ls1 mardyn: The massively parallel molecular dynamics code for large systems

Christoph Niethammer,[†] Stefan Becker,[‡] Martin Bernreuther,[†] Martin Buchholz,[¶] Wolfgang Eckhardt,[¶] Alexander Heinecke,[¶] Stephan Werth,[‡] Hans-Joachim Bungartz,[¶] Colin W. Glass,[†] Hans Hasse,[‡] Jadran Vrabec,[§] and Martin Horsch[*,‡]

High Performance Computing Center Stuttgart, Nobelstr. 19, 70569 Stuttgart, Germany, University of Kaiserslautern, Laboratory of Engineering Thermodynamics, Erwin-Schrödinger-Str. 44, 67663 Kaiserslautern, Germany, TU München, Chair for Scientific Computing in Computer Science, Boltzmannstr. 3, 85748 Garching, Germany, and University of Paderborn, Laboratory of Thermodynamics and Energy Technology, Warburger Str. 100, 33098 Paderborn, Germany

E-mail: martin.horsch@mv.uni-kl.de

Abstract

The molecular dynamics simulation code *ls1 mardyn* is presented. It is a highly scalable code, optimized for massively parallel execution on supercomputing architectures, and currently holds the world record for the largest molecular simulation with over four trillion particles. It enables the application of pair potentials to length and time

[*]To whom correspondence should be addressed
[†]High Performance Computing Center Stuttgart (HLRS), Germany
[‡]Laboratory of Engineering Thermodynamics (LTD), Univ. of Kaiserslautern, Germany
[¶]Scientific Computing in Computer Science (SCCS), TU München, Germany
[§]Thermodynamics and Energy Technology (ThEt), Univ. of Paderborn, Germany

scales which were previously out of scope for molecular dynamics simulation. With an efficient dynamic load balancing scheme, it delivers high scalability even for challenging heterogeneous configurations. Presently, multi-center rigid potential models based on Lennard-Jones sites, point charges and higher-order polarities are supported. Due to its modular design, *ls1 mardyn* can be extended to new physical models, methods, and algorithms, allowing future users to tailor it to suit their respective needs. Possible applications include scenarios with complex geometries, e.g. for fluids at interfaces, as well as non-equilibrium molecular dynamics simulation of heat and mass transfer.

1 Introduction

The molecular dynamics (MD) simulation code *ls1 mardyn* (large systems 1: molecular dynamics) is presented here. The *ls1 mardyn* program is an interdisciplinary endeavor, whose contributors have backgrounds from engineering, computer science and physics, aiming at studying challenging scenarios with up to trillions of molecules. In the considered systems, the spatial distribution of the molecules may be heterogeneous and subject to rapid unpredictable change. This is reflected by the algorithms and data structures as well as a highly modular software engineering approach. The source code of *ls1 mardyn* is made publicly available as free software under a two-clause BSD license.[1]

Molecular modelling and simulation has become a powerful computational method[2,3] and is applied to a wide variety of areas such as thermodynamic properties of fluids,[4] phase equilibria,[5,6] interfacial properties,[7] phase transitions,[8,9] transport coefficients,[10] adsorption,[11,12] mechanical properties of solids,[13] flow phenomena,[14,15] polymer properties,[16] protein folding,[17,18] or self-assembly.[19] The sound physical basis of the approach makes it extremely versatile. For a given force field, the phase space can be explored by molecular dynamics simulation under a variety of boundary conditions, which allows gathering information on all thermodynamic states and processes on the molecular level. If required, external forces (e.g. an electric field) can be imposed in addition to the intermolecular interactions.

MD simulation has an extremely high temporal and spatial resolution of the order of 10^{-15} seconds and 10^{-11} meters, respectively. This resolution is useful for studying physical phenomena at small length scales, such as the structure of fluid interfaces. With a time discretization on the femtosecond scale, rapid processes are immediately accessible, while slower processes may require particularly devised sampling techniques such as metadynamics.[20] The number of molecules is also a challenge for molecular simulation. While systems of practical interest contain extremely large numbers of molecules, e.g. of the order of 10^{23}, the largest ensembles that can be handled today are of the order of 10^{12} molecules.[21] This limitation is usually addressed by focusing on representative subvolumes, containing a limited number of molecules, to which an appropriate set of boundary conditions is applied. Depending on the type of information that is determined, e.g. transport properties[22] or phase equilibria[5,6] of bulk phases, a number of molecules of the order of 1 000 may be sufficient. However, non-equilibrium scenarios such as condensation[23,24] or mass transfer through nanoporous membranes[15,25] require much larger simulation volumes.

There are so many scalable MD codes available that a comprehensive discussion would be beyond the scope of the present paper. For the development of MD codes, as for any software, there are trade-offs between generality and optimization for a single purpose, which no particular implementation can completely evade. Several popular MD simulation environments are tailored for studying biologically relevant systems, with typical application scenarios including conformational sampling of macromolecules in aqueous solution. The relaxation processes of such systems are often several orders of magnitude slower than for simple fluids, requiring an emphasis on sampling techniques and long simulation times, but not necessarily on large systems.

The *AMBER* package,[26] for instance, scales well for systems containing up to 400 000 molecules, facilitating MD simulations that reach the microsecond time scale.[27] Similarly, *GROMACS*[28,29] and *NAMD*,[30] which also have a focus on biosystems, have been shown to perform quite efficiently on modern HPC architectures as well. *Tinker* was optimized for

biosystems with polarizable force fields,[31] whereas *CHARMM*,[32] which was co-developed by Nobel prize winner Martin Karplus, is suitable for coupling classical MD simulation of macromolecules with quantum mechanics.[33] The *LAMMPS* program[34–37] as well as *DL_POLY*,[38] which scales well for homogeneous fluid systems with up to tens of millions of molecules, and *ESPResSo*,[39] which emphasizes its versatility and covers both molecular and mesoscopic simulation approaches, are highly performant codes which aim at a high degree of generality, including many classes of pair potentials and methods. The *ms2* program performs well for the simulation of vapor-liquid equilibria and other thermodynamic properties,[4] but is limited to relatively small numbers of molecules. The *IMD* code,[40,41] which has twice before held the MD simulation world record in terms of system size, has a focus on multi-body potentials for solids.

With *ls1 mardyn*, which is presented here, a novel MD code is made available to the public. It is more specialized than most of the molecular simulation programs mentioned above. In particular, it is restricted to rigid molecules, and only constant volume ensembles are supported, so that the pressure cannot be specified in advance. Electrostatic long-range interactions, beyond the cut-off radius, are considered by the reaction field method,[42] which cannot be applied to systems containing ions. However, *ls1 mardyn* is highly performant and scalable. Holding the present world record in simulated system size,[21] it is furthermore characterized by a *modular structure*, facilitating a high degree of flexibility within a *single code base*. Thus, *ls1 mardyn* is not only a simulation engine, but also a framework for developing and evaluating simulation algorithms, e.g. different thermostats or parallelization schemes. Therefore, its software structure supports alternative implementations for methods in most parts of the program, including core parts such as the numerical integration of the equations of motion. The C++ programming language was used, including low level optimizations for particular HPC systems. In this way, *ls1 mardyn* has been proven to run efficiently on a variety of architectures, from ordinary workstations to massively-parallel supercomputers.

In a fluid system, neighborhood relations between molecules are always subject to rapid change. Thus, the neighbor molecules have to be redetermined throughout the simulation. For this purpose, *ls1 mardyn* employs a linked-cell data structure,[43–45] which is efficiently parallelized by spatial domain decomposition.[46,47] Thereby, the simulation volume is divided into subvolumes that are assigned to different processes. Interactions with molecules in adjacent subvolumes are explicitly accounted for by synchronized halo regions.[3]

Using *ls1 mardyn*, a wide range of simulation scenarios can be addressed, and pre-release versions of *ls1 mardyn* have already been successfully applied to a variety of topics from chemical and process engineering: Nucleation in supersaturated vapors[24,48–50] was considered with a particular focus on systems with a large number of particles.[23,51,52] On the SuperMUC, over four trillion molecules were simulated.[21] The vapor-liquid surface tension and its dependence on size and curvature was characterized.[49,53–56] The *ls1 mardyn* program was furthermore employed to investigate fluid flow through nanoporous membrane materials[57] and adsorption phenomena such as the fluid-solid contact angle in dependence on the fluid-wall interaction.[12,58]

Scenario generators for *ls1 mardyn* are available both internally, i.e. without hard disk input/output, and as external executables. The internal generators create the initial configuration directly in memory, which is distributed among the processes, facilitating a better scalability for massively-parallel execution. A generalized output plugin interface can be used to extract any kind of information during the simulation and to visualize the simulation trajectory with MegaMol[59,60] and other compatible tools.

