RESEARCH ARTICLE | OCTOBER 09 2020

MDBenchmark: A toolkit to optimize the performance of molecular dynamics simulations

Special Collection: Classical Molecular Dynamics (MD) Simulations: Codes, Algorithms, Force fields, and Applications , Chemical Physics Software Collection

Michael Gecht ; Marc Siggel ; Max Linke ; Gerhard Hummer ; Jürgen Köfinger

J. Chem. Phys. 153, 144105 (2020)
https://doi.org/10.1063/5.0019045

The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

MDBenchmark: A toolkit to optimize the performance of molecular dynamics simulations

Cite as: J. Chem. Phys. 153, 144105 (2020); doi: 10.1063/5.0019045
Submitted: 22 June 2020 • Accepted: 4 September 2020 •
Published Online: 9 October 2020

Michael Gecht,[1,a] Marc Siggel,[1] Max Linke,[1] Gerhard Hummer,[1,2] and Jürgen Köfinger[1,b]

AFFILIATIONS

[1] Department of Theoretical Biophysics, Max Planck Institute of Biophysics, Max-von-Laue-Straße 3, 60438 Frankfurt am Main, Germany
[2] Institute for Biophysics, Goethe University Frankfurt, Max-von-Laue-Straße 9, 60438 Frankfurt am Main, Germany

Note: This paper is part of the JCP Special Topic on Classical Molecular Dynamics (MD) Simulations: Codes, Algorithms, Force Fields, and Applications.
[a] **Electronic mail:** michael.gecht@biophys.mpg.de
[b] **Author to whom correspondence should be addressed:** juergen.koefinger@biophys.mpg.de

ABSTRACT

Despite the impending flattening of Moore's law, the system size, complexity, and length of molecular dynamics (MD) simulations keep on increasing, thanks to effective code parallelization and optimization combined with algorithmic developments. Going forward, exascale computing poses new challenges to the efficient execution and management of MD simulations. The diversity and rapid developments of hardware architectures, software environments, and MD engines make it necessary that users can easily run benchmarks to optimally set up simulations, both with respect to time-to-solution and overall efficiency. To this end, we have developed the software MDBenchmark to streamline the setup, submission, and analysis of simulation benchmarks and scaling studies. The software design is open and as such not restricted to any specific MD engine or job queuing system. To illustrate the necessity and benefits of running benchmarks and the capabilities of MDBenchmark, we measure the performance of a diverse set of 23 MD simulation systems using GROMACS 2018. We compare the scaling of simulations with the number of nodes for central processing unit (CPU)-only and mixed CPU–graphics processing unit (GPU) nodes and study the performance that can be achieved when running multiple simulations on a single node. In all these cases, we optimize the numbers of message passing interface (MPI) ranks and open multi-processing (OpenMP) threads, which is crucial to maximizing performance. Our results demonstrate the importance of benchmarking for finding the optimal system and hardware specific simulation parameters. Running MD simulations with optimized settings leads to a significant performance increase that reduces the monetary, energetic, and environmental costs of MD simulations.

I. INTRODUCTION

Molecular dynamics (MD) simulations have become an integral part of the molecular life sciences and material sciences. Their predictive power has been continuously increasing, thanks to methodological advances and the exponential growth of compute power. The latter is captured by Moore's law for the number of transistors of an integrated circuit and, across different technologies, by the law of accelerated returns.[1,2] This continuing growth translates into a similar growth of the temporal and spatial scales that can be assessed in MD simulations.[3] Thus, MD simulations are becoming more powerful in extending and connecting the different scales that are accessible to experimental methods.[4] As so-called computational microscopes,[5,6] MD simulations are widely used to make predictions and to analyze, design, and validate experiments.

The tools to perform MD simulations have reached a level of sophistication, which allows non-expert users to set up, run, and analyze simulations. Various software packages for

performing MD simulations, so-called MD engines, have been developed, e.g., ACEMD,[7] Amber,[8] CHARMM,[9] Desmond,[10] GROMACS,[11] HOOMD-blue,[12] LAMMPS,[13] NAMD,[14,15] and OpenMM.[16] These engines, each with a unique set of features, were designed to efficiently compute the time evolution of particles and take advantage of different hardware architectures and parallel computation to varying degrees.

Optimal settings for MD simulations aim to decrease the time-to-solution (TTS) and to increase the throughput of simulations and energy efficiency. To run an MD simulation on a compute cluster, the user has to specify numerous parameters to control the behavior of the underlying hardware and software. Moreover, optimal parameters might vary between different versions of the same MD engine and depend on the details of the molecular system under consideration. Users new to MD simulations might miss the details that are required to run efficient simulations on the high-performance compute clusters. A poor choice of parameters can notably degrade the simulation performance, usually measured in simulated time per day, e.g., ns/day, or increase it only by a small margin while wasting resources. Such inefficient simulations decrease the overall simulation throughput, lead to a higher electricity demand and operating cost, and ultimately increase the carbon footprint. It is therefore in the interest of every user to optimize their usage of hardware resources, and, at the same time, keep their TTS to a minimum. This daunting task of finding the optimal parameters for running MD simulations efficiently can only be tackled by thorough benchmarking.

Therefore, we need to enable individual users to run benchmarks conveniently and efficiently for their given settings. This complements the goals of systematic benchmark studies performed by experts.[17–20] For example, Kutzner *et al.* performed extensive benchmarks using GROMACS to determine the best performance-to-price ratio for a variety of MD systems and numerous hardware architectures.[19,20] They provided valuable guidelines for configuring and purchasing new clusters and for choosing optimal parameters. However, the rapid development of hardware, algorithms, and software and the variety of MD engines and simulation systems requires that users themselves are able to run benchmark studies efficiently.

We developed MDBenchmark, a standalone software package, implemented as a command-line interface (CLI), to conveniently set up, run, and analyze benchmarks of MD simulations. With this tool, users can run benchmarks and scaling studies for their specific molecular system, MD engine, and compute cluster. MDBenchmark was developed to streamline and simplify the process of finding the optimal run parameters and settings for any simulation and hardware stack. It takes care of submitting simulations to the queuing system, performs scaling studies by varying the number of nodes, automatically toggles the usage of central processing unit (CPUs) and/or graphics processing unit (GPUs), and scans the numbers of processes used for parallelization [message passing interface (MPI) ranks, open multi-processing (OpenMP) threads] if applicable. The package was designed for ease of use not only by expert users but also by researchers without prior detailed knowledge of the ins and outs of high-performance computing (HPC).

To illustrate the application and the capabilities of MDBenchmark and to highlight the value and necessity of running benchmarks, we report on an extensive scaling study of 23 MD simulations of varying sizes (~4×10^4 to ~4×10^6 atoms) and system compositions. We identify numbers of MPI ranks and OpenMP threads that produce the best performance for a range of system sizes, study the benefits of hyperthreading, and analyze when it is beneficial to use CPU-only or mixed CPU–GPU nodes and when to run multiple simulations on a single node. For this study, we use the GROMACS software suite as it is widely used, freely available, and highly optimized for different kinds of hardware. However, MDBenchmark has been designed such that different MD engines and job queuing systems can easily be added.

II. BACKGROUND

Current compute clusters are composed of compute nodes, each containing at least one CPU, an optional GPU, as well as gigabytes of dedicated random-access memory (RAM). These nodes are connected in a network such that data can be exchanged between nodes and calculations can be performed in parallel on multiple nodes.

Modern CPUs contain dozens of physical cores, where each core can perform computations independent from the others. In addition, a single physical core can often perform two computations at the same time, a feature called "hyperthreading."[21] When enabled, the number of physical cores is virtually doubled, i.e., for each physical core, two "logical cores" are introduced.

To use these heterogeneous resources efficiently and run calculations in parallel, two interfaces are widely used: *message passing interface* (MPI) and *open multi-processing* (OpenMP). MPI spawns processes, which we will refer to as ranks. A single MPI rank can comprise all cores of a single node or only a subset of them. By contrast, OpenMP creates computational threads, where each is composed only of a subset of cores available inside a MPI rank. OpenMP threads share the same memory.

Running a simulation on a computer cluster requires the user to submit a compute job to a queuing system. The user must configure a submission script that launches the MD engine over its CLI. Users have to define the correct parameters for the specific queuing system. Activating hyperthreading on the "Sun Grid Engine" (SGE) differs, for example, from activating it on the "Simple Linux Utility for Resource Management" (SLURM) queuing system.

The numbers of MPI ranks to OpenMP threads influence the performance of GROMACS simulations.[19,20] To take advantage of the parallel compute infrastructure, a simulation box is first divided into separate domains in a process called domain decomposition.[22] Each domain is regarded in an isolated manner, and information at the borders is communicated with the other domains. In a hybrid MPI-OpenMP approach, the calculations of a single domain are managed by a single MPI rank. This rank spawns multiple OpenMP threads, which then perform the actual calculations in each domain. Each MPI rank communicates with the ranks responsible for its neighboring domains. The number of ranks per node n_{ranks} times the number of threads per node n_{threads} gives the number of logical cores per node. If hyperthreading is deactivated, then the number of logical cores is equal to the number of physical cores. If activated, it is equal to twice the number of physical cores.

III. THE MDBENCHMARK SOFTWARE

The CLI of MDBenchmark provides access to four main functions (Fig. 1). In the first step, all parameters for the benchmark(s) are defined using the GENERATE command [Fig. 1(a)]. Here, the user chooses the MD engine, the numbers of nodes to perform scaling on, whether to use GPUs, and the numbers of MPI ranks. In addition, a run input file of an equilibrated MD simulation must be provided. A TPR file is sufficient for GROMACS. Different MD engines require different input files. For example, the NAMD, PSF, and PDB files have to be provided for NAMD. MDBenchmark automatically checks the availability of the requested MD engine and its installed version using the "Environment Modules" system.[23] This feature was put in place to safeguard against typos in the module name. If a module environment is not used on the compute cluster, the user can skip this availability check. MDBenchmark will prompt the user to confirm the action, before proceeding to create the folder structure.

The folder structure was intentionally designed with a nested hierarchy to allow users to access files themselves, if needed. Each requested MD engine is put into its own folder, with a subfolder denoting the engine's version and whether MDBenchmark is going to request CPU-only or mixed CPU–GPU nodes. The last subfolder layer separates the benchmarks by the numbers of nodes, MPI ranks, and OpenMP threads and by whether hyperthreading is enabled or disabled. Each of these subfolders then contains a copy of the run input file, as provided by the user, the job submission script containing all parameters and commands to run the benchmark, as well as a hidden folder holding all metadata in the JSON format. This metadata is managed through the datreant Python package.[24] We use it to define each benchmark as an entity and add our parameters as metadata. The package can be used to search and filter benchmarks through their Python API. This way, benchmarks can easily be grouped by distinct parameters for additional customized analysis by the user.

After benchmarks have been set up, they can be submitted to the queuing system with the SUBMIT command in the second step [Fig. 1(b)]. When called, it will traverse all subfolders and gather information on each benchmark. The user will be shown all benchmarks that are to be submitted, and they will be prompted to submit or cancel. If a benchmark was already submitted, it will be excluded from further submissions. The user can ask MDBenchmark to submit all benchmarks, ignoring their submission status.

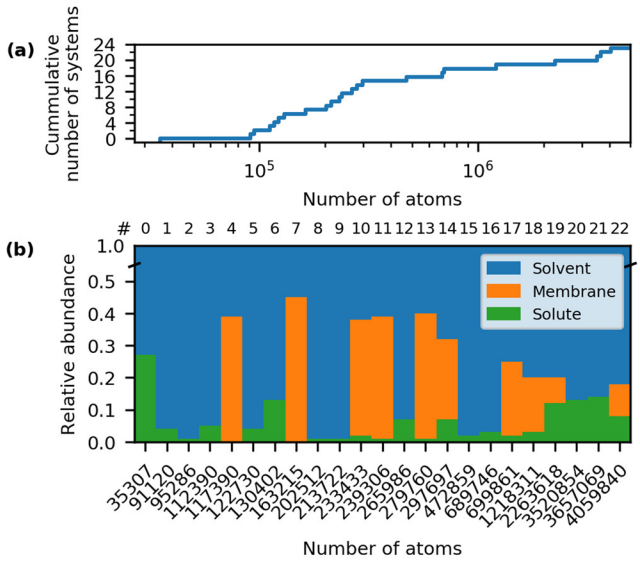

FIG. 2. Size and composition of simulation systems. (a) Cumulative number of systems as a function of the atom number. Most MD systems (65%) contain less than 300 000 atoms. (b) Relative abundance of system constituents as a function of the number of atoms. Systems are composed of 55%–99% solvent (blue). Lipid bilayers, if present, make up 8%–45% (orange) of the system's total number of atoms. All other solutes account for at most 27%, but usually less than 10% (green). Note that the numbers of atoms comprise all interaction sites, i.e., also the additional interaction sites for TIP4P-D water.

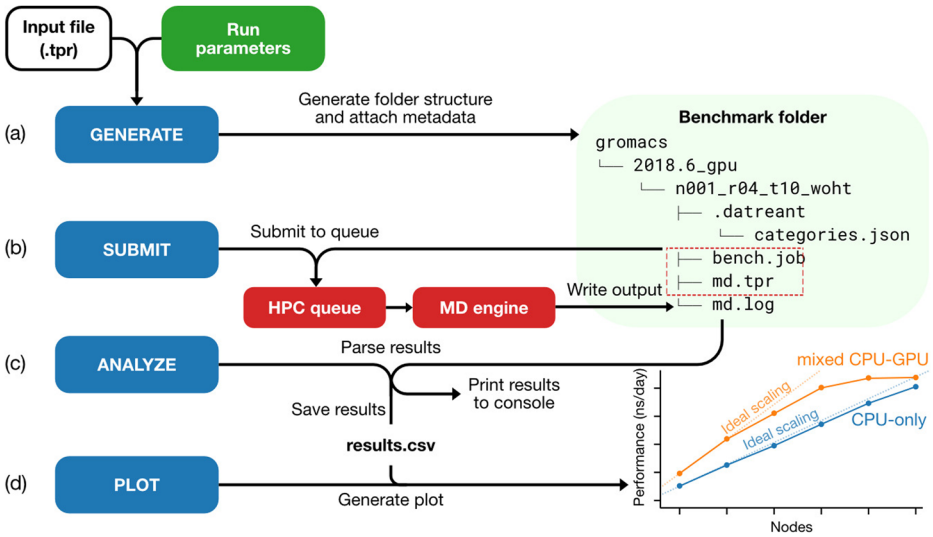

FIG. 1. Schematic representation of the implementation and workflow of MDBenchmark. (a) Run input files and benchmark parameters are supplied to the GENERATE command. For every benchmark, a folder with attached metadata is created. (b) The SUBMIT command sends the benchmark job to the HPC queue, starting the MD engine on the requested resources. (c) The log file, written by the engine, contains all necessary results and is parsed through the ANALYZE command. Results are shown in the console or saved to a CSV file. (d) The PLOT command can visualize the results in a plot using the CSV file as input. It produces a scaling plot with the performance (ns/day) as a function of the number of nodes, with every benchmark setting as a separate line.

Currently, MDBenchmark can submit jobs to the queuing systems SGE, SLURM, and IBM's LoadLeveler. The available queuing system will be automatically determined before submission. The submission files are conveniently implemented using the template syntax provided by the Jinja2 Python package, which facilitates the easy adaption to other HPC resources and different requirements. Further details are available in the MDBenchmark documentation.

After submission to the queuing system, the status of benchmarks can be checked with the ANALYZE command in the third step [Fig. 1(c)]. MDBenchmark will print all available information on each benchmark in a tabular format to the console. Every call to the ANALYZE command will parse the log files produced by the MD engine and extract information on the corresponding run. If a benchmark has already finished, its performance will be printed in the last column of the table. All available benchmark data can be saved in a CSV format to a file for future analysis.

In a fourth step, the name of the file containing the benchmark results can be provided as an argument to the PLOT command, which will produce a scaling plot [Fig. 1(d)]. The plot shows the performance (ns/day) as a function of the number of nodes, with each line representing a different combination of parameters (CPU-only/mixed CPU–GPU nodes, numbers of MPI ranks and OpenMP threads, and hyperthreading). A linear fit through the origin and the first data point is added to indicate ideal scaling. The PLOT command also accepts multiple CSV files from separate benchmarks to allow for straightforward comparison of different benchmark parameters. In addition, the CSV file can be easily read with the pandas Python package[25] for customized analysis.

Further detailed explanations of all available options can be found online in the MDBenchmark documentation (https://mdbenchmark.readthedocs.io/).

IV. METHODS

We used MDBenchmark to run benchmarks of atomistic MD simulations for 23 biomolecular systems. The system sizes range from ~4×10^4 to ~4×10^6 atoms [Fig. 2(a)]. 15 out of the 23 tested systems (65%) have less than 300 000 atoms. Due to the nature of biomolecular simulations, the 23 simulation systems studied here are mostly composed of solvent: 55%–99% of the system's total atoms belong to water molecules [Fig. 2(b)]. If present, lipid bilayers account for 8%–45% of the total number of atoms. Other solutes, i.e., proteins, nucleic acids, small molecules, and ions, make up only 0.1%–27%. These systems cover a broad range of force fields, water models, compositions, box geometries, and sizes (Table I). Their

TABLE I. Details of the fully atomistic simulation systems, for which we perform scaling benchmarks. The systems vary in size, composition, and box geometry, as well as in force fields and water models: CHARMM36,[36] CHARMM36m,[37] AMBER99SB*-ILDN,[38–40] AMBER99SB*-ILDN-Q,[38–42] TIP3P,[43] and TIP4P-D.[44] Box geometries are cuboids (C), rhombic dodecahedra (RD), and hexagonal prisms (H).

	Type	No. of atoms	Box geometry	System size (nm^3)	Force field	Water model	References
0	Protein in solution	35 307	RD	8.0 × 8.0 × 5.7	AMBER99SB*-ILDN	TIP3P	Unpublished
1	Protein in solution	91 120	C	9.9 × 9.9 × 9.9	CHARMM36	TIP3P	Unpublished
2	Protein in solution	95 286	C	10.0 × 10.0 × 10.0	CHARMM36	TIP3P	Unpublished
3	Protein in solution	112 390	C	10.0 × 10.0 × 10.0	AMBER99SB*-ILDN-Q	TIP3P	Unpublished
4	Protein in membrane	117 390	C	11.0 × 11.0 × 10.0	CHARMM36	TIP3P	Hofbauer *et al.*[45]
5[a]	Protein in solution	122 730	C	10.0 × 10.0 × 10.0	AMBER99SB*-ILDN-Q	TIP4P-D	Unpublished
6[a]	Dense protein solution	130 402	C	9.9 × 9.9 × 9.9	AMBER99SB*-ILDN-Q	TIP4P-D	Bülow *et al.*[46]
7	Empty membrane	163 215	C	13.1 × 13.1 × 9.1	CHARMM36m	TIP3P	Unpublished
8[a]	Protein in solution	202 512	RD	13.0 × 13.0 × 9.2	AMBER99SB*-ILDN-Q	TIP4P-D	Unpublished
9	dsDNA in solution	213 722	RD	14.5 × 14.5 × 10.3	AMBER99SB*-ILDN	TIP3P	Unpublished
10	Protein and membrane	233 433	C	14.6 × 14.6 × 10.5	CHARMM36m	TIP3P	Wu *et al.*[47]
11	Protein and membrane	239 306	C	14.6 × 14.6 × 10.9	CHARMM36	TIP3P	Unpublished
12[a]	Dense protein solution	265 986	C	12.6 × 12.6 × 12.6	AMBER99SB*-ILDN-Q	TIP4P-D	Bülow *et al.*[46]
13	Protein and membrane	279 760	C	15.9 × 15.9 × 10.8	CHARMM36	TIP3P	Unpublished
14	Protein in membrane	297 697	C	13.8 × 13.8 × 15.4	CHARMM36m	TIP3P	Hofmann *et al.*[48]
15[a]	Protein in solution	472 859	C	15.0 × 15.0 × 15.0	AMBER99SB*-ILDN-Q	TIP4P-D	Unpublished
16[a]	Dense protein solution	689 746	C	17.3 × 17.3 × 17.3	AMBER99SB*-ILDN-Q	TIP4P-D	Bülow *et al.*[46]
17	Protein in membrane	699 861	C	19.5 × 19.5 × 18.1	CHARMM36m	TIP3P	Unpublished
18	Protein and membrane	1 219 446	C	23.0 × 23.0 × 23.0	CHARMM36m	TIP3P	Unpublished
19	Protein and membrane	2 263 618	C	38.2 × 21.6 × 31.5	CHARMM36	TIP3P	Unpublished
20[a]	Dense protein solution	3 520 854	C	30.0 × 30.0 × 30.0	AMBER99SB*-ILDN-Q	TIP4P-D	Bülow *et al.*[46]
21[a]	Dense protein solution	3 657 069	C	30.4 × 30.4 × 30.4	AMBER99SB*-ILDN-Q	TIP4P-D	Bülow *et al.*[46]
22[b]	Protein and membrane	4 059 840	H	34.9 × 34.9 × 37.9	CHARMM36m	TIP3P	Turoňová *et al.*[26]

[a]Systems use the TIP4P-D water model.
[b]The timestep of this system was set to 4 fs.

exact setup can be found in the corresponding references, if already published. All systems were set up with a 1.2 nm cutoff for nonbonded interactions in the real space and a grid spacing of 0.12 nm for PME, with the exception of systems No. 6, No. 12, No. 16, No. 20, and No. 21 where a real space cutoff of 1.0 nm was used with a grid spacing of 0.16 nm. We used dynamic load balancing in all benchmarks to automatically tune the cutoff for nonbonded interactions and PME grid spacing. The listed values thus serve as lower and upper bounds, respectively. Systems No. 0–No. 21 used an integration time step of 2 fs and system No. 22 used 4 fs by doubling the hydrogen mass.[26] Note that the time step in fully atomistic simulations can also be increased by using virtual sites.[27,28]

For these 23 systems, we performed scaling studies in which we determine the performance $P(N)$ as a function of the number of nodes N. We vary the number of MPI ranks, n_{ranks}, which also determine the number of OpenMP threads per rank, n_{threads}, for activated and deactivated hyperthreading. On CPU-only nodes, GROMACS was allowed to dedicate about 25% of the available MPI ranks for the PME calculation. We did not specify individual PME ranks for mixed CPU–GPU benchmarks.

We use Amdahl's law as a simple model to summarize the results of our scaling studies.[29] This law describes the speed-up $S(N)$ of parallelized computations as a function of the number of nodes N, i.e.,

$$S(N) = \frac{1}{1 - p + \frac{p}{N}}, \tag{1}$$

where p is the fraction of the code that benefits from parallelization.

