JCTC Journal of Chemical Theory and Computation

Article

Cite This: J. Chem. Theory Comput. 2019, 15, 1827−1840

pubs.acs.org/JCTC

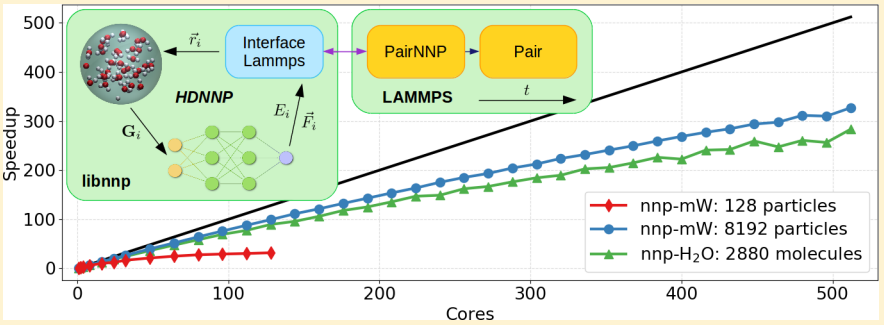
Library-Based *LAMMPS* Implementation of High-Dimensional Neural Network Potentials

Andreas Singraber,*,† Jörg Behler,‡ and Christoph Dellago*,†

†Faculty of Physics, University of Vienna, Boltzmanngasse 5, 1090 Vienna, Austria

‡Universität Göttingen, Institut für Physikalische Chemie, Theoretische Chemie, Tammannstraße 6, 37077 Göttingen, Germany

Supporting Information



ABSTRACT: Neural networks and other machine learning approaches have been successfully used to accurately represent atomic interaction potentials derived from computationally demanding electronic structure calculations. Due to their low computational cost, such representations open the possibility for large scale reactive molecular dynamics simulations of processes with bonding situations that cannot be described accurately with traditional empirical force fields. Here, we present a library of functions developed for the implementation of neural network potentials. Written in C++, this library incorporates several strategies resulting in a very high efficiency of neural network potential-energy and force evaluations. Based on this library, we have developed an implementation of the neural network potential within the molecular dynamics package *LAMMPS* and demonstrate its performance using liquid water as a test system.

1. INTRODUCTION

Over the past decades, molecular dynamics (MD) simulations have become a central tool for understanding how the properties of materials result from the interactions and dynamics of atoms. Ideally, the forces needed in MD simulations to integrate the equations of motion are determined by solving the electronic Schrödinger equation of the electrons in the field of the nuclei. Such an *ab initio* approach, however, is computationally very costly, and consequently forces are often computed from simple empirical potentials postulated based on physical considerations with parameters tuned to reproduce key properties of the material under study. While empirical potentials are available for many types of systems and can dramatically extend the time and length scales accessible to MD simulations, they may introduce systematic inaccuracies as a result of the approximate functional form. Complex chemical environments or the dynamic formation and breaking of covalent bonds are in particular difficult to model within this framework. In recent years, machine learning potentials[1] have become a new promising tool to bridge the gap between accurate and flexible but computationally expensive electronic structure methods and approximate but computationally convenient empirical force fields. Specifically, machine learning potentials have been used to predict physical properties of inorganic solids[2] or complex molecular systems with ab initio accuracy[3,4] and to model chemical dynamics, such as proton transfer[5] or dissociative chemisorption.[6,7] The general idea of this approach is to use statistical learning techniques to devise potentials with a very flexible functional form with many parameters and to train them using ab initio reference data. While some recent machine learning potentials such as the Gaussian approximation potential (GAP),[8,9] Coulomb-matrix-based methods,[10] or the spectral neighbor analysis potential (SNAP)[11] are based on kernel methods, artificial neural networks have been used for a longer time, as first applications date back to 1995[12] (see refs 13 and 14 for a comprehensive list of applications). In an approach proposed in 2007 by Behler and Parrinello,[15] the potential energy surface is represented using high-dimensional neural networks, which are trained to reproduce complex *ab initio* reference potential energy surfaces and allowed to perform simulations with large numbers of atoms. While during the training process the expensive calculation of energies and forces from the reference method is required, the final neural network potential (NNP) can predict the training data points and interpolate in between

Received: July 25, 2018
Published: January 24, 2019

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

them at a fraction of the computational cost. This ability can be used to drive a fast MD simulation with *ab initio* accuracy.

From a practical point of view, the NNP approach requires software to accomplish two central tasks. First, a training program is needed that adjusts the neural network parameters in such a way that the reference data is correctly reproduced. Once the training is done and the parametrization is ready, a computer code is needed to execute the neural networks and calculate the forces needed in the MD simulation. While in principle these two pieces of code can be developed independently from each other, this would possibly result in many lines of duplicated code and additional maintenance work. Thus, it is beneficial to pursue a library-based programming paradigm such that common functionalities and data structures can be collected and combined as needed by different applications. Following this approach, we have developed a C++ library for high-dimensional neural network potentials (HDNNPs). The present work focuses on the basic building blocks of the library and how they are implemented in order to optimize performance. In addition to the NNP core library we present its first application, an implementation of the method in the popular MD package *LAMMPS*,[16,17] and provide benchmark results for several systems simulated on a modern HPC cluster. Although similar implementations of the method by Behler and Parrinello written in Fortran,[18−21] Python/Fortran,[22,23] C++,[24,25] or as an extension to Tensorflow[26−29] have been developed recently, a clear illustration of the expected performance when applying the method is generally not provided. However, for a prospective user it is of significant importance to estimate the accessible time scales. Hence, in this work we demonstrate for typical system sizes and HDNNP parametrizations, but also for different levels of optimization and parallelization, how fast MD simulations can be run in practice. Unfortunately, a comparison of all available software is currently unfeasible as the implementations often differ in available features and technical details.

The C++ library and the LAMMPS interface are designed to work in conjunction with existing parametrizations[3,5,7,30−39] created with the *RuNNer* software.[21] Further applications based on the presented library, such as an HDNNP training program, are also available and will be described in an upcoming publication.[40] The NNP library and the *LAMMPS* implementation are available freely online as part of the *n2p2* neural network potential package at http://github.com/CompPhysVienna/n2p2.

The remaining article is organized as follows: in Section 2 the neural network potential method is shortly outlined. Section 3 focuses on the implementation and features of the NNP library as well as the *LAMMPS* interface. Finally, we present benchmarks for multiple test cases in Section 4 before we offer some conclusions in Section 5.

2. THE HIGH-DIMENSIONAL NEURAL NETWORK POTENTIAL METHOD

This section is intended as a concise summary of the HDNNP method and does not provide a detailed introduction, for which we refer the reader to refs 41 and 42.

2.1. Atomic Neural Networks. The high-dimensional neural network potential method introduced by Behler and Parrinello[15] is based on the assumption that the total potential energy E^{pot} can be written as the sum of individual atomic contributions

$$E^{\mathrm{pot}} = \sum_{j=1}^{N^a} E_j(\mathbf{G}_j) \tag{1}$$

where N^a is the number of atoms. The atomic energies E_j, which depend on the local environment of the atoms, are calculated from feed-forward neural networks as depicted in Figure 1.

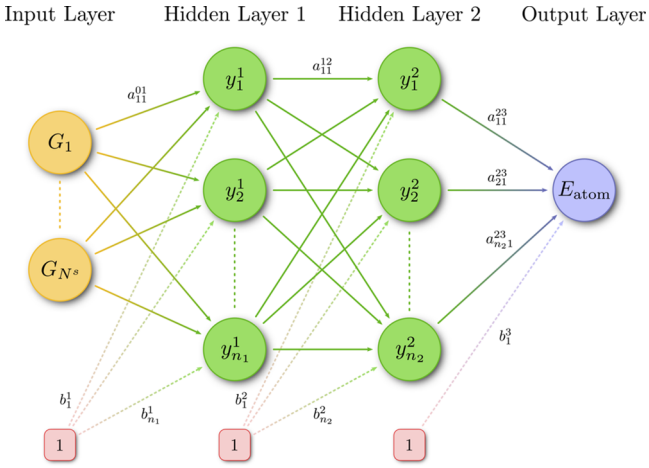


Figure 1. A simple feed-forward neural network with two hidden layers. Information about each atom's surroundings is entered in the input layer neurons on the very left. The neural network processes the data by successively computing the neurons in the hidden layers and presents the resulting atomic energy contribution in the single output layer neuron. Note that here we omitted the atom index j of the input and output layer label.

These networks consist of multiple layers of nodes, also called neurons, which are interconnected by weights representing the fitting parameters of the neural network. The neurons of the input layer of the neural networks are populated with sets of real numbers $\mathbf{G}_j = \{G_{j,1},...,G_{j,N_j^c}\}$, called symmetry functions,[42] that characterize the local environment of atom j (see Section 2.2). The neural network then propagates the data to neurons in the hidden layers with the simple rule

$$y_i^k = f_a\left(b_i^k + \sum_{j=1}^{n_l} a_{ji}^{lk} y_j^l\right) \tag{2}$$

where y_i^k denotes the value of neuron i in layer k, b_i^k is the associated bias, a_{ji}^{lk} is the weight connecting y_i^k with y_j^l, and n_l is the number of neurons in layer $l = k-1$. Thus, neuron values are computed from a weighted sum of all incoming connected neurons in the previous layer. In addition the bias is added to allow for a shift independent from input neurons. The activation function f_a is usually a nonlinear function (e.g., a hyperbolic tangent or the logistic function), which enables the neural network as a whole to reproduce complex nonlinear potential energy surfaces after weights and biases have been correctly adapted in a training stage. After processing the input data through the hidden layers, the recursion of eq 2 will ultimately result in the atomic energy contribution presented as the value of the single output neuron. Note that separate neural networks and typically also separate symmetry function setups are used to account for different chemical elements. For a given chemical element the symmetry function parameters,

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

the weight parameter values, and the neural network architecture are constrained to be identical. This construction ensures permutation symmetry because atoms of the same chemical element are treated equally.