This paper is organized as follows: Section 2 describes molecular models which are available in *ls1 mardyn*. Section 3 introduces the underlying computational methods. The implemented load balancing approach is discussed in detail in Section 4. A performance analysis of *ls1 mardyn* is presented in Section 5, including results obtained on two of the fastest HPC systems.

2 Interaction models in *ls1 mardyn*

Molecular motion has two different aspects: External degrees of freedom, corresponding to the translation and rotation with respect to the molecular center of mass, as well as internal degrees of freedom that describe the conformation of the molecule. In *ls1 mardyn*, molecules are modeled as rigid rotators, disregarding internal degrees of freedom and employing effective pair potentials for the intermolecular interaction. This modeling approach is suitable for all small molecules which do not exhibit significant conformational transitions. An extension of the code to internal degrees of freedom is the subject of a presently ongoing development, which is not discussed here. The microcanonical (NVE), canonical (NVT) and grand-canonical (μVT) ensembles are supported, whereby the temperature is (for NVT and μVT) kept constant by velocity rescaling.

The Lennard-Jones (LJ) potential

$$U_{\mathrm{LJ}}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{1}$$

with the size parameter σ and the energy parameter ε, is used to account for repulsive and dispersive interactions. It can also be employed in a truncated and shifted (LJTS) version.[2] LJ potential parameters for the unlike interaction, i.e. the pair potential acting between molecules of different species, are determined by the Lorentz and Berthelot combination rules,[61–63] which can be further adjusted by binary interaction parameters.[64–66]

Point charges and higher-order point polarities up to second order (i.e. dipoles and quadrupoles), are implemented to model electrostatic interactions in terms of a multipole expansion.[67,68] This allows an efficient computational handling while sufficient accuracy is maintained for the full range of thermophysical properties.[69] The Tersoff potential[70] can be used within *ls1 mardyn* in order to accurately describe a variety of solid materials.[71,72] As a multi-body potential, it is computationally more expensive than electrostatics and the LJ potential.

Any system of units can be used in *ls1 mardyn* as long as it is algebraically consistent and includes the Boltzmann constant $k_\mathrm{B} = 1$ as well as the Coulomb constant $k_C = 1/(4\pi\varepsilon_o) = 1$ among its basic units. Thereby, expressions for quantities related to temperature and the electrostatic interactions are simplified. The units of size, energy and charge are related (by Coulomb's law and the Coulomb constant unit) and cannot be specified independently of each other. A temperature is converted to energy units by using $k_\mathrm{B} = 1$, and vice versa. In this way, all other units are determined; for an example, see 1.

Table 1: A consistent set of atomic units (used by the scenario generators).

Boltzmann constant	$k_\mathrm{B} = 1$
Coulomb constant	$k_C = (4\pi\varepsilon_0)^{-1} = 1$
Unit length	$l_0 = 1\ a_\mathrm{H}$ (Bohr's radius) $= 5.29177 \times 10^{-11}$ m
Elementary charge	$q_0 = 1$ e $= 9.64854 \times 10^9$ C/mol
Unit mass	$m_0 = 1\,000\ u = 1$ kg/mol
Unit density	$\rho_0 = 1/l_0^3 = 11\,205.9$ mol/l
Unit energy	$E_0 = k_C q_0^2/l_0 = 4.35946 \times 10^{-18}$ J
Unit temperature	$T_0 = E_0/k_\mathrm{B} = 315\,775$ K
Unit pressure	$p_0 = \rho_0 E_0 = 2.94211 \times 10^{13}$ Pa
Unit time	$t_0 = l_0\sqrt{m_0/E_0} = 3.26585 \times 10^{-14}$ s
Unit velocity	$v_0 = l_0/t_0 = 1620.35$ m/s
Unit acceleration	$a_0 = l_0/t_0^2 = 4.96148 \times 10^{-15}$ m/s^2
Unit dipole moment	$D_0 = l_0 q_0 = 2.54176$ D
Unit quadrupole moment	$Q_0 = l_0^2 q_0 = 1.34505$ DÅ

3 Data structures and numerical integration

The computational core of every MD program is the calculation of forces and torques acting on the molecules, which are based on molecular models for the physical interactions. The choice of a suitable model depends on many factors, like the material to be simulated, the physical effects studied or the desired accuracy. Different models may require substantially different algorithms for the numerical solution of Newton's equations of motion. This can even necessitate major changes in the software structure, e.g. when long-range interactions

have to be considered explicitly in addition to short-range interactions, or when models with internal degrees of freedom are used instead of rigid molecular models.

In the present version of *ls1 mardyn*, only short-range interactions up to a specified cut-off radius are explicitly computed. The long-range contribution to the pressure and the energy is approximated by isotropic cut-off corrections, i.e. by a mean-field integral for the dispersive interaction, which is supplemented by the reaction field method[42] for dipolar molecules.

Calculating short range interactions in dynamic systems requires an efficient algorithm for finding neighbors. For this purpose, *ls1 mardyn* employs an adaptive linked-cell algorithm.[73] The basic linked-cell algorithm divides the simulation volume into a grid of equally sized cubic cells, which have an edge length equal to the cut-off radius r_c. This ensures that all interaction partners for any given molecule are situated either within the cell of the molecule itself or the 26 surrounding cells. Nonetheless, these cells still contain numerous molecules which are beyond the cut-off radius. The volume covered by 27 cells is 27 r_c^3, whereas the relevant volume containing the interaction partners is a sphere with a radius r_c, corresponding to $4\pi r_c^3/3 \approx 4.2\ r_c^3$. Thus, in case of a homogeneous configuration, only 16% of all pairs for which the distance is computed are actually considered for intermolecular interactions.

For fluids with computationally inexpensive pair potentials, e.g. molecules modeled by a single LJ site, the distance evaluation requires approximately the same computational effort as the force calculation. Reducing the volume which is examined for interaction partners can therefore significantly reduce the overall runtime. This can be achieved by using smaller cells with an edge length of e.g. $r_c/2$, which reduces the considered volume from 27 r_c^3 to 15.6 r_c^3, so that for a homogeneous configuration, 27% of the computed distances are smaller than the cut-off radius.

However, smaller cells also cause an additional effort, since 125 instead of 27 cells have to be traversed. This is only beneficial for regions with high density, where the cost of cell traversal is small compared to the cost of distance calculation. Moreover, many applications

of molecular dynamics, such as processes at interfaces, are characterized by a heterogeneous distribution of the molecules and thus by a varying density throughout the domain. To account for this, adaptive cell sizes depending on the local density[73] are (optionally) used in *ls1 mardyn*, cf. 1. Due to periodic boundary conditions, molecules leaving the simulation volume on one side re-enter it on the opposite side, and molecules near the volume boundary interact with those on the opposite side of the volume.

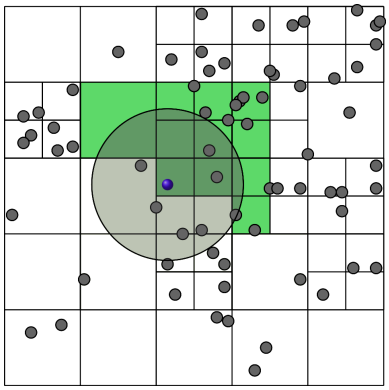


Figure 1: Adaptive cell sizes for an inhomogeneous molecule distribution. Cells that contain significantly more molecules than others are divided into smaller subcells. According to Newton's third law (*actio = reactio*), two interacting molecules experience the same force (in opposite directions) due to their mutual interaction, so that a suitable enumeration scheme can be employed to reduce the amount of cell pairs that are taken into account. Following such a scheme, it is sufficient to compute the force exerted by the highlighted molecule on molecules from the highlighted cells.[73]

