JAX-ReaxFF: A Gradient Based Framework for Extremely Fast Optimization of Reactive Force Fields

Mehmet Cagri Kaymak[1], Ali Rahnamoun[2], Kurt A. O'Hearn[1], Adri C. T. van Duin[3], Kenneth M. Merz, Jr.[2], and Hasan Metin Aktulga[1]

[1]*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA*
[2]*Department of Chemistry, Michigan State University, 578 S. Shaw Lane, East Lansing, MI 48824, USA*
[3]*Department of Mechanical Engineering, The Pennsylvania State University, University Park, State College, Pennsylvania 16802, United States*

Abstract

Molecular dynamics (MD) simulations facilitate the study of physical and chemical processes of interest. Traditional classical MD models lack reactivity to explore several important phenomena; while quantum mechanical (QM) models can be used for this purpose, they come with steep computational costs. The reactive force field (ReaxFF) model bridges the gap between these approaches by incorporating dynamic bonding and polarizability. To achieve realistic simulations using ReaxFF, model parameters must be optimized against high fidelity training data, typically with QM accuracy. Existing parameter optimization methods for ReaxFF consist of black-box techniques using genetic algorithms or Monte-Carlo methods. Due to the stochastic behavior of these methods, the optimization process can require millions of error evaluations for complex parameter fitting tasks, significantly hampering the rapid development of high quality parameter sets. In this work, we present JAX-ReaxFF, a novel software tool that leverages modern machine learning infrastructure to enable extremely fast optimization of ReaxFF parameters. By calculating gradients of the loss function using the JAX library, we are able to utilize highly effective local optimization methods, such as the limited Broyden–Fletcher–Goldfarb–Shanno (LBFGS) and Sequential Least Squares Programming (SLSQP) methods. As a result of the performance portability of JAX, JAX-ReaxFF can execute efficiently on multi-core CPUs, GPUs (or even TPUs). By leveraging the gradient information and modern hardware accelerators, we are able to decrease parameter optimization time for ReaxFF from days to mere minutes. JAX-ReaxFF framework

can also serve as a sandbox environment for domain scientists to explore customizing the ReaxFF functional form for more accurate modeling.

1 Introduction

Molecular dynamics (MD) is a widely adopted method to study physical and chemical processes at an atomistic level in a number of fields ranging from biophysics to chemistry to materials science [10]. Quantum mechanical (QM) simulations allow the geometries and energies to be predicted accurately by solving the Schrödinger's equation. However, the computational complexity and cost of the QM based methods make them only viable for simulating small systems for rather short periods of timeframes. Molecular dynamics (MD) simulations, on the other hand, enable the study of large systems for relatively long timeframes through a number of approximations. In this approach, atomic nucleus together with its electrons is treated as a unit particle and interactions between atoms are governed by a force field (FF), which is essentially a set of parameterized mathematical equations aimd at capturing well-known atomic interactions such as bonds, valence angles, torsion, van der Waals, and Coulomb interactions. These simplifications greatly reduce the overall computational cost, but an important measure of the predictive power of force fields is their fidelity, i.e., how well they can reproduce the results of QM calculations and experimental studies. Development of high fidelity force fields relies heavily on optimization of various force field parameters based on carefully selected quantum-chemical and experimental reference data. With the help of these approximations and careful training, MD methods have proven to be successful in atomistic simulations with billions of degrees of freedom [19].

Classical MD models as implemented in highly popular MD software such as Amber [5], LAMMPS [38], GROMACS [15] and NAMD [29] are based on the assumption of static chemical bonds and, in general, static charges. Therefore, they are not applicable to study phenomena where chemical reactions and charge polarization effects play a significant role. To address this gap, reactive force fields (e.g., ReaxFF [40], REBO [4], Tersoff [37]) have been developed. These bond order potentials allow bonds to form and break throughout the simulation and they can dynamically assign partial charges to atoms using suitable charge models such as the electronegativity equalization method (EEM) [24]. The functional forms for reactive potentials are significantly more complex than their classical counterparts due to the presence of dynamic bonds and charges. For instance, ReaxFF has a formulation that contains more than 100 parameters for a simulation with 3 elements, and is about two orders of magnitude more expensive than a typical Lennard-Jones potential. Consequently, training reactive force fields is an even more difficult task due to the need to capture complex phenomena such as charge distributions and reactions, and due to the large number of parameters involved.

We focus on the ReaxFF method, which is one of the most impactful reactive force fields, if not the most impactful one [40, 33]. If there is an existing Reax

force field for a similar purpose, it could be sufficient to fine-tune that for the new target application. When a new force field needs to be developed from scratch, multiple passes over the training data may be necessary, i.e., based on the quality of the resulting force field, the training data itself may need to go through revisions. In both scenarios, the training speed is crucial. As such, development of high-quality and fast optimization methods for ReaxFF has been an active research topic, first starting with the sequential one-parameter parabolic interpolation method (SOPPI) by van Duin [8], and then continuing with various global optimization methods such as genetic algorithms (GAs) [7, 18, 23], simulated annealing (SA) [16, 17], evolutionary algorithms (EAs) [39], particle swarm optimization (PSO) [12]. More recently, machine learning based search methods have been employed for this purpose [6, 14, 27].

Inspired by developments in machine learning, specifically in the field of automatic differentiation, we present a new software called JAX-ReaxFF that enables extremely fast optimization of Reax force field parameters [1]. JAX is an auto-differentitation software by Google for high performance machine learning research [3], it can automatically differentiate native Python and NumPy functions. Leveraging this capability, JAX-ReaxFF automatically calculates the derivative of a given fitness function, which essentially measures the root mean squared deviation (RMSD) of a force field against a reference dataset, from Python-based implementation of the ReaxFF potential energy terms with respect to the set of force field parameters to be optimized. By learning the gradient information of the high dimensional optimization space (which generally includes tens to over a hundred parameters), JAX-ReaxFF can employ the highly effective local optimization methods such as the Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [41] and Sequential Least Squares Programming (SLSQP) [20] optimizer. The gradient information alone is obviously not sufficient to prevent the local optimizers from getting stuck in a local minima, but when combined with a multi-start approach, JAX-ReaxFF can greatly improve the training efficiency (measured in terms of the number of fitness function evaluations performed) and significantly reduce the time spent on optimizing ReaxFF parameters.

Another important advantage of JAX is its architectural portability enabled by the XLA technology [32] used under the hood. Hence, JAX-ReaxFF can run efficiently on various architectures, including graphics procesing units (GPU) and tensor processing units (TPU), through automatic thread parallelization and vector processing. As we demonstrate through extensive tests, JAX-ReaxFF can reduce the overall training time by up to three orders of magnitude compared to the existing global optimization schemes, while achieving similar (or better) fitness scores and yielding accurate simulation results.

Beyond speeding up force field optimization, the Python based JAX-ReaxFF software provides an ideal sandbox environment for domain scientists, as they can go beyond parameter optimization and start experimenting with the functional forms of the ReaxFF interactions, or add/remove interactions as desired.

[1] The code is available on GitHub: https://github.com/cagrikymk/JAX-ReaxFF

Since evaluating the gradient of the new functional forms with respect to atom positions gives forces, scientists are freed from the burden of coding the lengthy and bug-prone force calculation parts. Through automatic differentiation of the fitness function as explained above, parameter optimization for the new set of functional forms can be performed without any additional effort by the domain scientists. After parameter optimization, they can readily start running MD simulations to test the macro-scale properties predicted by the modified set of functional forms as a further validation test before production-scale simulations, or go back to editing the functional forms if desired results cannot be confirmed in this sandbox evironment provided by JAX-ReaxFF. As such, we anticipate JAX-ReaxFF to be an indispensible tool for reactive molecular modeling and simulation.

2 Background and Related Work

Before going into the details of JAX-ReaxFF, we provide some background on ReaxFF and existing software for ReaxFF parameter optimizations.

