Understanding the Scalability of Molecular Simulation Using Empirical Performance Modeling

Sergei Shudler[1(✉)], Jadran Vrabec[2], and Felix Wolf[3]

[1] Argonne National Laboratory, Lemont, IL 60439, USA
sshudler@anl.gov
[2] Thermodynamics and Process Engineering, Technical University Berlin, 10587 Berlin, Germany
vrabec@tu-berlin.de
[3] Laboratory for Parallel Programming, Technical University Darmstadt, 64289 Darmstadt, Germany
wolf@cs.tu-darmstadt.de

Abstract. Molecular dynamics (MD) simulation allows for the study of static and dynamic properties of molecular ensembles at various molecular scales, from monatomics to macromolecules such as proteins and nucleic acids. It has applications in biology, materials science, biochemistry, and biophysics. Recent developments in simulation techniques spurred the emergence of the computational molecular engineering (CME) field, which focuses specifically on the needs of industrial users in engineering. Within CME, the simulation code *ms2* allows users to calculate thermodynamic properties of bulk fluids. It is a parallel code that aims to scale the temporal range of the simulation while keeping the execution time minimal. In this paper, we use empirical performance modeling to study the impact of simulation parameters on the execution time. Our approach is a systematic workflow that can be used as a blueprint in other fields that aim to scale their simulation codes. We show that the generated models can help users better understand how to scale the simulation with minimal increase in execution time.

Keywords: Molecular dynamics · Performance modeling · Parallel programming

1 Introduction

Molecular dynamics simulation is a fundamental approach for understanding the behavior of matter at the molecular level. In physics, molecular dynamics is used to study the behavior and interactions between single atoms. In biomedical research, scientists simulate macromolecules such as proteins and viruses to better understand cell structures in organisms, as well as to design better medical

A. Bhatele et al. (Eds.): ESPT/VPA 2017/2018, LNCS 11027, pp. 125–143, 2019.
https://doi.org/10.1007/978-3-030-17872-7_8

drugs. In chemistry and chemical engineering, molecular simulations are used to understand and predict thermodynamic properties of fluid mixtures.

Some of the well-known molecular simulation packages are LAMMPS [7,22], NAMD [8,11], and GROMACS [6,9]. Although all these codes are based on the same principle, they have different aims and target different scientific fields. LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator and is a versatile code designed to be easily modified or extended with new functionality. It supports both solid-state materials (e.g., metals) as well as soft matter (e.g., biomolecules and polymers). The primary objective of LAMMPS is providing a platform for further research in molecular simulation. NAMD stands for Nanoscale Molecular Dynamics and is implemented in Charm++ [19]. It is specifically designed to simulate large biomolecular systems such as viruses. Similar to NAMD, GROMACS (GROningen MAchine for Chemical Simulations) is designed to simulate biomolecular structures such as proteins, lipids, and nucleic acids. This code is most often used for simulation of protein folding. One notable example is the *Folding@home* [5] project, which is a massively distributed computing effort that exploits the idle time of processing elements of personal computers owned by a large group of volunteers worldwide.

To support chemical engineering needs, recent advances in simulation techniques of fluids ushered in a new field of Computational Molecular Engineering (CME). It falls under the category of simulation-based engineering and aims to adapt existing simulation techniques, optimized for soft matter physics, to the requirements of the chemical and process engineering industry [17]. Rather than providing scientific insight, the goal of CME is to provide a systematic approach to replace experiments that are otherwise too complex, hazardous, or expensive.

LAMMPS, NAMD, and GROMACS, albeit powerful and flexible, focus in most cases on scaling the size of the molecular system rather then the simulation time. On the other hand, chemical engineering in general and thermodynamics in particular have more benefit from longer running simulations. Furthermore, industrial applications require a proportional increase both in size of the system and simulation time.

One of the simulation packages in CME is *ms2* (molecular simulation: 2nd generation) [15,16,23]. It is aimed at industrial users and samples the full set of thermodynamic properties of bulk fluids. Since these properties can reliably be calculated from a relatively smaller number of molecules (i.e., the order of 10^4), *ms2* is not designed for larger molecular systems. The challenge, therefore, is to keep the execution time (i.e., time-to-solution) of *ms2* low as various other parameters of the simulation increase.

In this paper, we use empirical performance modeling to understand the impact of simulation parameters on the execution time of *ms2*. Empirical performance modeling produces human-readable performance models from real measurements. It has been extensively studied before [12,13,24,25], but in this work, we focus on specific challenges related to modeling the performance of a CME code. The produced models can help users select appropriate input values for the simulation such that the execution time stays within potential constraints. The

workflow we provide can also help developers optimize individual computational procedures during simulation. We make the following specific contributions:

- Systematic and reliable workflow that can be used as a blue-print in performance engineering efforts of simulation codes in other fields
- Identification of pitfalls in the process of producing measurements for empirical modeling
- Exhaustive set of two-parameter and three-parameter models of execution time for the *ms2* application

The remainder of the paper is organized as follows. Section 2 provides a brief overview of the design of *ms2*. Next, Sect. 3 discusses the modeling methodology in detail. In Sect. 4, we describe the experimental setup for evaluating the methodology, and then provide a detailed analysis of the results in Sect. 5. Finally, we review related work in Sect. 6, before drawing conclusions in Sect. 7.

2 ms2 Application

The *ms2* simulation application offers a choice between two fundamental molecular simulation techniques, namely, Monte-Carlo (MC) and Molecular Dynamics (MD) [15,16,23]. The MC technique investigates the behavior of molecular ensembles stochastically. In other words, during each iteration the MC technique displaces molecules in the volume randomly, such that the probability of accepting a displacement is chosen in a way that allows obtaining a representative set of configurations. By repeating this step a large number of times, the MC technique generates a Markov chain of configurations. From this chain, static (i.e., time-independent) thermodynamic properties of the simulated molecular ensembles can be obtained. The MD technique, on the other hand, relies on the numerical solution of Newton's equations of motion. In each time step, the technique evaluates intermolecular interactions (i.e., forces and torques) that are then used to determine the spatial displacement of all molecules during the time step. Each time step results in a new configuration. Ordered chronologically, the sequence of configurations represents an approximation of the molecular propagation process such that both static and dynamic thermodynamic properties can be calculated.