After the calculation of all pairwise interactions, the resulting force and torque acting on each molecule is obtained by summation. Newton's equations of motion are solved numerically for all molecules to obtain the configuration in the next time step. Most common methods to integrate these equations are single-step methods, where a new position at the time $t + \delta t$ is calculated from the position, velocity and acceleration at the time t. This is repeated for a specified number of time steps n up to the time $t + n\,\delta t$. Usually, algorithms based on the (Størmer-)Verlet method[74,75] are used. Instead, *ls1 mardyn* employs the leapfrog method,[76] which is algebraically equivalent to the Verlet method but more accurate

numerically. Positions $\boldsymbol{r}_i$ and velocities $\dot{\boldsymbol{r}}_i$ are calculated by

$$\dot{\boldsymbol{r}}_i\left(t+\frac{\delta t}{2}\right) = \dot{\boldsymbol{r}}_i\left(t-\frac{\delta t}{2}\right)+\delta t\,\ddot{\boldsymbol{r}}_i(t), \tag{2}$$

$$\boldsymbol{r}_i(t+\delta t) = \boldsymbol{r}_i(t)+\delta t\,\dot{\boldsymbol{r}}_i\left(t+\frac{\delta t}{2}\right). \tag{3}$$

For molecules which are not rotationally symmetric, the equations for angular momentum $\boldsymbol{j}$ and orientation $\boldsymbol{q}$ (with $\boldsymbol{q}$ being a quaternion)[2] are applied as well. In analogy to Eqs. (2) and (3) for the translational motion, the rotational motion is described by

$$\boldsymbol{j}_i\left(t+\frac{\delta t}{2}\right) = \boldsymbol{j}_i\left(t-\frac{\delta t}{2}\right)+\delta t\,\boldsymbol{\tau}_i(t), \tag{4}$$

$$\boldsymbol{q}_i(t+\delta t) = \boldsymbol{q}_i(t)+\delta t\,d\dot{\boldsymbol{q}}_i\left(t+\frac{\delta t}{2}\right), \tag{5}$$

where $\boldsymbol{\tau}_i$ is the torque divided by the rotational moment of inertia.

4 Parallelization and load balancing

4.1 Load balancing based on domain decomposition

A parallelization scheme using spatial domain decomposition divides the simulation volume into a finite number of subvolumes, which are distributed to the available processing units. Usually, the number of subvolumes and the number of processing units are equal. This method scales linearly with the number of molecules and is therefore much better suited for large systems than other methods like force or atom decomposition.[34,46,77]

For heterogeneous scenarios, it is not straightforward that all processes carry a similar share of the total workload. In simulation scenarios containing coexisting liquid and vapor phases, the local density within the simulation volume can differ significantly, e.g. by a factor of 1 000. The number of pairwise force calculations scales quadratically with the density. Therefore, the computational costs for two subvolumes of equal size may differ

by a factor of a million, resulting in many idle processes unless an efficient load balancing scheme is employed. Depending on the simulation scenario, it may be sufficient to apply a static load balancing scheme which is adjusted only once, or to rebalance the decomposition dynamically, e.g. every 100 to 1 000 time steps.

Like in other parts of *ls1 mardyn*, an interface class is available for the domain decomposition, allowing for the generic implementation of different decomposition approaches and therefore facilitating the implementation of load-balancing strategies based on domain decomposition. Several strategies were implemented in *ls1 mardyn* and evaluated for nucleation processes.[73] The strategy based on trees turned out to be the most efficient one. It is available in the current version of *ls1 mardyn* as described in the remainder of the present section.

4.2 Load costs

The purpose of load balancing is to decompose and distribute the simulation volume such that all processes need the same computing time. Such a decomposition requires a method to guess or measure the load that corresponds to a specific subvolume. The linked-cell algorithm, which is used to identify neighbor molecules, introduces a division of the simulation volume into cells. These cells are the basic volume units for which the load is determined. On the basis of the computational cost for each of the cells, a load balancing algorithm can group cells together such that p subvolumes of equal computational cost are created, where p is the number of processing units. In 2, a 2D example is given for a simulation volume divided into 8×8 cells. This volume is being partitioned along cell boundaries into two subvolumes which will then be assigned to different processes. The implementation in *ls1 mardyn* requires each subvolume to cover at least two cells in each spatial dimension.

In a typical simulation, the largest part of the computational cost is caused by the force and distance calculations. If N_i and N_j denote the number of molecules in cells i and j,

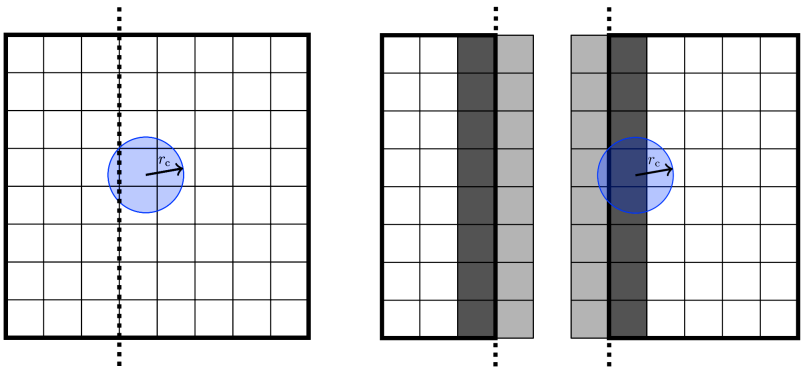


Figure 2: Left: The simulation volume (within the bold line) is divided into cells by the linked-cell algorithm (thin lines) where the cell edge length is the cut-off radius r_c. The simulation volume is divided into two subvolumes along cell boundaries (dotted line). Right: Halo cells (light shaded cells) are introduced storing copied molecule data from adjacent boundary cells (dark shaded cells).

respectively, the number of distance calculations $n_d(i)$ for cell i can be estimated by

$$n_d(i) \approx \frac{N_i}{2} \left(N_i + \sum_{j \in \text{neigh}(i)} N_j \right) . \tag{6}$$

The first term in Eq. (6), i.e. $N_i^2/2$, corresponds to the distance calculations within cell i. The second term represents the calculation of distances between molecules in cell i and an adjacent cell j.

While Eq. (6) can be evaluated with little effort, it is far more demanding to predict the number of force calculations. Furthermore, communication and computation costs at the boundary between adjacent subdomains allocated to different process can be significant. They depend on many factors, in particular on the molecule density at the boundary. Therefore, even if the load on all compute nodes is uniform and remains constant, the location of the subvolume boundaries has an influence on the overall performance. For a discussion of detailed models for the respective computational costs, the reader is referred to Buchholz.[73] In the present version of *ls1 mardyn*, the computational costs are estimated on the basis of the number of necessary distance calculations per cell according to Eq. (6).

4.3 Tree-based decomposition

The distribution of cells to processes is in principle straightforward. One way is to bring the cells into a linear order (e.g. row-wise), walk through the ordered list and sum up the load. Having reached $1/p$ of the total load, the cells may be grouped together to a subvolume and assigned to a process, ensuring that all processes carry a similar load. The problem with this naive approach is that it creates subvolumes with large surface to volume ratios. A homogeneous system with a cubic volume containing $100 \times 100 \times 100$ cells, distributed to 100 processes, would for instance be decomposed to 100 subvolumes with the thickness of a single cell so that all cells would be boundary cells. In such a case, the additional costs for boundary handling and communication are prohibitively high.

To overcome this problem, a hierarchical decomposition scheme was implemented in *ls1 mardyn*. This decomposition is similar to k-d trees,[78] which are known to achieve a good performance in general simulation tasks[79] as well as in the special case of particle simulations.[80,81] The simulation volume is recursively bisected into subvolumes with similar load by planes which are perpendicular to alternating coordinate axes.[82] To determine the optimal devision plane, the load distribution for every possible division plane is computed and the one resulting in the minimal load imbalance is selected. This procedure is recursively repeated until a subvolume is assigned to each process.

In case of extremely large simulation volumes, however, initial decompositions are determined following a simplified procedure, until a sufficiently small subvolume size is reached. Thereby, the volume is decomposed into equally sized subvolumes, and the number of processes per subvolume is assigned according to the estimated load for the respective subvolume.

5 Performance

Targeting large-scale parallel runs, *ls1 mardyn* has been designed for both good single-core and parallel efficiency. While the code was written in a portable way, which allows to

build and execute the program on every standard Linux or Unix system, we focus here on the HPC systems given in 2 for the performance analysis. In the following sections, we especially explain the influence of the compiler used to build *ls1 mardyn* on its performance, the overhead of the parallelization as well as its scalability.

Table 2: HPC platforms used for performance measurements.

System, location	Processor type	Interconnect	Cores
hermit, Stuttgart	AMD Opteron 6276 (Interlagos, 16 cores @2.3 GHz)	Cray Gemini	113 664
laki (NH), Stuttgart	Intel Xeon X5560 (Gainestown, 4 cores @2.8 GHz)	InfiniBand	5 600
laki (SB), Stuttgart	Intel Xeon E5-2670 (Sandy Bridge, 8 cores @2.6 GHz)	InfiniBand	3 072
SuperMUC, Garching	Intel Xeon E5-2680 (Sandy Bridge, 8 cores @2.7 GHz)	InfiniBand	147 456

5.1 Sequential performance

The compiler used to build the code has a large impact on its performance. 3 shows results obtained with a serial version of *ls1 mardyn* employing different compilers on the SB and NH partitions of *laki* as well as on *hermit*. The test scenarios were a LJ vapor (at $kT/\varepsilon = 0.7$ and $\rho\sigma^3 = 0.044$) consisting of 40 000 molecules and ethylene oxide in a liquid state (at $T = 285$ K and $\rho = 19.4$ mol/l) with 65 536 molecules. As can be seen, the sequential program runs fastest on the Sandy Bridge based *laki* system and built with the GNU compiler. Unless noted otherwise, the GNU compiler was also used for all further studies discussed below.

The computational complexity of the linked-cell algorithm and domain decomposition scheme used in *ls1 mardyn* is $\mathcal{O}(N)$. To evaluate the efficiency of the implementation, runs with different numbers of molecules were performed. The results in 4 show that in the present case, the implementation scales almost perfectly with $\mathcal{O}(N)$, as the execution time per molecule is approximately constant.

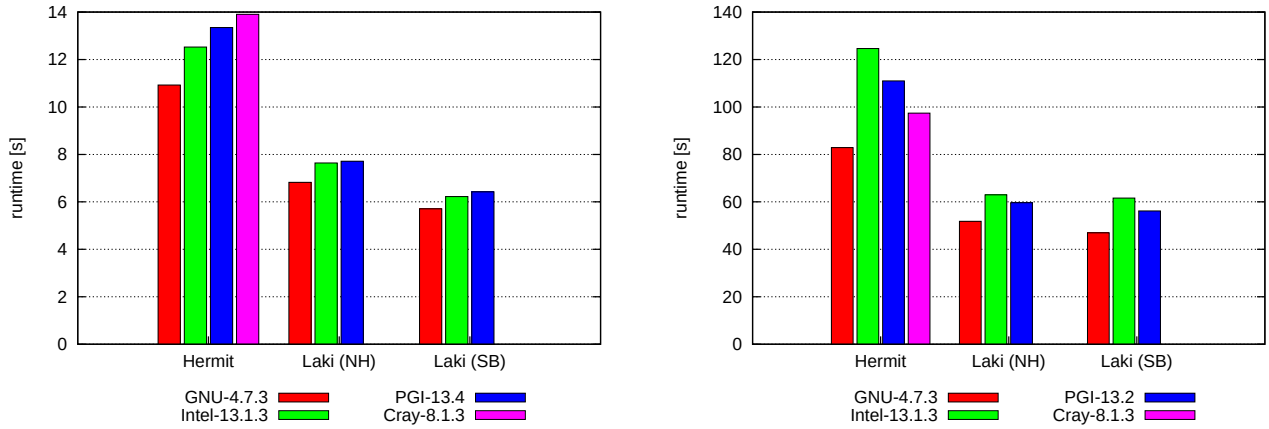


Figure 3: Sequential execution times of *ls1 mardyn* on various platforms with different compilers. Scenarios: LJ vapor with $N = 40\,000$, $\rho\sigma^3 = 0.044$, and $kT/\varepsilon = 0.7$ (left) as well as liquid ethylene oxide with $N = 65\,536$, $\rho = 19.4$ mol/l, and $T = 285$ K (right).

5.2 Sequential to parallel overhead

For the scalability evaluation of *ls1 mardyn*, different target scenarios with a varying degree of complexity were considered, produced by the internal scenario generators, cf. 5.

- **Homogeneous liquid:** Ethylene oxide at a density of $\rho = 16.9$ mol/l and a temperature of $T = 375$ K. The molecular model for ethylene oxide consists of three LJ sites and one point dipole.[69]

- **Droplet:** Simulation scenario containing a LJTS nanodroplet (cut-off radius $r_c = 2.5$ σ) surrounded by a supersaturated vapor at a reduced temperature of $kT/\varepsilon = 0.95$.

- **Planar interface:** Simulation of a planar vapor-liquid interface of the LJTS fluid (cut-off radius $r_c = 2.5$ σ) at a reduced temperature of $kT/\varepsilon = 0.95$.

In the scenarios, the number of molecules was varied. They were simulated on the platforms given in 2 for 1 000 time steps and with disabled final I/O.

Parallelization is associated with additional complexity due to communication and synchronization between the different execution paths of the program. In comparison with sequential execution on a single processing unit, this introduces an overhead. To determine the magnitude of this overhead for *ls1 mardyn*, the *planar interface* scenario with $N = 102\,400$

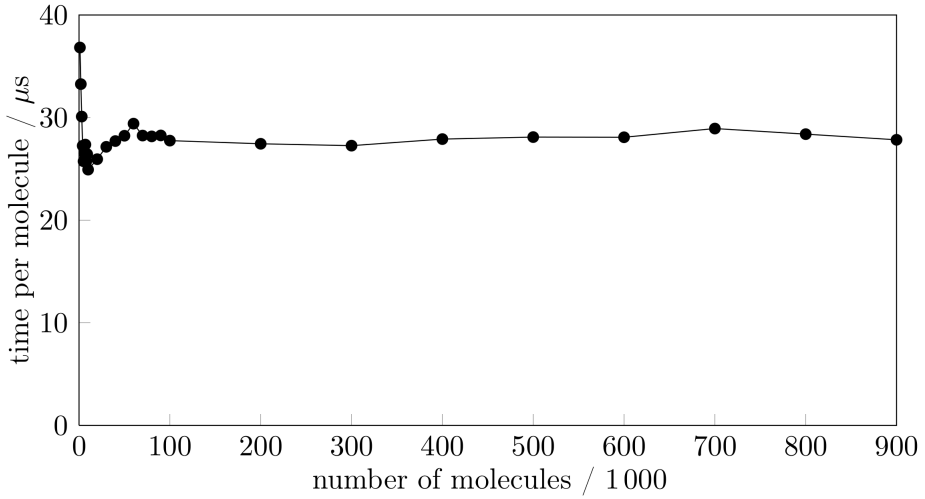


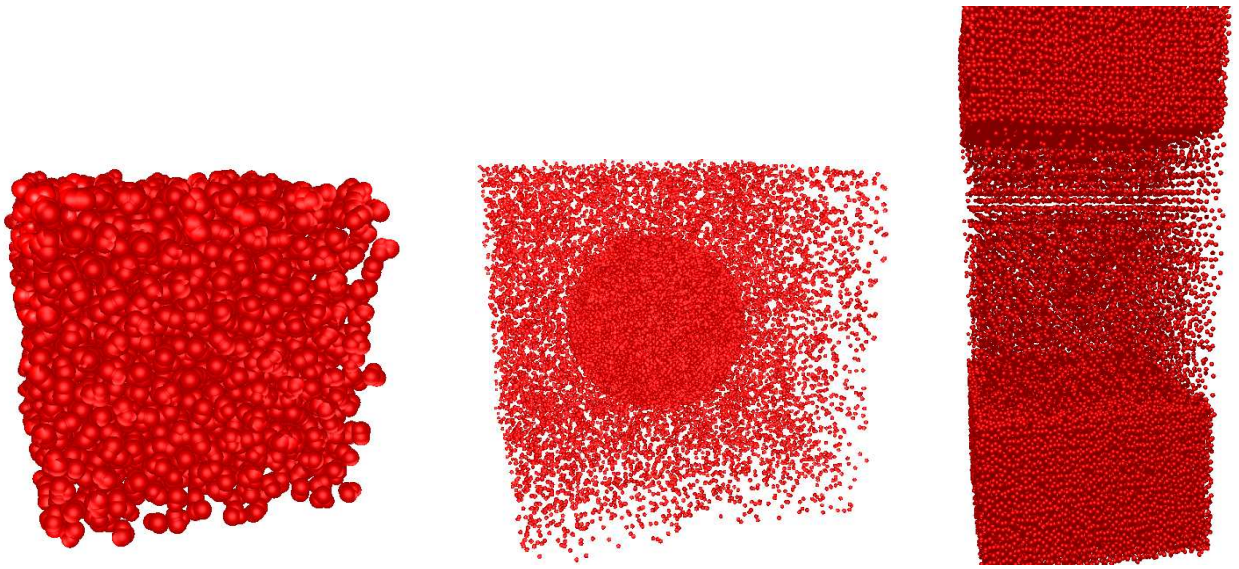
Figure 4: Sequential execution time of *ls1 mardyn* per molecule, for simulations of a homogeneous LJ fluid at $kT/\varepsilon = 0.95, \rho\sigma^3 = 0.6223$ with different system sizes on *laki* (SB).