2.1 ReaxFF Overview

ReaxFF divides the total potential energy into various parts, including bonded and non-bonded interactions as shown in Eq. (1). The model takes atom coordinates and required force field parameters for the set of elements present in the system as input, and calculates all terms constituting the potential energy together with the corresponding forces. The derivative of each potential energy term with respect to atom positions gives forces that are fundamental to the MD simulation. There is a number of ReaxFF implementations with different features and architectural support such as the original Fortran Reax code [40], PuReMD [1, 2, 21], GULP [13] and LAMMPS [30].

$$\begin{aligned} E_{\text{system}} &= E_{\text{bond}} + E_{\text{lone-pair}} + E_{\text{over}} + E_{\text{under}} + E_{\text{val}} + E_{\text{pen}} \\ &\quad + E_{\text{tors}} + E_{\text{conj}} + E_{\text{Hbond}} + E_{\text{vdWaals}} + E_{\text{Coulomb}} \,. \end{aligned} \tag{1}$$

An important aspect of ReaxFF that separates it from classical MD models are the notions of bond orders and dynamic partial charges (not shown in Eq. (1)). The bond order concept is used to determine the bond strength between pairs of atoms given their element types and distances. These pairwise bond orders are then subjected to corrections that take into account the information about all atoms surrounding each atom to obtain the predicted bonding information in a system. The corrected bond order constitutes the main input for common potential energy terms such as bond energy (E_{bond}), valence angle energy (E_{val}), and torsion angle energy (E_{tors}). However, in a dynamic bonding model, since atoms may not attain their optimal coordinations, additional terms such as lone pair ($E_{\text{lone-pair}}$), over/under-coordination ($E_{\text{over}}, E_{\text{under}}$), three-body penalty ($E_{\text{pen}}$) and four-body conjugation (E_{conj}) energies are needed. For systems with Hydrogen bonds, a special energy term (E_{Hbond}) is used. The van der Waals energy

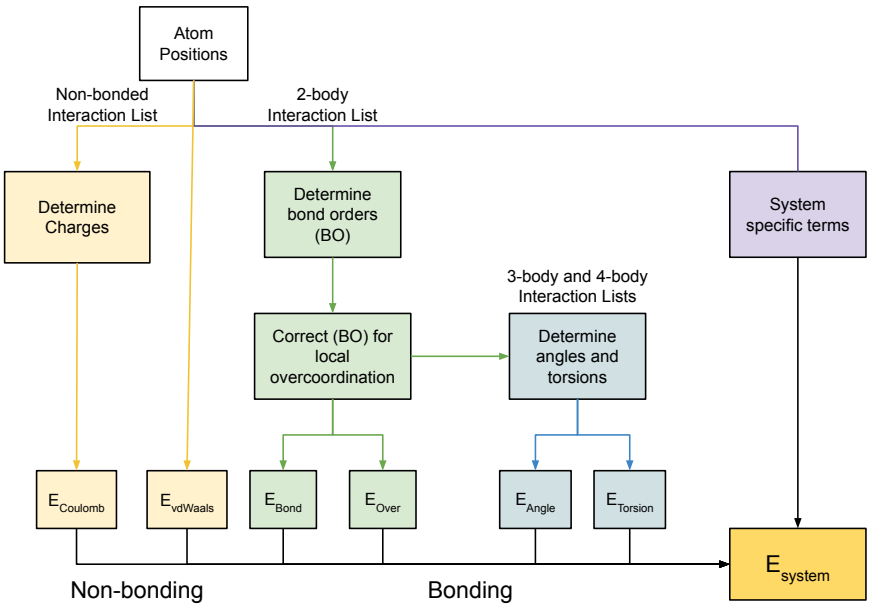


Figure 1: ReaxFF Model

(E_{vdWaals}), which is based on the Morse potential, and the electrostatic energy term (E_{Coulomb}), which uses shielded and range-limited interactions based on dynamic charges calculated from charge models such as EEM [24], constitute the non-bonded terms in ReaxFF. Typically, bonded interactions are truncated at 5Å, hydrogen bonds are effective up to 7.5Å and non-bonded interactions are range limited to 10-12Å depending on the system. Fig. 1 summarizes the calculations performed within a ReaxFF step.

2.2 ReaxFF Training

ReaxFF parameters are grouped by the number of atoms involved in the interaction (e.g., single-body, two-body, three-body and four-body) in addition to the system-wide global parameters. Based on the distances and angles between atoms and corresponding model parameters, bonded, 3-body, 4-body, H-bond and non-bonded interaction lists are formed dynamically at each timestep. For every interaction, corresponding parameters from the parameter set based on the element types of the atoms involved are used to calculate the E_{system}. As described in detail in [33], there exist parameter sets for different kinds of simulations such as combustion, aqueous systems, metals and biological systems. Even if there already is a parameter set for a simulation, it might require further tuning for a particular application. In some cases, the model needs to be trained from scratch which is a complex task. In general, as the number of elements in a parameter set increases, force field optimization becomes harder due to the

Table 1: Examples for commonly used training items. Identifiers (e.g., ID1) refer to structures/molecules.

Type	Training Item	Target	Description
Charge	ID1 1	0.5	Charge for atom 1 (in elementary charge)
Energy	ID1 - ID2/2 - ID3/3	50	Energy difference (in kcal/mol)
	ID1	-150	
	ID3/2 - ID1/3	30	
Geometry	ID1 1 2	1.25	Distance between atom 1 and 2 (in Å)
	ID2 1 2 3	120	Valence angle between atom 1, 2 and 3 (in degree)
	ID3 1 2 3 4	170	Torsion angle between atom 1, 2, 3 and 4 (in degree)
Force	ID1 1	0.5 0.5 0.5	Forces on atom 1 (in kcal/mol Å)
	ID2	1.0	RMSG (in kcal/mol Å)

increasing number of parameters involved.

ReaxFF training procedure requires three different inputs: i) *geometries*, a set of atom clusters crucial in describing the system of interest (e.g., bond stretching, angle and torsion scans, reaction transition states, crystal structures, etc.), ii) *training data*, properties of these atom clusters (such as energy minimized structures, relative energies for bond/angle/torsion scans, partial charges and forces) calculated using high-fidelity QM models, iii) model parameters to be optimized along with a fitness function that combines different types of training items as follows:

$$\text{Error}(m) = \sum_{i=1}^{N} \left(\frac{x_i - y_i}{\sigma_i} \right)^2 . \tag{2}$$

In Eq. (2), m is the model with a given set of force field parameters, x_i is the prediction by model m, y_i is the ground truth as calculated by QM, and σ_i^{-1} is the weight assigned to each training item.

Table 1 summarizes commonly used training data types and provides some examples. An energy-based training data item uses a linear relationship of different molecules (expressed through their identifiers) because relative energies rather than the absolute energies drive the chemical and physical processes. For structural items, geometries must be energy minimized as accurate prediction of the lowest energy states is crucial. For other training item types, energy minimization is optional but usually preferred.

2.3 Related Work

Existing force field optimization methods for ReaxFF employ gradient-free black-box optimization methods such as Genetic Algorithms (GA) and Evolutionary Algorithms (EA). These methods perform a global search in a high dimensional space and come with a high computational cost because they do not utilize any gradient information, but rather rely on error evaluations at different points in the search space.

The earliest ReaxFF optimization tool is the sequential parabolic parameter interpolation method (SOPPI) [8]. SOPPI uses a one-parameter-at-a-time approach where consecutive single parameter searches are performed until a certain

convergence criteria is met. The algorithm is simple, but as the number of parameters increases, the number of one-parameter optimization steps needed for convergence increases drastically. Also, the success of this method is very dependent on the initial guess and the order of the parameters to be optimized. Due to these drawbacks of SOPPI, various global methods such as genetic algorithms (GAs) [7, 18, 23], simulated annealing (SA) [16, 17], evolutionary algorithms (EAs) [39], particle swarm optimization (PSO) [12] and machine learning based search methods [6, 14, 27, 34] have been investigated for ReaxFF optimization. For an explanation and evaluation of the most promising of these methods, we refer readers to the prior work by Shchygol et al. [35, 36]. These methods have been proven to be successful for ReaxFF optimization. However, due to the absence of any gradient information, these global search methods require a high number of potential energy evaluations, as such they can be very costly.

With the emergence of optimized tools for machine learning to calculate gradients of complex functions automatically, a method called Intelligent-ReaxFF has been proposed to leverage these tools to train Reax force fields [14]. They use the TensorFlow library to calculate the gradients and optimize a force field. However, the method does not have the flexibility of the previously mentioned methods in terms of the training data. The force field only can be trained to match the ReaxFF energies to the absolute energies in the reference data; relative energies, charges or forces cannot be used in the training, essentially limiting its usability. Also since it does not filter out the unnecessary 2-body, 3-body and 4-body interactions before the optimization step, it is significantly slower than JAX-ReaxFF.

3 Proposed Method

3.1 Overview

JAX library performs auto-differentiation on native Python code. As such implementation of the ReaxFF energy expressions (see Eq. (1)) in Python forms the core of JAX-ReaxFF. Once the individual energy expressions and the training error function are provided, JAX can easily calculate the gradient of the training error function with respect to the ReaxFF parameters to be optimized. As mentioned earlier, atomic forces can also be part of the training dataset, these can be calculated using the gradients of ReaxFF energy expressions with respect to atom positions, too.

Molecular systems used for force field training tend to have a small number of atoms compared to regular MD runs. Using a software designed for running simulations with thousands of atoms (such as PuReMD or LAMMPS/ReaxFF) to run several small scale simulations introduces overheads. Optimizations in these software (such as optimized sparse solvers for atomic charges, fast neighbor list generation algorithms and distributed computation) would actually increase the overall run-time for small systems and result in unnecessarily complex code. Even though vanilla Python code tends to be slower than optimized Fortran or

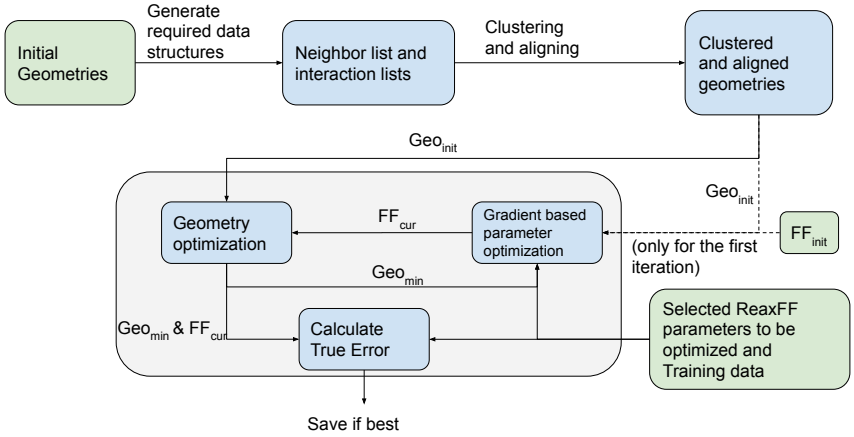


Figure 2: JAX-ReaxFF execution flow graph.

C code, when the auto-diff functionality, benefits of targeting small geometries and the just-in-compiled XLA support (discussed in Section 3.3 are considered, the advantages of JAX-ReaxFF outweighs the performance loss from not using Fortran or C.

While gradient-based optimization functionality is straight-forward to achieve using JAX as described above, there are a number of important considerations to realize an efficient (from a runtime point-of-view) and a scalable (from a memory utilization perspective) parameter optimization framework. Fig. 2 gives an overview of the task-flow in JAX-ReaxFF. After the neighbor list and interactions lists are calculated for the input geometries (Section 3.2, we cluster the inputs based on the size of their interaction lists and align them properly in memory to ensure efficient single instruction multiple data (SIMD) parallelization (Section 3.3). After these preparation steps, the main optimization loop is executed until convergence or the maximum number of optimization steps are reached (which typically takes only tens of iterations). During the parameter optimization loop, some molecules might require energy minimization before the final calculation to prevent overfitting and also to tune the parameters based on more likely states of the atom clusters as the lower energy states are more likely to be observed. Hence, the main optimization loop contains a "gradient-based optimization" step followed by a "geometry optimization" step. We discuss each step in more detail in the ensuing subsections.

3.2 ReaxFF Model Implementation

In ReaxFF implementations for MD simulations, neighbor and interaction lists are created based on the atom positions and the fixed force field parameters. Due to the dynamic nature of interactions in ReaxFF, accurate and fast calculation of energy terms (especially the higher order ones such as valence angle and torsion)

is critical. Differently from regular ReaxFF MD simulations, the force field is also dynamic during parameter optimization, hence adding to the challenges of developing an efficient implementation.

Pair-wise bonded interactions: We illustrate the challenges using bond order calculations as an example. As shown in Fig. 1, all bonded interactions depend on the corrected bond order term. Initially, if the distance between two atoms is less than a given cutoff, typically 5 Å, the uncorrected bond orders (BO) are calculated according to Eq. (3), where r_{ij} is the distance between the atom pair i-j, and p_{bo1-6}, r_o^σ, r_o^π, and $r_o^{\pi\pi}$ are the corresponding parameters.

$$BO'_{ij} = BO^{\sigma}_{ij} + BO^{\pi}_{ij} + BO^{\pi\pi}_{ij} = \exp\left[p_{bo1}\left(\frac{r_{ij}}{r_o^\sigma}\right)^{p_{bo2}}\right] + \exp\left[p_{bo3}\left(\frac{r_{ij}}{r_o^\pi}\right)^{p_{bo4}}\right] + \exp\left[p_{bo5}\left(\frac{r_{ij}}{r_o^{\pi\pi}}\right)^{p_{bo6}}\right] \tag{3}$$

Normally, if the uncorrected bond order is greater than a certain threshold, it is added to the initial bond list and subsequently bond order corrections are applied based on the neighborhood of the atoms forming the bond. In the context of parameter optimization though, whether the pair i-j will form a bond above the given threshold also depends on the values of those parameters. Furthermore, if a given molecular structure in the training dataset requires geometry optimization (as is needed for most structural properties), atom positions change as well. JAX requires an expensive recompilation if the interaction list sizes change. Therefore we create the interaction lists once before the optimization starts and use masks to ignore the unwanted elements throughout the parameter and/or geometry optimization steps. For this purpose, for every unique element pair, the maximum possible distance where a given pair can have a valid bond order is found. If some BO related parameters are included in the optimization, values which maximize the BO term are selected from the specified parameter ranges. Then through a distance scan, the maximum possible distance is determined as the cutoff for inclusion of bond orders between that pair of elements. For geometries that require minimization, the maximum calculated distance is extended by a buffer distance to be able to accommodate potential atom position changes.

Higher Order Bonded Interactions: Similar logic is applied for other types of interactions. In a given molecule with N atoms, when there is no trimming, there will be $O(N^3)$ 3-body and $O(N^4)$ 4-body interactions. Trimming these interaction lists is required to decrease the computational and memory costs. 3-body and 4-body interaction lists are built using the corrected BO term. Another threshold is applied to the bonds forming the 3-body and 4-body interaction lists. Since the higher order bonded interactions are built using the corrected BO terms, the thresholds are also based on the previously described maximum possible BO terms. Further trimming of the lists is possible by scan-

ning multiple distances and angles, but due to the increased computational complexity, only BO term based trimming is employed.

Non-bonded Interactions: It is assumed that there is a non-bonded interaction between every atom in the system since the non-bonded interaction cutoff (which is typically 10 Å) is much larger than the size of molecular/crystal structures used for training. Therefore, non-bonded interactions form an $N * N$ matrix. If the system has periodic boundary conditions, the box dimensions are a, b and c, and non-bonded interaction cutoff is r, then the size of the tensor for non-bonded interactions will become $N*N*(2*\lceil \frac{r}{a} \rceil+1)*(2*\lceil \frac{r}{b} \rceil+1)*(2*\lceil \frac{r}{c} \rceil+1)$. The part after $N * N$ accounts for the periodic boundaries.

Evaluation of the Potential Energy: Once interaction lists are created as described above, they stay constant throughout the optimization with unwanted interactions simply being masked out. Although masking wastes some computational power, it avoids the expensive reneighboring, interaction list recreation and recompilation steps as force field parameters evolve. It also leads to a simplified codebase, because the interaction list generation part can be separated from the force field optimization process. The interaction list creation is always performed on the CPU using multiprocessing, regardless of whether a hardware accelerator is used for the optimization part or not.

To calculate the potential energy, a similar approach to the standalone ReaxFF code is followed with the exception of charge equilibration. The Electronegativity Equalization Method (EEM) used for distributing partial charges requires the solution of a system of linear equations (for details see [24]) which is solved using preconditioned iterative solvers for large systems [28]. However, since the number of atoms is small for the training set structures, we use a direct *LU* factorization that is easier to implement and auto-differentiate.

3.3 Clustering and Alignment for SIMD Parallelization

JAX uses Accelerated Linear Algebra (XLA), a domain specific compiler for linear algebra, to achieve hardware portability. Using XLA, JAX-ReaxFF can easily run on multi-core CPUs, GPUs or TPUs without any code changes. JAX offers vectorization (*vmap*) and parallelization (*pmap*) support to take full advantage of the underlying architecture. While *vmap* is aimed at Single Instruction Multiple Data (SIMD) parallelism using which multiple small computations can be merged into batches to achieve high device utilization; *pmap* targets Multiple Instruction Multiple Data (MIMD) parallelism.

Our target architecture has been GPUs as they provide significant performance advantages over multi-core CPUs and have become the mainstream hardware accelerators. However, attaining high performance on GPUs requires some important considerations. Since parameter optimization requires efficient execution of several small atomic structures as opposed to running one big MD simulation in parallel, JAX-ReaxFF leverages the *vmap* support to accelerate the energy and gradient calculations. The keys for efficient vectorization

in JAX-ReaxFF are the pre-calculation of interaction lists that remain static throughout optimization (as described in the previous subsection), clustering of input geometries with similar computational demands together (explained below) and alignment of the interaction lists of geometries in the same cluster (by padding as necessary) for efficient memory accesses. As mentioned before, unwanted/unnecessary interactions in these static lists are masked during the energy and gradient calculations so that they do not affect the results.

To cluster the input geometries for efficient vectorization, a modified version of the K-Means algorithm is used. The distance formula for geometry x and cluster center y with size s_y is

$$\begin{aligned}\text{Dist}\,(x,y) = s_y \cdot \big(&c_1 \cdot \max\left(n_{\{2,x\}}, n_{\{2,y\}}\right) + c_2 \cdot \max\left(n_{\{3,x\}}, n_{\{3,y\}}\right) \\ &+ c_3 \cdot \max\left(n_{\{4,x\}}, n_{\{4,y\}}\right) \\ &+ c_4 \cdot \max\left(n_{\{5,x\}} n_{\{1,x\}}^2, n_{\{5,x\}} n_{\{1,y\}}^2\right)\big)\end{aligned} \tag{4}$$

where n_1, n_2, n_3, n_4 and n_5 are the numbers of atoms, 2-body interactions, 3-body interactions, 4-body interactions and periodic boxes within the long range cutoff, respectively. The coefficients c_1 through c_4 are indicators of the relative computational cost of their corresponding ReaxFF kernels. They can be determined empirically to accurately represent the computational costs in a given training set for a particular architecture. The cost of each cluster is determined by the computationally most expensive geometry within the cluster. This is determined by the max value between the current cluster center y and geometry x.

After initializing k cluster centers randomly, each geometry is assigned to these clusters based on the unique distance metric where the distance is an indicator of the change in computational load after assigning geometry x to cluster center y as shown in Eq. (4). The new center for each cluster is determined by the computationally most expensive geometry within the cluster; cluster centers determine the amount of padding needed for memory alignment of interaction lists for all geometries in their cluster. After centers are updated, a new iteration is performed where each geometry is reassigned to the closest cluster. Unlike the original K-Means algorithm, the order of geometries affects the result, therefore input geometries are shuffled after each iteration for randomization. The process continues until the cluster centers do not change anymore. Also to ensure high performance, the clustering algorithm is executed multiple times starting from different random initial cluster centers and the one where the total wasted computation (which can be determined by the total amount of padding) is minimal is chosen as the final clustering of the geometries. Although the algorithm does not guarantee optimality, empirical results are satisfactory.

The compilation time of JAX increases drastically with the number of clusters because JAX unrolls the loop that iterates through the clusters during compilation. Also, if the wasted computation does not increase significantly, a smaller number of clusters is more preferable for GPUs since improving SIMD parallelism is easier within clusters. For these reasons, the number of clusters

Algorithm 1 Clustering Algorithm

$C_{best} \leftarrow$ Keep track of the best so far
for $r = 1, 2, \ldots R$ **do**
    $C_{cur} \leftarrow$ Initialize the cluster centers by selecting a random geometry as the center for each cluster
    for $i = 1, 2, \ldots I$ **do**
        $C_{prev} \leftarrow C_{cur}$
        Shuffle G
        for each $g \in G$ **do**
            Assign g to c_i where Dist(g, c_i) is minimum
            Update the cluster centers
        end for
        if $C_{cur} == C_{prev}$ **then**
            Break
        end if
    end for
    if Cost(C_{cur}) < Cost(C_{best}) **then**
        $C_{best} \leftarrow C_{cur}$
    end if
end for

is selected automatically based on Algorithm 2. Unless the computational gain from a higher number of cluster centers is not significant, smaller number of clusters is preferred.

Algorithm 2 Clustering Algorithm 2

$k_{max} \leftarrow$ Maximum number of clusters
$R \leftarrow$ Number of repetitions for the clustering algorithm
$I \leftarrow$ Number of iterations for the clustering algorithm
$C_{selected} \leftarrow$ Selected clustering of the geometries
for $k = 1, 2, \ldots k_{max}$ **do**
    $Cost_k, C_k \leftarrow Clustering(G, k, I, R)$
    if $|Cost_k - Cost_{k-1}|/Cost_{k-1} < tolerance$ or $k == k_{max}$ **then**
        $C_{selected} \leftarrow C_k$
        Break
    end if
end for

3.4 Gradient Based Local Optimization

After the final clusters are formed, parameter optimization is performed using gradient based local optimizers with multi-start as depicted in Fig. 2. Vectorization based parallelism is employed for both energy minimization and parameter

optimization steps shown in this figure.

For gradient based optimization to work, JAX traces the error function from Eq. (2) and computes the gradients of the parameters. However, since typically many geometries require geometry optimization, tracing the gradients through the optimization step is more error prone due to the complex functional form of the ReaxFF. To remedy this problem, we separate the geometry optimization from the error minimization. The error function without the geometry optimization can be thought as a surrogate model since it is a fast way to approximate the true error where the geometry optimization is done as well. The approach accelerates the training and does not require tracing the gradients through the geometry optimization step.

The optimization algorithm starts from the initial geometries (Geo_{init}) and the initial force field (FF_{init}) then the force field is iteratively improved. For each iteration of the training loop shown in Algorithm 3, two different local optimizations are performed, one being local geometry optimization using the steepest descent method and the main one being minimization of the fitness error on the energy minimized geometries by updating the force field parameters using various local optimization method such as L-BFGS-B and SLSQP. Both of these methods are classified as quasi-Newton methods where the Hessian matrix is approximated by successive gradient calculations [20, 41]. Error minimization step uses the Geo_{min} and the optimized force field (FF_{cur}) from the last iteration and after applying the selected gradient-based algorithm, FF_{cur} gets updated with the newly trained force field. This step uses the surrogate model where the error is calculated with only the single step calculations. After that the geometry optimization step starts from Geo_{init} and yields optimized geometries (Geo_{min}) using FF_{cur}. The true error is calculated right after the geometry optimization, if there are any geometries that require it. After each iteration, the true error (E_{cur}) for FF_{cur} on the training data is calculated. If E_{cur} is lower than the lowest error so far (E_{best}), FF_{cur} is saved as the best force field (FF_{best}). The error on the surrogate gets closer to the true error as parameters converge because changes in parameters become minimal. One disadvantage of separating the energy minimization from the local optimization is that the fitness score for the geometry items will be be ignored by the local optimization since the atom positions will not change. This introduces a discrepancy between the true error and the surrogate one. However, if the training data has multiple items related to the geometry based items as a result of potential energy surface scans (PES), the discrepancy could be minimized. As it is demonstrated through numerical experiments, the surrogate approach works well in practice for a variety of training tasks which have geometry based items.

4 Evaluation

We evaluate the performance of JAX-ReaxFF, as well as the quality of the resulting force fields using published datasets with different characteristics.

Algorithm 3 Gradient Based Local Optimization

$FF_{cur} \leftarrow FF_{init}$
for $iteration = 1, 2, \ldots$ **do**
    $FF_{cur} \leftarrow$ Locally minimize the error through the selected gradient-based algorithm using Geo_{min} and starting from FF_{cur}. Geo_{min} is fixed.
    $Geo_{min} \leftarrow$ Geometry optimize the structures starting from the initial geometries Geo_{init} with the current model FF_{cur}
    $E_{cur} \leftarrow$ Calculate the current error using Geo_{min} and FF_{cur}
    if $E_{cur} < E_{best}$ **then**
        $E_{best} \leftarrow E_{cur}$
        $FF_{best} \leftarrow FF_{cur}$
    end if
    if $|E_{cur} - E_{prev}|/E_{prev} < 0.001$ **then**
        $FF_{cur} \leftarrow$ Add small uniform noise to FF_{best}
    end if
    $E_{prev} \leftarrow E_{cur}$
end for

4.1 Experimental Setup

Training Tasks We identified three training tasks[2] that form a well-rounded test bench with their varying degrees of complexity. These tasks include different system types (Cobalt, a metal; Silica, an amorphous material; Disulfide, a molecular system), different types and numbers of items in the training datasets, and different number of parameters with their respective ranges to be optimized. While structures in the Cobalt and Silica datasets mostly require energy minimization, those in the Disulfide case mostly require single-step energy evaluations. Table 2 summarizes the specifications of the selected training tasks. JAX-ReaxFF currently does not support training items with simulation cell optimization, as such these are ignored. This only affects the Silica dataset which has 5 of them (out of 296 cases).