Although MC is more limited in terms of accessible thermodynamic properties, it is a technique that can be parallelized easily (i.e., embarrassingly-parallel problem) since each process can generate an independent Markov chain and all chains have to be gathered only once at the end of the simulation run. To parallelize the MD technique, on the other hand, one has to parallelize the interaction calculation. For this purpose, *ms2* relies on force decomposition as proposed by Plimpton [22]. Instead of traditional domain decomposition, the interaction matrix is rearranged such that the interacting molecules are almost equally distributed in the matrix. Assuming n is the number of molecules and p is the number of processes, each process is responsible for $\frac{n}{p}$ columns of the interaction matrix. Figure 1 presents a schematic of this interaction matrix. Each gray cell

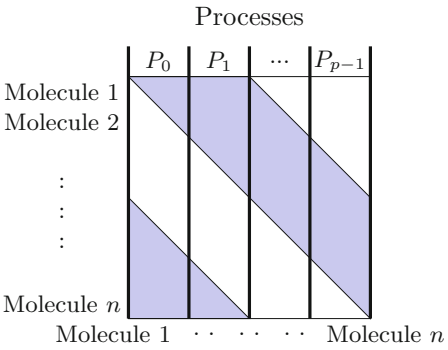


Fig. 1. Parallelization in *ms2* using force decomposition. The shaded area shows the pair interactions that have to be calculated for the simulation to proceed. Vertical black lines delimit the range of molecules for which a single process calculates the interactions.

represents an interaction between a pair of molecules that has to calculated for the simulation to proceed to the next time step. The assumption is that each process stores all molecule data (coordinates, momenta, etc.) locally. However, each process calculates only the interaction for a subgroup of molecules—exactly the group of molecules delimited by the black vertical lines in the figure. In this way, the work load is distributed almost equally between the processes. The root process then reduces all the resulting interaction components to sum up the molecular forces exerted on each individual molecule. For both MC and MD parallelization, the *ms2* application uses MPI [15]. Specifically, the MPI collective operations Barrier, Bcast, Reduce, and Allreduce are employed.

3 Methodology

In this section, we describe the methodology to produce performance models for the execution time of *ms2*. In general, the methodology follows the practice established by earlier studies. Specifically, we draw upon past experience in modeling the isoefficiency functions of task-based applications [25].

Figure 2 provides an overview of our methodology. In general, we can identify three separate phases: selecting parameters, benchmarking, and empirical modeling. Code instrumentation is an optional step that should be included if the aim is to produce models and derive predictions for specific parts of the code rather than the simulation as a whole. The subsections below cover the phases in the workflow in more detail.

3.1 Simulation Parameters

The *ms2* application has a group of parameters that characterize the simulation scenario. The most important parameters identify the type of the simulated

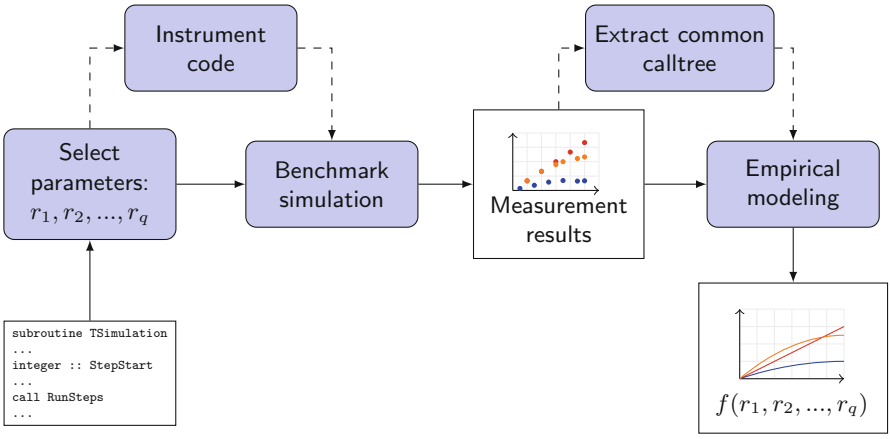


Fig. 2. Modeling workflow.

molecule (i.e., a molecular model), the number of molecules, density (or the simulated volume), temperature, and the number of time steps.

To accurately describe the interaction between molecules, *ms2* uses potential functions that describe different interaction types. Each molecular model specifies the placement of an *interaction site* on the molecule and the type of this site. A Lennard-Jones (LJ) site represents dispersive and repulsive interactions, while point charge (PC), point dipole (PD), and point quadrupole (PQ) sites represent electrostatic interactions. The complexity of molecular models depends on the molecules they represent. A simple Ar (argon) atom has only a single LJ site. A more complex CO_2 (carbon dioxide) molecule consists of three LJ sites and one PQ site. A $(CH_3)_2CO$ (acetone) molecule, however, has four LJ sites, one PD site, and one PQ site. *ms2* calculates the interaction forces between pairs of the same type of sites. For example, the interaction between two CO_2 molecules will be a combination of nine interactions between the LJ sites (three sites in each molecule equals nine different pairs) and one interaction between the PQ sites.

Using appropriate interaction sites in a molecular model is crucial for obtaining correct thermodynamic properties. However, from a computational point of view, the difference between calculating the interaction between any of the different sites is small. Furthermore, the calculation does not depend on any other simulation parameter. A far more important factor is the total number of sites in a molecule, since the computation time of molecule interaction grows quadratically with the number of sites of each type.

Following an analysis of the *ms2* design, we identified the group of parameters that should be considered for modeling the execution time of *ms2* (i.e., independent variables in our modeling):

- n: number of molecules in the simulation; range: 10^3–10^4
- m: number of interaction sites; range: 1–8
- d: density of the fluid; range: 0.001–0.9 (in reduced units σ^{-3})

– c: cut-off radius; range: 1–10 (in reduced units σ)
– p: number of MPI processes

The values for different parameters can be provided in SI units, but internally, the *ms2* application uses reduced quantities for the calculations. For example, the reducing unit for length σ is on the order of 3 Å (i.e., $3 \cdot 10^{-10}$ m).