LJ sites was executed over 1 000 time steps on the *hermit* system, both with the sequential and the MPI parallel version of the code, but using only a single process. Execution of the sequential program took 530.9 s, while the MPI parallel version took 543.4 s. This indicates that the overhead due to imperfect concurrency amounts to around 2% only.

5.3 Scalability

Scaling studies were carried out with the *homogeneous liquid* scenario on the entire *hermit* system, using the standard domain decomposition method, i.e. all processes were assigned equal volumes. The results presented in 6 show that *ls1 mardyn* scales favorably in the present case.

As discussed above, load balancing is of major importance for inhomogeneous molecule distributions. Strong scaling experiments were therefore carried out for the *planar interface* and *droplet* scenarios. The droplet was positioned slightly off the center of the simulation volume to avoid symmetry effects. The scenarios were run for 1 000 time steps, and the decomposition was updated every 100 time steps. The results are presented in 7 and show a clear advantage of the dynamic tree-based decomposition, making the simulation up to four times as fast as the static decomposition into subdomains with equal volume.



(a) *Homogeneous liquid* ($N = 2\,048$) (b) *Droplet* ($N = 46\,585$) (c) *Planar interface* ($N = 102\,400$)

Figure 5: Scenarios used during the performance evaluation of *ls1 mardyn*.

In addition to comparing the run times, the effectiveness of the dynamic load balancing implementation in *ls1 mardyn* is supported by traces revealing the load distribution between the processes. 8 shows such traces, generated with *vampirtrace*, for 15 processes of a *droplet* scenario simulation on the *hermit* system. For the trivial domain decomposition, 12 out of 15 processes are waiting in MPI routines most of the time, while the remaining three processes have to carry the bulk of the actual computation. In contrast, the k-d decomposition exhibits a more balanced distribution of computation and communication.

5.4 Trillion particle simulation

A version of *ls1 mardyn* was optimized for simulating single-site LJ particles on the *SuperMUC* system,[21] one of the largest x86 systems worldwide with 147 500 cores and a theoretical peak performance of more than 3 PFLOPS. It is based on a high-performance FDR-10 InfiniBand interconnect by Mellanox and composed of 18 so-called islands, each of which consists of 512 nodes with 16 Intel Sandy Bridge EP cores at 2.7 GHz clock speed (turbo mode disabled) sharing 32 GB of main memory.

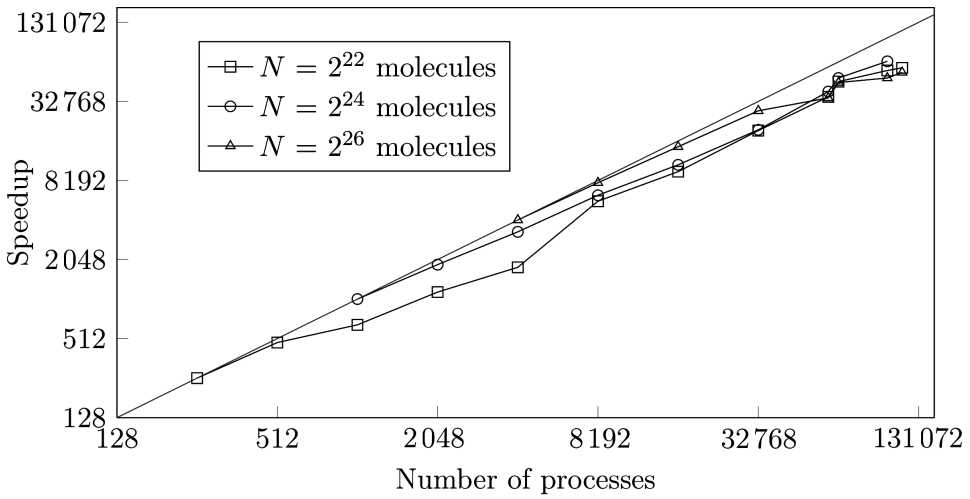


Figure 6: Scaling of *ls1 mardyn* on *hermit* with the *fluid* example. The starting points of the plots are placed on the diagonal, i.e. normalized to a parallel efficiency of 100 %, neglecting the deviation from perfect scaling for the respective reference case with the smallest number of processes.

Main features of the optimized code version include a lightweight shared-memory parallelization and hand-coded intrinsics in single precision for the LJ interactions within the kernel. The kernels were implemented in AVX128 (rather than AVX256), mainly for two reasons: First, the architecture of the Intel Sandy Bridge processor is unbalanced with respect to load and store bandwidth, which may result in equal performance for both variants. Second, AVX128 code usually shows better performance on the AMD Bulldozer architecture, where two processor cores share one 256-bit floating-point unit.

To evaluate the performance with respect to strong scaling behavior, a scenario with $N = 9.5 \times 10^8$ particles was studied, which fits into the memory of two nodes, as 18 GB per node are needed. Thereby, a cut-off radius of $r_c = 5\ \sigma$ was employed. 9 shows that a very good scaling was achieved for up to 32 768 cores using 65 536 threads. Built with the Intel compiler, the implementation delivered a sustained performance of 113 GFLOPS, corresponding to 8 % single-precision peak performance at a parallel efficiency of 53% compared to 32 cores (64 threads). In addition, a weak scaling analysis with $N = 1.6 \times 10^7$ particles per node was performed, where a peak performance of 12.9% or 183 TFLOPS was achieved at a parallel efficiency of 96 % when scaling from 1 to 32 768 cores.

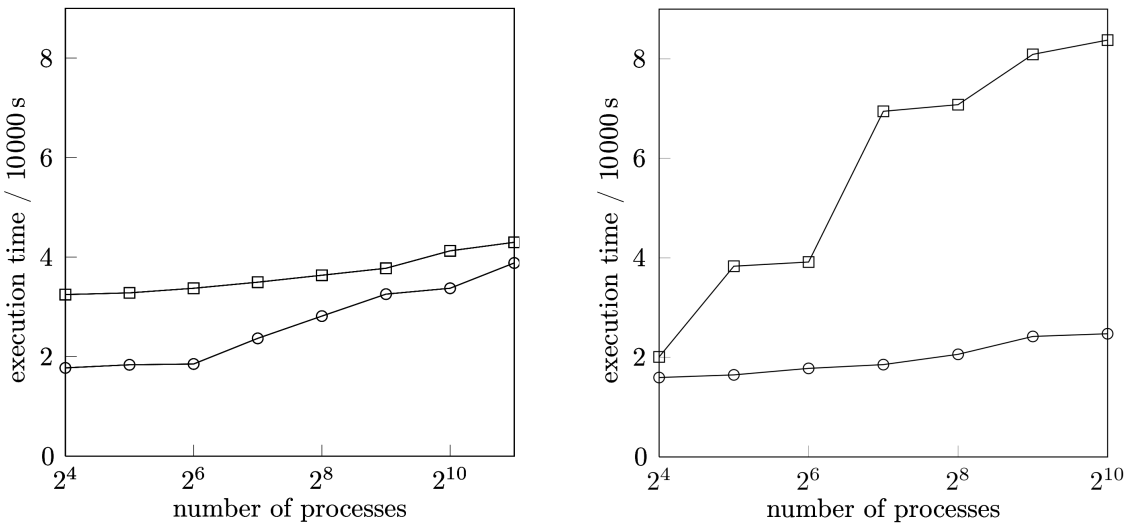
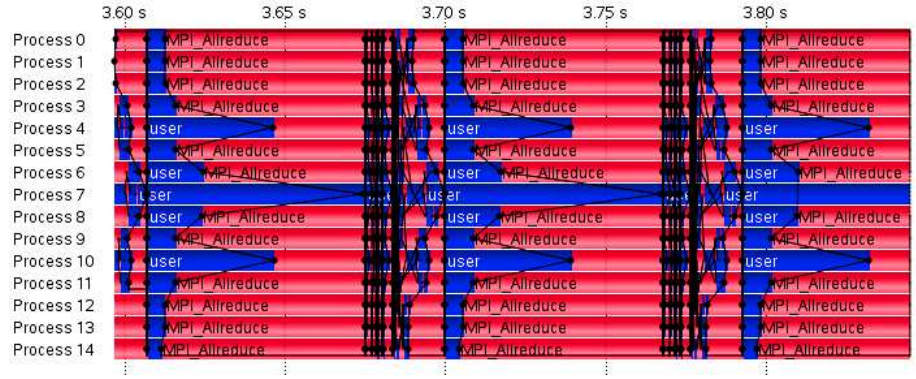


