. Rev. E,* 66: 36702. DOI: https://doi.org/10.1103/PhysRevE.66.036702
24. **Whitmer, JK** and **Luijten, E** 2010 Fluid–solid boundary conditions for multiparticle collision dynamics. *J. Phys. Condens. Matt.,* 22(10): 104106. DOI: https://doi.org/10.1088/0953-8984/22/10/104106
25. **Lamura, A, Gompper, G, Ihle, T** and **Kroll, DM** 2001 Multi-particle collision dynamics: Flow around a circular and a square cylinder. *EPL (Europhysics Letters),* 56: 319. DOI: https://doi.org/10.1209/epl/i2001-00522-9
26. **Andersen, HC** 1983 Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations. *J. Comp. Phys.,* 52: 24. DOI: https://doi.org/10.1016/0021-9991(83)90014-1
27. **Hamilton, CH** 2006 Compact hilbert indices. Technical Report CS-2006–07, Dalhousie University. URL https://www.cs.dal.ca/research/techreports/cs-2006-07.
28. **Salmon, JK, Moraes, MA, Dror, RO** and **Shaw, DE** 2011 Parallel random numbers: As easy as 1, 2, 3. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11,* pages 16:1–16:12 (ACM, New York, NY, USA). ISBN 978-1-4503-0771-0. DOI: https://doi.org/10.1145/2063384.2063405
29. **de Buyl, P** 2015–2016 random module. URL https://github.com/pdebuyl/random_module.
30. **Colberg, PH** 2013–2015 nano-dimer. URL https://colberg.org/nano-dimer/.
31. **Howse, JR, Jones, RAL, Ryan, AJ, Gough, T, Vafabakhsh, R** and **Golestanian, R** 2007 Self-motile colloidal particles: From directed propulsion to random walk. *Phys. Rev. Lett.,* 99: 48102. DOI: https://doi.org/10.1103/PhysRevLett.99.048102
32. **de Buyl, P** and **Kapral, R** 2013 Phoretic self-propulsion: a mesoscopic description of reaction dynamics that powers motion. *Nanoscale,* 5: 1337–1344. DOI: https://doi.org/10.1039/c2nr33711h

How to cite this article: de Buyl, P, Huang, M-J and Deprez, L 2017 RMPCDMD: Simulations of Colloids with Coarse-grained Hydrodynamics, Chemical Reactions and External Fields. *Journal of Open Research Software,* 5: 3, DOI: https://doi.org/10.5334/jors.142

Submitted: 17 August 2016 **Accepted:** 07 December 2016 **Published:** 11 January 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS