Performance Enhancement of the Dynamical Optimization Framework for Simulating Ions near Polarizable Nanoparticles

***Abstract*—Simulating the dynamics of ions near polarizable nanoparticles gets notoriously difficult due to the need to solve the Poisson equation at every simulation timestep. Recently, a dynamical optimization framework was developed that bypassed this obstacle by representing the polarization charge density as virtual dynamic variables, and evolving them in parallel with the physical dynamics of ions. In this paper, the performance of this framework is enhanced using a judicious combination of parallel computing and machine learning techniques. We develop a hybrid OpenMP/MPI implementation scalable for over 1000 cores that generates stable simulation of thousands of ions and virtual variables for over 10 million timesteps. We use machine learning to predict the virtual system parameters that optimize the stability and accuracy of the simulation. The enhanced framework improves performance by over 175$\times$ depending on system size and configuration. Extraction of ion distributions near nanoemulsions demonstrate the success of the methodology.**

***Index Terms*—MPI/OpenMP, Hybrid Parallelization, Machine Learning, Nanoscale Simulations**

I. Introduction and Motivation

Accurate knowledge of ion distributions and associated electrostatic interactions is necessary to understand nanoscale phenomena such as protein conformational changes [1], DNA precipitation [2], nanoparticle self-assembly [3], stability of emulsions [4], and charging/discharging processes in supercapacitor systems [5]. This information helps guide the synthesis of materials in biomimetic nanocontainers and extraction of heavy metal ions from wastewater, and is useful to understand biological processes such as cell signaling and transport of ions across cell membrane.

Many of the aforementioned biological and synthetic nanoparticle (NP) systems get polarized in the presence of electric fields generated by the ions and/or external charged objects. Examples include proteins within an aqueous cellular medium, emulsions where oil and water are partitioned, and gold NPs dispersed in water. Simulating the dynamics of ions in the presence of these polarizable NPs using coarse-grained models is challenging due to the need to compute polarization (induced) charges in order to propagate the ion configuration. This computation typically involves solving a second-order differential equation, the Poisson equation, in 3-dimensional space at *each* simulation timestep. This makes the use of conventional nanoscale simulation methods to extract ion distributions very time consuming and inefficient. Because of these computational challenges, the problem of simulating ions near polarizable NPs continues to be a subject of intense research [6]–[11]. Resolving these challenges by enhancing a recently introduced simulation framework [12]–[14] via a judicious combination of parallel computing and machine learning methods constitutes the main focus of this paper.

A dynamical optimization framework was developed that enabled the replacement of the expensive solution of the Poisson equation at each simulation step with an on-the-fly computation of polarization effects [12], [13]. In this simulation framework, inspired by the Car-Parrinello method for simulating ion-electron systems, the energy functional of the polarization charge density was dynamically optimized resulting in the physical dynamics of ions in parallel with the update of the virtual variables characterizing the polarization (induced) charge density. The virtual system was evolved in a manner that kept the induced charges close to the free-energy minimum ("ground state") corresponding to each new ionic configuration visited along the dynamics. The advantages associated with the on-the-fly computation of polarization effects in conjunction with the reduction in computational costs achieved by solving for the scalar induced density variable enabled the study of electrolyte ion solutions near polarizable NPs [12], [14] using this framework.

The effects of nanoconfinement and ion concentrations in synthetic and biological NP systems typically lead to ion distributions that reach constant bulk value within a few nanometers away from the NP. This enables the study of ion densities near NP surfaces to be performed by including thousands of ions in the simulation cell. Methods with the scaling of $O(n^2)$ in the number of particles are feasible for such applications and have been often employed [7]–[9], [15], [16]. The original dynamical optimization framework employed a shared memory model (OpenMP) to reduce the computing time associated with evaluating the forces and energies necessary to propagate the dynamics of the extended system involving the ions and virtual variables. In this paper, we outline a hybrid implementation scheme that utilizes MPI for distributed memory parallelism, OpenMP for shared memory parallelism, and memory optimization techniques to enhance the performance of the framework. Using this hybrid approach, the computational time for simulating 60 ions up to 10 ns was reduced from 268 hours to 12 hours. For 1454 ions, this time was reduced from 6112 hours to 60.5 hours. The applicability of the original method was extended to systems with up to 4362 ions and surface mesh points ranging from 482 to 1692.

The stability, accuracy, and efficiency of the dynamical

optimization framework depends on the correct initialization of the virtual system. In the original implementation, the parameters characterizing the virtual system are found by a tedious process of trial and error that was informed by in-domain experience. The applicability of the original method is limited by the absence of a framework that automates the process of selecting the "good" parameters *ab initio*. Inspired by the recent progress in the use of machine learning (ML) in materials physics and engineering applications [17]–[19], we describe an approach that employs ML to predict the initial parameters characterizing the virtual system that keep the polarization charge density close to the ground state of the configuration visited by the ions during the entire simulation run. The Neural Network (NN) based ML procedure was able to predict the optimal parameters with 95.6% accuracy.

The *ab initio* parameter selection is seamlessly coupled with the aforementioned hybrid openMP/MPI parallelization scheme. The enhanced simulation framework generates stable dynamics of several thousand ions in the presence of polarized NPs for over 10 million steps ($t > 10$ ns) with computational time reducing from thousands of hours to tens of hours yielding a maximum speed up of ~ 179. This combination of parallel computing and ML in the context of nanoscale simulation of ions is, to our knowledge, the first of its kind and paves way for employing the simulation framework for developing online applications for web-based platforms like nanoHUB [20] where the user interacts with the simulation software under limited interaction with the developer and/or in-domain expert. An application was recently developed on nanoHUB that employs high-performance computing resources to simulate the ionic structure near unpolarizable NPs [21]. The unique features of the framework presented in this paper extend such calculations to polarizable NP systems and enable the development of associated applications for nanoHUB.

II. Related Work

A. Nanoscale Simulation of Ions near Polarizable Materials

The problem of evaluating polarization effects in simulation of charged systems has been extensively explored by several research groups using different approaches [6]–[9], [12], [22]–[25]. Explicit simulation of solvent (environment) and NP molecules is possible [25] using advanced computational techniques such as fast multipole methods and local electrostatics algorithms [26]. However, many phenomena can not be suitably investigated using fully atomistic models due to the prohibitively large number of degrees of freedom associated with such systems. This has led to the study of coarse-grained models that treat ions explicitly but replace the molecular structure of the solvent and the NP with continuous dielectric environments. Systems where the different material parts are adequately captured by piecewise-uniform dielectric permittivities (e.g. NP and solvent, protein and cellular medium) have attracted particular attention [7]–[9], [12], [13], [22], [27], [28]. For this particular case, solving for the polarization or the induced charge density reduces the computational costs because the unknown induced charge density lives only on the interface (boundary) between the NP and the surrounding medium, thus reducing the full three-dimensional electrostatic calculation to a two-dimensional surface computation. Below, we discuss the techniques in computing ion distributions in these specific systems inspired from a variety of approaches [7], [8], [12]–[14], [22], [23]. We focus on methods that are widely applicable and are not limited by the choice of NP geometry or dielectric permittivity profile.

We first outline the methods based on variational approach to the problem of evaluating the polarization effects as these techniques are most closely related to the work presented here. In this approach, one transforms the original problem of solving the Poisson differential equation into an optimization problem. A variety of functionals employing various electrostatic quantities as field variables have been proposed to formulate the variational optimization problem [8], [26], [29]–[35]. Allen et al. [8] optimized a functional of the induced surface charge density $\omega(\mathbf{s})$ at each MD step to solve the Poisson equation and propagate ions. As their functional did not evaluate to the true energy of the system, they were not able to take advantage of the dynamical optimization schemes. Marchi et al. [6] worked with a true energy functional of the polarization vector and implemented a dynamical optimization framework to propagate ion dynamics in parallel with the evaluation of polarization vector fields. They also employed acceleration strategies to obtain a scaling of $O(N \log N)$ in number of particles. However, the choice of the polarization vector as the variable field needed a three-dimensional specification leading to increased computational costs; see Sec. V-A for more details.

Another class of methods of computing $\omega(\mathbf{s})$ involve transforming the problem into a matrix formulation; examples include the Induced Charge Computation (ICC) methods [7] which use matrix inversions to solve for $\omega(\mathbf{s})$. Evaluating matrix inversion involves $O(M^3)$ calculations where M is the number of surface mesh elements. Techniques to improve upon this scaling have been subsequently developed [22]. Alternatively, iterative methods to solve the matrix equation have been proposed [28]. In particular, the generalized minimum residual method solves the matrix equation without explicitly constructing the inverse matrix and converges to the correct $\omega(\mathbf{s})$ at each simulation timestep within 4 - 5 iterations (for spherically-shaped NPs) [10].

The evaluation of $\omega(\mathbf{s})$ in all the above approaches needs the ionic configuration to be static and requires considerable computational effort, whether in the form of matrix inversions or iteration steps convergence, at each simulation step to guarantee the overall stability of the simulation. In Sec. III, we present the details of recently developed dynamical optimization framework that enables the simultaneous (on-the-fly) updates of $\omega(\mathbf{s})$ and the ionic configuration in the same simulation step.

B. Parameter Prediction using Machine Learning

Many recent developments in the use of machine learning in materials science and engineering and physics [17], [18], [36], [37] have inspired us to investigate the use of ML to predict parameters to initialize the virtual system and enable the simulation of ions near polarizable nanomaterials. Employing machine learning (ML) abstractions for parameter prediction and tuning have been extensively employed in performance enhancement of bigdata or deep learning frameworks. In addition, ML based approaches have become popular in other domain areas including protein folding [38], insurance results prediction [39], brain wave classification [40], and image classification [41]. Recently, ML was applied to discover interesting areas of parameter space in self assembly of colloidal materials [19]. With the goal of identifying an appropriate ML technique for our specific application, we reivew some of the related work in using ML in the context of parameter prediction.

Ding et al. [38] have conducted a comprehensive evaluation of multi-class protein folding recognition using support vector machines (SVM) and neural networks (NN). They used a dataset containing 27 SCOP folds to train SVM and NN classifiers to improve the prediction accuracy by 14-110%. They investigated a large number of issues associated with different classes to report overall accuracy of 56% by using a dataset which had $< 25\%$ sequence identity. Balakrishnan et al. [40] have proposed a fast and simple prediction method for two class Brain-Computer Interface (BCI) simulations using multi-layer perceptron. They performed their experiments on two channel EEG data and used STFT for feature extraction. Classification accuracy was found to be 100% for training data and 74% for testing data. They reviewed few variations of NN that were used to design BCI systems based on EEG data. Multilayer perceptron was identified as the most widely used NN for parameter prediction in BCI. Denil et al. [41] have also used multilayer perceptron and convolutional deep learning NN to predict the parameters found in image classification task. The multilayer perceptron was able to obtain an accuracy of 95%.

A case for machine learning to optimize multi-core performance has been studied by Ganapathi et al. [42] to enhance the state of art auto-tuning approaches. Existing auto-tuning approaches are scalable, automatic, and produce high-quality code but they suffer from two major drawbacks. First drawback is the large size of the parameter space to explore: a single application, a single compiler, a specific set of compiler flags, and homogeneous cores alone would have 40 million configurations. Secondly, most existing auto-tuners only focus to minimize overall running time and not the efficiency (e.g, energy and power consumption). The proposed solution utilizes statistical machine learning (SML) approaches to infer models from large quantities of data. Kernel Canonical Correlation Analysis (KCCA) is used as the SML algorithm that effectively identifies the relationship between a set of optimization parameters and a set of resultant performance metrics. Using this approach, the six-month long search time was reduced to two hours on 7-point and 27-point stencil code. Similarly, Bergstra et al. [43] have employed ML for predictive auto-tuning of the Filterbank correlation kernel with boosted regression trees.

Yigitbasi et al. [44] have focused on employing ML-based auto-tuning for diverse MapReduce applications and cluster configurations in Hadoop framework. Their work shows that support vector regression model (SVR) has good accuracy and is also computationally efficient for performance modeling of MapReduce applications. They compared the existing Starfish auto-tuner (a cost-based model) to the SVR based approach and reported comparable and, in some cases, even better performance. Liao et al. [45] have also focused on optimizing MapReduce in Hadoop framework using a search-based ML called Gunther. Their approach utilizes a genetic algorithm designed to identify parameter settings that contribute to near-optimal job execution time.

This literature review identified the multilayer perceptron approach as a suitable method for enhancing the dynamical optimization framework presented here; this method has given comparable, and in many cases, better results than other ML approaches. In Sec. IV-A, we describe the results of using the multilayer perceptron ML to identify good initial parameters for the virtual system in the simulation of ions near polarizable NPs using the dynamical optimization framework.

C. Code Acceleration via Parallel Computing Techniques

The techniques used for parallelizing simulation frameworks in different science and engineering domain applications can be broadly categorized into three approaches: shared memory model (OpenMP), distributed memory model (MPI) and hybrid model (OpenMP and MPI). These methods have been in use on clusters of multi-core symmetric multiprocessing (SMP) nodes for many years. Rabenseifner et al. [46] have provided a comprehensive evaluation of different performance improvement factors and degradation factors in high-performance computing (HPC) with respect to a modern hierarchical hardware design. They employed pure MPI, pure OpenMP, and hybrid models to analyze the cases where a hybrid programming model could be the most effective solution offering reduced communication and memory consumption, as well as improved load balance. Their work shows that parallel programming model should consider combining distributed memory parallelization (on the node interconnect) with shared memory parallelization (inside each node).

Parallel programming approach using the hybrid programming model has been applied for an implicit finite-element methodology in groundwater transport simulations [47], [48]. The original program was decomposed using a domain decomposition strategy to enable parallelization for distributed memory model using MPI. Loop-level parallelism was implemented with several loop modifications using OpenMP directives inside each MPI process to enable shared memory model. Parallel performance results were compared using four different architectures and the hybrid approach was shown to

outperform both pure MPI and pure OpenMP performance when using it with SMP cluster architectures.