To obtain analytical expressions for the atomic forces, one simply applies the chain rule to the total potential energy expressed in eq 1

$$\vec{F}_i = -\vec{\nabla}_i E^{\mathrm{pot}} = -\sum_{j=1}^{N^a} \vec{\nabla}_i E_j = -\sum_{j=1}^{N^a}\sum_{k=1}^{N_j^s} \frac{\partial E_j}{\partial G_{j,k}} \vec{\nabla}_i G_{j,k} \tag{3}$$

where $\vec{\nabla}_i$ is the gradient with respect to the Cartesian coordinates of atom i. Note that the individual atomic energy contributions E_j do not have any physical meaning attached to them as they are not observables but merely a vehicle of the method to construct the total potential energy. In particular, only E^{pot} is typically known from a reference method and can therefore be used during training. However, splitting up the total energy as shown in eq 1 comes with a major benefit: scalability. A neural network potential trained for small system sizes is capable of predicting the potential energy surface for larger configurations by simply adding new summands in eq 1, as long as the atomic environments are properly included in the training set. Training a neural network potential, i.e., optimizing the weight and bias parameters, requires a data set of structures and corresponding potential energies, as well as optionally the forces, computed using the reference method of choice (for instance density functional theory). The procedure to generate appropriate training sets and the optimization algorithm itself are out of scope of this work and have been described elsewhere.[41,43]

2.2. Symmetry Functions. To provide the atomic neural networks with information about each atom's surroundings, sets of atom-centered symmetry functions (ACSFs)[42] are computed from the Cartesian coordinates of the atoms. Here, the term "symmetry" refers to the invariance of these functions with respect to global translation, rotation, and index permutation. The use of such symmetry functions ensures that by construction also the total energy is invariant with respect to these operations as required. A variety of symmetry functions have been proposed in the past for the prediction of potential energy surfaces[42,44] or structure detection.[45] Note that symmetry functions are only one set of possible descriptors of the local atomic environment used in machine learning approaches.[1,9,10,19,46]

Symmetry functions can be organized in two groups: Radial symmetry functions are computed only from distances to neighbor atoms, whereas angular symmetry functions also involve angles between triplets of atoms (one central and two neighboring atoms). However, we note that in general symmetry functions for high-dimensional NNPs are many-body functions depending simultaneously on a large number of degrees of freedom in the atomic environments. Here, we recapitulate three frequently used basic types of symmetry functions, one radial variant (G^{radial}) and two angular variants ($G^{\mathrm{ang.n.}}$ and $G^{\mathrm{ang.w.}}$) proposed by Behler in 2011.[42]

The calculation of the radial symmetry function G_i^{radial} for a given atom i involves summing over all neighboring atoms

$$G_i^{\mathrm{radial}} = \sum_{j\neq i} \mathrm{e}^{-\eta(r_{ij}-r_s)^2} f_c(r_{ij}) \tag{4}$$

where $r_{ij} = |\vec{r}_i - \vec{r}_j|$ is the distance between atoms i and j, $f_c(r)$ is a cutoff function, and η and r_s are free parameters determining the spatial shape of the symmetry function. The functional form of angular symmetry functions is similar but requires a double sum over all neighbors with corresponding exponential and cutoff function terms, and the angle $\theta_{ijk} = \angle(j,i,k)$ enters as central term to distinguish different angular environments. The angular symmetry function comes in two flavors, the "narrow" version is defined as

$$G_i^{\mathrm{ang.n.}} = 2^{1-\zeta} \sum_{\substack{j,k\neq i \\ j<k}} (1 + \lambda \cos\theta_{ijk})^{\zeta} \mathrm{e}^{-\eta(r_{ij}^2+r_{ik}^2+r_{jk}^2)} f_c(r_{ij}) f_c(r_{ik}) f_c(r_{jk}) \tag{5}$$

whereas the "wide" variant

$$G_i^{\mathrm{ang.w.}} = 2^{1-\zeta} \sum_{\substack{j,k\neq i \\ j<k}} (1 + \lambda \cos\theta_{ijk})^{\zeta} \mathrm{e}^{-\eta(r_{ij}^2+r_{ik}^2)} f_c(r_{ij}) f_c(r_{ik}) \tag{6}$$

omits the factor $\mathrm{e}^{-\eta r_{jk}^2} f_c(r_{jk})$ and thus effectively reduces the damping for large angles and neighbor atoms far from each other but not from the central atom. Both angular symmetry functions depend on the free parameters η, ζ, and λ.

As apparent in eq 3, calculating atomic forces from the neural network potential energy surface requires computation of the gradient of symmetry functions with respect to the Cartesian coordinates. Analytic expressions of symmetry function gradients are presented in the Supporting Information. The cutoff functions f_c mentioned above ensure a smooth decay of symmetry functions and their derivatives when neighbor atoms reach the cutoff radius r_c. Different functional forms have been proposed in the literature, e.g. involving the cosine[15]

$$f_c(r) = \begin{cases} \frac{1}{2}\left[\cos\left(\frac{\pi r_{ij}}{r_c}\right) + 1\right], & \text{for } r \le r_c \\ 0, & \text{for } r > r_c \end{cases} \tag{7}$$

or the hyperbolic tangent[42]

$$f_c(r) = \begin{cases} \tanh^3\left(1 - \frac{r_{ij}}{r_c}\right), & \text{for } r \le r_c \\ 0, & \text{for } r > r_c \end{cases} \tag{8}$$

See Section 3.3 for additional choices based on polynomials.

3. IMPLEMENTATION

In general, extending an existing MD software with a new interaction type can be accomplished following two different strategies. First, one could implement all new functionality within the given framework, i.e. add new functions or classes to the code which calculate energies and forces for this particular interaction. In this way no computational overhead and no new dependencies will be added to the MD software. For interactions represented by simple functional forms with a small number of tunable parameters (e.g., pairwise potentials) this is the obvious design choice. For technically more complex interactions a library approach seems preferable. Here, all

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

functions, data types, and variables related to the interaction are gathered in an external library, which provides an interface for the MD program to pass on required information, e.g. atomic positions or neighbor lists. Energies and forces are then computed within the library and returned back to the calling MD code. While this scheme introduces some overhead, it also comes with major advantages. The functionality introduced by the library is not restricted to a single MD software anymore but may be interfaced by multiple codes. The library as common code basis then ensures that changes made to internal routines (e.g., bug fixes, performance improvements, etc.) are automatically carried over to all interfacing programs. With numerous required computational steps (e.g., symmetry function calculation, neural network execution) and parameters (e.g., symmetry function types and parameters, neural network weights and topology) the neural network potential method clearly qualifies as a complex interaction, and a library strategy is suitable. The following sections introduce our NNP library implementation and present the *LAMMPS* interface.

3.1. NNP Library. The neural network potential library (libnnp) constitutes the foundation of our implementation. It is written in C++ adhering to the C++98 standard and utilizes features from the Standard Template Library (STL). Also, there are no dependencies to third-party libraries which simplifies its portability to different environments. The library provides classes supporting documentation generation from source code via doxygen.[47] They can be easily extended, or new classes can be derived to introduce additional functionality in the future. Although all classes are bundled together in the NNP library, it is still helpful to imagine them as part of two groups with different purpose.

First, some classes are intended for storing the data of the atomic configurations:

- `Structure` class: Holds information for one configuration, i.e., contains the geometry of the periodic box and the positions of all atoms (a vector of type `Atom`) as well as member functions to compute neighbor lists.
- `Atom` class: Stores the coordinates of an individual atom, its neighbor list (a vector of type `Atom::Neighbor`), and temporary results from symmetry function calculation.
- `Atom::Neighbor` class: Within the scope of the `Atom` class this nested class contains the relative location of an atom's neighbor, temporary symmetry function results, and the cutoff function cache (see Section 3.3).

The other group of classes organizes the neural network potential calculation depending on the large variety of possible user settings. The general layout of these classes is schematically shown in Figure 2.

- `Mode` class: The top-level class of the NNP library containing the complete information required to calculate energies and forces for configurations of type `Structure`. Existing and future applications should use its interface or inherit from here.
- `Element` class: Since the HDNNP method allows different neural network topologies and symmetry function setups for each element, all per-element components are grouped within this class. The `Mode` class contains as many instances of `Element` as there are elements in the system.

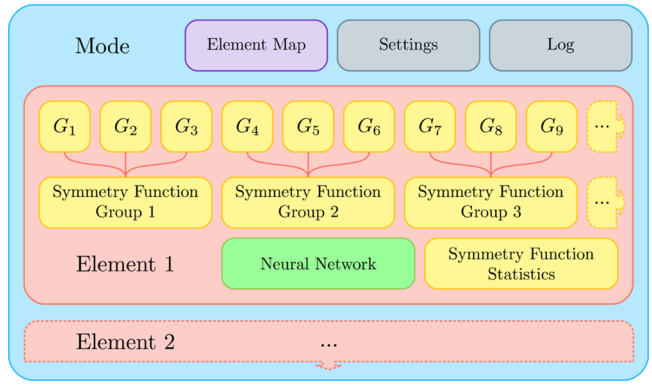


Figure 2. Top-level class layout of the neural network potential library `libnnp`: The `Mode` class contains all parts required to calculate the HDNNP energy and forces for a given configuration. Neural networks and symmetry functions (boxes labeled with G_i) are gathered per element. Symmetry functions with similar parameters are collected in symmetry function groups.