From Amdahl's law we express the performance $P(N) = S(N)P(1)$ as

$$P(N) = \frac{P^{\max}}{1 + \frac{p}{N(1-p)}}, \tag{2}$$

where the maximum performance $P^{\max} = \lim_{N\to\infty} P(N)$ is given by

$$P^{\max} = \frac{P(1)}{1-p}. \tag{3}$$

The ideal scaling is determined by the performance for $N = 1$ node as

$$P^{\text{id}}(N) = NP(1), \tag{4}$$

which becomes

$$P^{\text{id}}(N) = N(1-p)P^{\max}. \tag{5}$$

We use Amdahl's law to estimate the performance that can be achieved by increasing the number of nodes, while being reasonably efficient, i.e., close to ideal scaling as given by Eq. (4). For parallelization to be efficient, we demand that the performance is a fraction f of the ideal performance, i.e.,

$$P(N) = fP^{\text{id}}(N), \tag{6}$$

and solve for N. We obtain

$$N(f) = \frac{1 - fp}{f(1-p)}. \tag{7}$$

The performance corresponding to a fraction f of the ideal scaling is then given by $P(N(f))$ as

$$P(f) = (1 - fp)P^{\max}. \tag{8}$$

We performed benchmarks with GROMACS 2018[11] without the built-in thread-MPI library but with external MPI libraries and enabled OpenMP support. AVX_512 SIMD instructions were enabled at compile time with GCC 8.3 and CUDA 10.1. Note that AVX2_256 SIMD instructions might be beneficial in some cases, for example, for simulations with a small number of atoms per core. We used nodes with two Intel® Xeon® Gold 6148 CPUs (2.40 GHz) for all benchmarks, with additionally two NVIDIA Quadro RTX 5000 graphic cards in mixed CPU–GPU nodes. All nodes were connected with a 100 Gb/s OmniPath interconnect. Benchmarks were run on specific numbers of MPI ranks and OpenMP threads for a total wall time of 15 min, if not mentioned otherwise. We used SLURM as the queuing system for all benchmarks on the available supercomputer. Example submission files for both CPU-only and mixed CPU–GPU nodes can be found in the Appendix.

All data generated in this study was managed and analyzed using datreant,[24] MDAnalysis,[30,31] NumPy,[32] Pandas,[25] SciPy,[33] and IPython.[34] Plots were generated with Matplotlib.[35]

V. RESULTS

Using MDBenchmark, we first examine the dependence of the performance estimates on the run time of the benchmarks. We then use two exemplary systems to discuss in detail the performance scaling and its dependence on the number of MPI ranks and on hyperthreading. We present results for the 22 largest systems and show how optimal parameters vary with the system size. Finally, we investigate the performance when running multiple simulations on a single node for all 23 systems.

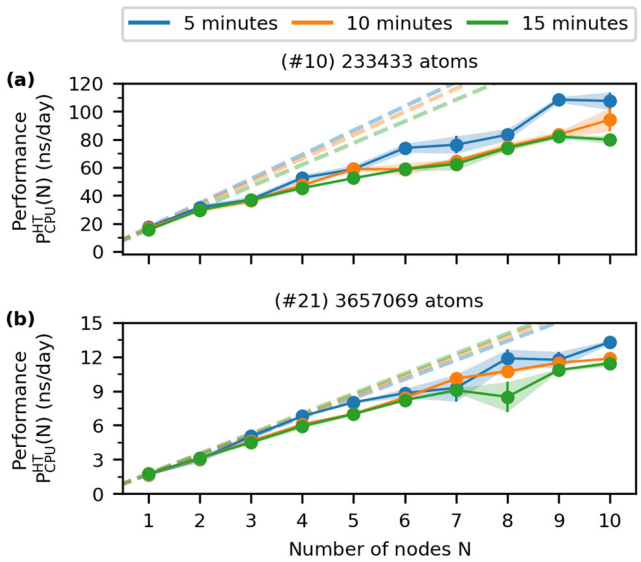

FIG. 3. Dependence of the performance estimates on the run times of the benchmarks for CPU-only nodes. MD simulations with (a) 233k and (b) 3.6M atoms scaled from 1 to 10 nodes and run for 5 (blue), 10 (orange), and 15 (green) min. Each data point shows the average of five independent runs with error bars and filled-in area showing the standard deviation. Transparent dashed lines show ideal scaling according to Eq. (4).

A. Run time of benchmarks

Ideally, a benchmark run provides the most accurate performance estimate (ns/day) in the least amount of time. Short benchmarks facilitate a higher throughput, a more efficient use of the limited computing resources, and thus allow us to sample a broader range of parameters.

To set the length of a benchmark run, we can either use a fixed number of steps, as previously done by Kutzner *et al.*,[19,20] or set the run time explicitly. In a GROMACS simulation, the first hundreds to thousands of steps can be used to balance computational load between different ranks using dynamic load balancing. This auto-tuning is enabled by default and beneficial for overall simulation performance. However, a benchmark will be aborted if this process has not finished after half of the available compute time. For larger systems, the auto-tuning process takes longer. Thus, we decided to set the run time explicitly as jobs with fixed, short run times can be given higher priority by the queuing systems.

We determined the minimum run time of the benchmarks from scaling studies. For two systems using N = 1–10 nodes, we used run times of 5 min, 10 min, and 15 min (Fig. 3) using CPU-only nodes. The results for the shortest run times of 5 min can deviate significantly from the results for 10 min and 15 min. The shortest run time can be used for a first screening, while longer run times seem to be necessary to obtain accurate results. The benchmark results can be influenced, for example, by the amount of traffic handled by the network infrastructure at run time. In the following, all reported benchmarks were run for 15 min independent of the system size.

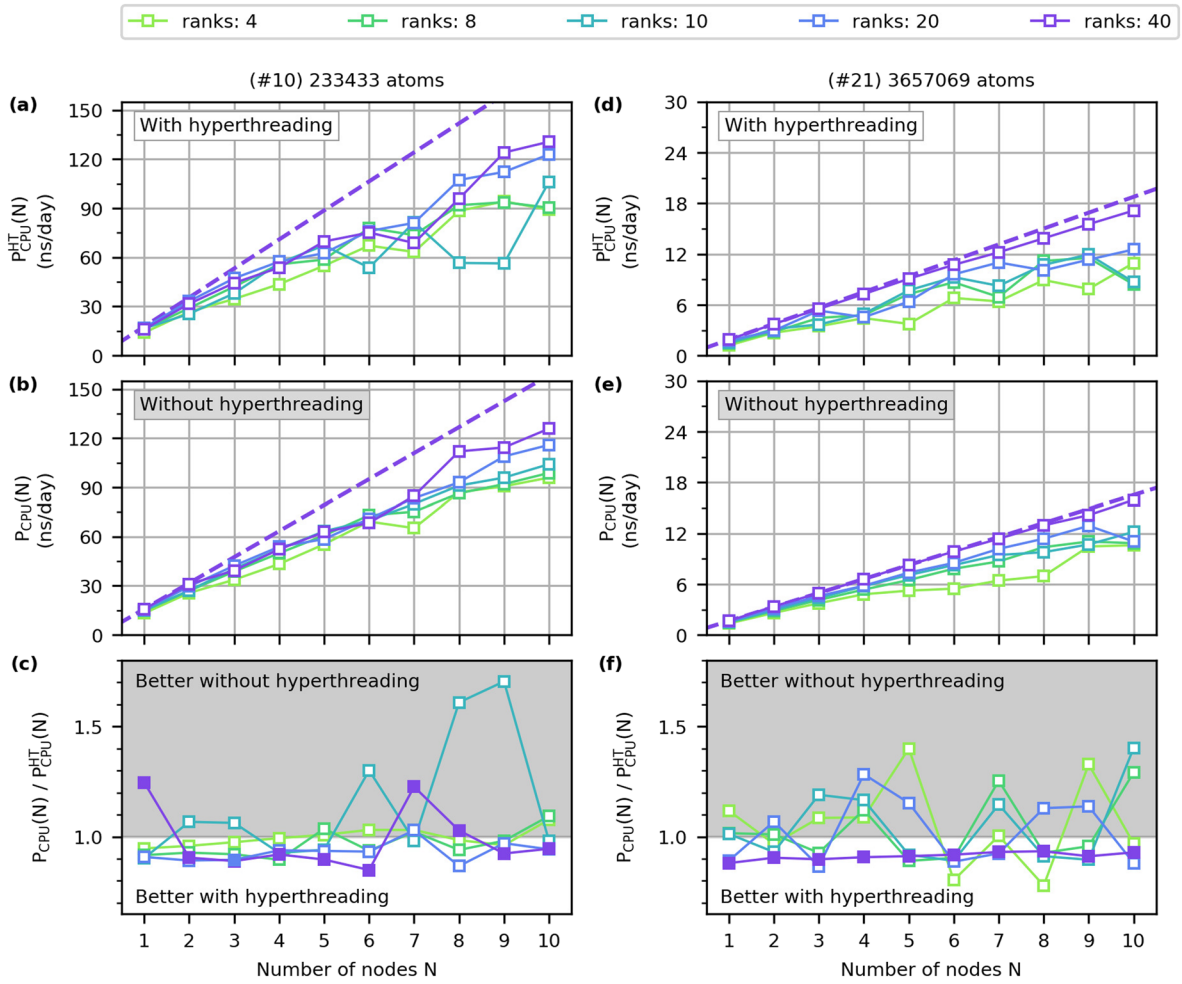

FIG. 4. Scaling of the performance $P_{CPU}(N)$ with the number N of CPU-only nodes for different numbers of MPI ranks n_{ranks} (colors) with [(a) and (d)] and without [(b) and (e)] hyperthreading. Ideal scaling was estimated according to Eq. (4) (dashed lines: strongest ideal scaling as thick dashed line). [(a) and (b)] Prototypical membrane protein system with 233k atoms and [(d) and (e)] dense protein solution with 3.6M atoms. Both systems scale the best with n_{ranks} = 40. [(c) and (f)] Ratios between the performance without and with hyperthreading. For the best performing rank settings (filled squares) of n_{ranks} = 40, both systems generally benefit from running simulations with hyperthreading.