The first three parameters, namely, n, m, and d, are part of the parameters that determine the simulation scenario. The last two, namely, c and p, are optimization and execution parameters. The cut-off radius c defines the radius around a molecule within which the interactions with other molecules are calculated explicitly. Decreasing c results in less computational effort to evaluate the interactions for each molecule, since less neighbors have to be considered. Basically, the cut-off radius provides a trade-off between the accuracy of thermodynamic properties and the runtime of the simulation.

To simplify benchmarking and modeling, we used synthetic molecular models with m LJ sites. Each such model is comparable to a model of a real molecule with the same number of interaction sites m, independent of the site type.Therefore, performance models based only LJ sites (synthetic molecules) are a viable proxy for performance models based on ensembles of real molecules. The advantage of the former is the ease of generating synthetic molecular models. The LJ sites in the synthetic molecule were placed at the vertices of a regular polygon with m edges and an edge $s = 0.1$ Å. This allowed us to conveniently generate molecular models for arbitrary values of m. If the center of the polygon was at $(0, 0)$, then the coordinates (x_i, y_i) of vertex i are given by (k is the circumradius from the center of the polygon to one of the vertices):

$$r = \frac{s}{2\sin(\frac{\pi}{m})}$$
$$x_i = r \cdot \cos(2\pi\frac{i}{m})$$
$$y_i = r \cdot \sin(2\pi\frac{i}{m})$$

The list of parameters above is not exhaustive. Additional parameters of *ms2* are the specified boundary conditions (i.e., simulated ensemble type), the length and the number of time steps, frequency of writing results and errors to a disk, temperature, and so on. Some of these parameters, such as temperature, have little influence on the computational cost. The number of time steps influences the execution time, but the relation is simply linear such that this parameter does not have to be considered in the modeling process. The same rationale also applies to other parameters omitted from the list above.

3.2 Benchmarking

Once the group of independent parameters has been identified, we can move to the benchmarking phase. However, one optional step before benchmarking is

instrumentation. By instrumenting the relevant regions of the code (i.e., functions, kernels, or code blocks), one can produce a model for each region. In this way, for example, we can obtain a model for the execution time of a single time step in *ms2*. Whether such high-resolution modeling is needed depends on the application and the analysis goals. In the *ms2* case, a model for a single time step makes little sense, since a simulation with a single time step is useless. The aim of the simulation is to simulate an evolving environment of molecules. Nevertheless, we ran benchmarks with both an instrumented and an uninstrumented version of *ms2* to evaluate our workflow. We used Score-P [20] for the instrumentation since it integrates easily with the Extra-P [4] modeling tool (discussed in the next subsection) and provides flexible instrumentation approaches. In other words, one can either automatically instrument all of the regions in the code or manually instrument just the most relevant ones.

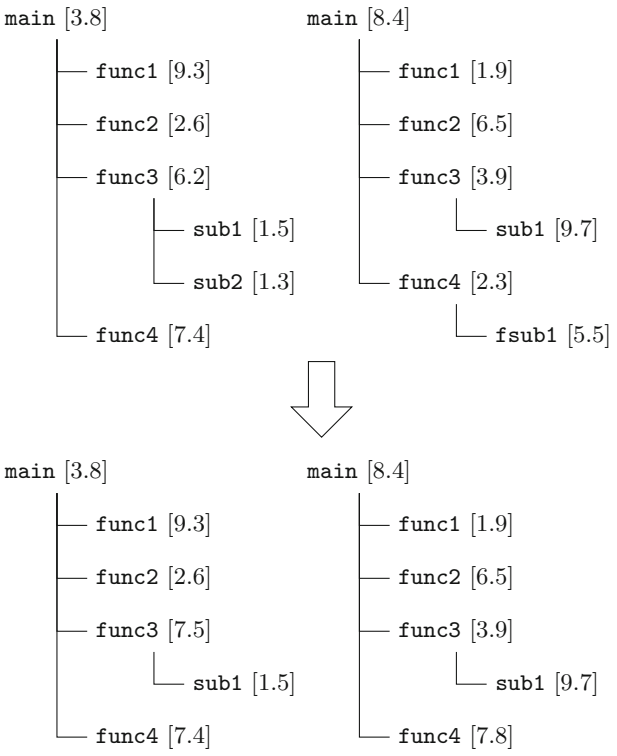


Fig. 3. Common calltree extraction from two different trees (top). The two trees at the bottom have a common structure.

During the execution of an instrumented application Score-P creates a calltree, where the root node is the first (main) function called and each new node (i.e., *cnode*) represents a called subfunction. An edge between nodes represents

a caller-callee relation. Once the application terminates, Score-P writes a performance profile to disk. Each profile is a CUBE [2] file that contains performance data arranged in three dimensions—metrics, calltree (cnodes), and system (processes/threads). Basically, there is a measurement value for each combination of (metric, cnode, process/thread). When Extra-P is used to generate a model of the execution time from these data, it collapses the system dimension by taking the maximum value and generates a separate model for each cnode. Therefore, it is important to ensure that the calltree structure is similar across all of the data used for modeling. Since the application is executed with different values for independent parameters, differences in calltrees are inevitable. To extract a common calltree, we used the `cube_commoncalltree` utility provided with CUBE. This utility looks for cnodes that do not appear in every calltree and then merges them into the parent cnode by adding the inclusive value of performance data of the child cnode to the parent's exclusive value. An inclusive value is a sum of values for the cnode itself and all of its descendants, whereas an exclusive value includes only the cnode itself, without its descendants. This computation is repeated as long as non-common cnodes are present.

Figure 3 shows an example of extracting a common calltree from two different trees. The numbers in brackets are example values for some metric (e.g., execution time). If a node has child nodes, then the number in the bracket is the exclusive value, otherwise it is an inclusive value. Note that if a node has no children, then the exclusive value is the same as the inclusive one. The figure shows that the value for the `sub2` node was merged into its parent `func3`, and the exclusive value of `func3` was updated accordingly. In a similar way, `fsub1` was merged into `func4`.

It is important to note that common calltree extraction is necessary only if all code regions are instrumented. If we do not use instrumentation at all or manually instrument just some regions of the code that are always executed, we can skip the common calltree extraction.