- `NeuralNetwork` class: A simple implementation of a feed-forward neural network with selectable number of hidden layers and nodes per hidden layer. Linear, hyperbolic tangent, logistic, and softplus activation functions are available.
- `SymmetryFunction` class: The base class for all symmetry functions providing common member variables and functions. Actual symmetry function implementations are inherited from here. New symmetry functions can be easily added by copying, renaming, and modifying existing child classes.
- `SymmetryFunctionGroup` class: Similar to `SymmetryFunction` this is the base class for all symmetry function groups (see Section 3.2).
- `CutoffFunction` class: A collection of different cutoff functions.
- `SymmetryFunctionStatistics` class: This class collects statistics of symmetry functions during runtime.
- `Settings` class: A map collecting all user-provided settings in the form of keyword-string pairs.
- `Log` class: All NNP library output intended for the computer screen is passed on to this simple logging class. From there it is possible to redirect all information to multiple targets (screen, C-style file pointers, or C++ file streams) or even completely silence the output.
- `ElementMap` class: This class provides a convenient mapping from atomic numbers to element strings and vice versa for all used atom types. Often required to construct other classes.

The NNP library features some basic thread-level parallelization via OpenMP in the `Mode` and `Structure` classes. Their respective member functions for neighbor list, symmetry function, and force calculation automatically distribute the workload when running on a multicore processor. Although OpenMP support is only provided for a few functions in the library, these cover the most time-consuming computations. Hence, applications will benefit immediately from this simple form of parallelization.

3.2. Symmetry Function Groups. The procedure to compute the HDNNP energy for a given atomic configuration is easily outlined: the outermost loop is running over all atoms

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

in the structure, as required by eq 1. For each atom, the symmetry function set is calculated and subsequently used as an input for the corresponding neural network. The atomic energy contributions delivered by the neural networks are then summed up to yield the total potential energy. Computing analytic forces complicates this procedure slightly, but the computation of symmetry functions and the evaluation of the neural network remain as the two main computational building blocks. It turns out that the calculation of symmetry functions for a given atom is usually by far more time-consuming than the evaluation of the attached neural network. In particular, HDNNP setups with numerous angular symmetry functions are computationally dominated by the double loop structure in eqs 5 and 6. Naturally, in our effort to create an efficient HDNNP implementation we focused our optimizations on the corresponding parts of the code. Taking into consideration the symmetry function setup of existing HDNNPs,[3,7,33,36,38,48−50] possible strategies to restructure naive loop implementations and utilize temporary memory for intermediate results, we decided for a simple but effective method to reduce the computational cost via *symmetry function groups*. Here, we will now outline their construction and explain how performance benefits from their usage.

The basic idea is to avoid unnecessary loops over the same neighbor lists for symmetry functions with different parameters. As an example, consider a partial list (Table 1) of angular symmetry functions for oxygen atoms used in a NNP developed for water and ice.[3]

Table 1. Oxygen Angular Symmetry Function Parameters e_1, e_2, η/bohr^{-2}, λ, ζ, and r_c/bohr for the Bulk Water Model[3a]

no.	e_1	e_2	η	λ	ζ	r_c
17	H	O	0.001	−1	4.0	12.0
18	O	O	0.001	−1	4.0	12.0
19	H	O	0.001	1	4.0	12.0
20	O	O	0.001	1	4.0	12.0
21	H	H	0.010	−1	4.0	12.0
22	H	H	0.010	1	4.0	12.0
23	H	H	0.030	−1	1.0	12.0
24	H	O	0.030	−1	1.0	12.0
25	O	O	0.030	−1	1.0	12.0
26	H	H	0.030	1	1.0	12.0
27	H	O	0.030	1	1.0	12.0
28	O	O	0.030	1	1.0	12.0
29	H	H	0.070	−1	1.0	12.0
30	H	H	0.070	1	1.0	12.0

[a] e_1 and e_2 denote the chemical element of neighbor atoms, e.g., if e_1 = e_2 = H, the symmetry function is applied to triplets with central O atom and two neighboring H atoms.

It enumerates different neighbor atom combinations and parameters η, λ, ζ, and r_c. With the functional form in eq 5 in mind, two observations can be exploited for performance gains. First, all symmetry functions share the same cutoff r_c and thus the same list of neighboring atoms. It would be inefficient to calculate the cutoff function values $f_c(r_{ij})$, $f_c(r_{ik})$, and $f_c(r_{jk})$ as well as their derivatives repeatedly for each symmetry function. Instead, it is beneficial to reorder the lines in Table 1 and group entries together according to their neighbor species signature and cutoff value. Each of the (e_1,e_2,r_c)-combinations now forms a symmetry function group whose members correspond to the remaining parameter variations (here η, λ, and ζ). Table 2 shows the arrangement of the example symmetry functions in three groups.

Table 2. Oxygen Angular Symmetry Function Groups Formed from Table 1[a]

e_1	e_2	η	λ	ζ	r_c	no.	exp
H	H				12.0		
		0.010	−1	4.0		21	yes
		0.010	1	4.0		22	no
		0.030	−1	1.0		23	yes
		0.030	1	1.0		26	no
		0.070	−1	1.0		29	yes
		0.070	1	1.0		30	no
H	O				12.0		
		0.001	−1	4.0		17	yes
		0.001	1	4.0		19	no
		0.030	−1	1.0		24	yes
		0.030	1	1.0		27	no
O	O				12.0		
		0.001	−1	4.0		18	yes
		0.001	1	4.0		20	no
		0.030	−1	1.0		25	yes
		0.030	1	1.0		28	no

[a] The three blocks with (e_1,e_2,r_c)-entries correspond to the three symmetry function groups. The following lines list the parameters of member symmetry functions. The column "exp" denotes whether the exponential factor (see text) needs to be recalculated.

The procedure to compute HDNNP energies and forces described at the beginning of this section is now modified as follows: while the outermost loop is still running over all atoms, the symmetry function loop is replaced by a loop over symmetry function groups. For each group, the correct neighbors are extracted from the neighbor list, and the corresponding cutoff function terms are calculated. Only then, in the innermost loop, the distinct parts of individual symmetry functions, i.e., the terms $(1 + \lambda \cos \theta_{ijk})^{\zeta}$ and $e^{-\eta(r_{ij}^2+r_{ik}^2+r_{jk}^2)}$, are computed. At second glance another optimization step for angular symmetry functions is possible since it is common that multiple symmetry functions with equal η but different λ and ζ parameters are included. Thus, once calculated for a given η, the term $e^{-\eta(r_{ij}^2+r_{ik}^2+r_{jk}^2)}$ can be reused for multiple symmetry functions.

Grouping of symmetry functions works similarly for eqs 4 and 6. The NNP library implements all three basic symmetry function groups and uses them per default. Note that the concept behind the optimizations described in this section is likely to be applicable also to future symmetry functions or other descriptors of local atomic environment since common features will usually imply more efficient computation pathways than the sequential calculation.

3.3. Cutoff Functions and Caching. As described in Section 3.2, in a naive implementation of an NNP cutoff functions for identical distances are repeatedly calculated for each symmetry function. The introduction of symmetry function groups reduces the number of cutoff function calls significantly since temporary results are reused to compute multiple symmetry functions in a row. Still, different groups with equal cutoff function radius r_c may loop over similar neighbor combinations and thus require the re-evaluation of cutoff functions. This computational overhead is avoided by

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

the NNP library because the `Atom::Neighbor` class is equipped with additional storage for cutoff function values and their derivatives. During the symmetry function calculation the first symmetry function group encountering a particular "central to neighbor atom situation" will calculate f_c and $\frac{df_c}{dr}$ and store the results. Other groups will then later retrieve the values from the cache instead of recalculating them.

The NNP library implements several cutoff functions, most of which (except the hyperbolic tangent in eq 11) also support the definition of an inner cutoff radius $r_{ci} := \alpha r_c$ via a parameter $\alpha \in [0,1)$. The cutoff function returns 1 up to the inner cutoff and then falls off to 0 up to the actual cutoff r_c with the desired functional form

$$f_c(r) = \begin{cases} 1, & \text{for } 0 \le r < r_{ci} \\ f(x), & \text{for } r_{ci} \le r < r_c \\ 0 & \text{for } r \ge r_c \end{cases} \tag{9}$$

where $x := \frac{r - r_{ci}}{r_c - r_{ci}}$. Among the implemented cutoff functions some are based on transcendental functions

$$f^{\cos}(x) = \frac{1}{2}[\cos(\pi x) + 1] \tag{10}$$

$$f_c^{\tanh}(r) = \tanh^3(1 - r/r_c) \tag{11}$$

$$f^{\exp}(x) = \exp\left\{1 - \frac{1}{1 - x^2}\right\} \tag{12}$$

where $f^{\cos}(x)$ and $f_c^{\tanh}(r)$ have been successfully used to parametrize HDNNPs.[3,7,31,33,39] Note that for $f_c^{\tanh}(r)$ eq 9 does not apply as at $r = 0$ it starts off from a value not equal to 1 and also shows a nonzero derivative (see Figure 3).

Nevertheless, $f_c^{\tanh}(r)$ is a valid symmetry function cutoff choice with a continuous second derivative at the cutoff radius. This may be an important feature when calculating properties from MD simulations which are dependent on higher order derivatives of the potential energy.[51] The third cutoff function $f^{\exp}(x)$ provides even infinite continuous derivatives at the cutoff radius. Because the above cutoff functions involve the time-consuming computation of transcendental functions, we propose and encourage the use of alternatives based on simple polynomials:

$$f^{\text{poly1}}(x) = x^2(2x - 3) + 1 \tag{13}$$

$$f^{\text{poly2}}(x) = x^3(x(15 - 6x) - 10) + 1 \tag{14}$$

$$f^{\text{poly3}}(x) = x^4(x(x(20x - 70) + 84) - 35) + 1 \tag{15}$$

$$f^{\text{poly4}}(x) = x^5(x(x(x(315 - 70x) - 540) + 420) - 126) + 1 \tag{16}$$

The coefficients of the cutoff functions $f^{\text{poly}n}$ are chosen in such a way that n defines the order of the highest derivative which goes to zero at the boundaries, i.e.

$$\frac{d^k f^{\text{poly}n}}{dx^k}(0) = \frac{d^k f^{\text{poly}n}}{dx^k}(1) = 0 \qquad \forall\ 0 < k \le n \tag{17}$$

For instance, $f^{\text{poly2}}(x)$ may be used instead of $f_c^{\tanh}(r)$ for future HDNNP generation as both feature a continuous second derivative, but $f^{\text{poly2}}(x)$ is generally faster. Figure 3 shows the shape of cutoff functions and their derivatives.

