Load Balanced Particle Simulation with Automated Algorithm Selection

Philipp Neumann, Fabio Gratl, Steffen Seckler, and Hans-Joachim Bungartz

Abstract *ls1 mardyn* is a molecular dynamics (MD) simulation framework for multiphase and multicomponent investigations at small scales with application in process engineering. *AutoPas* is a library that employs auto-tuning for various particle simulation algorithms, data structures, and node-level parallelization patterns.

In the last year, we have focussed on improving the interfaces of *AutoPas* to support massively parallel, distributed-memory simulations. In this scope, we have revised the incorporation of *AutoPas* into *ls1 mardyn* and extended *ls1 mardyn* by diffusive load balancing.

1 Introduction

Molecular dynamics (MD) simulations have become a valuable tool for engineering applications. Based on Newton's laws of motion

$$\frac{d^2\mathbf{x}_p}{dt^2} = \frac{1}{m_p}\mathbf{F}_p,$$

P. Neumann (✉)
Helmut-Schmidt-Universität Hamburg, Department of Mechanical Engineering,
Holstenhofweg 85, 22043 Hamburg, Germany
e-mail: philipp.neumann@hsu-hh.de

F. Gratl · S. Seckler · H.-J. Bungartz
Technical University of Munich, Department of Informatics,
Boltzmannstr. 3, 85748 Garching, Germany
e-mail: gratl@in.tum.de

S. Seckler
e-mail: seckler@in.tum.de

H.-J. Bungartz
e-mail: bungartz@in.tum.de

W. E. Nagel et al. (eds.), *High Performance Computing in Science and Engineering '20*,
https://doi.org/10.1007/978-3-030-80602-6_33

the positions $\mathbf{x}_p$ of individual particles p can be tracked virtually. These particles move due to forces $\mathbf{F}_p = \sum_{q \neq p} \mathbf{F}_{pq}$, arising from interacting forces between particles p, q. We will concentrate on short-range interactions: only particles within a prescribed distance—the so-called *cut-off radius* r_c—interact with each other. This can be efficiently realized algorithmically by sorting the particles into Cartesian grid cells (linked cell method) that exhibit the mesh size r_c, or by storing potential interaction partners for each particle in lists (Verlet lists) [9]; to avoid the expensive construction of particle pair lists, these lists are only built every few time steps and comprise all particle pairs within a distance $r_c + s$ where s is the so-called skin radius. If s is rather large, the lists have to be built less frequently, but this comes at increasing particle pair traversal cost.

Amongst others, MD simulations can be used to sample thermodynamic properties from large systems of small molecules. This allows, e.g., to determine equations of state for complex fluids, to investigate bubble formation [5], interfacial flows [6] or droplet coalescence [10].

In a long-standing, interdisciplinary collaboration of computer scientists and mechanical engineers, the MD simulation software *ls1 mardyn* has evolved to assess such systems [7, 15]. *ls1 mardyn* supports vectorization, shared- as well as distributed-memory parallelism [2, 14, 16] including dynamic load balancing which allows accounting for load imbalances due to heterogeneous particle distributions or due to the use of heterogeneous hardware [11, 12].

Besides load balancing on distributed-memory systems, another grand challenge is given by the fact that a great variety of algorithmic realizations are available to simulate short-range MD systems [1, 4, 8, 9, 17]. An optimal algorithm/data structure layout/parallelization approach (at shared-memory level) strongly depends on both the underlying hardware and the actual problem to solve. With new hardware rapidly evolving and with the heterogeneity of hardware design still increasing in the HPC sector, finding and incorporating the best algorithm into a simulation software is thus demanding.

Over the last years, the authors have developed the particle simulation library *AutoPas* [3], which provides various data structures, particle traversal schemes, and shared-memory parallelization approaches and which automatically selects the optimal combination thereof at run time.

In the following, we detail recent developments that build upon and extend the load balancing capabilities of *ls1 mardyn* as well as our prototypical integration of *AutoPas* and *ls1 mardyn* from the previous report [13]. We discuss the integration of an alternative load balancing library, cf. Sect. 2, and elaborate on challenges when integrating *AutoPas* and *ls1 mardyn* for distributed-memory simulations, cf. Sect. 3. We close with a summary and an outlook to future work in Sect. 4.

Parts of this work have been submitted for publication [11].