Hardware Setup All the CPU experiments reported here were conducted on a computer with two Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz (2x14 cores) CPU processors, 128 GB 2133 MHz DDR4 Ram. The GPU experiments were conducted on a computer with Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz, 16 GB 3000 MHz DDR4 Ram and single 1080-TI GPU card (11 GB GDDR5X memory). For the baseline methods, OGOLEM version 1.0with sPuReMD backend is used. The proposed method is implemented in Python 3.7 and utilizes JAX version 0.1.76 and NumPy version 1.16.4.

[2]The datasets which are provided in the Supplementary Information of [35] can be downloaded from https://ndownloader.figstatic.com/files/18698201.

Table 2: Datasets. N_{par} is the number of parameters to optimize, N_{strc} is the number of structures in the training dataset and N_{minim} is the number of geometries to be energy minimized. C, G, F, P and E are the number of charge based training items, geometry based items, force based items, cell parameter based items and energy based items, respectively.

Training Data	N_{par}	N_{strc}	N_{minim}	C	G	F	P	E
Cobalt [22]	12	146	130	0	0	0	0	144
Silica [9]	67	302	221	5	26	0	6	265
Disulfide [26]	87	231	10	0	255	4401	0	219

Baseline Results We compare the performance and training accuracy of JAX-ReaxFF to those of methods by Shchygol et al. [35], namely the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), Monte Carlo Force Field (MCFF) optimizer and Genetic Algorithm (GA) techniques described therein. MCFF optimizer utilizes the simulated annealing approach to slowly modify the parameters and act based on the change in the error value. The remaining two approaches are population based and are inspired by the basic principles of biological evolution. In GA, a population of candidate solutions for a given optimization problem is evolved towards better solutions. Typically, evolution happens through random mutations and cross-overs between selected candidate solutions. In CMA-ES, new solutions are sampled from a multivariate normal distribution. The pairwise dependencies between the parameters are captured by the covariance matrix and as the search progresses, CMA-ES updates the covariance matrix. All three approaches use ReaxFF model as a black box and find the direction solely from the function evaluations. Shchygol et al. [35] has compared these methods on different training tasks without focusing on tuning them and repeated the experiments multiple times with different starting conditions. Since they have provided an important test bench to compare different optimizers for ReaxFF, we follow the same approach to evaluate JAX-ReaxFF.

Since the exact software and hardware from [35] are not accessible, execution times for the baseline methods are approximated on the hardware described before.We have calculated the time per true error evaluation for each training task using OGOLEM with sPuReMD backend and multiplied this by the total number of error evaluations presented in [35]. This approximation is a lower bound for CMA-ES and MCFF since they have a lower level of parallelism unlike genetic algorithm where each evaluation is independent from each other.

The initial guess is an important factor which could change the results drastically. It is especially important for gradient based optimizations because these methods cannot move through the space freely as they need to follow the direction of the gradients. To show the capabilities of JAX-ReaxFF, we experimented with all three initialization methods from Shchygol et al. [35], namely random, educated and literature based initial guesses. For random initial guesses, initial values are sampled from a uniform distribution based on the given parameter

ranges. To produce educated guesses, prior information from the previous related force fields is used as it is described further in [35]. For the literature based initial guesses, the force fields developed previously using the same training data are used. To give more reliable results, each initialization method is repeated ten times. For the educated and literature based initial guesses, small amount of uniform random noise is added to the parameters without violating the range restrictions. For each parameter p, the noise value is sampled from $[\frac{-1k}{10}, \frac{1k}{10}]$ where k is the length of the given range for parameter p. For the random initial guesses, uniform sampling is done ten times to produce the guesses.

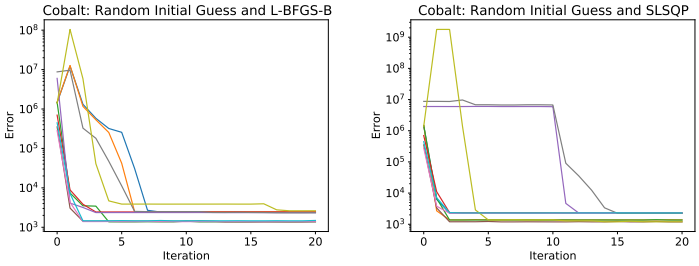
4.2 Runtime and Training Evaluation

In JAX-ReaxFF, as mentioned above, two different gradient based optimization algorithms are available, L-BFGS-B and SLSQP. For both L-BFGS-B and SLSQP, the maximum number iterations is set to 100. This iteration number is for the step 3 of Algorithm 3. For L-BFGS-B, the maximum number of iterations for the line search is set to 20 and the maximum number of variable matrix corrections to approximate the Hessian matrix is set to 20. For the rest of the control parameters, the default values from the SciPy library are used. The iteration count for the main optimization loop of Algorithm 3 is set to 20 where the local error minimization and the geometry optimization steps are iteratively repeated this many times. Therefore, for all of the experiments for Jax-ReaxFF, the true error calculation with geometry optimization is done 20 times since the local error minimization only uses single step calculations.

4.2.1 Cobalt Training

Cobalt testcase has only energy-based training items. About 90% of these items require energy minimization, yet the training error does not fluctuate as shown in Fig. 3. This shows that the surrogate error is close to the true error for this dataset. Otherwise, the error would fluctuate between iterations since the surrogate error is used for the error minimization in each iteration. For some of the random runs, SLSQP does not show any progress initially. One possible explanation is that when the initial parameters are from a non-smooth part of the optimization space that cause high gradients, the optimizer fails to escape (Fig. 3b). Small noise that is added when a stall in progress is detected stimulates progress as expected.

In Table 3, we compare the convergence of JAX-ReaxFF against the black box approaches. We observe significantly faster convergence in terms of both the number of evaluations required as well as the time taken, while obtaining similar or better training errors as measured by best and median scores. The entire optimization process takes slightly more than a minute on the GPU.



(a) L-BFGS-B (b) SLSQP

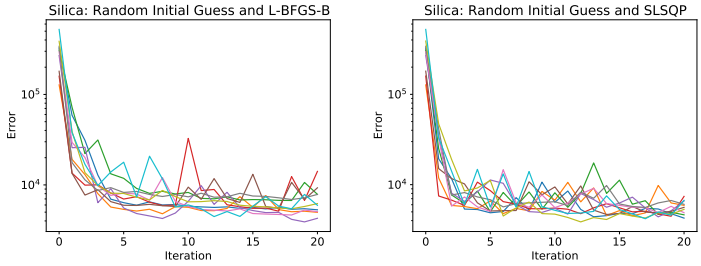
Figure 3: Convergence of the local optimizers for the Cobalt dataset

Table 3: Cobalt training results.

Method	Initial Guess	Best Score	Median Score	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B	rand	1368	2334	480	20	23.5	1.2
	edu	1352	1499	418			
	lit	1366	1446	450			
SLSQP	rand	1191	2253	513	20	24.8	1.3
	edu	1168	1188	618			
	lit	1187	1189	637			
Genetic Algorithm	rand	1346	1645	-	200k	3913	-
	edu	1349	1424				
	lit	1345	1483				
CMA-ES	rand	1150	1894	-	45k	880	-
	edu	1159	1491				
	lit	1180	2320				
MCFF	rand	1422	2104	-	45k	880	-
	edu	1532	2092				
	lit	1360	1405				

4.2.2 Silica

The silica training set includes energy, charge and geometry matching based items. 73% of the items require energy minimization. As shown in Fig. 4, unlike the Cobalt case, the error fluctuates more between iterations, possibly because of unstable geometries as parameters are converging (note that here the number of parameters to be optimized are significantly higher than the Cobalt case) and the presence of geometry matching items in the training set. Although the single point evaluation based surrogate model ignores the simulation cell optimization items, the proposed method is able to minimize the error comparable to that of black box methods, while taking a fraction of their execution times.



(a) L-BFGS-B (b) SLSQP

Figure 4: Convergence of the local optimizers for the Silica dataset

Table 4: Silica training results.