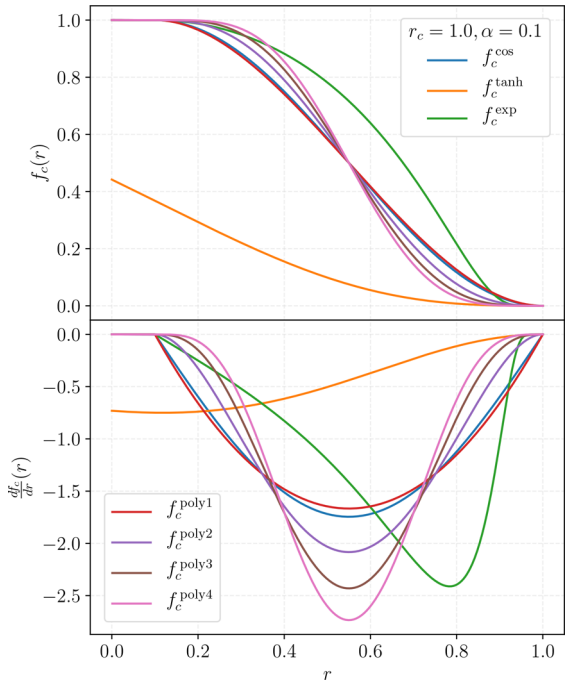


Figure 3. Cutoff functions (top) and their derivatives (bottom) implemented in the NNP library. For this visualization $r_c = 1.0$, and the shift parameter $\alpha = 0.1$. Note that $f_c^{\tanh}$ cannot be shifted as it sets off at $r = 0$ with a value not equal to 1.

3.4. *LAMMPS* Interface.

After having presented the structure and capabilities of our NNP library in the preceding sections, we will now move on with the details of the *LAMMPS* implementation. *LAMMPS* is a widely used molecular dynamics software, which supports numerous potentials and simulation techniques. The documentation is comprehensive, it uses a spatial decomposition algorithm together with message passing interface (MPI) for parallelization, and the modular design allows for an easy extension with new features. For these reasons, *LAMMPS* is an ideal framework for the implementation of the neural network potential method.

The *LAMMPS* developer guide[17] provides a starting point for any extension and describes the code design and top-level classes. The authors state that nonbonded interactions are implemented as subclasses of the `Pair` base class. The name does not imply a restriction to pairwise interactions; also many-body potentials are included. Thus, we introduce the derived class `PairNNP` which makes the neural network functionality available in the *LAMMPS* scripting language via the keywords `pair_style nnp`. As mentioned in the developer guide, during runtime at each time step the `PairNNP::compute` method is called, and lists of atoms and their neighbors are provided. If the simulation is run in parallel, each MPI process is handling only a group of "local" atoms for which the contributions to potential energy and atomic forces are calculated. They are accompanied by surrounding "ghost" atoms which are required for the interaction computation but are local to other processes. At each time step the information about per-process available atomic configurations needs to be rearranged into data structures compatible with the NNP library. The atomic energy and force computation is then performed internally by the NNP library, and the results are returned back to the *LAMMPS* `PairNNP` class. Technically this scheme is

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

implemented as follows: the class `InterfaceLammps` is derived from the NNP top-level class `Mode` described in Section 3.1, and an instance is added as a member of `PairNNP`. In contrast to its base class, `Interface-Lammps` contains an additional private `Structure` member and provides public functions to transfer atomic configurations and neighbor information from *LAMMPS* to this internal data storage. Once the data has been transferred, symmetry functions and atomic networks are computed via methods inherited from the `Mode` class. Finally the atomic energies and forces from local atoms are reported back to *LAMMPS*. See Figure 4 for a schematic representation of the class inheritance.

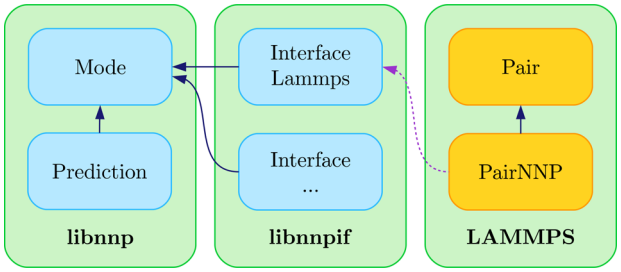


Figure 4. Schematic overview of class inheritance for the *LAMMPS* interface. Blue and orange boxes represent classes being parts of the NNP library and *LAMMPS*, respectively. Dark blue lines with arrows point from inherited to base classes. Here, multiple classes derived from `Mode` are shown. For example, `Prediction` is the basis of a standalone tool to predict energies and forces for a single configuration. The purple dotted line indicates that `PairNNP` instantiates `InterfaceLAMMPS`.

Special care has to be taken to correctly update the *LAMMPS* force arrays. Contributions of the form $\vec{\nabla}_i E_j$ (see eq 3), where i is a ghost atom and j is a local atom, must not be communicated and summed up during the interaction computation. Instead they are stored in force arrays of ghost atoms and later processed in a separate *LAMMPS* communication step. This ensures that the virial is correctly calculated by *LAMMPS*.[52]

4. BENCHMARKS

In the preceding sections we described the building blocks of the NNP library and explained how they are combined in order to implement the method in the molecular dynamics package *LAMMPS*. Here, we proceed with an overview of the efficiency obtained for our NNP implementation in a modern high-performance computing (HPC) environment. To provide a useful performance measurement we decided not to benchmark individual parts of the library but rather collect timing results for real-world application of different NN potentials in MD simulations with *LAMMPS* (version 16Mar18).

All calculations discussed below were performed on the Vienna Scientific Cluster (VSC), an HPC system equipped with dual-processor nodes (2x Intel Xeon E5-2650v2, 2.6 GHz, 8 cores per processor) interconnected via the Intel QDR-80 InfiniBand fabric. All source code was compiled with the Intel C++ compiler and linked to the Intel MPI library, both part of the Intel Parallel Studio XE 2017 Update 7. Compilation was performed with a small set of optimization flags (`-O3 -xHost -ipo`).

4.1. Neural Network Parametrizations.

Two neural network potentials were used for benchmarking, both modeling water and ice but with different number of atomic species. The first one is a model for liquid water and multiple ice phases[3] reproducing the potential energy surface of density functional theory with the RPBE functional[53] including van der Waals corrections according to the D3 method.[54] This NNP has been used to investigate the importance of van der Waals corrections to explain anomalous properties of water[3] and for the determination of the line of maximum density at negative pressures.[55] This model was trained with the *RuNNer* software[21] and will be abbreviated as "nnp-H_2O" in the remaining work. The other neural network potential ("nnp-mW") was developed purely for benchmarking purposes as it is not based on a computationally expensive *ab initio* method but rather on the coarse-grained monatomic model for water developed by Molinero and Moore,[56] also known as mW water. In this approach, water molecules are represented by single particles favoring tetrahedral coordination structures via the Stillinger-Weber interaction,[57] which includes terms depending on the angle between triplets of atoms. This system is ideal for our testing purposes, because it requires the use of angular symmetry functions to properly describe the atomic environments just like in the case of complex *ab initio* potential energy surfaces. Moreover, the mW model is implemented in *LAMMPS*, which makes it very easy to generate reference data for the training procedure. Note that naturally the applicability of this neural network potential is limited to benchmarking since the performance of the mW implementation exceeds that of the neural network representation by far due to its simpler functional form. However, as an effective one-component system the mW model is suitable for our purpose here, which is to demonstrate the achievable performance for single-component systems (in contrast to the nnp-H_2O model with a more complex symmetry function setup for two chemical elements).

General settings for both neural network potentials are compared in Table 3, and symmetry function parameters for nnp-H_2O, previously reported in ref 3, and resulting symmetry function group definitions are provided as Supporting Information. Note that since in the mW model water molecules are represented by single particles, only one corresponding "atomic" neural network for this atom type is required.

Table 3. Comparison of Neural Network Potential Settings for the Two Water Models[a]

setting	nnp-H_2O	nnp-mW
hidden layers	2	2
neurons per hidden layer	25	10
weight parameters	1376/1451 (H/O)	451
activation function	$\tanh(x)$	$\ln(1+e^x)$
symmetry function types	$G^{\text{radial}}/G^{\text{ang.n.}}$	$G^{\text{radial}}/G^{\text{ang.n.}}/G^{\text{ang.w.}}$
cutoff function	$f_c^{\tanh}$	$f^{\cos}$, $\alpha = 0$
cutoff radius r_c/bohr	12.0	12.0
symmetry functions	27/30 (H/O)	32

[a]Although two hidden layers are used, in both cases the number of hidden layer nodes is much smaller for the nnp-mW model since the complexity of the potential energy surface is expected to be much lower than for the nnp-H_2O model based on DFT data.