2 Load Balancing

So far, *ls1 mardyn* employed a kd-tree-based decomposition to balance the load between MPI processes. Its recursive bisectioning is an efficient means to partition and distribute loads. Yet, the implemented method bears some disadvantages. Due to successive bisectioning, every dimension is visited a multiple number of times in the partitioning process. Besides, a global collection of computational loads, that is particle or at least cell-averaged data, is required and re-balancing can potentially alter the overall topology of the simulation partitioning significantly from one to the next time step.

Due to these points, the *A Load-balancing Library (ALL)*[1] which is developed at the Jülich Supercomputing Center has been incorporated as an alternative to the existing kd-tree implementation. Still following the hierarchical/recursive splitting approach, *ALL* allows to split the domain into multiple subsections in one recursion step and limits the number of splits along every dimension to one. Moreover, to avoid rigorous repartitioning, diffusive load balancing was incorporated into *ls1 mardyn*. Since the diffusive load balancing algorithm operates (more or less) strictly locally, loads are migrated between neighboring processes only. This limits the significance of changes in the parallel topology.

The diffusive load balancing in conjunction with the *ALL*-based partitioner was evaluated in various scenarios, including

– scenario *droplet coalescence*: the coalescence of two nanodroplets that are suspended in a vapor phase, cf. Fig. 1(a),
– scenario *spinodal decomposition*: a fluid is rapidly cooled down and separates into vapor and liquid phases, cf. Fig. 1(b),
– scenario *exploding liquid*: a compressed, hot liquid expands in vacuum, cf. Fig. 1(c).

While no performance improvements were observed in the *spinodal decomposition*—which was expected, since the occurring inhomogeneities are rather fine-scale—, both kd-tree- and diffusive *ALL*-based load balancing significantly boosted performance in the *droplet coalescence*. This is in accordance with expectations since the merging process of the two droplets is relatively slow and thus, load changes occur on rather long time scales. In contrast, diffusive *ALL*-based load balancing outperformed the kd-tree approach significantly for the *exploding liquid*: here, the overall particle distribution in the entire computational domain changes drastically, with particles getting sucked outward at the beginning and bouncing back from the outer boundaries afterwards. This rather directed motion of the film fragments is well reflected by the diffusive load balancing algorithm, which effectively propagates loads between spatially neighbored subdomains and MPI processes, respectively.

[1] https://e-cam.readthedocs.io/en/latest/Meso-Multi-Scale-Modelling-Modules/modules/ALL_library/tensor_method/readme.html.

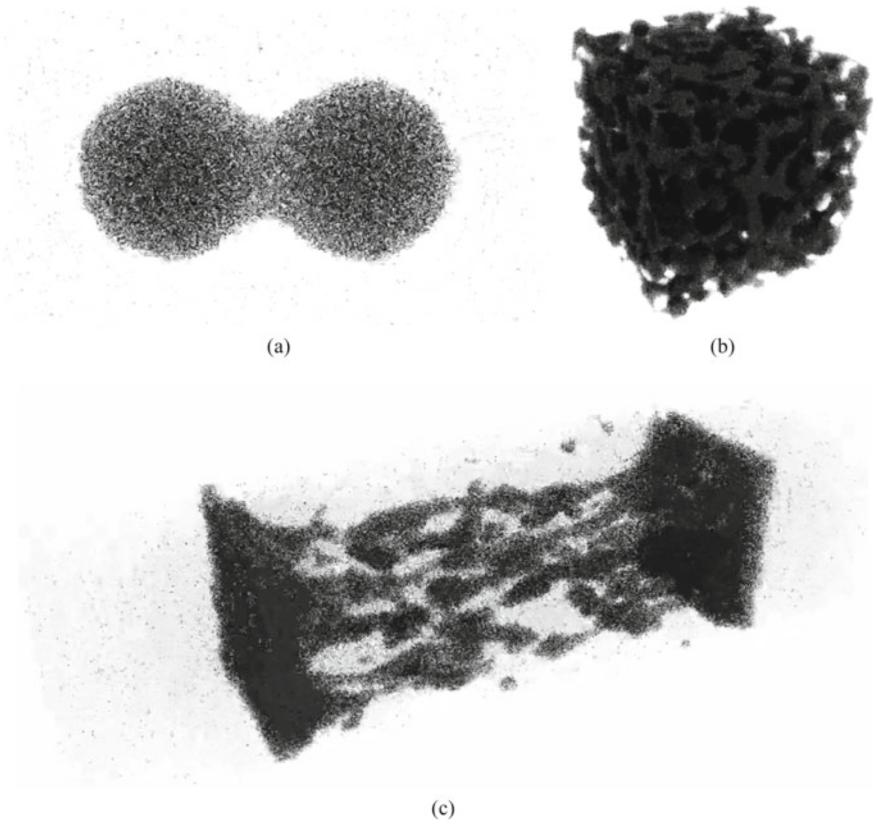
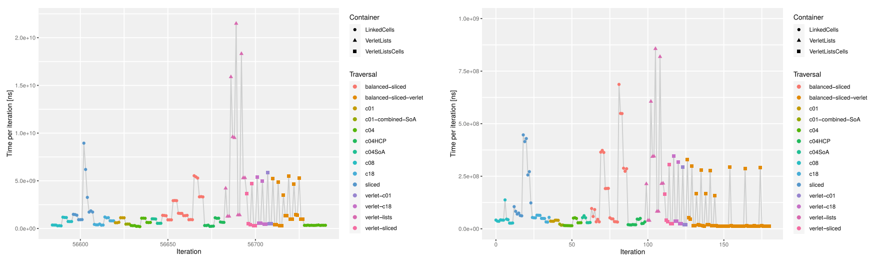