3.3 Empirical Modeling

The benchmarking phase is followed by the empirical modeling phase. Specifically, we use the performance-model generator in Extra-P [4], a tool for automated performance modeling of HPC applications. The model generator has already shown to confirm known performance models of real applications as well as discover previously unknown scalability bottlenecks [13,27], and has also been validated using a wide range of synthetic functions [12]. Furthermore, specific usage examples include modeling the performance of OpenMP constructs [18] and the isoefficiency functions of task-based programs [25].

A multi-parameter model aims to capture how a number of independent parameters, such as process count, problem size, and algorithmic parameters, influence a target metric, such as runtime, floating-point operations, and so on. The key concept of the modeling approach in Extra-P is the *performance model normal form* (PMNF) for multiple parameters [12]:

$$f(r_1, r_2, ..., r_q) = \sum_{k=1}^{n} c_k \cdot \prod_{l=1}^{q} r_l^{i_{k_l}} \cdot \log^{j_{k_l}}(r_l) \quad (1)$$

In this form, parameters r_l are represented by q combinations of monomials and logarithms, which are summed up in n different terms to form the model. The exponents i_{k_l} and j_{k_l} are chosen from sets $I, J \subset \mathbb{Q}$, respectively. Essentially, these sets define the scope of all possible terms. Consider, for example, $n = 3$, $q = 2$, and $I = \{0, 0.25, 0.5\}$, $J = \{0, 1\}$. In this case, the search space for possible terms would be $\{1, \log(r), r^{0.25}, r^{0.25}\log(r), r^{0.5}, r^{0.5}\log(r)\}$, and an example model could be: $f(r_1, r_2) = c_1 + c_2 \cdot r_1^{0.5} + c_3 \cdot r_1^{0.25} r_2^{0.25} \log(r_2)$.

The generator requires a set of measurements as input whose precise nature depends on the scaling objective (e.g., number of processes vs. input size, weak vs. strong). As a rule of thumb, it needs at least five different settings for each independent parameter. For example, if there is only a single parameter, such as the number of processes, we need to benchmark the application with five different values of this parameter. If there are two or more parameters, we need to benchmark the application for each combination of parameter values. This means at least 5^q measurements are required for q parameters. Each such measurement has to be repeated a number of times to obtain a statistically significant result. If k repetitions are required, the application has to be executed $k \cdot 5^q$ times. Figure 4 shows typical benchmarking results for two parameters. In this case, the number of MPI processes p and the number of molecules n were varied. The points represent parameter combinations for which execution times were measured. There are six different values for each parameter, which means that there are 36 combinations. Each of the 36 points represents a median value of $k = 10$ repetitions.

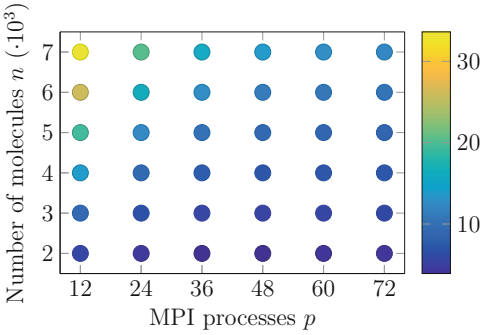


Fig. 4. Typical benchmark results for two parameters; in this case, the number of MPI processes p and the number of molecules n were varied. The color of each point represents the execution time in seconds. (Color figure online)

The modeling technique in Extra-P is based on an iterative modeling refinement process that stops when $\bar{R}^2$—the adjusted coefficient of determination—cannot be improved any further. The adjusted coefficient of determination is a

standard statistical fit factor $\in [0, 1]$ such that a value of 1 indicates a perfect fit. Since even small increases in n and q can lead to a prohibitively large search space of possible terms, the technique employs a heuristic that reduces the number of candidate models. Specifically, the search space of possible terms is generated from the best single parameter models for each individual parameter. This leads to a smaller number of candidate models, which greatly reduces the time for finding the best fitting model, but still retains a high degree of accuracy [12].

4 Experimental Setup

We performed our evaluation on Hazel Hen, a Cray XC40 system at the High Performance Computing Center Stuttgart (HLRS). The system has 7712 compute nodes with the Aries interconnect fabric and Dragonfly topology. Each node comprises two Intel Xeon E5-2680 v3 processors with 12 cores each and 128 GB of memory. In other words, there are 24 cores per node and more than 5 GB of memory per core on Hazel Hen.

The *ms2* application uses OpenMP to parallelize the calculation of interactions in each process. A performance audit of *ms2*, performed as part of the Performance Optimisation and Productivity project [3], suggested that the optimal number of OpenMP threads is four (i.e., four cores are used by one process). Following this observation, we used four OpenMP threads in all of our benchmarks. Consequently, there were six MPI processes per node.

4.1 Parameter Values

For each independent parameter discussed in Sect. 3.1, at least five different values have to be chosen. The following list specifies our choices:

- n: 2000, 3000, 4000, 5000, 6000, 7000
- m: 1, 2, 3, 4, 5, 6
- d: 0.05, 0.20, 0.35, 0.50, 0.65, 0.80
- c: 1, 2, 3, 4, 5, 6
- p: 12, 24, 36, 48, 60, 72

As discussed in Sect. 3.3, producing a model with all independent variables (e.g., $T(n, m, d, c, p)$) is not feasible since it would require at least 5^5 measurements. The alternative is to generate a series of two-parameter and three-parameter models that describe the application behavior and allow us to produce useful time predictions. For example, we can generate models $T(n, p)$, $T(n, m)$, $T(n, m, p)$, and so on. In each case, however, we vary only a subset of two or three parameters. The values for the other parameters have to remain constant throughout the benchmarking phase of each particular model. For example, if one runs benchmarks to generate the model $T(n, p)$, the values for m, d, and c have to remain constant.

4.2 Measurements Variability

Earlier studies showed that applications that run on Cray XC40 might experience a high degree of variability in execution time and performance [10,14,28]. The reason is that Cray XC40 uses the Dragonfly topology. It is a high-radix, low-diameter network that utilizes shared links and is designed to improve bandwidth and reduce packet latency. Furthermore, it uses adaptive routing and random node placement, both of which can alleviate congestion and achieve better load-balancing. However, the combination of these characteristics makes each application highly susceptible to the behavior of other applications that are being executed at the same time. In other words, a communication-intensive application can cause performance degradation in less-intensive applications executed concurrently.