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

For the mW model we chose a mixture of all three symmetry function types from Section 2.2 with parameters chosen along the guidelines given in ref 42 and listed in Table 4. Note that

Table 4. Symmetry Function Parameters for the nnp-mW Model[a]

no.	η	r_s	λ	ζ	r_c
		G^{radial}			
1	0.001	0.0			12.0
2	0.010	0.0			12.0
3	0.030	0.0			12.0
4	0.060	0.0			12.0
5	0.150	4.0			12.0
6	0.300	4.0			12.0
7	0.600	4.0			12.0
8	1.500	4.0			12.0
		$G^{\text{ang.n.}}$			
9	0.010		−1	1.0	12.0
10	0.010		1	1.0	12.0
11	0.010		−1	3.0	12.0
12	0.010		1	3.0	12.0
13	0.030		−1	1.0	12.0
14	0.030		1	1.0	12.0
15	0.030		−1	3.0	12.0
16	0.030		1	3.0	12.0
17	0.060		−1	1.0	12.0
18	0.060		1	1.0	12.0
19	0.060		−1	3.0	12.0
20	0.060		1	3.0	12.0
		$G^{\text{ang.w.}}$			
21	0.010		−1	1.0	12.0
22	0.010		1	1.0	12.0
23	0.010		−1	3.0	12.0
24	0.010		1	3.0	12.0
25	0.030		−1	1.0	12.0
26	0.030		1	1.0	12.0
27	0.030		−1	3.0	12.0
28	0.030		1	3.0	12.0
29	0.060		−1	1.0	12.0
30	0.060		1	1.0	12.0
31	0.060		−1	3.0	12.0
32	0.060		1	3.0	12.0

[a]Note, that parameters for $G^{\text{ang.n.}}$ and $G^{\text{ang.w.}}$ are chosen following the same scheme, i.e., all possible combinations of η = 0.01, 0.03, 0.06 bohr^{-2}, $\lambda = \pm 1$ and ζ = 1,3 are used for a fixed cutoff (r_c and r_s given in units of bohr).

there are many choices of symmetry function parameters equally suited to describe the local atomic environment, e.g. systematic approaches recently presented in refs 44 and 58. The performance is of course mostly determined by the number of symmetry functions (and their type) and not by the selected parameters. The corresponding symmetry function groups are listed in the Supporting Information. The training data set consists of 1991 configurations, where initially 1300 structures, each containing 128 mW particles, were taken directly from MD simulations of water and ice at different temperatures and pressures. The remaining configurations were obtained via artificial modifications (shearing and scaling of the simulation box, random displacements, and deletions of individual atoms) of the initial configurations. Training of the nnp-mW potential was performed with a training application based on the NNP library, which implements parallel Kalman filter based neural network training.[40] We excluded roughly 10% of the entire data set from training. The configurations in this "test" set are used to detect overfitting and assess the overall quality of the fit. Only if the root-mean-square errors (RMSEs) of both the training and the test set are sufficiently low and do not deviate much from each other is the neural network capable of reproducing the potential energy surface at and in between the sample points in the training data set. Excellent training results were achieved for the mW model: the RMSE of energies in the training/test set converged to 0.24/0.27 meV/atom, and the RMSE of training/test forces converged to 15.5/15.7 meV/Å.

4.2. Comparison of Performance Optimization Strategies. In order to compare the performance gains achieved through the performance optimization strategies described in Sections 3.2 and 3.3, we set up MD simulation runs under realistic conditions on the VSC. Starting configurations of liquid water with almost cubic box geometry were prepared for both models. For nnp-H_2O a snapshot containing 2880 water molecules from an equilibration run at 300 K and 1 bar was used. For the nnp-mW model, two configurations, one with 128 and another one with 8192 mW particles, were prepared at 298 K and zero pressure. These three test systems were propagated in time for 1000 time steps with the velocity Verlet algorithm,[59] which yields trajectories consistent with the microcanonical ensemble. The time steps were 0.5 and 10 fs for nnp-H_2O and nnp-mW, respectively.

Four *LAMMPS* binaries were created with different NNP-library optimizations enabled at compile time: the basic version without any optimization ("Basic"), one with the cutoff function cache enabled ("CFC"), another one making use of symmetry function groups ("SFG"), and finally one with both optimizations activated ("SFG+CFC"). For each binary the benchmark simulations were carried out on different numbers of VSC nodes starting from serial runs on a single node up to runs with 512 MPI tasks on 32 nodes. *LAMMPS* binaries automatically collected timing results, which were parsed from the output files.

Figure 5 shows the performance in units of time steps per second for all three test cases and each binary as a function of the number of MPI tasks. As is to be expected, in general the performance increases with the number of cores, but the relative speedup decreases for each additional core added due to parallelization overhead. Clearly, the naive implementation without any optimization yields the worst timing results. Caching cutoff function values gives a small speed gain, while the introduction of symmetry function groups significantly boosts the performance. Combining both optimization methods is only marginally better than the SFG-only version, yet a consistent improvement over all simulations is clearly visible. As it turns out, the relative performance gain over the basic version is comparable for different numbers of MPI tasks. Thus, it is practical to specify average values for each optimization method which are provided in Table 5.

Summarizing the results from Figure 5 and Table 5, we find that the *LAMMPS* implementation based on our NNP library performs best when both the cutoff function caching and symmetry function groups are enabled as on average the performance is increased by approximately a factor of 3. Naturally, the default compile options for the NNP library are set accordingly. Depending on neural network parametrization, system size, and number of neighbors within the cutoff sphere,

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

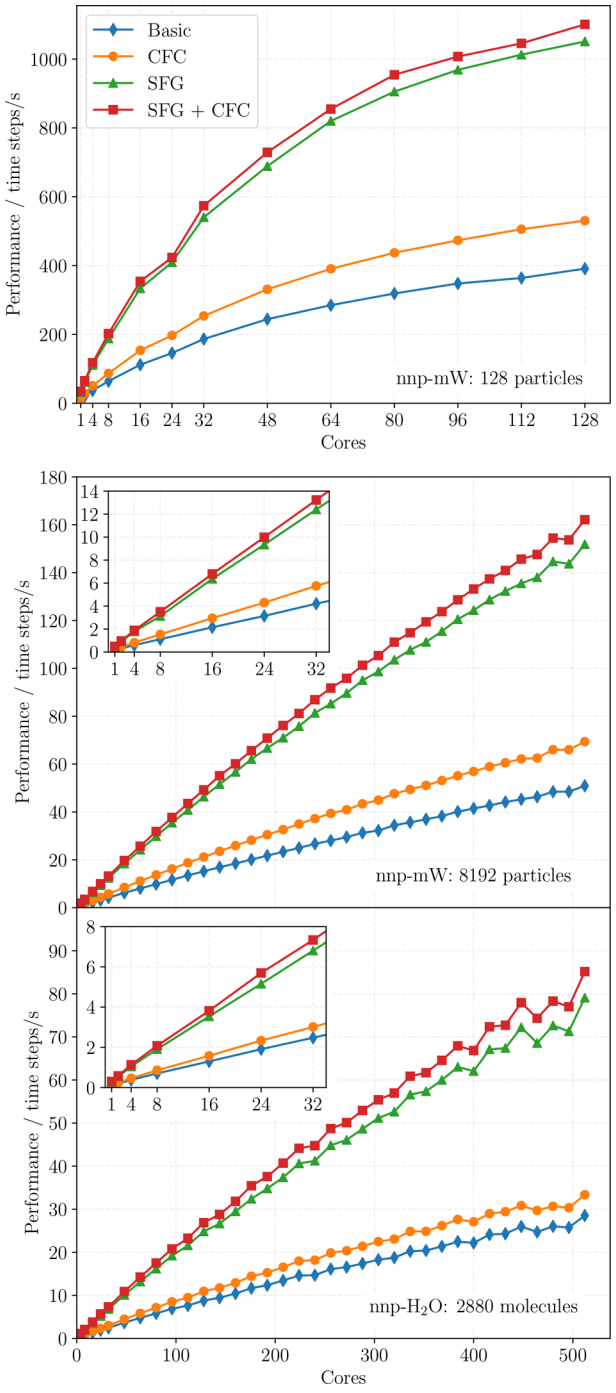


Figure 5. Benchmark results for the nnp-mW model with 128 (top) and 8192 (center) particles, as well as for the nnp-H_2O model with a system size of 2880 molecules (bottom). The performance, given in computed time steps per second, is shown as a function of the number of MPI tasks. In each panel four curves corresponding to the different optimization levels are shown.

the efficiency of the MPI parallelization may vary. Figure 6 shows the speedup and the parallel efficiency of the SFG+CFC version for all three test cases considered here.

4.3. MPI/OpenMP Hybrid Parallelization Performance. The parallel performance results obtained so far were solely based on the MPI parallelization strategy of LAMMPS. However, the NNP library offers an additional layer of thread-level parallelism via OpenMP (see Section 3.1) which can be

Table 5. Speedup Averaged from Data Shown in Figure 5[a]

system	CFC	SFG	SFG+CFC
nnp-H_2O	1.22	2.78	3.01
nnp-mW (128)	1.37	2.88	3.03
nnp-mW (8192)	1.38	3.01	3.21

[a]For each benchmark system and all optimized binaries (CFC, SFG, SFG+CFC), the average speedup factor with respect to the Basic program version is calculated from runs with different numbers of MPI tasks.

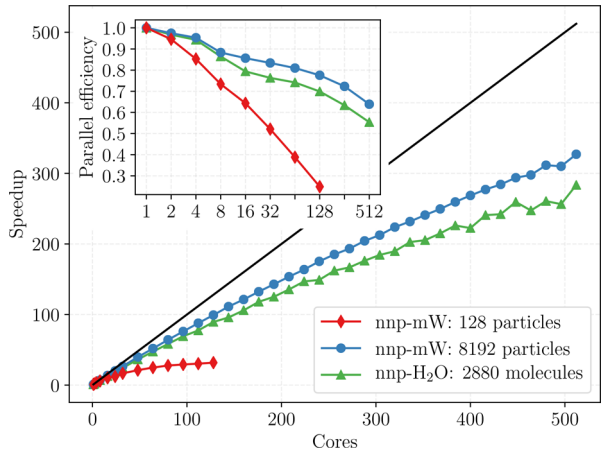


Figure 6. Speedup from parallelization via MPI and corresponding parallel efficiency (inset) for the fastest binary with all optimizations enabled (SFG+CFC). The solid line indicates the ideal speedup.

used to run hybrid MPI/OpenMP simulations with LAMMPS. This approach allows for the combining of the capabilities of MPI (internode communication) and OpenMP (intranode shared memory access) to reflect the hierarchical hardware setups found on many cluster systems (interconnected nodes with multicore processors). Typically, in such a hybrid simulation run each MPI process dynamically spawns multiple OpenMP threads and distributes the local workload to cores within each node. The total number of cores used is therefore the product of MPI processes and OpenMP threads. In comparison to a pure MPI run with the same number of cores, the potential benefit is a reduction of MPI communication overhead and a better scaling behavior. However, the actual performance depends strongly on the hardware setup and the implementation details.