Fig. 1 Scenarios considered in our work. (a) Droplet coalescence. (b) Spinodal decomposition. (c) Exploding liquid; the liquid is initially placed in a vertical box-like form in the middle of the domain and expands/breaks up towards the left and the right of the computational domain

3 Integrating AutoPas and Ls1 Mardyn for Distributed-Memory Simulations

In a prior report [13], we had commented on the integration of *AutoPas* into *ls1 mardyn*, enabling an "on-the-fly" exchange of data structures, particle pair traversal schemes, and parallelization methods during an MD simulation. *AutoPas* already comprised a few implementations and configuration possibilities in terms of data structures, linked cell-based particle traversal schemes, and linked cell-based shared-memory parallelization approaches. At this stage, it had already been possible to execute the *spinodal decomposition* in node-level simulations and to automatically tune over some configurations.

Over the last year, *AutoPas* was significantly extended. In particular, Verlet lists as an alternative to the linked cell method to traverse particle pairs for force interactions



(a) *Exploding liquid*, mid of simulation, inhomogeneous

(b) *Spinodal decomposition*, start of simulation, homogeneous

Fig. 2 Snapshot of simulation performance during the auto-tuning phase. In this phase, 48 algorithmic configurations are tested, employing an exhaustive search. Data points are colored by their traversal choice and the symbol is used to identify the particle container; note that further variants (AoS vs SoA data structures; Newton3-optimization to reduce computations; load balancing schemes) are partly employed and varied (resembling multiple occurrences of the same symbols in the plot). Always exactly three data points represent the same configuration, corresponding to three samples being taken. The x-axis shows the number of iterations, the y-axis shows the time for a single iteration. (a) *Exploding liquid*; the configuration, that is tested last, is chosen and is taken over for the following 5000 iterations. (b) *Spinodal decomposition*.

were incorporated, including a vectorizing variant thereof (cluster lists, as used in the software Gromacs, for example). This yields a total of > 70 discrete algorithmic configurations, cf. [11] for details.

In Fig. 2, the performance behavior of the exhaustive search tuning strategy is visualized. Every available algorithmic configuration is measured over three iterations each. Afterwards, the fastest one is selected for the next period of—in this case—5000 iterations. Verlet lists appear to be fluctuating in performance. This is, however, due to the fact that in their first iteration, the neighbor lists are built, which is taken into account here. As the two setups differ in their structure, differences in the performance profiles can be seen, especially between the faster configurations. It also becomes apparent, that there is still work to be done in a) improving the average performance of individual configurations and b) only testing promising configurations. Both of these tasks are subject to current research.

A specific challenge, however, arises when distributed-memory, that is MPI-parallel, MD simulations shall be supported by AutoPas. Due to potential particle distribution inhomogeneities over all MPI processes, cf. Sect. 2, every MPI process might require a different *AutoPas* configuration to execute at optimal performance, cf. Fig. 3. In particular, one MPI process might make use of Verlet lists, which requires less MPI-based particle data exchange with neighboring processes (i.e., some information only needs to be exchanged every few time steps), while another MPI process might employ linked cells which require particle data exchange in every time step. This also renders the handling of particles at the interface of the embedding simulation (*ls1 mardyn* in our case) and *AutoPas* challenging, since particle exchange and updates need to be carried out consistently.