Method	Initial Guess	Best Score	Median Score	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B	rand	3901	5214	1865	20	25.0	1.6
	edu	4143	4467	1385			
	lit	4315	5068	1929			
SLSQP	rand	3870	4498	2962	20	31.9	2.0
	edu	3977	4540	2839			
	lit	3857	4534	2938			
Genetic Algorithm	rand	3577	3738	-	200k	1632	-
	edu	3705	3817				
	lit	3593	3721				
CMA-ES	rand	3739	4753	-	45k	367	-
	edu	3747	4122				
	lit	3793	4298				
MCFF	rand	5059	6584	-	45k	367	-
	edu	5632	7127				
	lit	4885	6126				

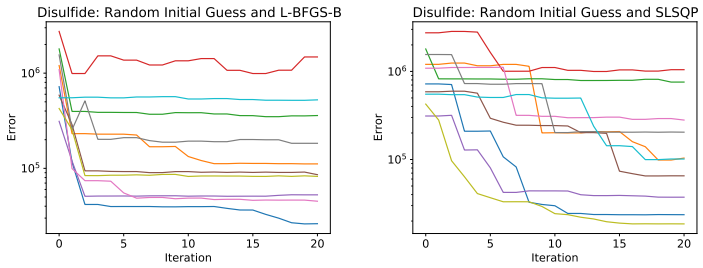
4.2.3 Disulfide

The disulfide training data is drastically different from the previous ones since it uses force matching for model optimization. In JAX-ReaxFF, forces are calculated by taking the derivative of the potential energy expressions with respect to atom positions

$$F_x = \frac{\partial E_p}{\partial x}$$

$$\frac{\partial (F_x - F_t)^2}{\partial p} = \frac{\partial (\frac{\partial E_p}{\partial x} - F_t)^2}{\partial p}$$

where F_x is the 3-dimensional force vector for atom x, F_t is the target force vector from the training dataset and p is the model parameter to be optimized. $\frac{\partial (F_t - F_x)^2}{\partial p}$ gives the gradients for the force matching items in the objective func-



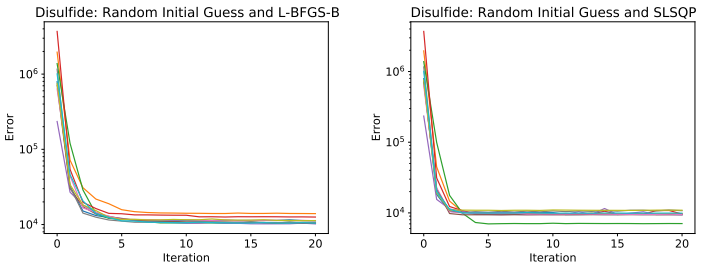
(a) L-BFGS-B (b) SLSQP

Figure 5: Convergence of the local optimizers for the Disulfide dataset before the modification

tion. However, we have observed that the final gradients for parameters from Eq. (3) result in extremely high values, $\sim 10^{17}$, while the other gradients are much lower. These high gradients stop the local optimizers from doing any progress as seen in Fig. 5. Since bond order parameters form the core of the dynamic bond concept in ReaxFF and therefore affect all types of bonded interactions, highly sensitive functional form for the bond order calculation [11], using relatively large parameter ranges and initial guesses being off of their ideal values are likely responsible for this behaviour. While not ideal, we excluded the bond order parameters from optimization and fixed their values to the literature ones. Among 87 parameters, 18 parameters are removed and the optimization is performed again with the remaining 69 parameters. As shown in Fig. 6, this improves results drastically, and JAX-ReaxFF can attain better scores than the baseline methods in significantly shorter time, but it should be noted that the comparison cases include all 87 parameters. This situation shows that gradient-based optimization is prone to failures for parameters with a large influence on the objective function like the bond order parameters. However, in practice training items for bond order optimization are easy to construct and their optimization can be performed independently prior to the actual optimization task. For this reason, we do not see this issue to be a major limitation for JAX-ReaxFF's practical use.

4.3 Force Field Evaluation

Fitness scores of optimized parameters can be seen as proxies, but the quality of the resulting force field parameter sets ultimately need to be validated through actual MD simulations and comparisons against experimental and/or QM data. These results show that the newly trained force fields are on par with those from the literature [35], while their training is 1-3 orders of magnitude faster.



(a) L-BFGS-B (b) SLSQP

Figure 6: Convergence of the local optimizers for the Disulfide dataset after the modification

Table 5: Disulfide training results. *The results are from the modified version of the training.

Method	Initial Guess	Best Score	Median Score	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B*	rand	10198	10920	1660	20	9.7	0.9
	edu	10313	10631	1600			
	lit	10438	10803	1503			
SLSQP*	rand	6986	9488	1187	20	8.9	0.8
	edu	9306	9635	1234			
	lit	10304	10494	1901			
Genetic Algorithm	rand	19285	20384	-	340k	878	-
	edu	18054	20150				
	lit	18524	21206				
CMA-ES	rand	8052	11371	-	45k	116	-
	edu	8727	11105				
	lit	9284	11120				
MCFF	rand	8507	11893	-	45k	116	-
	edu	9608	13393				
	lit	10605	13625				

5 Force Field Evaluation

MD simulations in this work are performed using Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) that is a molecular dynamics program from Sandia National Laboratories [30]. A relatively short time step of 0.5 fs was used in all the simulations. This is the recommended setting for ReaxFF simulations of systems that don't include light atoms like Hydrogen. All NVT ensemble (constant number of atoms, volume and temperature) simulations were performed using Nose-Hoover thermostat to control the temperature with a temperature damping parameter of 100 fs which determines how rapidly the temperature is relaxed. All NPT ensemble (constant number of atoms, pressure and temperature) simulations were performed using Nose-Hoover thermostat

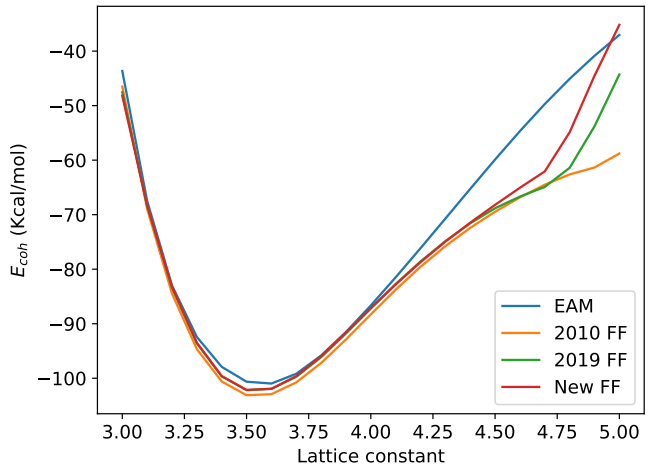


Figure 7: Variations in pure Cobalt single fcc crystal cohesive energy by variations of the lattice constant.

to control the temperature with a temperature damping parameter of 100 fs and Nose-Hoover barostat to control the pressure with a temperature damping parameter of 1000 fs.

5.1 Molecular dynamics simulations of pure Cobalt structure

We investigated the crystal lattice constant correlation with cohesive energy in crystals of fcc Cobalt for validation. The lattice constant was changed from 3Å to 5Å and the associated lattice cohesive energies were recorded (Fig. 7). The results of the fitted force field with the best fitness score were compared to two previously trained ReaxFF force fields for Cobalt [22, 35] and embedded atom method (EAM) force field [31]. To validate the quality of the force field in capturing the dynamics behavior, the annealing loop was generated for a pure Cobalt crystal structure and was compared to the available force fields. A cubic simulation box of 5x5x5 ideal fcc Cobalt unit cells was generated for annealing simulations using NPT ensemble between 1000K-3000K. After the NPT equilibration of the pure Cobalt crystal at 1000K, the system was subjected to NPT ensemble annealing between 1000K-3000K by 10 K/ps heating and cooling rate. A time step of 0.5 fs was used for the simulations. The changes in the system energy during this annealing loop is shown in Fig. 8. Three ReaxFF force fields showed similar dynamic evolution behavior for the pure Cobalt structure while the EAM force field showed a different dynamic evolution (Fig. 9).

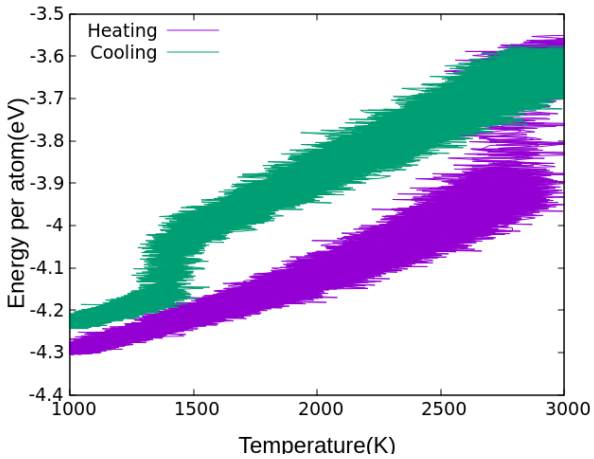


Figure 8: Annealing loop of a 5x5x5 fcc Cobalt crystal including 500 atoms using fitted ReaxFF force field with heating and cooling rate of 10 K/ps.