In Figure 7 we present benchmark results for the nnp-H_2O model and a system size of 2880 molecules. A binary with all optimizations (SFG+CFC) and additional OpenMP support was used, and timing information was collected for hybrid MPI/OpenMP runs with 1, 2, and 4 OpenMP threads per MPI process. While the communication overhead drops as expected with multiple threads, an overall speedup is unfortunately not observed. Instead, up to about 128 total cores used there is even a clear decrease in performance of about 5% and 15% for 2 and 4 threads, respectively. Nevertheless, the situation changes above 256 cores as the performance approaches that of the pure MPI run. Ultimately, for the maximum number of cores (1024) there is even a small speedup of about 7%. A potential explanation for the observed behavior is the incomplete parallelization at the OpenMP level competing against the communication overhead. Due to technical reasons the LAMMPS interface parallelizes only the symmetry function

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

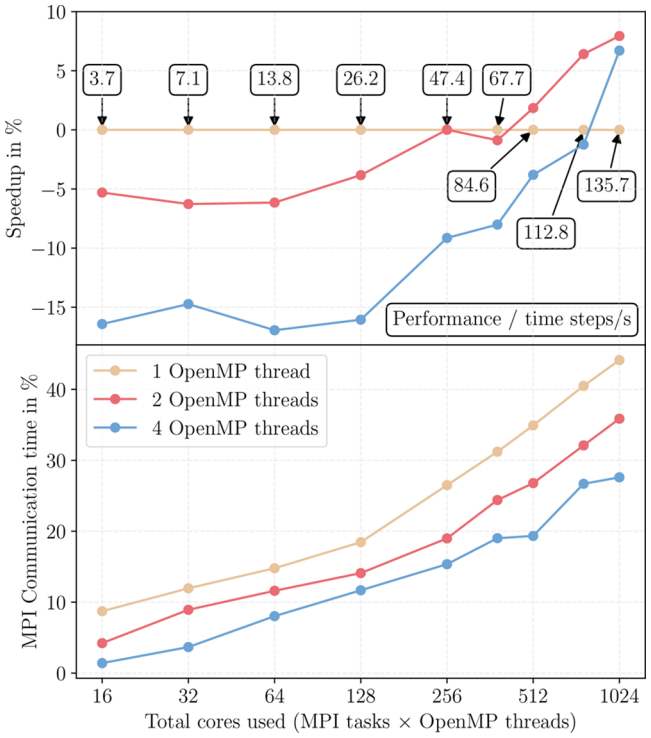


Figure 7. Speedup and communication overhead for hybrid MPI/OpenMP simulation runs. The bottom panel shows the expected reduction of MPI communication when using 2 or 4 OpenMP threads per MPI process. The speedup with respect to a pure MPI run is illustrated on the top panel. Numbers in black boxes denote the absolute performance in time steps per second (compare to bottom panel of Figure 5).

calculation via OpenMP while additional energy and force computation parts remain serially executed by each MPI process. Thus, for small numbers of total cores this disadvantage dominates. Only with many MPI processes involved, the communication reduction benefit outweighs the parallel performance loss due to serial code parts.

The results demonstrate that pure MPI simulations using the LAMMPS interface already scale well to many cores. However, some additional performance gain can be achieved using a MPI/OpenMP hybrid parallelization strategy if many MPI processes are used and the communication time would otherwise be substantial. With some additional programming efforts in the future, the NNP library may be tuned to reduce the amount of serial computations and further improve the hybrid parallelization performance. For the remainder of this Article we discuss only results obtained with pure MPI and without OpenMP parallelization.

4.4. Dependence of Performance on Neural Network Potential Settings. In the previous section we showed that both optimization strategies presented in Section 3 significantly increase the performance when compared to a naive implementation. Here, we address two more detailed questions on how the performance is affected by NN parametrization specifics:

1. Is it possible to further increase the efficiency of the NNP implementation speed by making use of different cutoff functions, i.e. switching from a transcendental functional form to cutoff functions based on computationally less demanding polynomials?
2. How big is the performance loss if for a given NN parametrization additional symmetry functions are introduced? This question usually arises in the creation of a new neural network potential when training results indicate that the spatial resolution is not sufficient and additional symmetry functions should be used. More radial symmetry functions will usually not influence performance, but the number of angular symmetry functions will require a balance between accuracy and speed.

In order to answer these questions we set up three additional NN parametrizations based on the nnp-mW model and ran benchmarks with the 8192 mW particle system. Regarding the first question, the modification "nnp-mW-f_c^{poly2}" uses the same NN settings as the original model with only one exception: the cosine cutoff function $f^{\cos}$ of all symmetry functions is replaced by the polynomial function f^{poly2} (leaving $\alpha = 0$ unchanged) as described in Section 3.3. To tackle the second question, two variants with additional angular symmetry functions were constructed from the original nnp-mW model. First, the variant "nnp-mW-η" adds new symmetry functions of type $G^{\text{ang.n.}}$ and $G^{\text{ang.w.}}$ with $\eta = 0.001$. The remaining parameters are set following the scheme from the original model, i.e. $\lambda = \pm 1$, $\zeta = 1,3$ and $r_c = 12.0$ bohr, see Table 6 for a list of new symmetry functions. Similarly, the variation "nnp-mW-ζ" extends the original model by new symmetry functions with $\zeta = 9$, see Table 7 for a list of additional symmetry function entries.

Table 6. Additional Symmetry Functions for the nnp-mW-η Variant

η	r_s	λ	ζ	r_c
		$G^{\text{ang.n.}}$		
0.001		−1	1.0	12.0
0.001		1	1.0	12.0
0.001		−1	3.0	12.0
0.001		1	3.0	12.0
		$G^{\text{ang.w.}}$		
0.001		−1	1.0	12.0
0.001		1	1.0	12.0
0.001		−1	3.0	12.0
0.001		1	3.0	12.0

Table 7. Additional Symmetry Functions for the nnp-mW-ζ Variant

η	r_s	λ	ζ	r_c
		$G^{\text{ang.n.}}$		
0.01		−1	9.0	12.0
0.01		1	9.0	12.0
0.03		−1	9.0	12.0
0.03		1	9.0	12.0
0.06		−1	9.0	12.0
0.06		1	9.0	12.0
		$G^{\text{ang.w.}}$		
0.01		−1	9.0	12.0
0.01		1	9.0	12.0
0.03		−1	9.0	12.0
0.03		1	9.0	12.0
0.06		−1	9.0	12.0
0.06		1	9.0	12.0

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

The three variants of the nnp-mW model were trained with the same data set as described in Section 4.1, and a comparable fit quality was achieved. Benchmark MD simulations were run with the 8192 mW particle system under the same conditions as in the previous section with all four *LAMMPS* binaries. Finally, the timing data were compared to those of the unmodified nnp-mW model, and the performance gain or loss was averaged over runs with different number of MPI tasks.

Table 8 shows the results for all variants and binaries in percent with respect to the performance of the nnp-mW model. With the results in the first line of the table, we are able to answer our first question posed. While the basic implementation would still benefit from the use of a computationally less expensive cutoff function, the advantage almost disappears when all NNP library optimizations are enabled. This is because the cutoff function calls are drastically reduced by caching and by the use of symmetry function groups. Regarding the second question about the performance loss caused by additional symmetry functions we find that the usage of symmetry function groups (binaries SFG and SFG+CFC) is clearly favorable. If symmetry function calls are executed sequentially without any reuse of intermediate results (binaries Basic and CFC), the expected maximum decrease in performance corresponds to the number of additional symmetry functions since the calculation of radial symmetry functions and the execution of other parts of the code can be neglected. In the case of 8 additional entries for the nnp-mW-η model, this amounts to 25%. Likewise for nnp-mW-ζ a maximum drop of 37.5% is to be expected. Indeed the observations presented in Table 8 closely match these expectations. On the other hand, the performance drop experienced with symmetry function groups enabled (columns SFG and SFG+CFC) is less pronounced as the corresponding numbers in Table 8 are consistently lower. This behavior is easily explained with the construction of symmetry function groups (see Section 3.2) in mind: With the given choice of parameters the additional symmetry functions can be appended to the list of symmetry function group members of the original nnp-mW model (see the Supporting Information) to form the symmetry function groups of the nnp-mW-η or nnp-mW-ζ model. Therefore, the additionally required computation steps involve only those that are not already part of calculations common to all group members, which effectively reduces the cost of the extra symmetry functions.

Table 8. Change of Performance Given in % with Respect to the Original nnp-mW Model[a]

variant	basic	CFC	SFG	SFG+CFC
nnp-mW-f_c^{poly2}	+37.8	+10.9	+8.4	+2.2
nnp-mW-η	−24.4	−24.2	−20.3	−21.6
nnp-mW-ζ	−33.8	−34.1	−28.9	−30.0

[a]Similarly to Table 5 the results are averages over different numbers of MPI tasks.