B. Optimizing performance for two exemplary systems

We evaluated how the performance scales with different numbers of nodes using either CPU-only or mixed CPU–GPU nodes. Using MDBenchmark, we ran benchmarks using 1–10 nodes and scanned the values of the MPI ranks with and without hyperthreading. In the following, we present detailed results for two exemplary systems of different size and composition (systems No. 10 and No. 21 in Table I): system No. 10, a prototypical membrane protein system with 233k atoms,[47] and system No. 21, a large 3.6M atom system of a dense protein solution using TIP4P-D as the water model.[46]

Our results show that the dependence of the performance on the number of MPI ranks is different for CPU-only nodes and for mixed CPU–GPU nodes. For CPU-only nodes with hyperthreading activated, both systems show the best performance for $n_{\text{ranks}} = 40$, consistently for all node numbers [Figs. 4(a) and 4(d)]. Thus, for CPU-only nodes, the optimal number of ranks is independent of the system size. By contrast, for mixed CPU–GPU nodes with hyperthreading activated, the optimal choice of n_{ranks} depends on the system size [Figs. 5(a) and 5(d)]. For the default parameters set by our software environment and queuing system, $n_{\text{ranks}} = 40$ and hyperthreading activated, the medium sized system with 233k atoms shows the worst performance [Fig. 5(a)]. We find that the optimal numbers of ranks yielding the highest performance are given by $n_{\text{ranks}} = 8$ as well as by $n_{\text{ranks}} = 10$. This example illustrates that blindly trusting the default values set by the software environment can decrease performance by more than half. For the larger systems of 3.6M atoms, we observe that a higher number of ranks n_{ranks}

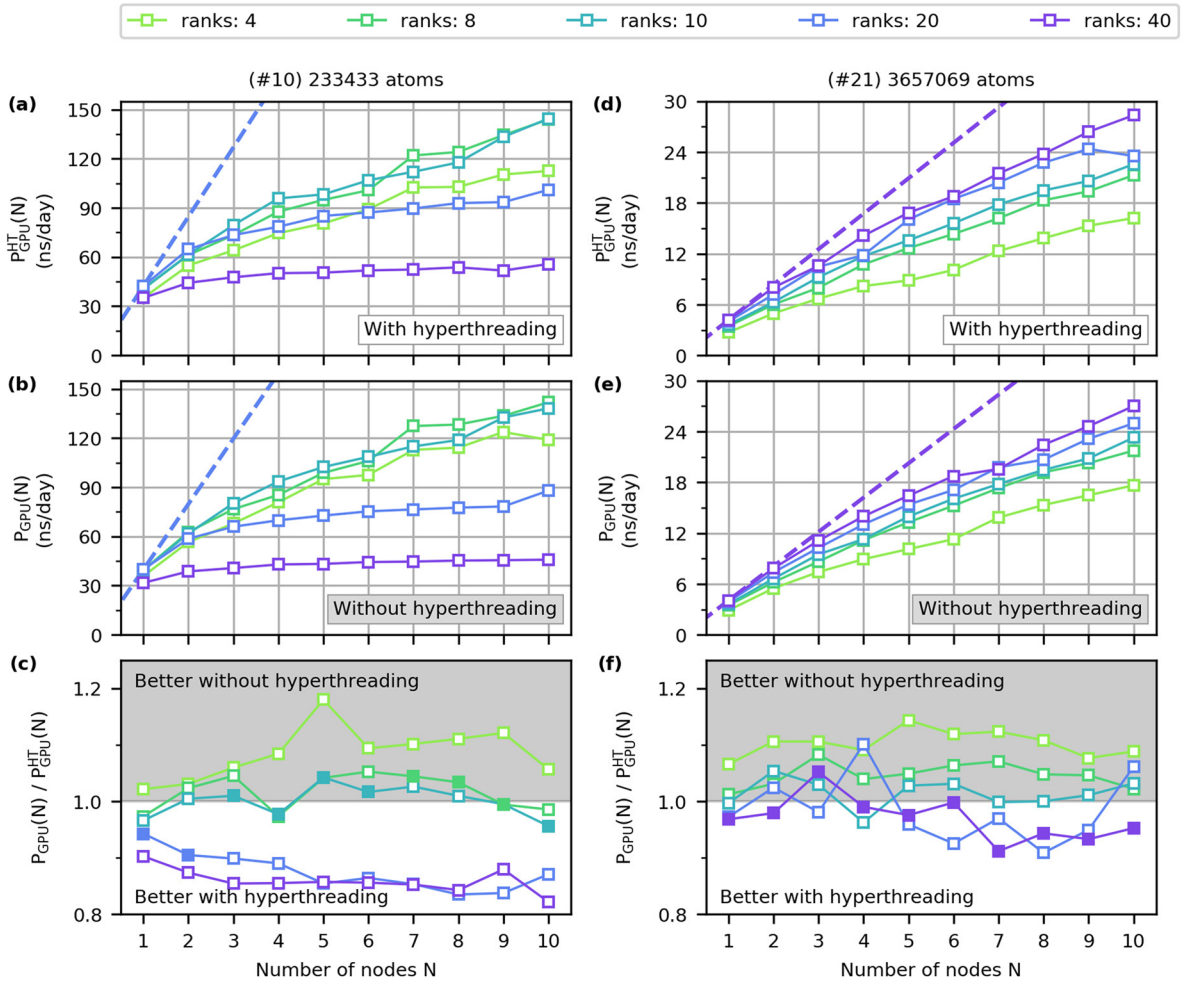

FIG. 5. Scaling of the performance $P_{GPU}(N)$ with the number N of mixed CPU–GPU nodes for different numbers of MPI ranks n_{ranks} (colors) with [(a) and (d)] and without [(b) and (e)] hyperthreading. Ideal scaling was estimated according to Eq. (4) (dashed lines: strongest ideal scaling as the thick dashed line). For the membrane protein system with 233k atoms, we obtain the best performance for $n_{\text{ranks}} = 8$, (a) with and (b) without hyperthreading. For the dense protein solution with 3.6M atoms, we obtain the best performance for $n_{\text{ranks}} = 40$, (c) with and (d) without hyperthreading. [(c) and (f)] Ratios between the performance without and with hyperthreading. (c) We find that for the 233k atom system, the performance for the optimal rank settings (filled symbols) does not benefit from hyperthreading. (f) For the 3.6M atom system, a significant increase in performance can be achieved by activating hyperthreading for the optimal rank settings (filled squares).

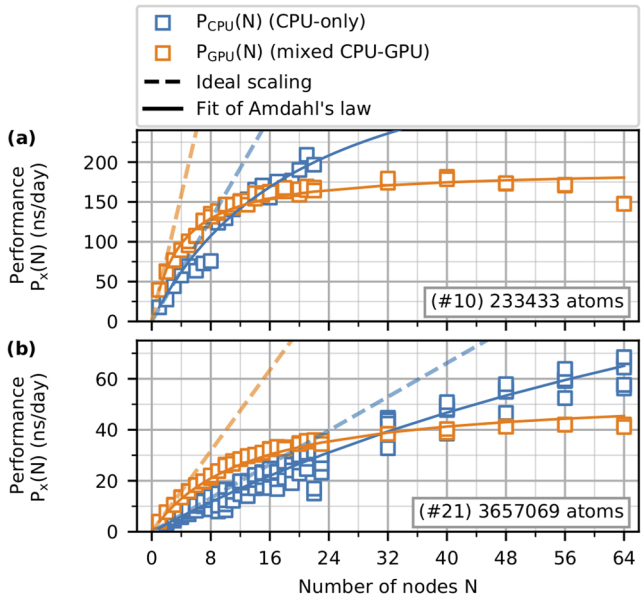

FIG. 6. Amdahl's law captures the scaling behavior. Performance $P_x(N)$ of (a) system No. 10 and (b) system No. 21 as a function of nodes N. Each panel shows results for the best performing n_{ranks} values on mixed CPU–GPU nodes without hyperthreading (orange) and for CPU-only nodes with 40 n_{ranks} and hyperthreading (blue) from Figs. 4 and 5. Each combination of nodes N was run in multiple independent simulations of 15 min each. All replicates are shown. Solid colored lines show the fits to the data using Eq. (2). Transparent dashed lines show ideal scaling according to Eq. (4), where we used the average of $P_x(1)$ over all replicates.

results in better performance and that the optimal value is actually the default setting of $n_{\text{ranks}} = 40$ and $n_{\text{threads}} = 2$ [Fig. 5(d)]. Note that the default settings will depend on the hardware architecture and configuration of the system.

With hyperthreading activated, the previously determined optimal values of n_{ranks} remain unchanged, both for CPU-only and mixed CPU–GPU nodes [Figs. 4(b), 4(e), 5(b), and 5(e)]. To quantify the effect of hyperthreading, we calculate the ratio of the absolute performance as $P_x(N)/P_x^{\text{HT}}(N)$, where $P_x(N)$ is the performance without hyperthreading and $P_x^{\text{HT}}(N)$ is the performance with hyperthreading. Here and in the following, subscript x = CPU denotes CPU-only and x = GPU denotes mixed CPU–GPU nodes. The superscript HT indicates that hyperthreading is activated, its absence that it is deactivated.

The impact of hyperthreading on performance depends sensitively on the system size, computer architecture, node number, and rank number. By and large, hyperthreading improves the performance on CPU-only nodes with optimized rank and thread numbers, as shown in Figs. 4(c) and 4(f). Only for the 233k atom system and $N = 1$, 7 or 8 nodes, we see deviations from this behavior. For mixed CPU–GPU nodes, we find that there is no significant benefit from hyperthreading, as shown in Figs. 5(c) and 5(f) for the 233k and 3.6M atom systems. Only for the larger system and large node numbers, the activation of hyperthreading leads to a small (~5%) performance increase for the optimal rank settings.

We now show that the scaling behavior, i.e., the dependence of the performance on the number of nodes, is well captured by Amdahl's law in the form of Eq. (2) for both hardware architectures (Fig. 6). This simple law captures the linear increase of the performance with small node numbers and the convergence to a plateau for larger node numbers. If node numbers become too large, the scaling breaks down and Amdahl's law cannot be applied. Using the optimal rank and hyperthreading settings for the two system's and hardware architectures considered here, we performed scaling studies up to 64 nodes. We fit Amdahl's law to the scaling curves using the single fit parameter p. We find that Amdahl's law fits the scaling curves reasonably well and that it can thus be used to summarize the results, as is done in Sec. V C.

A comparison of the performance scaling for the two hardware architectures shows that the mixed CPU–GPU nodes perform

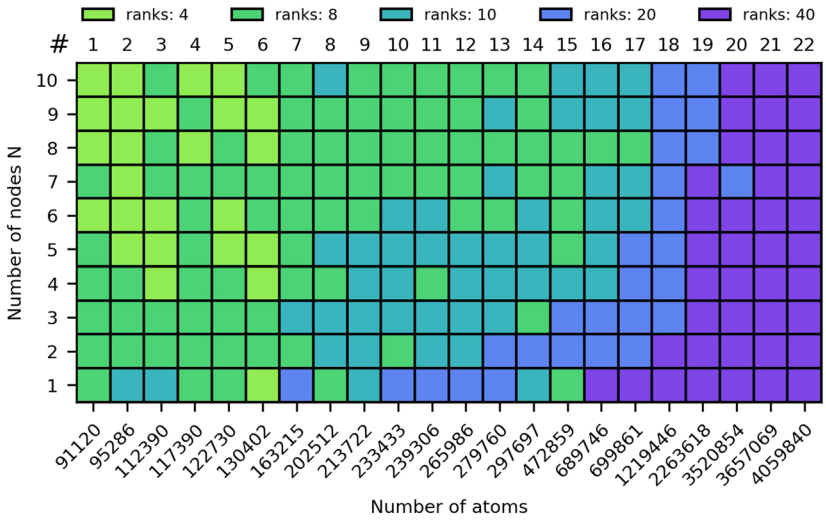

FIG. 7. On mixed CPU–GPU nodes, the optimal number of MPI ranks (colors) increases with the system size and decreases with increasing node numbers N. The same systems (columns) were scaled over different numbers of nodes (rows), using different n_{ranks} values. The top axis indicates the system numbers according to Table I.

better than CPU-only nodes for small node numbers only (Fig. 6). For the membrane protein system No. 10, mixed CPU–GPU nodes perform better up to $N = 14$ [Fig. 6(a)] and for the dense protein solution system No. 21 up to 32 nodes. For larger N, the performance increases only slowly for mixed CPU–GPU nodes such that CPU-only nodes achieve higher absolute performance. This behavior is consistent for all of the 22 systems considered here, and the point of equal performance shifts to higher N with increasing system size (Figs. 12 and 13). The location of this point of equal performance is also determined by the relative computational power of CPU and GPU. Note that both node architectures contain the same CPU. A less powerful CPU would thus shift the point of equal performance to higher node numbers. It is likely that the CPUs in the mixed CPU–GPU nodes are not fully used.[20] CPU and GPU utilization in HPC settings can be monitored with the hpcmd tool, for example.[49]