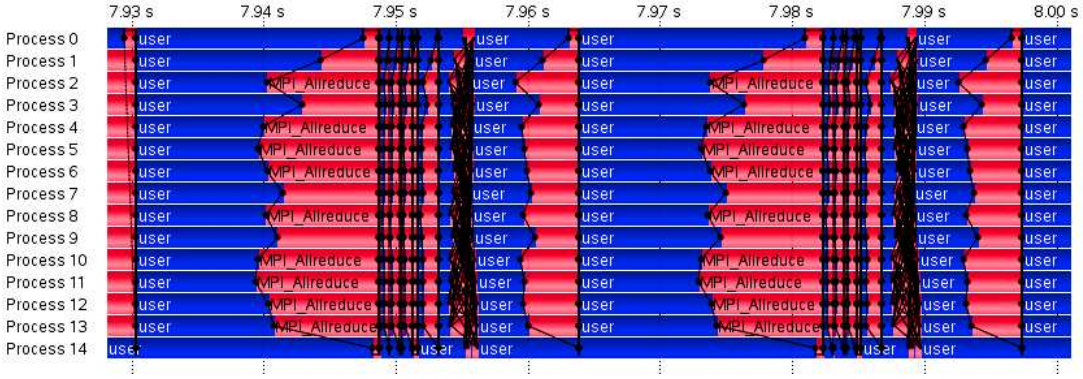
Figure 7: Accumulated execution time of *ls1 mardyn* for a strong scaling experiment on *hermit* using the *planar interface* scenario with $N = 5\,497\,000$ (left) and the *droplet* scenario with $N = 3\,698\,000$ (right). A straightforward static domain decomposition (□), which assigns subdomains with equal volumes to all processing units, is compared with the dynamic k-d tree based decomposition (○).

As the kernel was implemented using AVX128, the same scenario was executed on the Cray XE6 system *hermit* at HLRS, however, without shared-memory parallelization and built with the GNU compiler. A noteworthy feature of the Cray XE6 machine is its 3D torus network with Gemini interconnect, which directly plugs in to the HyperTransport 3 host interface for fast MPI communication. On *hermit*, the code achieved a parallel efficiency of 82.5% and 69.7 GFLOPS in case of strong scaling and 91.5 % and 76.8 TFLOPS or 12.8% peak performance for weak scaling, respectively, on 32 768 cores in comparison to 64 cores, i.e. two nodes.

As can be seen in 9, the scalability on *hermit* is superior, particularly for strong scaling. The Gemini interconnect allows for higher bandwidth and lower latency for MPI communications than the FDR-10 InfiniBand interconnect of *SuperMUC*. Furthermore, a 3D torus network is more favorable for the communication pattern of *ls1 mardyn* than the tree topology of *SuperMUC*, where the nodes belonging to each island (8 192 cores) communicate via a fully connected network, while for inter-island communication four nodes have to share a single uplink. This can also be seen in 9, where the scalability noticeably drops when going



(a) Trivial domain decomposition



(b) Tree-based domain decomposition

Figure 8: Traces for the *droplet* scenario on *hermit*, generated with *vampirtrace*. The program state over two time steps is shown for 15 parallel processes. Computation is indicated by blue colour, communication by red colour. Vertical lines indicate message passing between processes.

from 8 192 to 16 384 processes.

As described by Eckhardt et al.,[21] a larger weak scaling benchmark on the whole *SuperMUC* was performed with that code version. Simulating 4.125×10^{12} molecules, to our knowledge the largest MD simulation to date, with a cut-off radius of $r_c = 3.5\ \sigma$, one time step took roughly 40 s. For this scenario, a speedup of 133 183 compared to a single core with an absolute performance of 591.2 TFLOPS was achieved, which corresponds to 9.4 % peak performance efficiency.

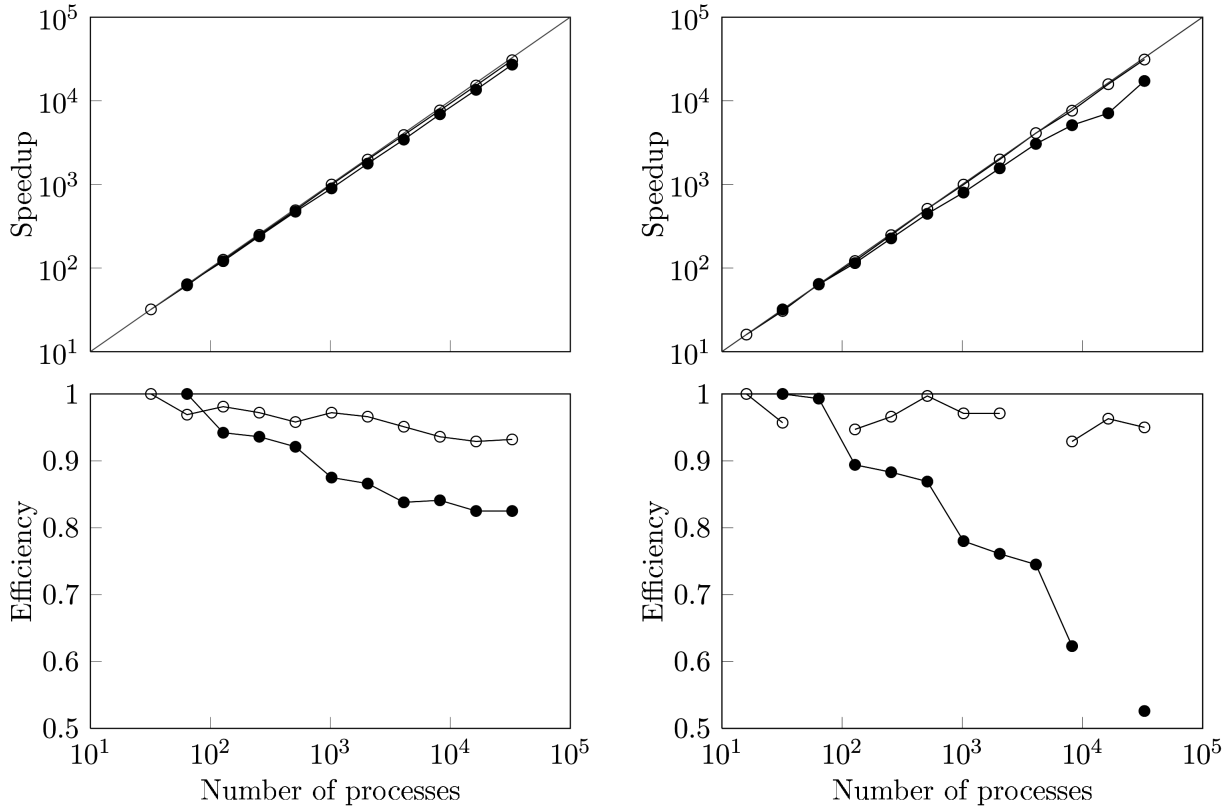


Figure 9: Weak scaling (○) and strong scaling (●) of *ls1 mardyn* on *hermit* (left) and *SuperMUC* (right), including the speedup (top) and the parallel efficiency (bottom), i.e. the speedup reduced by the number of processes. Almost ideal scaling was achieved in case of weak scaling, whereas a parallel efficiency of 53 % was obtained in the strong scaling tests on *SuperMUC* and 82.5 % on *hermit*, compared to two nodes.