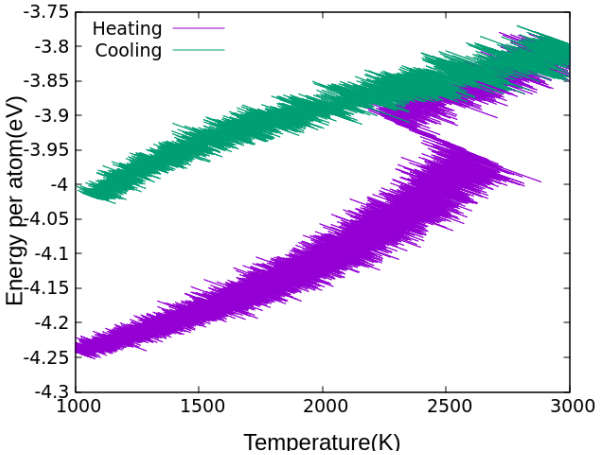


Figure 9: Annealing loop of a 5x5x5 fcc Cobalt crystal including 500 atoms using EAM force field with heating and cooling rate of 10 K/ps.

After completion of the annealing loop, structural evaluation showed that using the ReaxFF force fields resulted a considerable recrystallization in the pure Cobalt structure, while recrystallization was not observed when EAM force field was utilized (Fig. 10).

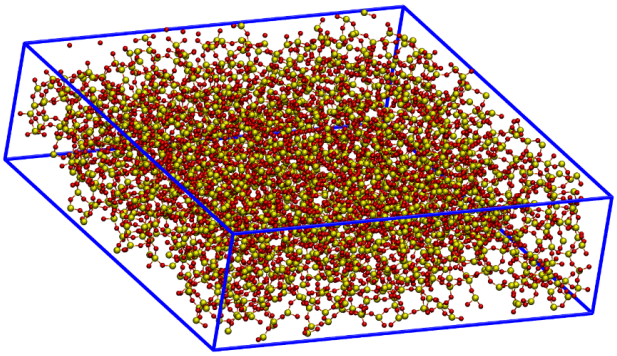


Figure 11: The amorphous silica structure including 2000 SiO_2 molecules and total of 6000 atoms. Silicon atoms are shown with yellow color and Oxygen atoms are shown with red color.

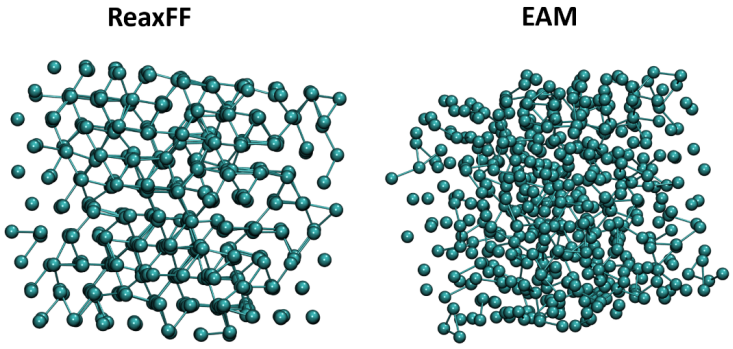


Figure 10: Final configurations of pure Co fcc crystals after annealing loop with 1000K-3000K temperature range.

5.2 Molecular dynamics simulations of pure Silica structure

To evaluate the quality of the silica optimized force field with the best fitness score, the amorphous silica structure introduced in the Fogarty et al. [9] was reconstructed. The amorphous silica system included 2000 SiO_2 molecules with an initial density of 2.2 g/cm^3 (Fig. 11). The amorphous silica system was energy-minimized to eliminate initial bad contacts. The system was then annealed twice between 300K and 4000K. The first annealing loop was performed using NVT ensemble with heating and cooling rate of 37 K/ps. The second annealing loop was performed in NPT ensemble between 300K-4000K using Nose–Hoover thermostat and barostat 1.01325 bar pressure. Similar to the NVT annealing,

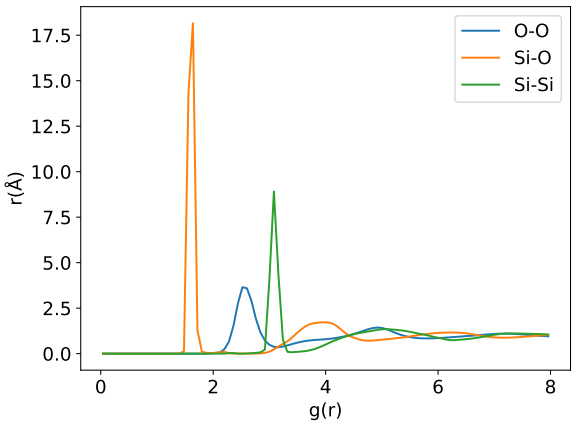


Figure 12: Radial distribution function of silicon-oxygen, oxygen-oxygen, and silicon-silicon for silica structure at the end of annealing and equilibration.

the heating and cooling rate was 37 K/ps. Finally, the silica system was equilibrated in NPT ensemble using T=300K and P=1.01325 bar for additional 200 ps as the production run. These calculations were performed using our fitted force field and two previous ReaxFF force fields introduced for such systems. The properties of the final configuration of these silica structures are compared in Table 6. The radial distributions of the final configuration of silica structure equilibrated using our fitted force field for Si-O, O-O and Si-Si are shown in Fig. 12. These results show good force field quality for silica structure using JAX-ReaxFF.

Table 6: Silica properties calculated using three different force fields. The experimental value reported for silica density is 2.2 g/cm^3 [25]

Property	2010 FF [9]	2019 FF [35]	New FF
Density (g/cm^3)	2.19	2.31	2.23
Si coordination	3.99	3.94	3.97
O coordination	1.99	1.97	1.99

5.3 Molecular dynamics simulations of molecules with Sulfur bonds

As the validation data is provided besides the training data for this task [26], the force field with the best fitness score on both the training and validation data is selected to limit overfitting. To test the validity of the force field containing Sulfur parameters updated using our proposed training method, we performed

minimum energy structure search for single molecules with different restraints. The results of the fitted force field were compared to two previously trained ReaxFF force fields[35, 26]. The restraints are applied on S-S bond of dimethyl disulfide (DMDS), S-C bond of dimethyl thioether (DMTE), H-S-H angle of Hydrogen sulfide (H_2S) and H-S-S-H torsion angle of Hydrogen disulfide (H_2S_2). These potential energy graphs are shown in Fig. 13.

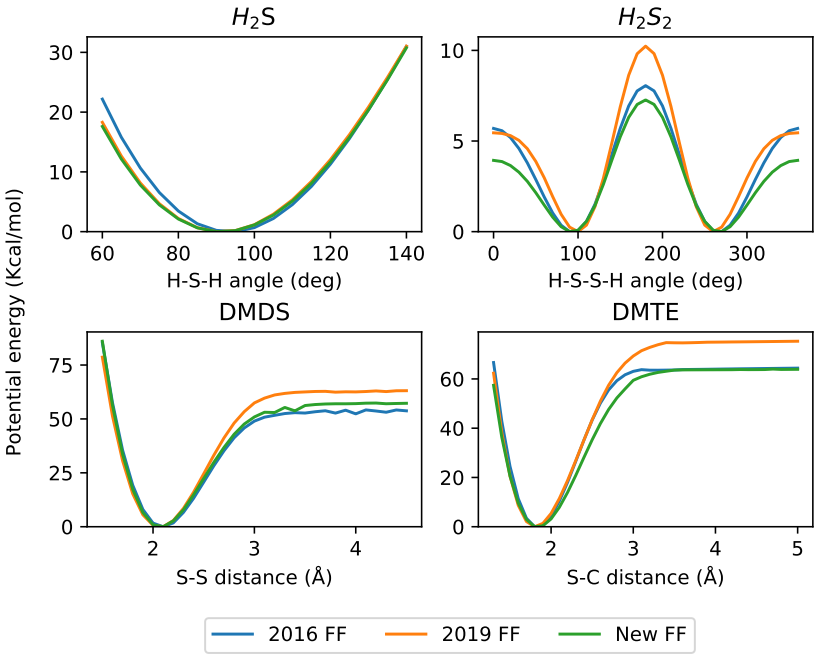


Figure 13: Potential energy graphs of energy minimized molecules including Sulfur bonds with different restraints, calculated with the updated force field and previously trained force fields.