4.5. Comparison with TIP4P/Ice. Among the large variety of empirical potentials proposed for water over the years, the TIP4P class of models[60−62] are among the most popular ones as they offer a good compromise between speed and agreement with experimental data. Here, we provide a benchmark comparison between the *LAMMPS* implementation of the TIP4P/Ice potential[61] and the nnp-H_2O model described in Section 4.1. Note that for the TIP4P/Ice benchmark *LAMMPS* was built with the USER-INTEL package enabled to ensure optimal performance on the VSC HPC system. A liquid water configuration with 2880 molecules was used to start the molecular dynamics simulations in both cases. The TIP4P/Ice simulation was run for 5000 time steps with Δt = 1.0 fs. Note that the time step for the TIP4P/Ice model can be larger than for the NNP simulations, because in the latter case molecules are fully flexible rather than rigid. Up to 128 MPI tasks were used, and the timing results were compared to those of the nnp-H_2O model with the fully optimized (SFG+CFC) binary as depicted in the bottom panel of Figure 5. The resulting Figure 8 shows the factor by which the TIP4P/Ice model outperforms the neural network potential approach. The single core performance in time steps per second differs by a factor 50. Most likely due to Ewald summation in the case of the TIP4P/Ice model the scaling behavior of the two methods is more in favor of the neural network potential method as the advantage shrinks to a factor 30 at still reasonable numbers of MPI tasks. In the world of molecular dynamics simulations this is a considerable performance difference. However, the neural network approach with its capability to reproduce *ab initio* reference potential energy surfaces brings us very close to time scales accessible to empirical potentials.

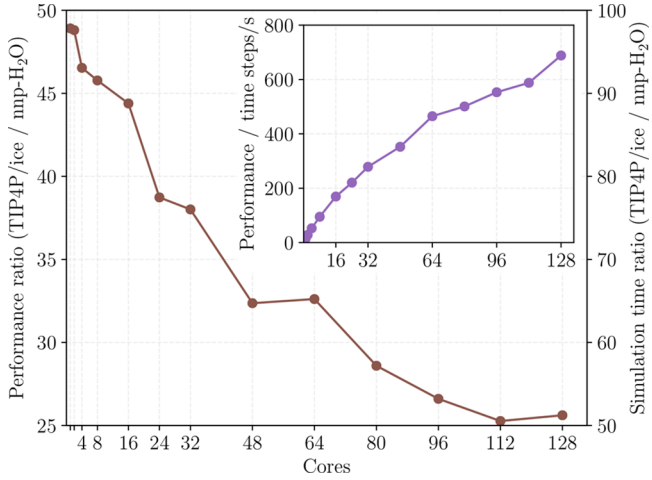


Figure 8. Relative speedup of the TIP4P/ice simulation with respect to the nnp-H_2O model. Note that the time steps for the two methods are different, i.e. Δt = 1 and 0.5 fs for TIP4P/Ice and nnp-H_2O, respectively. Thus, two different *y*-axes labels are provided: one for the bare speedup in time steps per second (left axis) and one for the actual simulation time ratio where the differing time steps are taken into account (right axis). Inset: Performance in time steps per second for the TIP4P/Ice potential.

5. CONCLUSION

In this work, we have presented a neural network potential library that serves as an efficient foundation for the application of Behler-Parrinello type neural network potentials. With its data types and class structure, the library provides everything necessary to develop new standalone programs as well as interfaces to external software. As an example, we have integrated our neural network potential implementation into the molecular dynamics program *LAMMPS*, making the diverse molecular dynamics techniques included in this powerful package available for NNP simulations. Due to the

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

library approach, the advantages of code optimizations within the library are automatically passed on to the interfacing software. We presented how the *LAMMPS* interface benefits from two optimization strategies: cutoff function caching and symmetry function groups. With both of these optimization options enabled we find that a speedup by a factor of 3 can be achieved on a modern HPC system in highly parallelized simulations.

With our *LAMMPS* implementation the possibility of the neural network potential method to simulate large systems over long time periods with the accuracy of the underlying *ab initio* potential energy surface can be fully exploited. As we demonstrated, it is feasible to run massively parallelized MD simulations of 2880 water molecules with a DFT neural network potential parametrization at rates of about 100 time steps per second. As the NNP performance is independent of the reference electronic structure method and the number of training data points, the same performance could be reached for high-level reference data from quantum chemical methods. With an appropriate time step this amounts to around 4 ns per day using 512 cores on a current HPC system.

The software presented here is available freely online as part of the *n2p2* neural network potential package at http://github.com/CompPhysVienna/n2p2. Example *LAMMPS* scripts, HDNNP parametrizations, and additional library-based applications for training and data set manipulation[40] are also provided within the package. Future development efforts will likely aim at an efficient implementation of symmetry functions computation on graphics cards which may open up the possibility to include the HDNNP method in GPU-based MD codes such as *HOOMD-blue*[63−65] or *OpenMM*.[66,67]

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acs.jctc.8b00770.

Expressions for derivatives of symmetry functions and symmetry function parameter tables (PDF)

■ AUTHOR INFORMATION

Corresponding Authors

*E-mail: andreas.singraber@univie.ac.at.
*E-mail: christoph.dellago@univie.ac.at.

ORCID

Andreas Singraber: 0000-0002-4330-1394
Jörg Behler: 0000-0002-1220-1542
Christoph Dellago: 0000-0001-9166-6235

Funding

A.S. is grateful for support by the VSC Research Center funded by the Austrian Federal Ministry of Science, Research and Economy (bmwfw). Financial support of the Austrian Science Fund (FWF) (SFB Vicom, F41) is gratefully acknowledged. J.B. thanks the DFG for a Heisenberg Professorship (BE3264/11-2). The results presented here have been achieved using the Vienna Scientific Cluster (VSC).

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors thank Clemens Moritz and Tobias Morawietz for many insightful discussions.

■ REFERENCES

(1) Behler, J. Perspective: Machine learning potentials for atomistic simulations. *J. Chem. Phys.* **2016**, *145*, 170901.
(2) Behler, J.; Martoňák, R.; Donadio, D.; Parrinello, M. Metadynamics Simulations of the High-Pressure Phases of Silicon Employing a High-Dimensional Neural Network Potential. *Phys. Rev. Lett.* **2008**, *100*, 185501.
(3) Morawietz, T.; Singraber, A.; Dellago, C.; Behler, J. How van der Waals interactions determine the unique properties of water. *Proc. Natl. Acad. Sci. U. S. A.* **2016**, *113*, 8368−8373.
(4) Cheng, B.; Engel, E. A.; Behler, J.; Dellago, C.; Ceriotti, M. Ab initio thermodynamics of liquid and solid water. *Proc. Natl. Acad. Sci. U. S. A.* **2019**, *116*, 1110−1115.
(5) Hellström, M.; Behler, J. Concentration-Dependent Proton Transfer Mechanisms in Aqueous NaOH Solutions: From Acceptor-Driven to Donor-Driven and Back. *J. Phys. Chem. Lett.* **2016**, *7*, 3302−3306.
(6) Behler, J.; Reuter, K.; Scheffler, M. Nonadiabatic effects in the dissociation of oxygen molecules at the Al(111) surface. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2008**, *77*, 115421.
(7) Shakouri, K.; Behler, J.; Meyer, J.; Kroes, G.-J. Accurate Neural Network Description of Surface Phonons in Reactive Gas−Surface Dynamics: N2 + Ru(0001). *J. Phys. Chem. Lett.* **2017**, *8*, 2131−2136.
(8) Bartók, A. P.; Payne, M. C.; Kondor, R.; Csányi, G. Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons. *Phys. Rev. Lett.* **2010**, *104*, 136403.
(9) Bartók, A. P.; Kondor, R.; Csányi, G. On representing chemical environments. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2013**, *87*, 184115.
(10) Rupp, M.; Tkatchenko, A.; Müller, K.-R.; von Lilienfeld, O. A. Fast and Accurate Modeling of Molecular Atomization Energies with Machine Learning. *Phys. Rev. Lett.* **2012**, *108*, 058301.
(11) Thompson, A. P.; Swiler, L. P.; Trott, C. R.; Foiles, S. M.; Tucker, G. J. Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials. *J. Comput. Phys.* **2015**, *285*, 316−330.
(12) Blank, T. B.; Brown, S. D.; Calhoun, A. W.; Doren, D. J. Neural network models of potential energy surfaces. *J. Chem. Phys.* **1995**, *103*, 4129−4137.
(13) Behler, J. Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations. *Phys. Chem. Chem. Phys.* **2011**, *13*, 17930−17955.
(14) Behler, J. First Principles Neural Network Potentials for Reactive Simulations of Large Molecular and Condensed Systems. *Angew. Chem., Int. Ed.* **2017**, *56*, 12828−12840.
(15) Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Phys. Rev. Lett.* **2007**, *98*, 146401.
(16) Plimpton, S. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.* **1995**, *117*, 1−19.
(17) Plimpton, S.; Thompson, A.; Moore, S.; Kohlmeyer, A. *LAMMPS Molecular Dynamics Simulator*, 16Mar18; 2004. http://lammps.sandia.gov (accessed Oct 12, 2018).
(18) Artrith, N.; Urban, A. An implementation of artificial neural-network potentials for atomistic materials simulations: Performance for TiO2. *Comput. Mater. Sci.* **2016**, *114*, 135−150.
(19) Artrith, N.; Urban, A.; Ceder, G. Efficient and accurate machine-learning interpolation of atomic energies in compositions with many species. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2017**, *96*, 014112.
(20) Artrith, N.; Urban, A. *aenet: Atomic Interaction Potentials Based on Artificial Neural Networks*, Version 2.0.3; 2018. http://ann.atomistic.net (accessed Oct 12, 2018).
(21) Behler, J. *RuNNer - A Neural Network Code for High-Dimensional Potential-Energy Surfaces*; Universität Göttingen, 2018. http://www.uni-goettingen.de/de/560580.html (accessed Oct 12, 2018).