The hybrid approach has also been used for groundwater model calibration using multicore computers [49]. The computational model for groundwater calibration utilizes between 100 – 1000 forward solutions, each entailing many nonlinear partial differential equations, leading to a computationally intensive problem to solve. First, the sequential program was profiled using GPROF and a single parallelizable loop was identified to account for over 97% of the total computational time. The OpenMP programming model was adopted for the identified parallelizable loop and the MPI programming model was used for parallelizing the Jacobian calculation and lambda search in the ground water calibration algorithm. Reported results indicate that the calibration time was reduced from weeks to a few hours by using this hybrid approach in $100 - 200$ compute cores.

Mininni et al. [50] focused on performance improvements of pseudospectral computations for fluid turbulence to achieve very high Reynolds numbers using MPI for distributed memory parallelism and OpenMP for shared memory parallelism. They used domain decomposition techniques to achieve numerical discretizations of the problem to implement the hybrid parallel programming approach on top of the original sequential program. The hybrid methodology provided good scalability up to 20,000 compute nodes with a maximum efficiency of 89%, and a mean of 79%. The cost of communication increased with the number of MPI processes, and the hybrid scheme reduced the number of MPI processes by utilizing OpenMP programming model. Further, for a given workload per MPI task, the hybrid scheme outperformed the pure MPI version in terms of speed up and computation time.

Finally, software packages such as ESPRESSO [51] and LAMMPS [52] use MPI techniques to simulate many soft matter systems including ions and nanoparticles. While these are versatile simulation platforms, they largely support simulations of ions near unpolarizable NPs using conventional molecular dynamics (MD) method. We note that features have been added recently in these open source packages that enable the simulation of ions in the presence of polarizable materials; for example, some methods outlined in Sec. II-A were implemented in ESPRESSO [22].

III. Dynamical Optimization Framework: Background and Key Features

We employed the Car-Parrinello molecular dynamics (CPMD) technique to implement the dynamical optimization of the energy functional to generate the propagation of ionic degrees of freedom in tandem with an accurate update of the polarization charges [12], [13] . Here we provide the details of this methodology; these will help clarify the use of enhancement strategies outlined in sections IV-A and IV-B.

A. Variational Functional and the Extended Lagrangian

We focus on the case of ions near a polarizable NP in a solvent environment where the NP and the solvent are modeled as materials of different, but uniform, permittivities. For this system, the discretized form for the functional is obtained by meshing the NP surface into M finite elements. We then assign an average induced charge density ω_k, an area a_k, and a normal vector n_k to each element k. The functional takes the form:

$$\mathscr{F}[\{\omega_k\}] = \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j=1\\ j\neq i}}^{N} q_i \overset{\circ\circ}{K}_{\mathbf{r}_i,\mathbf{r}_j} q_j + \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{M} q_i \overset{\circ\bullet}{K}_{\mathbf{r}_i,\mathbf{s}_k}\omega_k a_k + \frac{1}{2}\sum_{k=1}^{M}\sum_{l=1}^{M}\omega_k \overset{\bullet\bullet}{K}_{\mathbf{s}_k,\mathbf{s}_l}\omega_l a_k a_l, \quad (1)$$

where $\mathbf{s}_k$ is the position vector of the k^{th} finite element and N is the number of ions that are represented with the density $\rho(\mathbf{r}) = \sum_{i=1}^{N} q_i\delta(\mathbf{r}-\mathbf{r}_i)$. Here, q_i and $\mathbf{r}_i$ are, respectively, the charge and position vector of the i^{th} point charge. The terms $\overset{\circ\circ}{K}$, $\overset{\circ\bullet}{K}$, and $\overset{\bullet\bullet}{K}$ in the above equation are, respectively, the effective potentials of interaction between two ions, between an ion and an induced charge, and between two induced charges; explicit expressions can be found in the original papers [12], [13].

To implement the dynamic (on the fly) optimization of $\mathscr{F}[\{\omega_k\}]$ as ions are moved to their new configuration, we include $\mathscr{F}[\{\omega_k\}]$ as the potential energy part of a Lagrangian $\mathcal{L}$ that is extended to include a fictitious kinetic energy term corresponding to the M virtual variables associated with the surface induced charge density values $\{\omega_k\}$. This extended Lagrangian $\mathcal{L}$ is given by:

$$\mathcal{L} = \sum_{i=1}^{N}\frac{1}{2}m_i\dot{\mathbf{r}}_i^2 + \sum_{k=1}^{M}\frac{1}{2}\mu_k\dot{\omega}_k^2 - \mathscr{F}[\omega_k;\mathbf{r}_i] - \mathscr{H}[\mathbf{r}_i]. \quad (2)$$

The first term is the kinetic energy of N ions with masses m_i and position $\mathbf{r}_i$. The second term is a kinetic energy of the virtual system, with μ_k being the mass of the k^{th} virtual variable. The electrostatic potential energy $\mathscr{F}[\{\omega_k\}]$ of the system constitutes the third term. And the final term contains a set of Lennard-Jones potentials to model the steric interactions of the ions and the NP surface.