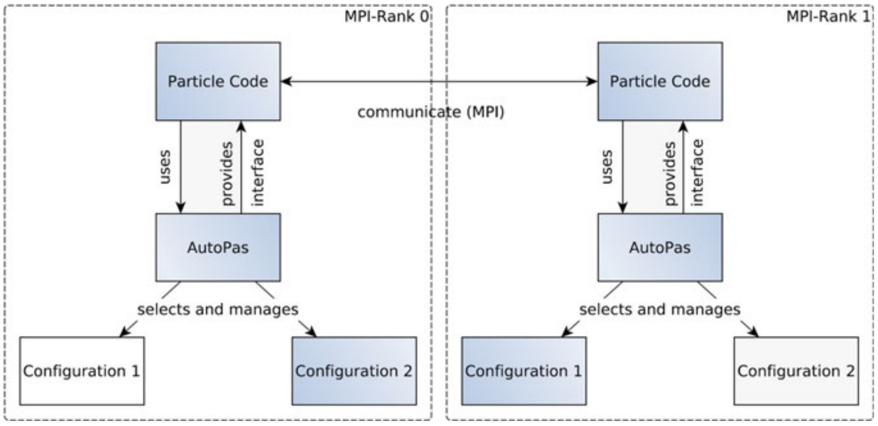


Fig. 3 MPI-parallel use of *AutoPas* in *ls1 mardyn*. *AutoPas* may execute different configurations on different MPI processes. Yet, a common interface and semantics is required to bridge all potential particle simulation configurations of *AutoPas*

To solve this, the interface to *AutoPas* was designed such that it follows the Verlet list idea, in the sense that the particle container of *AutoPas* is only being updated every few time steps. This implies that the linked cell implementation of *AutoPas* needs to operate on slightly enlarged cells, corresponding to the skin radius s that is typically added to the cut-off radius r_c in the Verlet list approach to avoid frequent list rebuilds. Yet, refined linked cells are also supported, which means that more neighbor cells need to be searched. Although the Verlet-like approach comes with rather good code maintainability features and performance, the overall usability of the library, especially the particle exchange in the *AutoPas*-embedding MD simulation, becomes slightly more advanced.

Putting things together, we have combined *ALL*-based diffusive load balancing and *AutoPas*-based automated node-level algorithm selection. Figure 4 shows the different algorithm choices per subdomain of an exemplary load balanced domain decomposition for the scenario *droplet coalescence*. We simulated the three aforementioned scenarios *droplet coalescence*, *exploding liquid* and *spinodal decomposition* with this simulation technology on two platforms. Due to the HLRS machine HAWK getting set up in 2019, we first established and tuned simulations on SuperMUC-NG at LRZ during that time. Figure 5 shows the scalability for the *exploding liquid* on both machines. We can first observe that, expectedly, the per-node performance on HAWK is significantly better than on SuperMUC-NG in terms of simulated time steps per second. The *ALL*-based load balancing with auto-tuning via *AutoPas* exhibits best performance on SuperMUC-NG compared to the other configurations. This approach also scales well up to 8 nodes on HAWK. For larger node counts, however, the curve currently flattens. Reasons for this lie in one node featuring 8 NUMA domains; since we use one MPI process per NUMA domain, this yields significantly more MPI processes on HAWK compared to SuperMUC-NG.