Table 1. Variability (i.e., coefficient of variation (CV)) of measurements for generating the model $T(n, p)$. The columns specify values for the time step number and cut-off radius, as well as whether the code was instrumented and a compact placement (CP) of nodes was used.

Time steps	Cut-off	Instr.	CP	CV
3,000	2.0			3.3%
3,000	2.0	✓		13.5%
30,000	2.0			10.5%
3,000	4.0			6.2%
3,000	4.0	✓		58.7%
40,000	4.0	✓		26.6%
40,000	4.0	✓	✓	8.3%

For empirical modeling, the execution of the application for any combination of parameter values has to be repeated k times (see Sect. 3.3). In our evaluation, we set $k = 10$, and sometimes $k = 5$ to reduce the total time to obtain the measurements. The purpose of these repetitions was to increase the statistical significance of the measurements. However, a high degree of variability between repetitions indicates a high level of noise, which makes modeling far less accurate [13].

Table 1 shows how various factors influence the variability of the measurements. In this case, the measurements were performed to generate the model $T(n, p)$, which means repeated executions for different combinations of parameter values for n and p. Variability was measured as the coefficient of variance (CV) between the repetitions for each combination of values. The CV is defined as the ratio of the standard deviation to the mean and shows the extent of samples variability in relation to the mean.

The two leftmost columns in Table 1 specify the values for the number of time steps and the cut-off radius, respectively. The column *Instr* specifies whether the

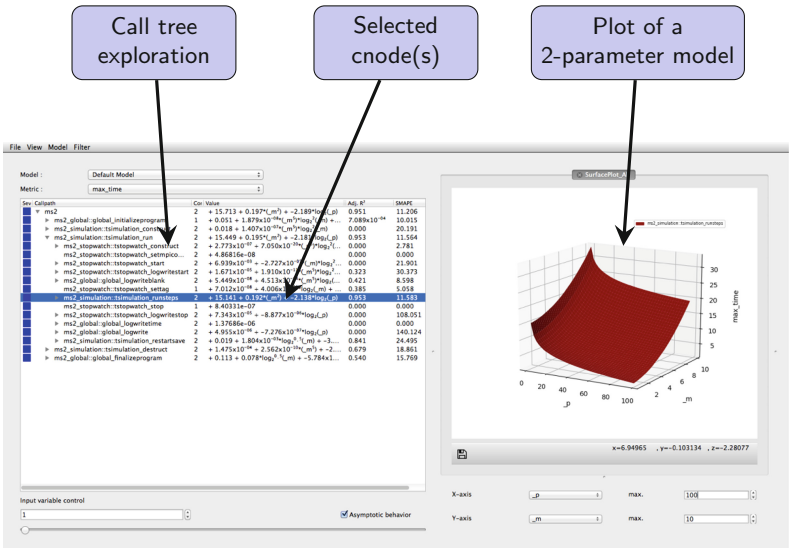


Fig. 5. Graphical user interface of Extra-P.

code was fully instrumented and column *CP* specifies whether the nodes were placed compactly in the machine, that is on the same blade and in the same chassis. The table indicates that increasing the number of time steps increases the variability. It also shows that fully instrumenting the code increases the variability as well. However, one factor which helps reduce the CV is placing the nodes physically together. These observations can be explained by the Dragonfly topology studies [10,14,28] and the low intensity of communication in *ms2* [15]. The longer the *ms2* code runs, the more time it is under the influence of other communication-heavy applications on the machine, which causes the variability to increase. This is also the reason why placing the nodes together reduces the variability—only local communication links are used in such cases. Unfortunately, this placement mode is not generally available on Hazel Hen as it has a negative effect on the utilization of the system.

Full instrumentation has a minimal perturbation in terms of performance, but has a significant effect on the variability. One likely explanation is that code instrumentation uses more communication to collate some of the measured data while the program runs. As discussed earlier, full instrumentation is not needed to model the total simulation time. Consequently, to reduce the variability we ran uninstrumented code with 3,000 time steps and cut-off radius of 2.0 (when it is not a parameter in the model).

5 Result Analysis

In this section, we discuss the results of our evaluation. As described above, we chose to focus on the execution time of full simulations rather than the execution

Table 2. 2-parameter models for the execution time of the *ms2* application.

Model	Fixed parameters	Model	$\bar{R}^2$
$T(n,m)$	$d = 0.84$, $c = 2.0$, $p = 72$	$4.41 + 8.03 \cdot 10^{-5} \cdot m \cdot n \cdot \log n$	0.99
$T(p,m)$	$n = 4{,}000$, $d = 0.84$, $c = 2.0$	$6.6 + 3.21 \cdot m^2 - 0.42 \cdot m^2 \cdot \log p$	0.92
$T(p,d)$	$n = 4{,}000$, $m = 1$, $c = 2.0$	$20.67 - 2.2 \cdot \log p$	0.88
$T(p,c)$	$n = 4{,}000$, $m = 1$, $d = 0.84$	$33.83 + 0.05 \cdot c^3 - 4.89 \cdot \log p$	0.79
$T(n,c)$	$m = 1$, $d = 0.84$, $p = 36$	$-0.99 + 0.06 \cdot c^3 + 1.81 \cdot 10^{-5} \cdot \log^2 n$	0.95
$T(m,c)$	$n = 4{,}000$, $d = 0.84$, $p = 36$	$-23.49 + 10.09 \cdot m + 0.22 \cdot c^3 \cdot m$	0.95