B. CPMD Equations of Motion and Implementation Details

The following Euler-Lagrange equations of motion can be derived from $\mathcal{L}$:

$$m_i\ddot{\mathbf{r}}_i = -\nabla_{\mathbf{r}_i}\mathscr{F}[\{\omega_k\};\mathbf{r}_i] \quad (3)$$

$$\mu_k\ddot{\omega}_k = -\nabla_{\omega_k}\mathscr{F}[\{\omega_k\};\mathbf{r}_i] \quad (4)$$

These equations capture the Car-Parrinello idea of *on the fly* optimization using MD. In the same timestep as the ions are moved by the force $-\nabla_{\mathbf{r}_i}\mathscr{F}[\{\omega_k\};\mathbf{r}_i]$, the induced charges are updated via a *force* $-\nabla_{\omega_k}\mathscr{F}[\{\omega_k\};\mathbf{r}_i]$ which is obtained from the same potential energy function $\mathscr{F}[\{\omega_k\}]$ that generates the force on the ions. Using these equations, the dynamics for the extended system – ions and the virtual variables – is generated using the standard MD technique implemented via the velocity-Verlet algorithm with the timestep Δ. Ideally,

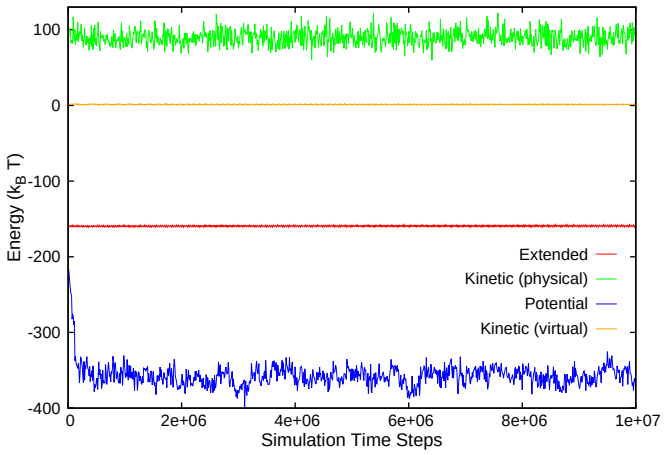

Fig. 1. Energy profiles for the simulation of 1454 ions and 1082 grid points for 10 ns. They show the first key feature of our framework.

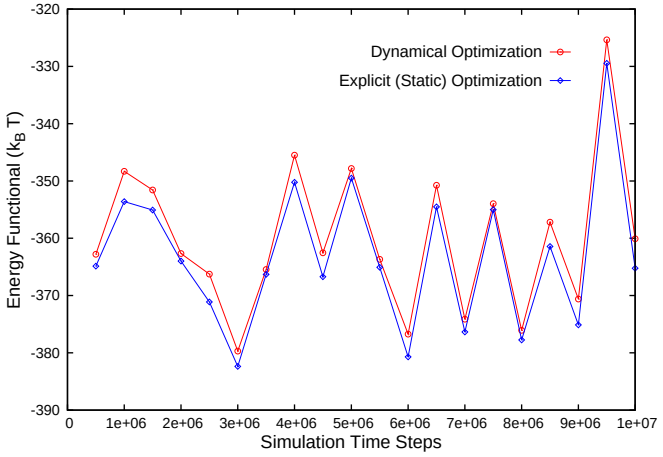

Fig. 2. Tracking of the induced density via functional matching feature for 10 million time steps for a system of 1454 ions and 1082 grid points.

Δ is similar to the value used in the MD simulation of the unpolarizable system ($\Delta = 1$ femtoseconds for an MD simulation of electrolyte ions in water at room temperature).

To simulate the behavior of the ions at temperature T, we couple the extended system to a set of Nosé-Hoover thermostats (this coupling modifies the equations of motion (3) and (4) similar to a canonical MD routine). This two-temperature approach is a standard feature of CPMD [53]–[55]. The ions couple to a thermostat at temperature T, while the virtual system is coupled to one at T_v. Virtual masses μ_k are chosen to be proportional to the areas of the mesh points. The value of the proportionality constant μ depends on the particular system under study (NP charge, dielectric profile, ion valencies, etc.) The parameters T_v and μ are optimized to ensure the stability and accuracy of the simulation. Further technical details of the method can be found in Ref. [12].