6 Conclusion

We presented a new software called JAX-ReaxFF that enables extremely fast optimization of Reax force field parameters by leveraging recent advances in machine learning. JAX-ReaxFF uses several innovative techniques for high performance on architectures with GPUs. Clustering similar geometries together to maximize the SIMD parallelism while limiting the padding for alignment yields high parallelism, especially for single step evaluations. As it is described in Algorithm 3, by using single step energy evaluation based approximations to the error function and gradient information about the search space, we are able to decrease the convergence time significantly with the help of GPU acceleration. We have empirically showed that even though the local optimizer is

not fully aware of the geometry optimization, the overall algorithm converges with changes in parameter space becoming minimal as the algorithm progresses. Based on extensive experiments, we demonstrated that even if the starting initial guess is a poor one, gradient based local optimizers are able to improve the fitness of the force field drastically (with one exception being the core bond order parameters). Combined together, these innovations allow users to optimize Reax force fields in mere minutes. This is notable because existing methods require several days for obtaining essentially same quality force fields. Finally, JAX-ReaxFF provides a utility not available in other similar tools; its auto-diff functionality enables the study of the new functional forms for the interactions of the ReaxFF model without explicitly implementing the force calculations and the optimizer, since both forces and parameter gradients can automatically be calculated by JAX.

References

[1] Hasan Metin Aktulga et al. "Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques". In: *Parallel Computing* 38.4-5 (2012), pp. 245–259.

[2] Hasan Metin Aktulga et al. "Reactive molecular dynamics: Numerical methods and algorithmic techniques". In: *SIAM Journal on Scientific Computing* 34.1 (2012), pp. C1–C23.

[3] James Bradbury et al. "JAX: composable transformations of Python+ NumPy programs, 2018". In: *URL http://github. com/google/jax* (2020), p. 18.

[4] Donald W Brenner et al. "A second-generation reactive empirical bond order (REBO) potential energy expression for hydrocarbons". In: *Journal of Physics: Condensed Matter* 14.4 (2002), p. 783.

[5] David A Case et al. *Amber 2021*. University of California Press, 2021.

[6] Chaitanya M Daksha et al. "Automated ReaxFF parametrization using machine learning". In: *Computational Materials Science* 187 (), p. 110107.

[7] Mark Dittner et al. "Efficient global optimization of reactive force-field parameters". In: *Journal of computational chemistry* 36.20 (2015), pp. 1550–1561.

[8] Adri CT van Duin, Jan MA Baas, and Bastiaan Van De Graaf. "Delft molecular mechanics: a new approach to hydrocarbon force fields. Inclusion of a geometry-dependent charge calculation". In: *Journal of the Chemical Society, Faraday Transactions* 90.19 (1994), pp. 2881–2895.

[9] Joseph C Fogarty et al. "A reactive molecular dynamics simulation of the silica-water interface". In: *The Journal of chemical physics* 132.17 (2010), p. 174704.

[10] Daan Frenkel and Berend Smit. *Understanding molecular simulation: from algorithms to applications*. Vol. 1. Elsevier, 2001.

[11] David Furman and David J Wales. "A well-behaved theoretical framework for ReaxFF reactive force fields". In: *The Journal of Chemical Physics* 153.2 (2020), p. 021102.

[12] David Furman et al. "Enhanced particle swarm optimization algorithm: Efficient training of reaxff reactive force fields". In: *Journal of chemical theory and computation* 14.6 (2018), pp. 3100–3112.

[13] Julian D Gale, Paolo Raiteri, and Adri CT van Duin. "A reactive force field for aqueous-calcium carbonate systems". In: *Physical Chemistry Chemical Physics* 13.37 (2011), pp. 16666–16679.

[14] Feng Guo et al. "Intelligent-ReaxFF: Evaluating the reactive force field parameters with machine learning". In: *Computational Materials Science* 172 (2020), p. 109393.

[15] Berk Hess et al. "GROMACS 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation". In: *Journal of chemical theory and computation* 4.3 (2008), pp. 435–447.

[16] Pierre O Hubin et al. "Parameterization of the ReaxFF reactive force field for a proline-catalyzed aldol reaction". In: *Journal of Computational Chemistry* 37.29 (2016), pp. 2564–2572.

[17] Eldhose Iype et al. "Parameterization of a reactive force field using a Monte Carlo algorithm". In: *Journal of computational chemistry* 34.13 (2013), pp. 1143–1154.

[18] Andres Jaramillo-Botero, Saber Naserifar, and William A Goddard III. "General multiobjective force field optimization framework, with application to reactive force fields for silicon carbide". In: *Journal of Chemical Theory and Computation* 10.4 (2014), pp. 1426–1439.

[19] Jaewoon Jung et al. "Scaling molecular dynamics beyond 100,000 processor cores for large-scale biophysical simulations". In: *Journal of computational chemistry* 40.21 (2019), pp. 1919–1930.

[20] Dieter Kraft et al. "A software package for sequential quadratic programming". In: (1988).

[21] Sudhir B Kylasa, Hasan Metin Aktulga, and Ananth Y Grama. "PuReMD-GPU: A reactive molecular dynamics simulation package for GPUs". In: *Journal of Computational Physics* 272 (2014), pp. 343–359.

[22] Matthew R LaBrosse, J Karl Johnson, and Adri CT van Duin. "Development of a transferable reactive force field for cobalt". In: *The Journal of Physical Chemistry A* 114.18 (2010), pp. 5855–5861.

[23] Henrik R Larsson, Adri CT van Duin, and Bernd Hartke. "Global optimization of parameters in the reactive force field ReaxFF for SiOH". In: *Journal of computational chemistry* 34.25 (2013), pp. 2178–2189.

[24] Wilfried J Mortier, Swapan K Ghosh, and S Shankar. "Electronegativity-equalization method for the calculation of atomic charges in molecules". In: *Journal of the American Chemical Society* 108.15 (1986), pp. 4315–4320.

[25] RL Mozzi and n BE Warren. "The structure of vitreous silica". In: *Journal of Applied Crystallography* 2.4 (1969), pp. 164–172.

[26] Julian Müller and Bernd Hartke. "ReaxFF reactive force field for disulfide mechanochemistry, fitted to multireference ab initio data". In: *Journal of chemical theory and computation* 12.8 (2016), pp. 3913–3925.

[27] Hiroya Nakata and Shandan Bai. "Development of a new parameter optimization scheme for a reactive force field based on a machine learning approach". In: *Journal of computational chemistry* 40.23 (2019), pp. 2000–2012.

[28] Kurt A O'Hearn, Abdullah Alperen, and Hasan Metin Aktulga. "Fast Solvers for Charge Distribution Models on Shared Memory Platforms". In: *SIAM Journal on Scientific Computing* 42.1 (2020), pp. C1–C22.

[29] James C Phillips et al. "Scalable molecular dynamics with NAMD". In: *Journal of computational chemistry* 26.16 (2005), pp. 1781–1802.

[30] Steve Plimpton. "Fast parallel algorithms for short-range molecular dynamics". In: *Journal of computational physics* 117.1 (1995), pp. 1–19.

[31] GP Purja Pun and Y Mishin. "Embedded-atom potential for hcp and fcc cobalt". In: *Physical Review B* 86.13 (2012), p. 134116.

[32] Amit Sabne. "XLA: Compiling Machine Learning for Peak Performance". In: (2020).

[33] Thomas P Senftle et al. "The ReaxFF reactive force-field: development, applications and future directions". In: *npj Computational Materials* 2.1 (2016), pp. 1–14.

[34] Mert Y Sengul et al. "INDEEDopt: a deep learning-based ReaxFF parameterization framework". In: *npj Computational Materials* 7.1 (2021), pp. 1–9.

[35] Ganna Shchygol et al. "ReaxFF Parameter Optimization with Monte-Carlo and Evolutionary Algorithms: Guidelines and Insights". In: *Journal of Chemical Theory and Computation* 15.12 (2019), pp. 6799–6812.

[36] Ganna Shchygol et al. "Systematic comparison of Monte Carlo Annealing and Covariance Matrix Adaptation for the optimization of ReaxFF parameters". In: *ChemRxiv* (2018).

[37] JJPRB Tersoff. "Modeling solid-state chemistry: Interatomic potentials for multicomponent systems". In: *Physical review B* 39.8 (1989), p. 5566.

[38] Aidan P Thompson et al. "LAMMPS-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Computer Physics Communications* 271 (2022), p. 108171.

[39] Tomas Trnka, Igor Tvaroska, and Jaroslav Koca. "Automated training of ReaxFF reactive force fields for Energetics of Enzymatic reactions". In: *Journal of chemical theory and computation* 14.1 (2018), pp. 291–302.

[40] Adri CT Van Duin et al. "ReaxFF: a reactive force field for hydrocarbons". In: *The Journal of Physical Chemistry A* 105.41 (2001), pp. 9396–9409.

[41] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997), pp. 550–560.