C. Size dependence of optimal parameters

As we have shown above for two exemplary systems, the optimal values for n_{ranks} depend on the number of atoms in a system for mixed CPU–GPU nodes. We further validate these observations with additional scaling results of additional 20 systems on mixed CPU–GPU nodes without hyperthreading, varying the n_{ranks} values for different numbers of nodes N (Fig. 12). We also present scaling benchmarks for each system with the best performing settings on CPU-only nodes ($n_{\text{ranks}} = 40$ with hyperthreading). We exclude the smallest system No. 0 with 35k from the following scaling

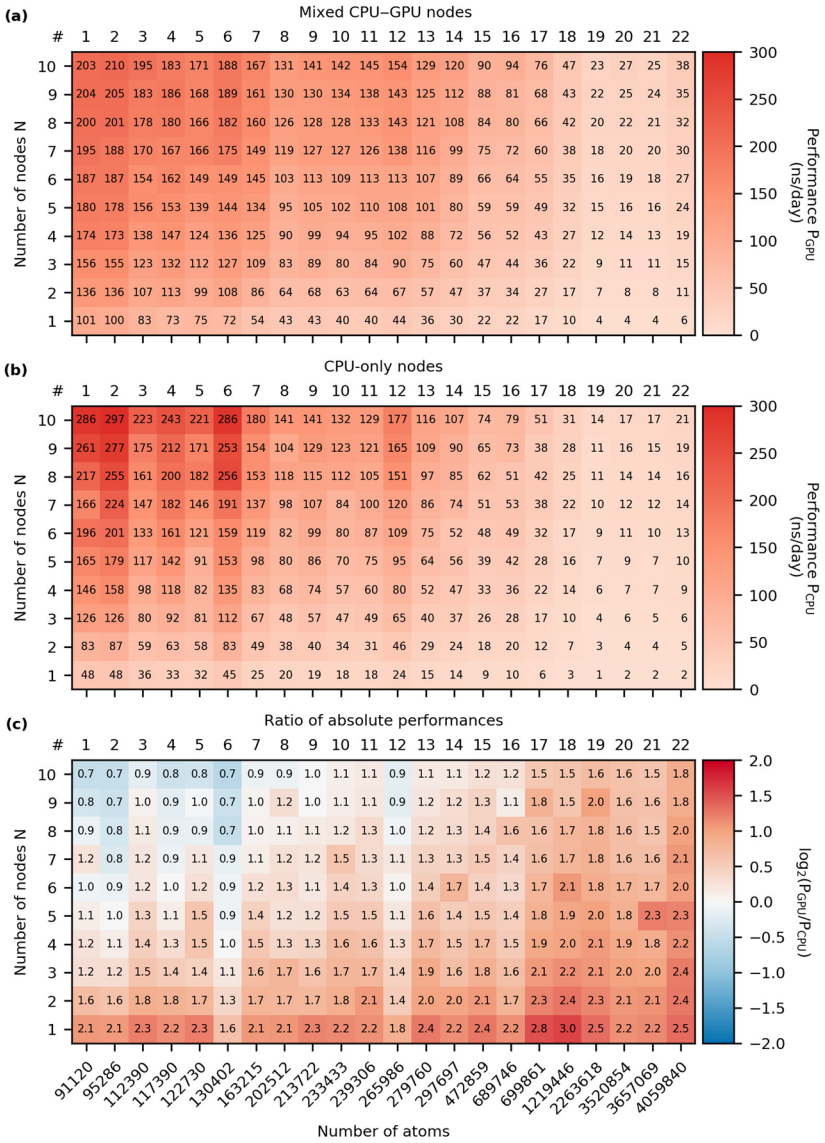

FIG. 8. Comparison of the performance on (a) mixed CPU–GPU nodes and (b) CPU-only nodes. The numbers in the fields show the performance values in ns/day. (c) The performance ratio $P_{\text{GPU}}(N)/P_{\text{CPU}}(N)$ is shown on a logarithmic (base 2) color scale. The color scale is centered at equal performance ($P_{\text{GPU}}(N)/P_{\text{CPU}}(N) = 1$, white). The numbers in the cells are the actual values of the ratios. Performance data were taken for the best performing n_{ranks} values on mixed CPU–GPU nodes without hyperthreading (Fig. 7) for each node individually. For CPU-only nodes, $n_{\text{ranks}} = 40$ and hyperthreading was activated. The top axis indicates the system numbers according to Table I. Note that for system No. 22, we used a time step of 4 fs and doubled the hydrogen mass.

analysis. Whereas this system scales nicely for CPU-only nodes, it scales poorly on mixed CPU–GPU nodes, and even using a second node is highly inefficient (see Figs. 12 and 13). Moreover, not all settings of our scaling study can be used for such small systems, and simulations with some of these settings will not run.

Our benchmark results for the 22 systems consistently show that for mixed CPU–GPU nodes, the optimal numbers of n_{ranks} decreases with increasing node number and increases with the system size (Fig. 7). Note that the number of OpenMP threads is given by the number of physical cores divided by the number of MPI ranks if hyperthreading is deactivated and by the number of logical cores divided by the number of MPI ranks if hyperthreading is activated. The observed trends are clear and consistent and provide guidelines for the optimal rank settings. However, the deviations from these monotonic trends also show that for a specific system, it is beneficial to run benchmarks for numbers of ranks close to the optimal values. The performance in Fig. 12 is in some cases degraded for 7 and 9 nodes, which can be attributed to an unfavorable domain decomposition. In general, settings with an even node number appear to be favorable.

The absolute performance for the 23 systems follow the trends as exemplified above for the 233k and 3.6M atom systems (Fig. 8). Mixed CPU–GPU nodes [Fig. 8(a)] perform better than CPU-only nodes [Fig. 8(b)] only for fairly small numbers of nodes, as indicated by their performance ratio [Fig. 8(c)]. With the exception of systems 6 and 12, the speed-up factor is larger than two for all simulations on a single node. For two nodes, a speed-up larger than two is achieved for nearly all system sizes larger than 236 K atoms, with system No. 12 and No. 16 being the exception. As we discuss in the following, these exceptions are due to aggressive tuning of the simulation settings.

The overall scaling of the performance with the number of nodes and system size is fairly smooth and quite monotonic. However, the dense protein solutions (systems No. 6, No. 12, No. 16, No. 20, and No. 21) systematically deviate from the overall trends for CPU-only nodes [Fig. 8(b)], which also becomes noticeable in the performance ratios [Fig. 8(c)]. The reason is that in these simulations, the cutoff distance in the real space interactions has been reduced and the grid spacing in the PME calculation increased compared to the other systems. With these settings, the performance could be increased significantly on CPU-only nodes. However, the performance on mixed CPU–GPU nodes remains largely unaffected. Note that in addition to the dense protein solutions listed here, also systems No. 5, No. 8, and No. 15 use the TIP4P-D[44] water model. However, these systems do not show deviations from the overall performance trends.

For all system sizes, simulations on CPU-only nodes scale much better with the number of nodes than mixed CPU–GPU nodes (Fig. 9). We quantify the computational efficiency of choosing N nodes by calculating the ratio of the actual performance to the performance we would get for ideal scaling. We determine ideal scaling using Eq. (5), using values of p from fits of Amdahl's law to the performance data. Instead of fitting to the scaling curve for the n_{ranks} value, which gives the overall best performance, we

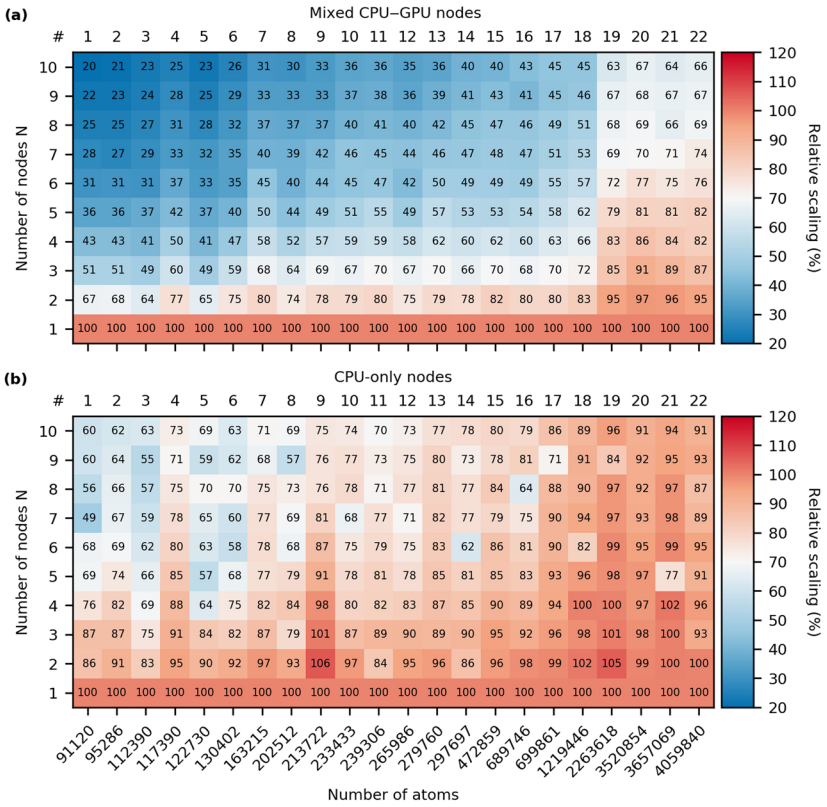

FIG. 9. Simulation performance scales differently on different architectures. Relative scaling of (a) mixed CPU–GPU nodes and (b) CPU-only nodes to their best performing settings. (a) For most system sizes, simulations scale efficiently over a couple of nodes only. The largest systems scale well up to seven nodes. (b) All system sizes scale well over multiple nodes. For the largest systems, the efficiency decreases only little for the shown node range of up to ten nodes. The numbers in each cell show the scaling efficiency to the ideal scaling of the best performing setting, as shown in Fig. 7. The top axis indicates the system numbers according to Table I. Note that for system No. 22, we used a time step of 4 fs and doubled the hydrogen mass.

generate an optimal scaling curve by first identifying for each node number N the maximum performance over all n_{ranks} values. We then identify the rank giving the optimal performance given N, as shown in Fig. 7. Finally, we fit Amdahl's law to these optimal scaling curves.

We next use the fits of Amdahl's law to the performance data to summarize the overall trends and the differences between the two hardware architectures (Fig. 10). We extract the values of the single fit parameter p and the maximum performance [Figs. 10(a) and 10(b)]. We find that for both architectures, the values of p increase with increasing system size. For CPU-only nodes, $p \gtrsim 0.9$ for all sizes. For mixed CPU–GPU nodes, p increases from ~0.5 for 10^5 atoms to >0.9 for ~2×10^6 atoms. Note that in all cases, the values of p for CPU-only nodes are larger than for mixed CPU–GPU nodes. Although we cannot expect the estimates for the maximum performance to be highly accurate, they are useful to summarize the observed trends [Fig. 10(b)]. For CPU-only nodes, the maximum performance decreases from ~545 ns/day to ~157 ns/day with increasing system size. The maximum performance of mixed CPU–GPU nodes is always smaller than the maximum performance for CPU-only nodes and decreases from ~226 ns/day for the smallest system considered here to ~50 ns/day–80 ns/day for the largest system sizes.