A system with N ions near an unpolarizable NP surface effectively translates into a system with M additional degrees of freedom in the case of polarizable NP. The associated computational costs scale roughly as $O((N+M)^2)$. The prefactor for this scaling can be significantly reduced using parallelization. With OpenMP parallelization, the framework produced simulations with improved scaling for small system sizes. For example, for a system with $N = 60$ ions and $M = 484$ grid points, the CPU time is $\tau = 60$ milliseconds per timestep for a simulation performed on a 16 core CPU node with OpenMP shared memory multiprocessing. For large systems, e.g. $N = 1454$ ions and $M = 1082$ grid points, $\tau = 162$ milliseconds per timestep. In Sec. IV-B, we provide the strategies that combine OpenMP and MPI parallelization models to enhance the performance of this framework.

C. Key Features of the Dynamical Optimization Framework

The dynamics derived from the Lagrangian $\mathcal{L}$ has a conserved quantity associated with it: the extended (total) energy of the system. This is given as:

$$\mathscr{E} = \sum_{i=1}^{N} \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 + \sum_{k=1}^{M} \frac{1}{2} \mu_k \dot{\omega}_k^2 + \mathscr{F}[\omega_k; \mathbf{r}_i] + \mathscr{H}[\mathbf{r}_i] + \mathcal{T} + \mathcal{T}_v. \tag{5}$$

Here, $\mathcal{T}$ and $\mathcal{T}_v$ are the energy terms associated with the thermostats controlling the temperature of the physical and virtual systems in the simulation. Two more energies are important to isolate – the energy $\mathscr{R}$ of the real (physical) system at constant T given by

$$\mathscr{R} = \sum_{i=1}^{N} \frac{1}{2} m_i \dot{\mathbf{r}}_i^2 + \mathscr{F}[\omega_k; \mathbf{r}_i] + \mathcal{T}, \tag{6}$$

and the kinetic energy of the virtual system:

$$\mathscr{K} = \sum_{k=1}^{M} \frac{1}{2} \mu_k \dot{\omega}_k^2. \tag{7}$$

In the optimal scenario:

$$\mathscr{K} \sim 0 \tag{8}$$

That is, the real system should be unaffected as much as possible by the presence of virtual system and thus its energy should be approximately conserved ($\mathscr{R} \sim \mathscr{E}$). The usual conservation of energy $\mathscr{E}$ and the approximate conservation of $\mathscr{R}$ reflected in (8) constitute the first key feature of the CPMD-based dynamical optimization framework. The energy profiles of a successful simulation are shown in Fig. 1; the extended energy $\mathscr{E}$ is constant and the virtual kinetic energy stays stable and close to 0 throughout the entire simulation run (for 10 ns). In practice, this feature is incorporated in the simulation by appropriately choosing values of μ and $T_v \ll T$. These parameters are selected to control any rise in the "momentum" (heating up) associated with the virtual system as the simulation progresses.

This feature is encoded in two important quantities, R and R_v, which are used to assess the stability and accuracy of the

simulation. R measures the ratio of the fluctuations in $\mathscr{E}$ and the fluctuations in the kinetic energy of the physical system. R_v is the ratio of the fluctuations in $\mathscr{E}$ and the fluctuations in the $\mathscr{K}$. For good energy conservation, $R < 0.05$ as noted in the literature [6]. For stability, we demand a similar passing test for the virtual system and enforce $R_v < 0.15$. The latter inequality implicitly satisfies the requirement that $\mathscr{K}$ is kept close to the value dictated by the low temperature T_v.

The second important feature relates to the effectiveness of our scheme to reproduce the correct polarization charge distribution. At regular intervals during the course of the simulation, we collect and store the ion coordinates and induced charge densities. Then, we carry out an ordinary (static) minimization of the functional $\mathscr{F}$ to explicitly determine the (numerically) exact induced density, and compare it to the distributions obtained via dynamical optimization during the simulation. This tracking of the induced density distributions on the NP surface can be assessed by evaluating the matching of $\mathscr{F}$ optimized on the fly with the functional that is optimized explicitly. This functional matching is the second key feature. In practice, we compute the functional deviation, f_d, which measures the average difference between the dynamically optimized functional $\mathscr{F}$ and the potential energy functional obtained via direct (static) minimization. To pass the test of stability and accuracy, we enforce $f_d < 1\%$.

Quantities R, R_v, and f_d enter the ML based procedure described in Sec. IV-A to determine the initial selection of optimal parameters.