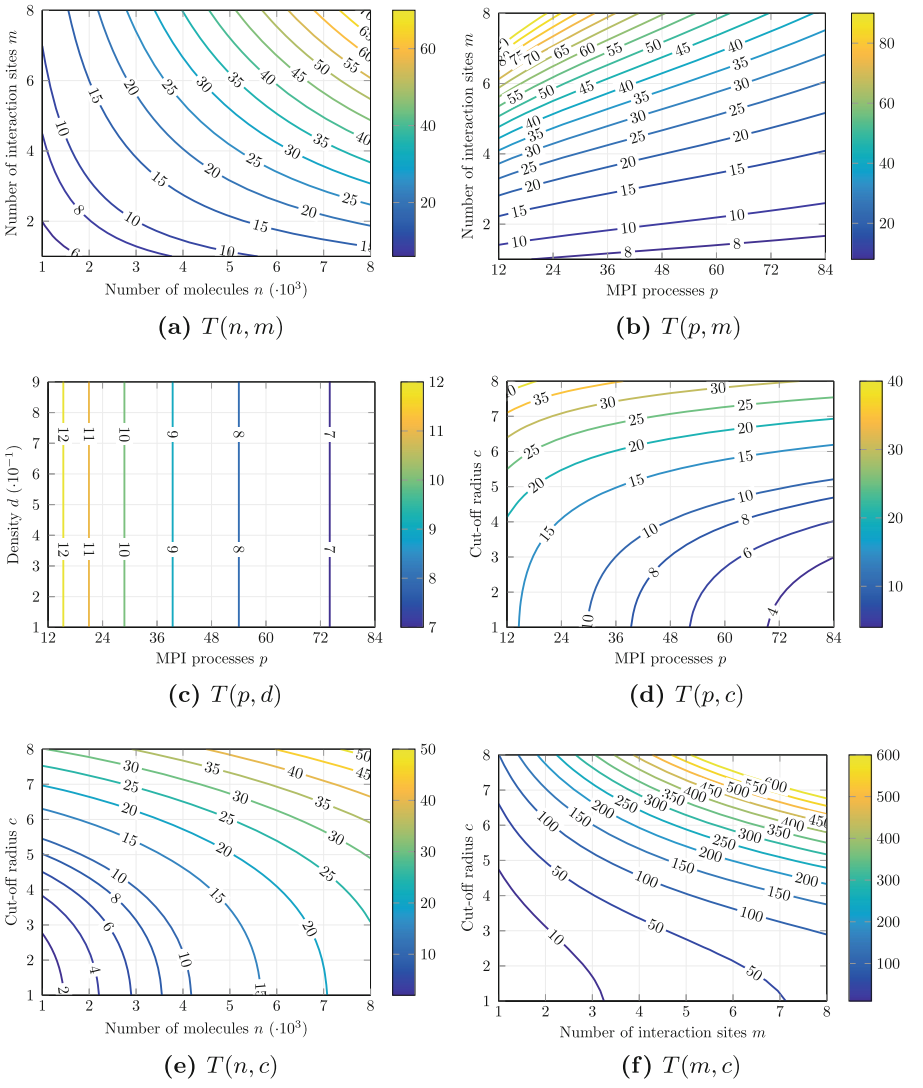
Table 3. 3-parameter models for the execution time of the *ms2* application.

Model	Fixed parameters	Model	$\bar{R}^2$
$T(p,n,m)$	$d = 0.84$, $c = 2.0$	$62.28 + 2.03 \cdot 10^{-8} \cdot m^2 \cdot n^{1.5} \cdot \log^2 n - 9.63 \cdot \log p$	0.83
$T(n,m,c)$	$d = 0.84$, $p = 72$	$9.24 + 5.71 \cdot 10^{-6} \cdot n \cdot \log n \cdot c^2 \cdot \log c \cdot m$	0.88

time of individual steps, so that there was no need for full instrumentation of the code. Nevertheless, Sect. 3 provides a detailed description of how the workflow can support full instrumentation for later analysis of *ms2* or other codes. To complete the discussion on full instrumentation, we briefly present the Extra-P graphical user interface (GUI) that allows users to analyze the model at each cnode. Figure 5 shows a screenshot of the Extra-P window. The left part is divided into two areas. The upper area has a dropdown box that shows the selected metric (e.g., time, visits, etc.) and allows users to choose a different metric. The lower area contains the calltree with a model for each cnode and fit factors such as $\bar{R}^2$ (adjusted coefficient of determination) besides each model. By clicking on any one of the cnodes, the plot of the corresponding model is displayed in the right part of the Extra-P window. The figure shows an example plot for a two-parameter model $T(p,m)$. It is a three-dimensional surface where the vertical axis is the time dimension.

The Extra-P GUI provides a convenient way to explore and compare multiple models from the same calltree. However, when no instrumentation is involved, Extra-P provides a programmatic interface to produce models directly from the measurement results. We used this interface in our evaluation and produced a set of models summarized in Tables 2 and 3. The leftmost column specifies the independent parameters in each model and the following column specifies the values of the parameters that were fixed in each case. The second column from the right shows the two-parameter and three-parameter models produced by Extra-P and the rightmost column specifies the adjusted coefficient of determination.

Unsurprisingly, the model $T(p,m)$ shows that the execution time increases in quadratic proportion to the number of interaction sites m. Furthermore, all models with the cut-off radius c as an independent parameter show that the time increases in cubic proportion to c. This is because every increase in the cut-off



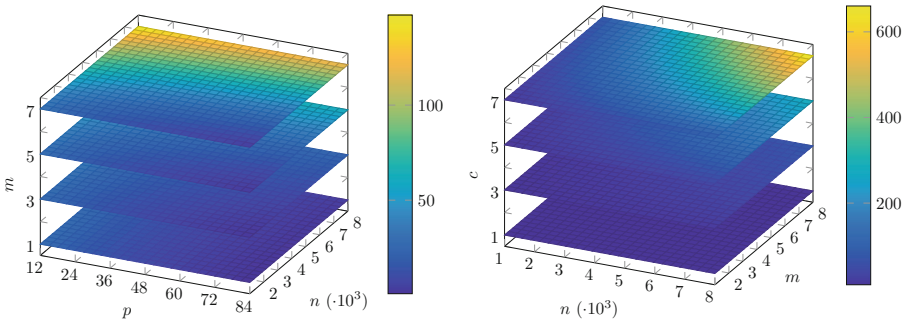
(a) $T(n, m)$ **(b)** $T(p, m)$

(c) $T(p, d)$ **(d)** $T(p, c)$

(e) $T(n, c)$ **(f)** $T(m, c)$

Fig. 6. Contour lines of the plots for 2-parameter models of the *ms2* execution time. All times are in seconds.

radius leads to a cubic increase in the cut-off volume around each particle, which also means a cubic increase in the number of particles in the cut-off sphere. These results confirm our expectations about the factors that influence the simulation. Although m depends on the simulated fluid and cannot be reduced without breaking the simulation, the cut-off radius c is an important optimization factor and should be as minimal as the simulation goals permit.