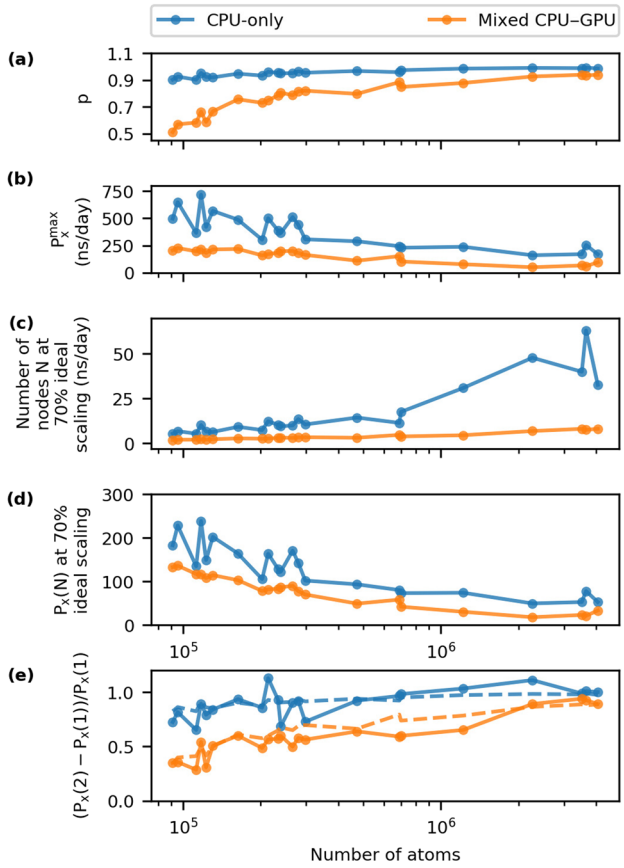

FIG. 10. Summary of the dependence on the performance on the atom number for CPU-only nodes (blue) and for mixed CPU–GPU nodes (orange). We fit Amdahl's law to the scaling data for 1–10 nodes using the single fit parameter p. (a) The fitted values of p increase with increasing atom numbers. (b) The estimates for the maximum performance decrease with increasing system size. CPU-only nodes always have larger values than mixed CPU–GPU nodes. (c) The number of nodes at which we reach 70% of the ideal performance [Eq. (7)] and (d) the performance at 70% of the ideal scaling [Eq. (8)] provide guidance for setting up benchmarks and simulations. (e) The relative performance increases when going from a single node to two nodes, directly calculated from the respective performance values. Dashed lines indicate the relative performance obtained from the fits of Amdahl's law. Note that for the largest system (No. 22), we used a time step of 4 fs and doubled the hydrogen mass.

The number of nodes for which the parallel performance stays above 70% of the ideal performance [$f = 0.7$; see Eq. (6)] increases with the system size much more quickly for CPU-only nodes than for mixed CPU–GPU nodes [Fig. 10(c)]. The estimate of this critical node number is quite sensitive to the quality of the fit because the performance changes only slowly with the number of nodes in this regime [Fig. 10(d)]. These performance estimates show that the highest performance, i.e., shortest TTS, with at least a 70% parallel performance is achieved with CPU-only nodes in all cases. Note that the choice of $f = 0.7$ is somewhat arbitrary as it depends on the trade-off between performance and efficiency. However, the trends observed for $f = 0.7$, e.g., the increase in $N(f)$ with increasing node numbers and that $N(f)$ is larger for CPU-only nodes than for mixed CPU–GPU nodes, do not depend on the exact value of f.

Note that for mixed CPU–GPU nodes, the performance increase from one to two nodes is usually small compared to the performance on one node [Fig. 10(e)]. Only for systems larger than 2M atoms, the performance increase from one to two nodes exceeds 90% of the performance of a single node. For the smallest systems, this increase is only about 35% [see also Fig. 8(a)]. By contrast, for CPU-only nodes, this increase is closer to 100% for all system sizes [see also Fig. 8(b)]. The relative performance increase calculated directly from the performance values agrees well with the results from the fits of Amdahl's law.

The small increase in performance on mixed CPU–GPU nodes when going from one to two nodes indicates that a single simulation does not use the resources of a single node efficiently.[20] To investigate this issue, we ran benchmarks of $n = 2$, 4, and 8 identical simulations on a single mixed CPU–GPU node for all systems, including system No. 0, and optimized the number of ranks. We find that we can achieve the highest total performance, given by the sum of the performances of the individual simulations running on a single node, for $n = 4$ simulations on a single node [Fig. 11(a)]. Interestingly, for $n = 8$ simulations on a single node, we consistently get the same total performance per node as for $n = 4$. Note that for $n = 1$ the optimal number of ranks decreases with an increasing number of jobs per node. Compared to running a single simulation, we obtain four times the total performance for the smallest system size of 35k atoms considered here [Fig. 11(b)]. For systems with hundreds of thousands of atoms, the total performance is about two times larger than the performance for $n = 1$. For the largest system sizes, we can still gain up to 50% in total performance by running $n = 4$ simulations instead of $n = 1$.

Running multiple simulations on a single node generally decreases the performance of an individual simulation even though the total performance accumulated on this node increases [Fig. 11(c)]. We observe two exceptions from this behavior. System

No. 0 with 35k atoms shows a large performance increase of 60% if we run one simulation per GPU instead of running one simulation per node. That is, for such a small system, we best use a single domain. We see a similar behavior for system No. 6, which is the smallest of the systems where a small real space cutoff of 1 nm and a larger grid spacing of 0.16 nm for PME are used. For this system, a simulation with 1 rank on a single GPU has a 30% higher performance than a single simulation using 4 ranks on two GPUs. In addition, we have run two identical simulations for systems No. 0 and No. 1 on a single CPU-only node using 40 ranks with hyperthreading. Running two simulations of system No. 0 with 35k atoms on a single node (1 CPU per simulation) leads to a total performance increase of 68%, whereas the larger system No. 1 with 91k atoms still gains 10% of additional performance with this setting. The results for multiple simulations on a single node nicely illustrate the benefits of running benchmarks using MDBenchmark as they reveal large performance gains for settings, which we naively might not have considered.

VI. CONCLUSIONS

High-performance computing in general, and MD simulations in particular, are fast growing and highly dynamic fields. In a rapidly changing environment of hardware, software, and systems, running MD simulations efficiently thus requires continuous benchmarking and monitoring of the simulation performance. The MDBenchmark toolkit presented here has been designed to simplify the benchmarking process. Its design is open to different MD engines and queuing systems, acknowledging the fact that it is becoming a common practice that a single user uses different MD engines on various high-performance compute clusters.

The performance of an MD simulation depends on many factors, of which only some are controlled by the user. Even for a given MD engine and hardware configuration, the performance is sensitive to the choice of the underlying algorithms and the simulation parameters. An example for the latter presented here is the increase in performance of the dense protein solutions compared to similarly sized systems, which was achieved by aggressively tuning simulation parameters. However, such tuning has to be done with great care and thorough validation, and it is our general recommendation that non-experts refrain from such fine-tuning. Activating an enhanced sampling method can affect the performance and the scaling with the number of nodes dramatically. While efficiency usually changes with the version of the MD engine itself, variations in the hardware drivers, compilers, and interfaces for parallelization can also have huge effects on the performance. For example, we observed that an update of the NVIDIA driver for the GPU increased the performance on mixed CPU–GPU nodes by up to ~20%.

Thus, our extensive performance scaling study surveyed only a small region in a high-dimensional parameter space. For example, we have not explored the effects of offloading specific calculations, i.e., PME, to separate GPU ranks or systematically investigated how tuning the cutoff parameters affects the performance. Note that offloading of PME and nonbonded interaction calculations will give large performance gains on systems where the CPU is weaker than the GPU. It would be also interesting to perform a systematic benchmark study for the widely used coarse-grained MARTINI model,

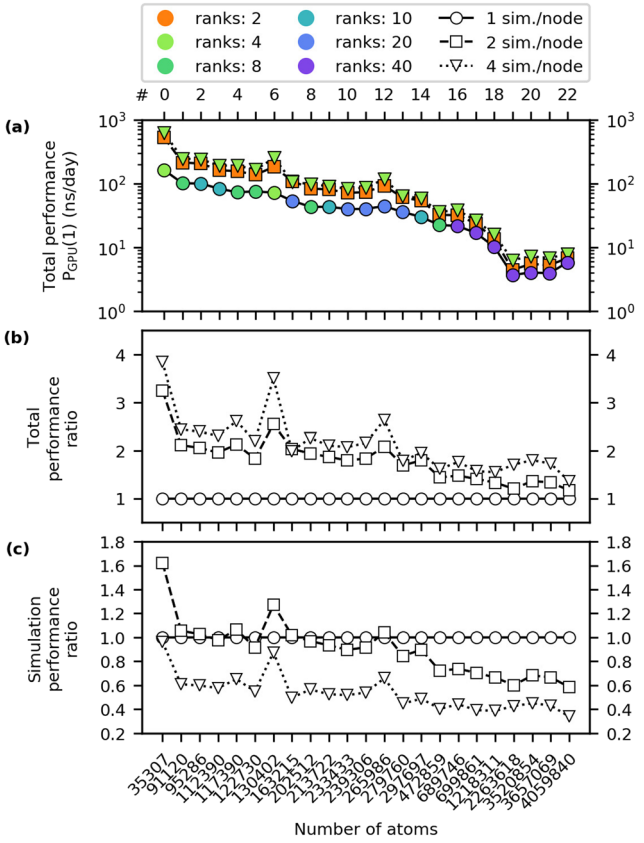

FIG. 11. Running *n* equivalent simulations on a single node increases the total performance per node in comparison to a single simulation on a single node ($n = 1$) for all system sizes. (a) The total performance for a single simulation per node ($n = 1$, circles, solid lines), two simulations per node ($n = 2$, squares, dashed lines), and four simulations per node ($n = 4$, triangles, dotted lines) for the optimal rank/thread combinations. Lines serve as a guide to the eye. (b) The ratio of the total performance per node when running *n* and $n = 1$ simulations per node ($n = 1, 2, 4$). (c) The ratio of the simulation performance with *n* and $n = 1$ equivalent simulations running on a single node. The performance averaged over all *n* simulations was divided by the performance of a single ($n = 1$) simulation.

which does not use PME.[50] Nevertheless, our extensive quantification of the performance scaling provides guidelines for GROMACS users, reveals general trends, and serves as a point of reference for performance comparison, also for users of other MD engines.

Our results illustrate that benchmarking is necessary to find optimal parameters and to identify inefficiencies due to singular deviations from observed scaling trends. In the case of GROMACS 2018, for example, the proper choice of the number of MPI ranks is crucial. For mixed CPU–GPU nodes, this choice depends on the system size, the number of nodes, and the number of simulations run on a single node. We find that hyperthreading is generally beneficial for CPU-only runs with 91k atoms or more. By contrast, for mixed CPU–GPU nodes, it depends on the system size and node number whether hyperthreading leads to performance gains. Even though the results presented here serve as guidelines, confirming these settings for the specific simulation system and the resources available is inevitable.