IV. Enhanced Dynamical Optimization Framework

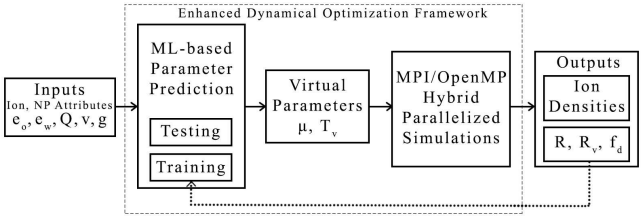

Fig. 3. System overview of enhanced dynamical optimization framework.

We now describe the performance enhancement of the dynamical optimzation framework using a combination of the state-of-the-art parallelization techniques and the ML-based scheme for prediction of virtual system parameters. Figure 3 shows the overview of the implemented methodology. First, the ion and NP attributes characterizing the system input properties are fed to the ML-based parameter prediction procedure. The predicted virtual system parameters are used to run the actual simulation that is parallelized using the MPI/OpenMP hybrid programming model. The output of the simulations are the ion densities that characterize the ionic structure near polarizable NPs. In addition, during the simulation, the quantities R, R_v, and f_d are computed and saved to be used for retraining the ML procedure after a set number of simulation runs are executed.

A. Machine Learning Virtual System Parameters

As reviewed in Sec. II-B, many ML techniques are available for parameter tuning and prediction. We adopt one of these approaches, the multilayer perceptron neural network, for automatic *ab initio* prediction of virtual system parameters in the dynamical optimization framework for a given set of physical system parameters. In this section, we describe the data preparation, data preprocessing techniques, method of feature extraction and classification techniques as well as their validation based on Python programs. Figure 4 shows the suggested framework to predict the desired parameters.

1) Data Preparation and Preprocessing: Inputs to the CPMD program are classified into two categories; physical parameters and computational parameters. In our research, we created a data set using 13600 simulation runs to collect all input parameters (which includes physical, virtual, and computational parameters such as mesh size M and simulation timestep Δ). As identified in Sec. III-C, success of the framework implemented via the CPMD simulation is determined by three quantities: R, R_v, and f_d. Acceptable threshold values for R, R_v, and f_d are 0.05, 0.15, and 1 percent respectively. So these three parameters are converted to a binary value to represent the final output. Min max normalization filter was applied to normalize the input data in the data preprocessing stage.

2) Feature extraction and classification: As shown in figure 4, the conventional multilayer perceptron algorithm with one hidden layer was implemented in Python for binary classification. Hidden layer outputs were wrapped with *tanh* function while *softmax* function was used in the output layer for the algorithm. Using a trial and error process, hyper parameters such as number of hidden layers and learning rate were set to 6 hidden units and 0.05 respectively. The weights in the hidden layer and output layer were initialized to be random values using a seed at the beginning.

B. OpenMP/MPI Hybrid Parallelization

Dynamics of the physical system (ions) and the virtual system in CPMD simulation starts with pre-calculated parameters and associated variables initialized. Following velocity-Verlet algorithm, at each simulation step, the code first updates the velocity of all ions and virtual variables for half timestep $\Delta/2$ and then it updates their positions for Δ. Next it calculates the forces on each ion and virtual variable, and finally it updates the velocity for the next $\Delta/2$. During this propagation, the energies and other thermodynamical quantities are monitored at periodic intervals. Once the system reaches equilibrium, the positions of the ions are stored periodically to generate ionic density profiles at the end of the simulation run.

The original dynamical optimization framework implemented via CPMD with built-in OpenMP parallelization takes approximately 450 hours using 16 cores to run a simulation of a typical system (1454 ions and 1082 grid points) for 10 million timesteps (that is, 10 ns). This runtime is high and prohibits the use of the method to simulate dynamics of ions for longer times in order to extract accurate data for evaluating