(a) $T(p, n, m)$ **(b)** $T(n, m, c)$

Fig. 7. Stacked plots of 3-parameter models of the *ms2* execution time. All times are in seconds.

The model $T(p, d)$ has no terms with density d, which suggests that increasing d has a minimal or no effect on the execution time at all. This is surprising since increasing the density leads to a linear increase in the number of particles in the cut-off sphere. One reasonable explanation is that not all the molecules in the cut-off sphere are taken into the account during the interaction calculation. The interaction matrix, discussed in Sect. 2, is arranged in a way that makes each process calculate only part of the interactions. Therefore, additional interaction calculations that are caused by a higher number of molecules in the cut-off sphere are distributed evenly between the processes leading only to a slight increase in the wallclock time of the simulation.

Furthermore, all models containing the number of processes p clearly show that increasing p leads to just a logarithmic improvement in the execution time. Although an increase in the number of processes means less interactions have to calculated by each single process, the cost of communication (i.e., MPI collective operations) still increases. This suggests that to achieve shorter execution time, we might look at changing other parameters rather than p.

Figure 6 depicts the plots of two-parameter models from Table 2 as contour lines. The label on each line specifies the time value along that line. The shape of the lines and their density provide a visual cue as to how fast the execution time increases and which parameter has more impact on this increase. Furthermore, each contour line shows how both parameters have to be increased so that the execution time remains constant. Figures 6b and d, for example, show that increasing p reduces the execution time. However, the shape of the contour lines in these figures is different. For higher p values, increasing c leads to faster increase in the runtime in Fig. 6b compared to equivalent increase of m in Fig. 6d. As another example, Figs. 6e and f show that the impact of increasing both m and c is much more severe than increasing n and c at the same time.

Figure 7 depicts three-parameter models from Table 3, namely, $T(p, n, m)$ and $T(n, m, c)$. As these functions are four-dimensional entities it is not straightforward to visualize them. The figure shows 3 axes—one for each parameter—and

horizontal slices of data for different values of the z axis. The colors represent execution times. For example, the topmost slice in Fig. 7a represents the function $T(p, n, 7)$. The differences between the slices show the impact of increasing the parameter represented by the vertical axis, that is m in Fig. 7a and c in Fig. 7b. The figures suggest that one should find a balance between different parameters to keep the execution time under a certain threshold.

6 Related Work

The empirical modeling approach that forms the basis for Extra-P was first proposed by Calotoiu et al. [13] in the context of identifying scalability bugs. A scalability bug is a part of the program in which scaling behavior is unintentionally poor. In this study, the authors produced scaling models of execution time as a function of the MPI process count, but no other independent parameters were considered. In another study, Vogel et al. [27] used both Score-P and Extra-P to analyze the scalability of the whole UG4 framework, which simulates drug diffusion through the human skin. The authors showed that Extra-P was able to produce over 10,000 models—spanning the whole calltree—in less than a minute. Each model was a scaling model of execution time as a function of the number of MPI processes. These studies had only one independent parameter and did not have to overcome pitfalls that arise when dealing with multiple parameters.

The capability to produce empirical models with multiple parameters was introduced by Calotoiu et al. [12]. This functionality is based on a number of important heuristics that make the approach feasible in practice. The authors performed their evaluation using a number of scientific codes that were executed on a Blue Gene/Q system. They produced multi-parameter models of execution time and floating point operations. In the present study, we go one step further and provide a systematic workflow that can be readily applied in performance engineering of simulation codes.

Shudler et al. [24] proposed a framework, based on empirical modeling, for validating performance expectations of HPC libraries. The framework targets both developers and users, and provides a systematic method that allows, with as little effort as possible, to evaluate whether the observed behavior corresponds to the expected behavior. The authors focused on scaling models with one independent parameter, namely, the number of MPI processes. The benefits of this framework for performance engineering inspired the methodology developed in this work. Furthermore, past experience of Shudler et al. [25] in modeling the isoefficiency functions of task-based applications [25] provided important guiding points for designing the present workflow.

Singh et al. [26] and Marathe et al. [21] used machine learning techniques to model the effects of various input parameters on the performance of scientific codes. The authors showed that some of these techniques can handle a large parameter space without the costs associated with our methodology. However, machine learning techniques are inherently black-box, meaning that users can

use the models to obtain predictions, but the models themselves provide little insight. We use a transparent technique that produces human-readable models, which indeed can provide additional insights.

7 Conclusion

In this paper, we propose a versatile methodology for understanding the performance of simulation codes in particular and scientific codes in general. It is based on a systematic workflow for producing empirical performance models. Empirical performance modeling is a proven technique for automated generation of performance models from the results of code benchmarking. Using our methodology, we generated two-parameter and three-parameter models for the execution time of *ms2*, a molecular dynamics code for studying thermodynamic properties of bulk fluids. The models provide insight on the impact of various parameters on the execution time. They also show in which situations the impact is compounded, for example, increasing both the number of interaction sites and the cut-off radius leads to a much higher increase in execution time.

Besides providing insight, the generated performance models are analytical expressions that can be used to calculate the execution time for given parameter values. In other words, the models allow us to predict the performance of the application. This capability was employed in the TaLPas project [1], which aims to provide a solution for fast and robust simulation of many, potentially dependent particle systems in a distributed environment. Specifically, performance prediction is used to support a purpose-built scheduler in the process of finding optimal execution configurations for individual simulation runs.

The study also identifies potential pitfalls in the workflow and provides suggestions for overcoming them. Specifically, we discuss the necessity of extracting a common calltree from performance profiles, and also provide guidelines for performing the benchmarking. Furthermore, we highlight the influence of various factors on the variability of the measurements and the importance of reducing it to obtain accurate models.

Acknowledgements. This work was supported by the German Research Foundation (DFG) through the Program Performance Engineering for Scientific Software and the ExtraPeak project, by the German Federal Ministry of Education and Research (BMBF) through the TaLPas project under Grant No. 01IH16008D, and by the US Department of Energy through the PRIMA-X project under Grant No. DE-SC0015524. The authors would like to thank the partners of the TaLPas project for fruitful discussions. Finally, the authors would also like express their gratitude to the High Performance Computing Center Stuttgart (HLRS) and the University Computing Center (Hochschulrechenzentrum) of Technische Universität Darmstadt for providing access to machines Hazel Hen and Lichtenberg, respectively.