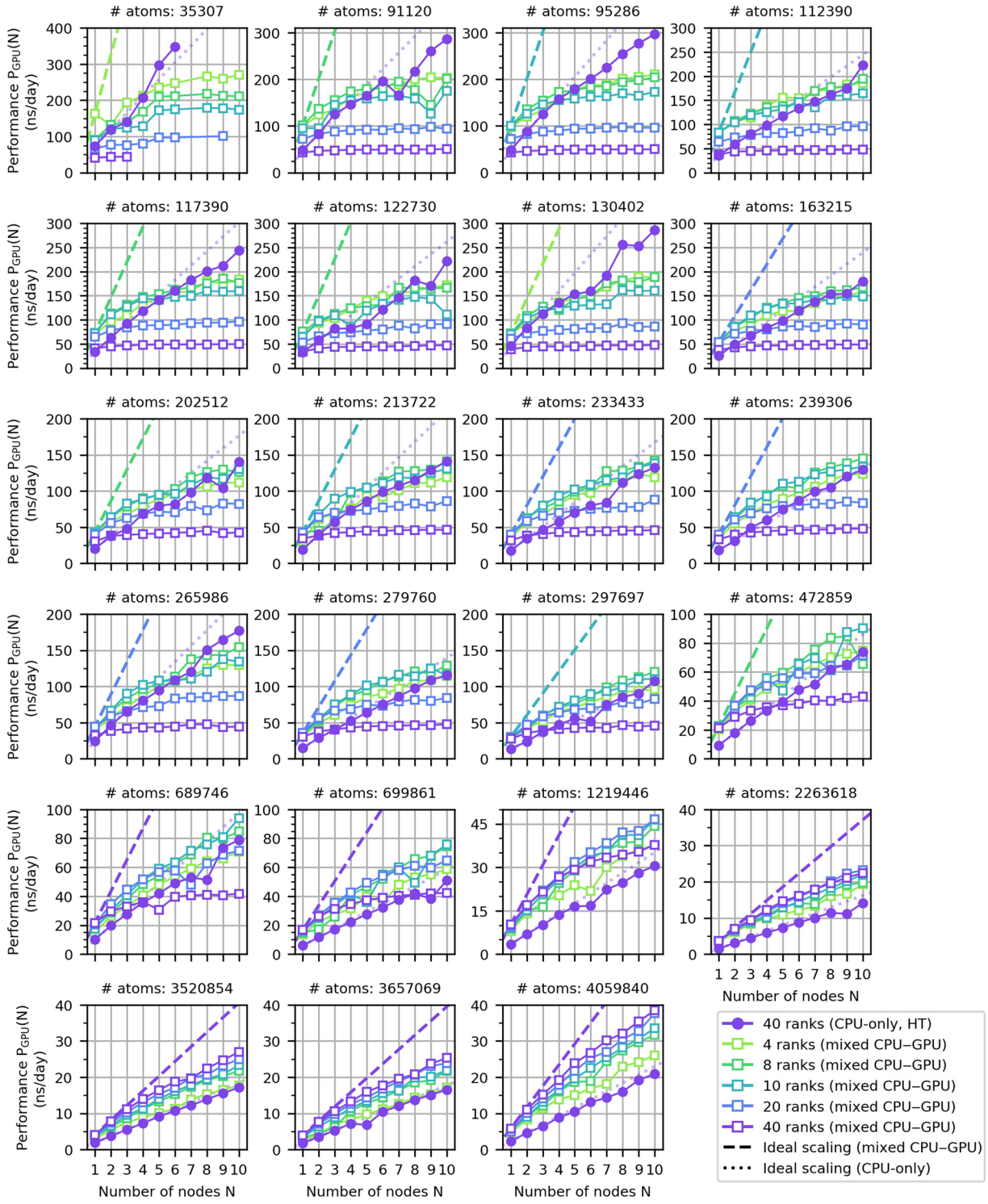

FIG. 12. For mixed CPU–GPU nodes, the optimal choice of n_{ranks} depends on the system size. Performance $P_{GPU}(N)$ of 23 MD simulations with varying sizes as a function of nodes N for different numbers of MPI ranks, n_{ranks}, on mixed CPU–GPU nodes without hyperthreading (colored squares). The best performing setting with CPU-only nodes is shown as reference (circles). Each data point in the performance plot shows one independent run of 15 min each. Transparent dotted and dashed lines show ideal scaling for CPU-only and mixed CPU–GPU benchmarks according to Eq. (4). Note that for system No. 22 with 4059840 atoms, we used a time step of 4 fs and doubled the hydrogen mass.

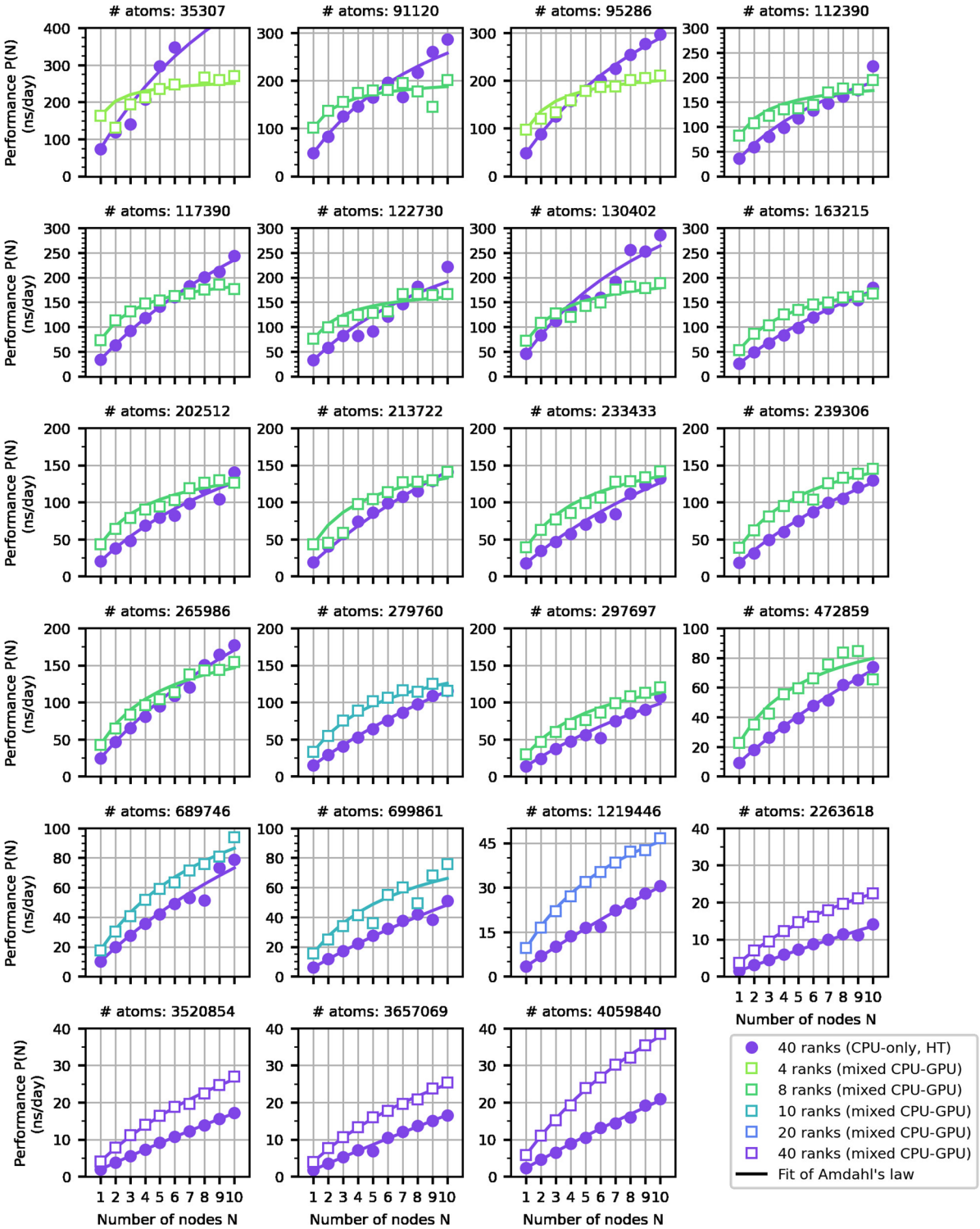

FIG. 13. GPUs scale to higher N for larger systems. Performance $P(N)$ of 23 MD simulations with varying sizes as a function of nodes N. Each panel shows results for the best performing number of MPI ranks, n_{ranks}, on mixed CPU–GPU nodes without hyperthreading (colored squares) and for CPU-only nodes with 40 n_{ranks} and hyperthreading (filled circles) from Fig. 12. The solid colored line shows the interpolated fit to the data using Eq. (2), respectively. The intersection of both curves increases to higher N with increasing number of atoms. Note that for system No. 22 with 4059840 atoms, we used a time step of 4 fs and doubled the hydrogen mass.

Our scaling results for a single simulation across multiple nodes and for multiple simulations on a single node highlight the necessity of balancing the goals of maximizing overall efficiency and of obtaining long trajectories within a reasonable amount of time. For mixed CPU–GPU nodes, running four simulations on a single node gives higher total performance but shorter individual trajectories. If having a long continuous production run is not critical, then this mode of operation is preferred. If it is, one may be willing to trade off longer trajectories against decreased efficiency. Reliable benchmark results, as provided by MDBenchmark, help in making an informed decision in this trade-off.

In its current version, MDBenchmark can scan parameters such as the number of nodes, the numbers of MPI ranks and OpenMP threads, and the activation of hyperthreading, which are set when submitting the job. However, to achieve the best performance, it is also necessary to tune parameters that are set in the configuration files of the respective MD engines. Currently, these configuration files have to be provided by the user. Ideally, future versions of MDBenchmark perform scans over simulation parameters specified in the configuration files automatically and validate the results.

The monetary and environmental costs of molecular simulations are significant, and even small relative performance improvements have a large absolute effect on the overall cost and efficiency. For many, MDBenchmark might be a first step to start continuously monitoring and evaluating the efficient use of their hardware resources. It is fair to assume that within a typical research group, with a mix of members with essentially no experience and members who are experts in running simulations, resources can be easily wasted if insufficient attention is paid to monitoring simulation efficiency. Running benchmarks as a rule for any new system and the comparison with existing benchmarks could greatly reduce the risk of wasteful use of resources.

Thus, ideally, a tool like MDBenchmark additionally collects the benchmark information generated by the users in a single database accessible to all. This information should be supplemented by the actual performance data of production runs. In principle, the MDBenchmark toolkit could be already used to perform production runs and thus automatically collect performance information. With this kind of information, inefficiencies can be identified quickly, and the database can provide accurate guidelines for setting up simulations. Such guidelines also serve to keep the number of necessary benchmarks to a minimum.

The design of MDBenchmark embraces the philosophy that we should always choose the best tool for the task at hand. Ideally, we can easily switch between different MD engines to take advantage of their unique features.[7–16] MDBenchmark is open to all MD engines (and queuing systems). We hope that the community will appreciate the design and capabilities of the provided framework to run and analyze benchmarks such that they contribute their expert knowledge by adding their favorite MD engines and queuing systems. Ultimately, running simulations more efficiently translates into doing better science.

The source code of MDBenchmark is freely available under the GPLv3 license at https://github.com/bio-phys/mdbenchmark. It can be installed either via the *pip* or *conda* package managers using the PyPI or conda-forge repositories, respectively. The code is accompanied by an extensive documentation that is hosted at https://mdbenchmark.readthedocs.io/. Detailed instructions for adding currently unsupported MD engines can also be found in the documentation.