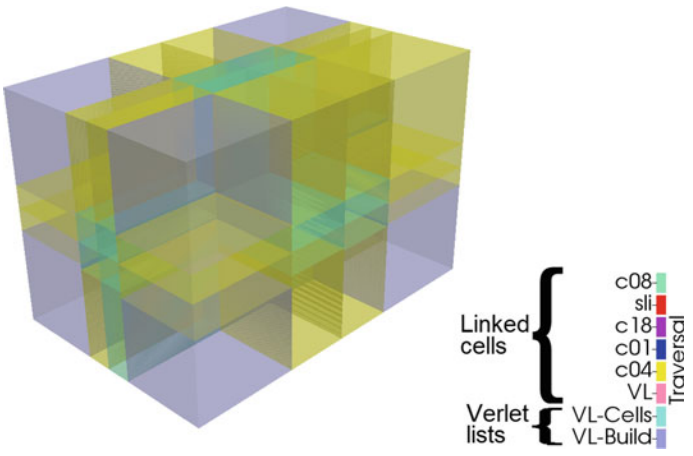


Fig. 4 Algorithm selection on a load-balanced simulation of *droplet coalescence*. c08, sli, c18, c01, c04 correspond to OpenMP parallelization schemes that operate on linked cells. In contrast, VL, VL-Cells, VL-Build are parallel schemes that operate on Verlet lists

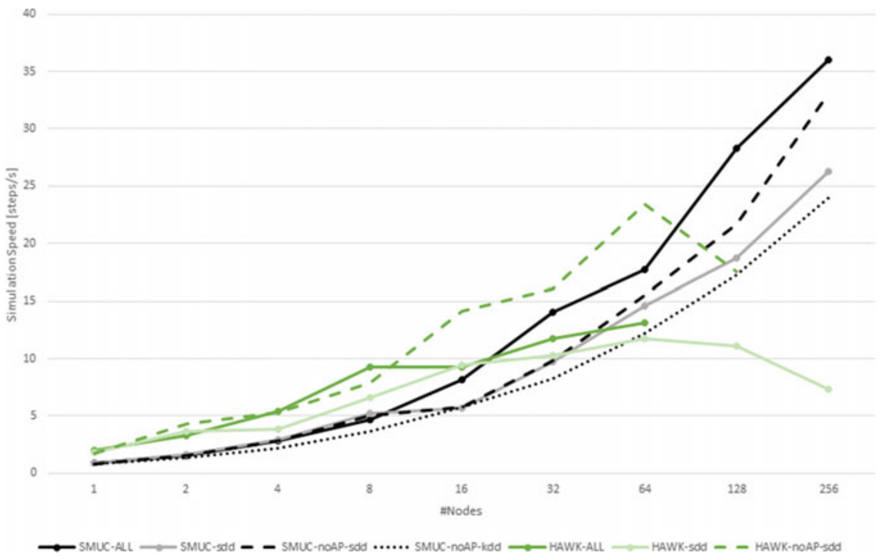


Fig. 5 Scalability of the scenario *exploding liquid* on two platforms SuperMUC-NG and HAWK. –ALL: *ALL*-based diffusive load balancing; –sdd: standard domain decomposition; –noAP-sdd: standard domain decomposition with *ls1 mardyn* kernels instead of using *AutoPas*; –noAP-kdd: kd-tree decomposition with *ls1 mardyn* kernels instead of using *AutoPas*

Besides, logging was not deactivated for *AutoPas* on both machines, which impedes performance even more on HAWK, again, due to the increased number of MPI processes. More detailed analysis and performance tuning on HAWK is subject to the current investigation.

4 Summary and Outlook

We presented recent advances in our particle simulation software *ls1 mardyn*. Automated algorithm selection has been enabled in *ls1 mardyn* via the library *AutoPas*, yielding optimal node-level performance. We further improved parallel performance by incorporating diffusive balancing. The node-level throughput on HAWK suggests very high performance for the three scenarios that we discussed in the future. However, performance gains in the multi-node case are currently not optimal on HAWK, due to the recent porting of our software to this architecture. This requires further performance analysis and more tuning of the number of MPI processes and OpenMP threads per node. Current and future work further focus on techniques to improve auto-tuning in *AutoPas*: testing all combinations of algorithms is tedious and requires many time steps. Therefore, Bayesian and other data analysis methods are being evaluated in this scope.

Acknowledgements The presented work was carried out in the scope of the Large-Scale Project *Extreme-Scale Molecular Dynamics Simulation of Droplet Coalescence*, acronym GCS-MDDC, of the Gauss Centre for Supercomputing. Financial support by the Federal Ministry of Education and Research, project *Task-based load balancing and auto-tuning in particle simulations (TaLPas)*, grant numbers 01IH16008A/B, is acknowledged.