References

1. BMBF project TaLPas - Task-based Load Balancing and Auto-tuning in Particle Simulations. https://wr.informatik.uni-hamburg.de/research/projects/talpas/start. Accessed 22 May 2018
2. Cube 4.x series. http://www.scalasca.org/software/cube-4.x/download.html. Accessed 22 May 2018
3. European Union's Horizon 2020 project POP - Performance Optimisation and Productivity. https://pop-coe.eu. Accessed 25 June 2018
4. Extra-P – Automated Performance-modeling Tool. http://www.scalasca.org/software/extra-p. Accessed 22 May 2018
5. Folding@home. https://foldingathome.org/. Accessed 04 July 2018
6. GROMACS: Molecular Dynamics Package. http://www.gromacs.org/. Accessed 03 July 2018
7. LAMMPS: Molecular Dynamics Simulator. http://lammps.sandia.gov/. Accessed 03 July 2018
8. NAMD: Scalable Molecular Dynamics. http://www.ks.uiuc.edu/Research/namd/. Accessed 03 July 2018
9. Berendsen, H., van der Spoel, D., van Drunen, R.: GROMACS: a message-passing parallel molecular dynamics implementation. Comput. Phys. Commun. **91**(1), 43–56 (1995). https://doi.org/10.1016/0010-4655(95)00042-E
10. Bhatele, A., Jain, N., Livnat, Y., Pascucci, V., Bremer, P.T.: Analyzing network health and congestion in dragonfly-based supercomputers. In: Proceedings of the 30th IEEE International Parallel & Distributed Processing Symposium (IPDPS), pp. 93–102. IEEE Computer Society, May 2016. https://doi.org/10.1109/IPDPS.2016.123
11. Phillips, J.C., et al.: Scalable molecular dynamics with NAMD. J. Comput. Chem. **26**(16), 1781–1802 (2005). https://doi.org/10.1002/jcc.20289
12. Calotoiu, A., et al.: Fast multi-parameter performance modeling. In: Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–10. IEEE, September 2016. https://doi.org/10.1109/CLUSTER.2016.57
13. Calotoiu, A., Hoefler, T., Poke, M., Wolf, F.: Using automated performance modeling to find scalability bugs in complex codes. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC), pp. 45:1–45:12. ACM, November 2013. https://doi.org/10.1145/2503210.2503277
14. Chunduri, S., et al.: Run-to-run Variability on Xeon Phi based cray XC systems. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC), pp. 52:1–52:13. ACM, November 2017. https://doi.org/10.1145/3126908.3126926
15. Deublein, S., et al.: *ms*2: a molecular simulation tool for thermodynamic properties. Comput. Phys. Commun. **182**(11), 2350–2367 (2011). https://doi.org/10.1016/j.cpc.2011.04.026
16. Glass, C.W., et al.: *ms*2: a molecular simulation tool for thermodynamic properties, new version release. Comput. Phys. Commun. **185**(12), 3302–3306 (2014). https://doi.org/10.1016/j.cpc.2014.07.012
17. Horsch, M., Niethammer, C., Vrabec, J., Hasse, H.: Computational molecular engineering as an emerging technology in process engineering. Methods Appl. Inform. Inf. Technol. **55**(3), 97–101 (2013). https://doi.org/10.1524/itit.2013.0013
18. Iwainsky, C., et al.: How many threads will be too many? On the scalability of OpenMP implementations. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015. LNCS, vol. 9233, pp. 451–463. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_35

19. Kale, L.V., Krishnan, S.: CHARM++: a portable concurrent object oriented system based on C++. In: Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pp. 91–108. ACM (1993). https://doi.org/10.1145/165854.165874
20. Knüpfer, A., et al.: Score-P - a joint performance measurement run-time infrastructure for periscope, scalasca, TAU, and vampir. In: Brunst, H., Müller, M., Nagel, W., Resch, M. (eds.) Tools for High Performance Computing, pp. 79–91. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-31476-6_7
21. Marathe, A., et al.: Performance modeling under resource constraints using deep transfer learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2017, pp. 31:1–31:12. ACM (2017). https://doi.org/10.1145/3126908.3126969
22. Plimpton, S.: Fast parallel algorithms for short-range molecular dynamics. J. Comput. Phys. **117**(1), 1–19 (1995). https://doi.org/10.1006/jcph.1995.1039
23. Rutkai, G., et al.: *ms*2: a molecular simulation tool for thermodynamic properties, release 3.0. Comput. Phys. Commun. **221**, 343–351 (2017). https://doi.org/10.1016/j.cpc.2017.07.025
24. Shudler, S., Calotoiu, A., Hoefler, T., Strube, A., Wolf, F.: Exascaling your library: will your implementation meet your expectations? In: Proceedings of the 29th ACM International Conference on Supercomputing (ICS), pp. 165–175. ACM, June 2015. https://doi.org/10.1145/2751205.2751216
25. Shudler, S., Calotoiu, A., Hoefler, T., Wolf, F.: Isoefficiency in practice: configuring and understanding the performance of task-based applications. In: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 131–143. ACM, February 2017. https://doi.org/10.1145/3018743.3018770
26. Singh, K., İpek, E., McKee, S.A., de Supinski, B.R., Schulz, M., Caruana, R.: Predicting parallel application performance via machine learning approaches: research articles. Concurr. Comput.: Pract. Exper. **19**(17), 2219–2235 (2007). https://doi.org/10.1002/cpe.1171
27. Vogel, A., et al.: 10,000 performance models per minute – scalability of the UG4 simulation framework. In: Träff, J.L., Hunold, S., Versaci, F. (eds.) Euro-Par 2015. LNCS, vol. 9233, pp. 519–531. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48096-0_40
28. Yang, X., Jenkins, J., Mubarak, M., Ross, R.B., Lan, Z.: Watch out for the bully!: job interference study on dragonfly network. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC), pp. 64:1–64:11. IEEE Press (2016). https://doi.org/10.1109/SC.2016.63