ACKNOWLEDGMENTS

The authors thank Dr. Markus Rampp, Dr. Klaus Reuter, and Dr. Sebastian Kehl for technical support and useful discussions. They thank Dr. Florian Blanc, Sören von Bülow, Daniel Chavez Rojas, Dr. Roberto Covino, Sergio Cruz, Dr. Sonya Hanson, Dr. Ahmadreza Mehdipour, Laura Schulz, and Jan Stuke for providing molecular dynamics systems. This study used the high-performance computing resources of the Max Planck Computing and Data Facility (MPCDF). The authors acknowledge financial support by the Max Planck Society and the Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz (LOEWE) DynaMem program of the state of Hesse (M.G., M.S. and G.H.).

APPENDIX: EXAMPLES FOR SLURM SUBMISSION SCRIPTS

Scripts 1 and 2 show example SLURM submission scripts for CPU-only nodes and mixed CPU-GPU nodes, respectively.

SCRIPT 1. SLURM submission script for a 15 min run on a CPU-only single node with 40 MPI ranks, 2 OpenMP threads, and hyperthreading enabled.

```
#!/bin/bash -1
#SBATCH -o ./benchjob.out .%j
#SBATCH -e ./benchjob.err .%j
#SBATCH -D ./
#SBATCH -J n001_r40_t02_wht
#
#SBATCH –nodes=1
# Set the number of tasks per node (=MPI ranks)
#SBATCH –ntasks - per - node=40
# Set the number of threads per rank (=OpenMP threads)
#SBATCH –cups - per - task=2
# Enable hyperthreading
#SBATCH –ntasks - per - core=2
#SBATCH –time=00:17:00

module purge
module load gcc
module load impi
module load cuda
module load gromacs/2018.8

# Set number of OpenMP threads and proper core pinning with
  hyperthreading
export OMP_NUM_THREADS=$SLURM_CUPS_PER_TASK
export OMP_PLACES=threads
export SLURM_HINT=multithread

# Run gromacs/2018.8 for 15 min
srun gmx_mpi mdrun -v -ntomp $OMP_NUM_THREADS
  -maxh 0.25 -resethway -deffnm md -noconfout
```

SCRIPT 2. SLURM submission script for a 15 min run on a single mixed CPU–GPU node with 20 MPI ranks, 2 OpenMP threads, and hyperthreading disabled.

```
1 #!/bin/bash -1
2 #SBATCH -o ./benchjob.out.%j
3 #SBATCH -e ./benchjob.err.%j
4 #SBATCH -D ./
5 #SBATCH -J n001_r20_t02_woht
6 #
7 #SBATCH –constraint="gpu"
8 #SBATCH –gres=gpu:rtx5000:2
9 #
10 #SBATCH –nodes=1
11 #Set the number of tasks per node (=MPI ranks)
12 #SBATCH –ntasks - per - node=20
13 # Set the number of threads per rank (=OpenMP threads)
14 #SBATCH –cups - per - task=2
15 #SBATCH –time=00:17:00
16
17 module purge
18 module load gcc
19 module load impi
20 module load cuda
21 module load gromacs/2018.8
22
23 # Set number of OpenMP threads and proper core pinning
    without hyperthreading
24 export OMP_NUM_THREADS=$SLURM_CUPS_PER_TASK
25 export OMP_PLACES=cores
26
27 # Run gromacs/2018.8 for 15 min
28 srun gmx_mpi mdrun -v -ntomp $OMP_NUM_THREADS
    -maxh 0.25 -resethway -deffnm md -noconfout
```

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] G. E. Moore, Electronics **38**, 114 (1965).
[2] R. Kurzweil, *The Age of Spiritual Machines: When Computers Exceed Human Intelligence*, A Penguin Book, Science/Technology (Penguin Books, 2000).
[3] M. Vendruscolo and C. M. Dobson, Curr. Biol. **21**, R68 (2011).
[4] R. O. Dror, M. Ø. Jensen, D. W. Borhani, and D. E. Shaw, J. Gen. Physiol. **135**, 555 (2010).
[5] E. H. Lee, J. Hsin, M. Sotomayor, G. Comellas, and K. Schulten, Structure **17**, 1295 (2009).
[6] R. O. Dror, R. M. Dirks, J. P. Grossman, H. Xu, and D. E. Shaw, Annu. Rev. Biophys. **41**, 429 (2012).
[7] M. J. Harvey, G. Giupponi, and G. D. Fabritiis, J. Chem. Theory Comput. **5**, 1632 (2009).
[8] R. Salomon-Ferrer, A. W. Götz, D. Poole, S. Le Grand, and R. C. Walker, J. Chem. Theory Comput. **9**, 3878 (2013).
[9] B. R. Brooks, C. L. Brooks, A. D. Mackerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A. R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R. W. Pastor, C. B. Post, J. Z. Pu, M. Schaefer, B. Tidor, R. M. Venable, H. L. Woodcock, X. Wu, W. Yang, D. M. York, and M. Karplus, J. Comput. Chem. **30**, 1545 (2009).
[10] K. J. Bowers, D. E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossvary, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, in *SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (IEEE, Tampa, FL, 2006), p. 43.
[11] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl, SoftwareX **1-2**, 19 (2015).
[12] J. A. Anderson, J. Glaser, and S. C. Glotzer, Comput. Mater. Sci. **173**, 109363 (2020).
[13] W. M. Brown, A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington, Comput. Phys. Commun. **183**, 449 (2012).
[14] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten, J. Comput. Chem. **26**, 1781 (2005).
[15] J. C. Phillips, D. J. Hardy, J. D. C. Maia, J. E. Stone, J. V. Ribeiro, R. C. Bernardi, R. Buch, G. Fiorin, J. Hénin, W. Jiang *et al.*, J. Chem. Phys. **153**, 044130 (2020).
[16] P. Eastman, M. S. Friedrichs, J. D. Chodera, R. J. Radmer, C. M. Bruns, J. P. Ku, K. A. Beauchamp, T. J. Lane, L.-P. Wang, D. Shukla, T. Tye, M. Houston, T. Stich, C. Klein, M. R. Shirts, and V. S. Pande, J. Chem. Theory Comput. **9**, 461 (2013).
[17] A. M. J. J. Bonvin, A. E. Mark, and W. F. van Gunsteren, Comput. Phys. Commun. **128**, 550 (2000).
[18] C. C. Gruber and J. Pleiss, J. Comput. Chem. **32**, 600 (2011).
[19] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller, J. Comput. Chem. **36**, 1990 (2015).
[20] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller, J. Comput. Chem. **40**, 2418 (2019).
[21] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, Intel Technol. J. **6**, 1 (2002).
[22] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, J. Chem. Theory Comput. **4**, 435 (2008).
[23] J. L. Furlani, in *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)* (San Diego, CA, 1991), pp. 141–152.
[24] D. L. Dotson, S. L. Seyler, M. Linke, R. J. Gowers, and O. Beckstein, in *Proceedings of the 15th Python in Science Conference*, edited by S. Benthall and S. Rostrup (Austin, TX, 2016), pp. 51–56.
[25] W. McKinney, in *Proceedings of the 9th Python in Science Conference*, edited by S. van der Walt and J. Millman (Austin, TX, 2010), pp. 51–56.
[26] B. Turoňová, M. Sikora, C. Schürmann, W. J. H. Hagen, S. Welsch, F. E. C. Blanc, S. von Bülow, M. Gecht, K. Bagola, C. Hörner, G. van Zandbergen, J. Landry, N. T. D. de Azevedo, S. Mosalaganti, A. Schwarz, R. Covino, M. D. Mühlebach, G. Hummer, J. Krijnse Locker, and M. Beck, Science eabd5223 (2020).
[27] K. A. Feenstra, B. Hess, and H. J. C. Berendsen, J. Comput. Chem. **20**, 786 (1999).
[28] P. Larsson, R. C. Kneiszl, and E. G. Marklund, J. Comput. Chem. **41**, 1564 (2020).
[29] G. M. Amdahl, SSCS **12**, 19 (2007).
[30] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, J. Comput. Chem. **32**, 2319 (2011).
[31] R. J. Gowers, M. Linke, J. Barnoud, T. J. E. Reddy, M. N. Melo, S. L. Seyler, J. Domański, D. L. Dotson, S. Buchoux, I. M. Kenney, and O. Beckstein, in *Proceedings of the 15th Python in Science Conference*, edited by S. Benthall and S. Rostrup (Austin, TX, 2016), pp. 98–105.
[32] S. van der Walt, S. C. Colbert, and G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011).
[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, Nat. Methods **17**, 261 (2020).
[34] F. Perez and B. E. Granger, Comput. Sci. Eng. **9**, 21 (2007).
[35] J. D. Hunter, Comput. Sci. Eng. **9**, 90 (2007).

[36]R. B. Best, X. Zhu, J. Shim, P. E. M. Lopes, J. Mittal, M. Feig, and A. D. MacKerell, Jr., J. Chem. Theory Comput. **8**, 3257 (2012).
[37]J. Huang, S. Rauscher, G. Nawrocki, T. Ran, M. Feig, B. L. de Groot, H. Grubmüller, and A. D. MacKerell, Nat. Methods **14**, 71 (2017).
[38]J. Wang, P. Cieplak, and P. A. Kollman, J. Comput. Chem. **21**, 1049 (2000).
[39]V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, Proteins **65**, 712 (2006).
[40]K. Lindorff-Larsen, S. Piana, K. Palmo, P. Maragakis, J. L. Klepeis, R. O. Dror, and D. E. Shaw, Proteins **78**, 1950 (2010).
[41]R. B. Best and G. Hummer, J. Phys. Chem. B **113**, 9004 (2009).
[42]R. B. Best, D. de Sancho, and J. Mittal, Biophys. J. **102**, 1462 (2012).
[43]W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, J. Chem. Phys. **79**, 926 (1983).
[44]S. Piana, A. G. Donchev, P. Robustelli, and D. E. Shaw, J. Phys. Chem. B **119**, 5113 (2015).
[45]H. F. Hofbauer, M. Gecht, S. C. Fischer, A. Seybert, A. S. Frangakis, E. H. K. Stelzer, R. Covino, G. Hummer, and R. Ernst, J. Cell Biol. **217**, 3109 (2018).
[46]S. von Bülow, M. Siggel, M. Linke, and G. Hummer, Proc. Natl. Acad. Sci. U. S. A. **116**, 9843 (2019).
[47]X. Wu, M. Siggel, S. Ovchinnikov, W. Mi, V. Svetlov, E. Nudler, M. Liao, G. Hummer, and T. A. Rapoport, Science **368**, 433–436 (2020).
[48]S. Hofmann, D. Januliene, A. R. Mehdipour, C. Thomas, E. Stefan, S. Brüchert, B. T. Kuhn, E. R. Geertsma, G. Hummer, R. Tampé, and A. Moeller, Nature **571**, 580 (2019).
[49]L. Stanisic and K. Reuter, in *Euro-Par 2019: Parallel Processing Workshops*, edited by U. Schwardmann, C. Boehme, D. B. Heras, V. Cardellini, E. Jeannot, A. Salis, C. Schifanella, R. R. Manumachu, D. Schwamborn, L. Ricci, O. Sangyoon, T. Gruber, L. Antonelli, and S. L. Scott (Springer International Publishing, Cham, 2020), pp. 613–625.
[50]D. H. De Jong, S. Baoukina, H. I. Ingólfsson, and S. J. Marrink, Comput. Phys. Commun. **199**, 1 (2016).