6 Conclusions

The massively parallel MD simulation code *ls1 mardyn* was introduced and presented. The *ls1 mardyn* program is designed to simulate homogeneous and heterogeneous fluid systems containing very large numbers of molecules. Fluid molecules are modeled as rigid rotators consisting of multiple interaction sites, enabling simulations of a wide variety of scenarios from noble gases to complex fluid systems under confinement. The code, which presently holds the world record for the largest MD simulation, was evaluated on large-scale HPC architectures. It was found to scale almost perfectly on over 140 000 cores for homogeneous scenarios. The dynamic load balancing capability of *ls1 mardyn* was tested with different scenarios, delivering a significantly improved scalability for challenging, highly heterogeneous

systems.

It can be concluded that *ls1 mardyn*, which is made publicly available as free software,[1] represents the state of the art in MD simulation. It can be recommended for large-scale applications, and particularly for processes at fluid interfaces, where highly heterogeneous and time-dependent particle distributions may occur. Due to the modularity of its code base, future work can adjust *ls1 mardyn* to newly emerging HPC architectures and further extend the range of available molecular modeling approaches and simulation methods. In this way, *ls1 mardyn* aims at driving the progress of molecular simulation in general, paving the way to the micrometer length scale and the microsecond time scale for computational molecular engineering.

Acknowledgement

The authors would like to thank A. Bode and M. Brehm for their help in accessing the supercomputing infrastructure at the Leibniz Supercomputing Center (LRZ) of the Bavarian Academy of Sciences and Humanities. They thank D. Mader for his contribution to developing the very first version of the *ls1 mardyn* program, S. Grottel, M. Heinen, D. Jenz and G. Reina for their work on libraries and tools, as well as C. Avendaño Jiménez, S. Eckelsbach, K. Langenbach, R. Lustig, S. K. Miroshnichenko, E. A. Müller, G. Rutkai, F. Siperstein, R. Srivastava and N. Tchipev for fruitful discussions. The present work was conducted under the auspices of the Boltzmann-Zuse Society for Computational Molecular Engineering (BZS), and the molecular simulations were carried out within the supercomputing project *pr83ri* on the *SuperMUC* at the LRZ, Garching, and within MMHBF2 on *hermit* and *laki* at the HLRS, Stuttgart. Financial support is acknowledged due to the IMEMO and SkaSim grants of the German Federal Ministry of Education and Research (BMBF), and the Reinhart Koselleck Program as well as the Collaborative Research Center MICOS (SFB 926) of the German Research Foundation (DFG).

References

(1) *Large systems 1: molecular dynamics*; http://www.ls1-mardyn.de/, accessed August 19, 2014.

(2) Allen, M. P.; Tildesley, D. J. *Computer Simulation of Liquids*; Clarendon: Oxford, 1987.

(3) Frenkel, D.; Smit, B. *Understanding Molecular Simulation*, 2nd ed.; Academic Press: San Diego, 2002.

(4) Deublein, S.; Eckl, B.; Stoll, J.; Lishchuk, S. V.; Guevara Carrión, G.; Glass, C. W.; Merker, T.; Bernreuther, M.; Hasse, H.; Vrabec, J. *Comput. Phys. Comm.* **2011**, *182*, 2350–2367.

(5) Möller, D.; Fischer, J. *Mol. Phys.* **1990**, *69*, 463–473.

(6) Vrabec, J.; Hasse, H. *Mol. Phys.* **2002**, *100*, 3375–3383.

(7) Rusanov, A. I.; Brodskaya, E. N. *J. Colloid Interf. Sci.* **1977**, *62*, 542–555.

(8) Rao, M.; Berne, B. J.; Kalos, M. H. *J. Chem. Phys.* **1978**, *68*, 1325–1336.

(9) Angélil, R.; Diemand, J.; Tanaka, K. K.; Tanaka, H. *J. Chem. Phys.* **2014**, *140*, 074303.

(10) Chialvo, A. A.; Debenedetti, P. G. *Phys. Rev. A* **1991**, *43*, 4289–4295.

(11) Sokołowski, S.; Fischer, J. *Phys. Rev. A* **1990**, *41*, 6866–6870.

(12) Horsch, M.; Heitzig, M.; Dan, C.; Harting, J.; Hasse, H.; Vrabec, J. *Langmuir* **2010**, *26*, 10913–10917.

(13) Rösch, F.; Trebin, H.-R. *Eur. Phys. Lett.* **2009**, *87*, 66004.

(14) Thompson, P. A.; Troian, S. M. *Nature* **1997**, *389*, 360–362.

(15) Frentrup, H.; Avendaño, C.; Horsch, M.; Salih, A.; Müller, E. A. *Mol. Sim.* **2012**, *38*, 540–553.

(16) Müller-Plathe, F. *ChemPhysChem* **2002**, *3*, 754–769.

(17) Lee, E. H.; Hsin, J.; Sotomayor, M.; Comellas, G.; Schulten, K. *Structure* **2009**, *17*, 1295–1306.

(18) Lindorff-Larsen, K.; Piana, S.; Dror, R. O.; Shaw, D. E. *Science* **2011**, *334*, 517–520.

(19) Engel, M.; Trebin, H.-R. *Phys. Rev. Lett.* **2007**, *98*, 225505.

(20) Laio, A.; Parrinello, M. *Proc. Nat. Acad. Sci.* **2002**, *99*, 12562–12566.

(21) Eckhardt, W.; Heinecke, A.; Bader, R.; Brehm, M.; Hammer, N.; Huber, H.; Kleinhenz, H.-G.; Vrabec, J.; Hasse, H.; Horsch, M.; Bernreuther, M.; Glass, C. W.; Niethammer, C.; Bode, A.; Bungartz, J. In *Supercomputing – Proceedings of the XXVIII. International Supercomputing Conference (ISC)*; Kunkel, J. M., Ludwig, T., Meuer, H. W., Eds.; Lecture Notes in Computer Science 7905; Springer: Heidelberg, 2013; pp 1–12.

(22) Guevara Carrión, G.; Vrabec, J.; Hasse, H. *J. Chem. Phys.* **2011**, *134*, 074508.

(23) Horsch, M.; Vrabec, J.; Bernreuther, M.; Grottel, S.; Reina, G.; Wix, A.; Schaber, K.; Hasse, H. *J. Chem. Phys.* **2008**, *128*, 164510.

(24) Horsch, M.; Vrabec, J. *J. Chem. Phys.* **2009**, *131*, 184104.

(25) Müller, E. A. *Curr. Opin. Chem. Eng.* **2013**, *2*, 223–228.

(26) Case, D. A.; Cheatham, I., T. E.; Darden, T.; Gohlke, H.; Luo, R.; Merz, j., K. M.; Onufriev, A.; Simmerling, C.; Wang, B.; Woods, R. *J. Comput. Chem.* **2005**, *26*, 1668–1688.

(27) Salomon Ferrer, R.; Götz, A. W.; Poole, D.; Le Grand, S.; Walker, R. C. *J. Chem. Theory Comput.* **2013**, *9*, 3878–3888.

(28) Berendsen, H. J. C.; van der Spoel, D.; van Drunen, R. *Comput. Phys. Comm.* **1995**, *91*, 43–56.

(29) Pronk, S.; Szilárd, P.; Schulz, R.; Larsson, P.; Bjelkmar, P.; Apostolov, R.; Shirts, M. R.; Smith, J. C.; Kasson, P. M.; van der Spoel, D.; Hess, B.; Lindahl, E. *Bioinformatics* **2013**, *29*, 845–854.

(30) Phillips, J. C.; Braun, R.; Wang, W.; Gumbart, J.; Tajkhorshid, E.; Villa, E.; Chipot, C.; Skeel, R. D.; Kalé, L.; Schulten, K. *J. Comput. Chem* **2005**, *26*, 1781–1802.

(31) Ren, P.; Wu, C.; Ponder, J. W. *J. Chem. Theory Comput.* **2011**, *7*, 3143–3461.

(32) Brooks, B. R.; Bruccoleri, R. E.; Olafson, B. D.; States, D. J.; Swaminathan, S.; Karplus, M. *J. Comput. Chem.* **1983**, *4*, 187–217.

(33) Brooks, B. R.; Brooks, I., C. L.; Mackerell, A. D.; Nilsson, L.; Petrella, R. J.; Roux, B.; Won, Y.; Archontis, G.; Bartels, C.; Boresch, S.; Caflisch, A.; Caves, L.; Cui, Q.; Dinner, A. R.; Feig, M.; Fischer, S.; Gao, J.; Hodoscek, M.; Im, W.; Kuczera, K.; Lazaridis, T.; Ma, J.; Ovchinnikov, V.; Paci, E.; Pastor, R. W.; Post, C. B.; Pu, J. Z.; Schaefer, M.; Tidor, B.; Venable, R. M.; Woodcock, H. L.; Wu, X.; Yang, W.; York, D. M.; Karplus, M. *J. Comput. Chem.* **2009**, *30*, 1545–1615.