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

(22) Khorshidi, A.; Peterson, A. A. Amp: A modular approach to machine learning in atomistic simulations. *Comput. Phys. Commun.* **2016**, *207*, 310−324.
(23) Peterson, A.; Khorshidi, A. *Amp: Atomistic Machine-learning Package*, version 0.6; 2017. https://amp.readthedocs.io (accessed Dec 5, 2018).
(24) Kolb, B.; Lentz, L. C.; Kolpak, A. M. Discovering charge density functionals and structure-property relationships with PROPhet: A general framework for coupling machine learning and first-principles methods. *Sci. Rep.* **2017**, *7*, 1192.
(25) Kolb, B.; Lentz, L. *PROPhet*, v1.2; 2018. http://doi.org/10.5281/zenodo.159046 (accessed Oct 12, 2018).
(26) Yao, K.; Herr, J. E.; Toth, D. W.; Mckintyre, R.; Parkhill, J. The TensorMol-0.1 model chemistry: a neural network augmented with long-range physics. *Chem. Sci.* **2018**, *9*, 2261−2269.
(27) Yao, K.; Herr, J.; Toth, D.; McIntyre, R.; Casetti, N.; Parkhill, J. *TensorMol*, TensorMol0.1; 2017. https://github.com/jparkhill/TensorMol (accessed Oct 12, 2018).
(28) Zhang, L.; Han, J.; Wang, H.; Car, R.; E, W. Deep Potential Molecular Dynamics: A Scalable Model with the Accuracy of Quantum Mechanics. *Phys. Rev. Lett.* **2018**, *120*, 143001.
(29) Zhang, L.; Han, J.; Wang, H.; Car, R.; E, W. *DeePMD-kit: A Deep Learning Package for Many-Body Potential Energy Representation and Molecular Dynamics*, v0.10.2; 2018. https://github.com/deepmodeling/deepmd-kit (accessed Oct 20, 2018).
(30) Artrith, N.; Morawietz, T.; Behler, J. High-dimensional neural-network potentials for multicomponent systems: Applications to zinc oxide. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2011**, *83*, 153101.
(31) Artrith, N.; Behler, J. High-dimensional neural network potentials for metal surfaces: A prototype study for copper. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2012**, *85*, 045439.
(32) Sosso, G. C.; Miceli, G.; Caravati, S.; Behler, J.; Bernasconi, M. Neural network interatomic potential for the phase change material GeTe. *Phys. Rev. B: Condens. Matter Mater. Phys.* **2012**, *85*, 174103.
(33) Artrith, N.; Hiller, B.; Behler, J. Neural network potentials for metals and oxides − First applications to copper clusters at zinc oxide. *Phys. Status Solidi B* **2013**, *250*, 1191−1203.
(34) Cheng, B.; Behler, J.; Ceriotti, M. Nuclear Quantum Effects in Water at the Triple Point: Using Theory as a Link Between Experiments. *J. Phys. Chem. Lett.* **2016**, *7*, 2210−2215.
(35) Gastegger, M.; Kauffmann, C.; Behler, J.; Marquetand, P. Comparing the accuracy of high-dimensional neural network potentials and the systematic molecular fragmentation method: A benchmark study for all-trans alkanes. *J. Chem. Phys.* **2016**, *144*, 194110.
(36) Natarajan, S. K.; Behler, J. Neural network molecular dynamics simulations of solid−liquid interfaces: water at low-index copper surfaces. *Phys. Chem. Chem. Phys.* **2016**, *18*, 28704−28725.
(37) Gabardi, S.; Baldi, E.; Bosoni, E.; Campi, D.; Caravati, S.; Sosso, G. C.; Behler, J.; Bernasconi, M. Atomistic Simulations of the Crystallization and Aging of GeTe Nanowires. *J. Phys. Chem. C* **2017**, *121*, 23827−23838.
(38) Quaranta, V.; Hellström, M.; Behler, J. Proton-Transfer Mechanisms at the Water−ZnO Interface: The Role of Presolvation. *J. Phys. Chem. Lett.* **2017**, *8*, 1476−1483.
(39) Schran, C.; Uhl, F.; Behler, J.; Marx, D. High-dimensional neural network potentials for solvation: The case of protonated water clusters in helium. *J. Chem. Phys.* **2018**, *148*, 102310.
(40) Singraber, A.; Morawietz, T.; Behler, J.; Dellago, C. Parallel multi-stream training of high-dimensional neural network potentials. *J. Chem. Theory Comput*, **2019** (submitted for publication).
(41) Behler, J. Constructing high-dimensional neural network potentials: A tutorial review. *Int. J. Quantum Chem.* **2015**, *115*, 1032−1050.
(42) Behler, J. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *J. Chem. Phys.* **2011**, *134*, 074106.
(43) Behler, J. Representing potential energy surfaces by high-dimensional neural network potentials. *J. Phys.: Condens. Matter* **2014**, *26*, 183001.
(44) Gastegger, M.; Schwiedrzik, L.; Bittermann, M.; Berzsenyi, F.; Marquetand, P. wACSF—Weighted atom-centered symmetry functions as descriptors in machine learning potentials. *J. Chem. Phys.* **2018**, *148*, 241709.
(45) Geiger, P.; Dellago, C. Neural networks for local structure detection in polymorphic systems. *J. Chem. Phys.* **2013**, *139*, 164105.
(46) Huo, H.; Rupp, M. *Unified Representation of Molecules and Crystals for Machine Learning*; 2017; arXiv:1704.06439. arXiv.org ePrint archive. https://arxiv.org/abs/1704.06439 (accessed Oct 13, 2018).
(47) van Heesch, D. *Doxygen*, 1.8.14; 2017. http://www.doxygen.nl (accessed Feb 6, 2019).
(48) Morawietz, T.; Behler, J. A Full-Dimensional Neural Network Potential-Energy Surface for Water Clusters up to the Hexamer. *Z. Phys. Chem.* **2013**, *227*, 1559−1581.
(49) Natarajan, S. K.; Morawietz, T.; Behler, J. Representing the potential-energy surface of protonated water clusters by high-dimensional neural network potentials. *Phys. Chem. Chem. Phys.* **2015**, *17*, 8356−8371.
(50) Hellström, M.; Behler, J. Structure of aqueous NaOH solutions: insights from neural-network-based molecular dynamics simulations. *Phys. Chem. Chem. Phys.* **2017**, *19*, 82−96.
(51) Zhou, X. W.; Jones, R. E. Effects of cutoff functions of Tersoff potentials on molecular dynamics simulations of thermal transport. *Modell. Simul. Mater. Sci. Eng.* **2011**, *19*, 025004.
(52) Thompson, A. P.; Plimpton, S. J.; Mattson, W. General formulation of pressure and stress tensor for arbitrary many-body interaction potentials under periodic boundary conditions. *J. Chem. Phys.* **2009**, *131*, 154107.
(53) Hammer, B.; Hansen, L. B.; Nørskov, J. K. Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1999**, *59*, 7413−7421.
(54) Grimme, S.; Antony, J.; Ehrlich, S.; Krieg, H. A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H-Pu. *J. Chem. Phys.* **2010**, *132*, 154104.
(55) Singraber, A.; Morawietz, T.; Behler, J.; Dellago, C. Density anomaly of water at negative pressures from first principles. *J. Phys.: Condens. Matter* **2018**, *30*, 254005.
(56) Molinero, V.; Moore, E. B. Water Modeled As an Intermediate Element between Carbon and Silicon. *J. Phys. Chem. B* **2009**, *113*, 4008−4016.
(57) Stillinger, F. H.; Weber, T. A. Computer simulation of local order in condensed phases of silicon. *Phys. Rev. B: Condens. Matter Mater. Phys.* **1985**, *31*, 5262−5271.
(58) Imbalzano, G.; Anelli, A.; Giofré, D.; Klees, S.; Behler, J.; Ceriotti, M. Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials. *J. Chem. Phys.* **2018**, *148*, 241730.
(59) Swope, W. C.; Andersen, H. C.; Berens, P. H.; Wilson, K. R. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *J. Chem. Phys.* **1982**, *76*, 637−649.
(60) Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. Comparison of simple potential functions for simulating liquid water. *J. Chem. Phys.* **1983**, *79*, 926−935.
(61) Abascal, J. L. F.; Sanz, E.; García Fernández, R.; Vega, C. A potential model for the study of ices and amorphous water: TIP4P/Ice. *J. Chem. Phys.* **2005**, *122*, 234511.
(62) Abascal, J. L. F.; Vega, C. A general purpose model for the condensed phases of water: TIP4P/2005. *J. Chem. Phys.* **2005**, *123*, 234505.
(63) Anderson, J. A.; Lorenz, C. D.; Travesset, A. General purpose molecular dynamics simulations fully implemented on graphics processing units. *J. Comput. Phys.* **2008**, *227*, 5342−5359.

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840

(64) Glaser, J.; Nguyen, T. D.; Anderson, J. A.; Lui, P.; Spiga, F.; Millan, J. A.; Morse, D. C.; Glotzer, S. C. Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Comput. Phys. Commun.* **2015**, *192*, 97−107.
(65) *The Glotzer Group HOOMD-blue*, v2.4.2; University of Michigan: 2018. http://glotzerlab.engin.umich.edu/hoomd-blue/index.html (accessed Jan 18, 2019).
(66) Eastman, P.; Swails, J.; Chodera, J. D.; McGibbon, R. T.; Zhao, Y.; Beauchamp, K. A.; Wang, L.-P.; Simmonett, A. C.; Harrigan, M. P.; Stern, C. D.; Wiewiora, R. P.; Brooks, B. R.; Pande, V. S. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLoS Comput. Biol.* **2017**, *13*, e1005659.
(67) *The OpenMM team OpenMM*, 7.3; Stanford University: 2018. http://openmm.org/ (accessed Jan 18, 2019).

DOI: 10.1021/acs.jctc.8b00770
J. Chem. Theory Comput. 2019, 15, 1827−1840