References

1. M. Abraham, T. Murtola, R. Schulz, S. Páll, J. Smith, B. Hess, E. Lindahl, GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX **1–2**, 19–25 (2015)
2. W. Eckhard et al., 591 TFLOPS multi-trillion particles simulation on SuperMUC, in *Supercomputing. ISC 2013*. ed. by J.M. Kunkel, T. Ludwig, H.W. Meuer. Lecture Notes in Computer Science. (Springer, Berlin, Heidelberg, 2013), pp. 1–12. https://doi.org/10.1007/978-3-642-38750-0_1
3. F. Gratl, S. Seckler, N. Tchipev, H.-J. Bungartz, P. Neumann, AutoPas: auto-tuning for particle simulations. in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, (2019), pp. 748–757
4. C. Hu, X. Wang, J. Li, X. He, S. Li, Y. Feng, S. Yang, H. Bai, Kernel optimization for short-range molecular dynamics. Comput. Phys. Commun. **211**, 31–40 (2017)
5. K. Langenbach, M. Heilig, M. Horsch, H. Hasse, Study of homogeneous bubble nucleation in liquid carbon dioxide by a hybrid approach combining molecular dynamics simulation and density gradient theory. J. Chem. Phys. **148**, 124702 (2018)
6. G. Nagayama, P. Cheng, Effects of interface wettability on microscale flow by molecular dynamics simulation. Int. J. Heat Mass Transf. **47**, 501–513 (2004)

7. C. Niethammer, S. Becker, M. Bernreuther, M. Buchholz, W. Eckhardt, A. Heinecke, S. Werth, H.-J. Bungartz, C. Glass, H. Hasse, J. Vrabec, M. Horsch, ls1 Mardyn: the massively parallel molecular dynamics code for large systems. J. Chem. Theory Comput. **10**(10), 4455–4464 (2014)
8. S. Páll, B. Hess, A flexible algorithm for calculating pair interactions on SIMD architectures. Comput. Phys. Commun. **184**(12), 2641–2650 (2013)
9. D. Rapaport, *The Art of Molecular Dynamics Simulation* (Cambridge University Press, Cambridge, 2004)
10. L. Rekvig, D. Frenkel, Molecular simulations of droplet coalescence in oil/water/surfactant systems. J. Chem. Phys. **127**, 134701 (2007)
11. S. Seckler, F. Gratl, M. Heinen, J. Vrabec, H.-J. Bungartz, P. Neumann, Autopas in ls1 mardyn: massively parallel particle simulations with node-level auto-tuning. J. Comput. Sci. **50**, 101296 (2021)
12. S. Seckler, N. Tchipev, H.-J. Bungartz, P. Neumann, Load balancing for molecular dynamics simulations on heterogeneous architectures. in *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*, (2016), pp. 101–110
13. S. Seckler, F. Gratl, N. Tchipev, M. Heinen, J. Vrabec, H.-J. Bungartz, P. Neumann, Load balancing and auto-tuning for heterogeneous particle systems using ls1 MARDYN. in: *High Performance Computing in Science and Engineering 2019* (2019), To be published
14. N. Tchipev, Algorithmic and Implementational Optimizations of Molecular Dynamics Simulations for Process Engineering (2020), Dissertation
15. N. Tchipev, S. Seckler, M. Heinen, J. Vrabec, F. Gratl, M. Horsch, M. Bernreuther, C.W. Glass, C. Niethammer, N. Hammer, B. Krischok, M. Resch, D. Kranzlmüller, H. Hasse, H.-J. Bungartz, P. Neumann, Twetris: twenty trillion-atom simulation. Int. J. High Perform. Comput. Appl. **33**(5), 838–854 (2019)
16. N. Tchipev, A. Wafai, C. Glass, W. Eckhardt, A. Heinecke, H.-J. Bungartz, P. Neumann, Optimized force calculation in molecular dynamics simulations for the intel Xeon phi, in *Euro-Par 2015: Parallel Processing Workshops. Euro-Par 2015*. ed. by S. Hunold. Lecture Notes in Computer Science. (Springer International Publishing, Cham, 2015), pp. 774–785. https://doi.org/10.1007/978-3-319-27308-2_62
17. X. Wang, J. Li, J. Wang, X. He, N. Nie, Kernel optimization on short-range potentials computations in molecular dynamics simulations, in *Big Data Technology and Applications. BDTA 2015*. ed. by W. Chen. Communications in Computer and Information Science. (Springer, Singapore, 2016), pp. 269–281. https://doi.org/10.1007/978-981-10-0457-5_25