(34) Plimpton, S. *J. Comput. Phys.* **1995**, *117*, 1–19.

(35) Brown, W. M.; Wang, P.; Plimpton, S. J.; Tharrington, A. N. *Comput. Phys. Comm.* **2011**, *182*, 898–911.

(36) Plimpton, S. J.; Thompson, A. P. *MRS Bulletin* **2012**, *37*, 513–521.

(37) Diemand, J.; Angélil, R.; Tanaka, K. K.; Tanaka, H. *J. Chem. Phys.* **2013**, *139*, 074309.

(38) Todorov, I. T.; Smith, W.; Trachenko, K.; Dove, M. T. *J. Materials Chem.* **2006**, *16*, 1911–1918.

(39) Limbach, H.-J.; Arnold, A.; Mann, B. A.; Holm, C. *Comput. Phys. Comm.* **2006**, *174*, 704–727.

(40) Stadler, J.; Mikulla, R.; Trebin, H.-R. *Int. J. Mod. Phys. C* **1997**, *8*, 1131–1140.

(41) Roth, J.; Gähler, F.; Trebin, H.-R. *Int. J. Mod. Phys. C* **2000**, *11*, 317–322.

(42) Saager, B.; Fischer, J.; Neumann, M. *Mol. Sim.* **1991**, *6*, 27–49.

(43) Quentrec, R.; Brot, C. *J. Comput. Phys.* **1973**, *13*, 430–432.

(44) Hockney, R. W.; Eastwood, J. W. *Computer Simulation Using Particles*; McGraw-Hill: New York, 1981.

(45) Schamberger, S.; Wierum, J.-M. In *Proceedings of the VII. International Conference on Parallel Computing Technologies (PaCT)*; Malyshkin, V., Ed.; Lecture Notes in Computer Science 2763; Springer: Heidelberg, 2003; pp 165–179.

(46) Bernreuther, M.; Vrabec, J. In *High Performance Computing on Vector Systems*; Resch, M., Bönisch, T., Benkert, K., Bez, W., Furui, T., Seo, Y., Eds.; Springer: Heidelberg, 2006; pp 187–195.

(47) Bernreuther, M.; Buchholz, M.; Bungartz, H.-J. In *Parallel Computing: Architectures, Algorithms and Applications – Proceedings of the XII. International Conference on Parallel Computing (ParCo)*; Joubert, G., Bischof, C., Peters, F., Lippert, T., Bücker, M., Gibbon, P., Mohr, B., Eds.; Advances in Parallel Computing 15; IOS: Amsterdam, 2008; pp 53–60.

(48) Horsch, M.; Vrabec, J.; Hasse, H. *Phys. Rev. E* **2008**, *78*, 011603.

(49) Vrabec, J.; Horsch, M.; Hasse, H. *J. Heat Transfer (ASME)* **2009**, *131*, 043202.

(50) Horsch, M.; Lin, Z.; Windmann, T.; Hasse, H.; Vrabec, J. *Atmospher. Res.* **2011**, *101*, 519–526.

(51) Grottel, S.; Reina, G.; Vrabec, J.; Ertl, T. *IEEE Transact. Vis. Comp. Graph.* **2007**, *13*, 1624–1631.

(52) Horsch, M.; Miroshnichenko, S.; Vrabec, J. *J. Physical Studies (L'viv)* **2009**, *13*, 4004.

(53) Horsch, M.; Hasse, H.; Shchekin, A. K.; Agarwal, A.; Eckelsbach, S.; Vrabec, J.; Müller, E. A.; Jackson, G. *Phys. Rev. E* **2012**, *85*, 031605.

(54) Werth, S.; Lishchuk, S. V.; Horsch, M.; Hasse, H. *Physica A* **2013**, *392*, 2359–2367.

(55) Horsch, M.; Hasse, H. *Chem. Eng. Sci.* **2014**, *107*, 235–244.

(56) Werth, S.; Rutkai, G.; Vrabec, J.; Horsch, M.; Hasse, H. *Mol. Phys.* **2014**, in press (DOI: 10.1080/00268976.2013.861086).

(57) Horsch, M.; Vrabec, J.; Bernreuther, M.; Hasse, H. In *Proceedings of the 6th International Symposium on Turbulence, Heat and Mass Transfer*; Hanjalić, K., Ed.; Begell House: New York, 2009; pp 89–92.

(58) Horsch, M.; Niethammer, C.; Vrabec, J.; Hasse, H. *Informat. Technol.* **2013**, *55*, 97–101.

(59) Grottel, S.; Reina, G.; Ertl, T. In *Proceedings of the IEEE Pacific Visualization Symposium*; Eades, P., Ertl, T., Shen, H.-W., Eds.; IEEE Computer Society, 2009; pp 65–72.

(60) Grottel, S.; Reina, G.; Dachsbacher, C.; Ertl, T. *Comp. Graph. Forum* **2010**, *29*, 953–962.

(61) Lorentz, H. A. *Ann. Phys. (Leipzig)* **1881**, *12*, 127–136, 660–661.

(62) Schnabel, T.; Vrabec, J.; Hasse, H. *J. Mol. Liq.* **2007**, *135*, 170–178.

(63) Berthelot, D. *Compt. Rend. Acad. Sci.* **1898**, *126*, 1703–1706, 1857–1858.

(64) Vrabec, J.; Huang, Y.-L.; Hasse, H. *Fluid Phase Equilib.* **2009**, *279*, 120–135.

(65) Stoll, J.; Vrabec, J.; Hasse, H. *AIChE J.* **2003**, *49*, 2187–2198.

(66) Vrabec, J.; Stoll, J.; Hasse, H. *Mol. Sim.* **2005**, *31*, 215–221.

(67) Stone, A. J. *Science* **2008**, *321*, 787–789.

(68) Gray, C. G.; Gubbins, K. E. *Theory of Molecular Fluids*; Oxford University Press, 1984; Vol. 1: Fundamentals.

(69) Eckl, B.; Vrabec, J.; Hasse, H. *Fluid Phase Equilib.* **2008**, *274*, 16–26.

(70) Tersoff, J. *Phys. Rev. Lett.* **1988**, *61*, 2879–2882.

(71) Tersoff, J. *Phys. Rev. B* **1989**, *39*, 5566–5568.

(72) Ghiringhelli, L. M.; Valeriani, C.; Los, J. H.; Meijer, E. J.; Fasolino, A.; Frenkel, D. *Mol. Phys.* **2008**, *106*, 2011–2038.

(73) Buchholz, M. Framework zur Parallelisierung von Molekulardynamiksimulationen in verfahrenstechnischen Anwendungen. Dissertation, Technische Universität München, 2010.

(74) Störmer, C. *Radium (Paris)* **1912**, *9*, 395–399.

(75) Verlet, L. *Phys. Rev.* **1967**, *159*, 98–103.

(76) Hockney, R. W. *Methods Comput. Phys.* **1970**, *9*, 136–211.

(77) Plimpton, S.; Hendrickson, B. In *Parallel Computing in Computational Chemistry*; Mattson, T. G., Ed.; ACS: Washington, D.C., 1995; pp 114–132.

(78) Bentley, J. L. *Comm. ACM* **1975**, *18*, 509–517.

(79) Simon, H. D.; Teng, S.-H. *SIAM J. Sci. Comput.* **1995**, *18*, 1436–1445.

(80) Bernard, P.-E.; Gautier, T.; Trystram, D. *Parallel Processing – Proceedings of the XIII. International Conference on Parallel and Distributed Processing (IPPS/SPDP)*; IEEE: Washington, D.C., 1999; pp 638–644.

(81) Fleissner, F.; Eberhard, P. In *Parallel Computing: Architectures, Algorithms and Applications – Proceedings of the XII. International Conference on Parallel Computing (ParCo)*; Joubert, G., Bischof, C., Peters, F., Lippert, T., Bücker, M., Gibbon, P., Mohr, B., Eds.; Advances in Parallel Computing 15; IOS: Amsterdam, 2008; pp 37–44.

(82) Berger, M. J.; Bokhari, S. H. *IEEE Transact. Comput.* **1987**, *C-36*, 570–580.