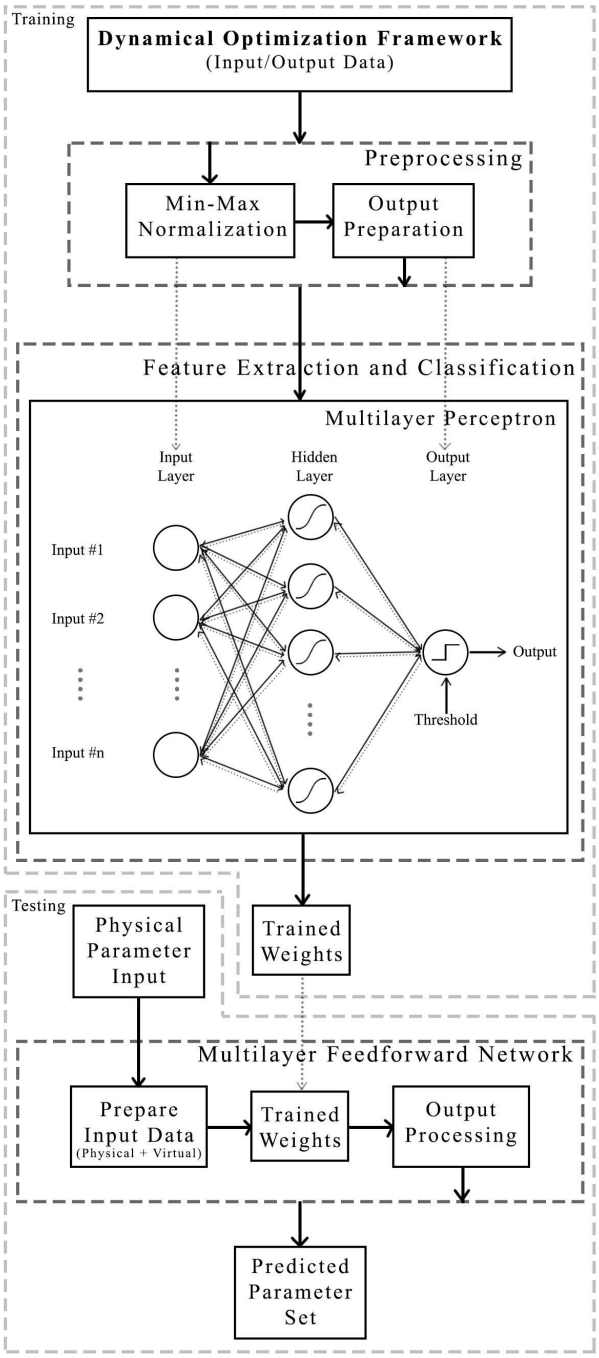

Fig. 4. ML procedure for determining the parameters for the virtual system defined in the dynamical optimization framework.

ionic correlations and understanding associated physics. We now outline the performance enhancement techniques we utilized to significantly reduce the computational time for this simulation framework.

First, the sequential program was evaluated for performance profiling in order to determine where it is spending most of the run time. The program was tested in the Indiana university BIGRed2 cluster, which features a hybrid architecture based on two Cray Inc., and it runs a proprietary variant of Linux called Cray Linux Environment. The performance profiling was done using Performance Counters for Linux (PERF). The report showed that the program was spending 64.33% of its total computation time on calculating the forces between the ions for each time step of the simulation. The report also revealed that no other major function call takes considerable computation time compared to force calculation. The second noticeable computation time was taken by object creation procedures, whereas the third highest computation time was consumed by basic vector operations such as addition, subtraction and scalar multiplication. We expect the time complexities of the three major components of the MD routine – update position, update velocity, and force calculation, to be $O(n)$, $O(n)$, and $O(n^2)$ respectively. Performance profiling results verified these time complexities and accordingly we focused on optimization and parallelization of the force calculation subroutine. Performance profiling did not highlight the energy calculation subroutine as it was not executed as frequently. If the energy sampling rate is increased, this subroutine will also increase the runtime as it is also $O(n^2)$ in time complexity.

1) Distributed memory parallelization with MPI: The MPI implementation uses one-dimensional (1D) problem decomposition considering the forces on each ion (due to other ions and induced charges), and the virtual forces on induced density variables (due to the physical and virtual particles in the system). Figure 5 shows the procedure applied to enable the distributed memory parallelization with MPI. In this approach, the simulation propagation happens sequentially in every process until each instances of the program inside the MPI processes need to calculate the current forces based on the position of the ions and other MD parameters. Inside each of these processes, the (partial) force calculation happens only for the boundary parameters defined for that particular process. After the partial force calculation has been completed, as explained in Fig. 5, the MPI collective operation MPI_Allgather was used to distribute the partial data of the forces among all processes. Subsequently, each process calculates other required MD parameters and moves on to perform the next iteration in the evolution of the system. Similarly MPI collective operation MPI_Allreduce was used to parallelize the energy computation subroutine.

2) Shared memory parallelization with OpenMP: The shared memory parallelization approach with OpenMP is based on a state-of-the-art loop level parallelism. Some modifications were made to the computationally intensive loops inside the force calculation procedure so as to improve the shared memory parallelization. The straightforward loop level parallelism was achieved by implementing OpenMP compiler directives on all nested loops. The OpenMP parallelization was applied at the outermost loops to reduce OpenMP overheads, such as thread generation and data copy. The dynamical distribution and the ordering of the loop index from a large task to a small task were applied to obtain better load balancing. Several memory optimization techniques such as using C-language arrays instead of C++ vectors (as it favors data locality) were used to improve the efficiency of the program [56]. Repetitive

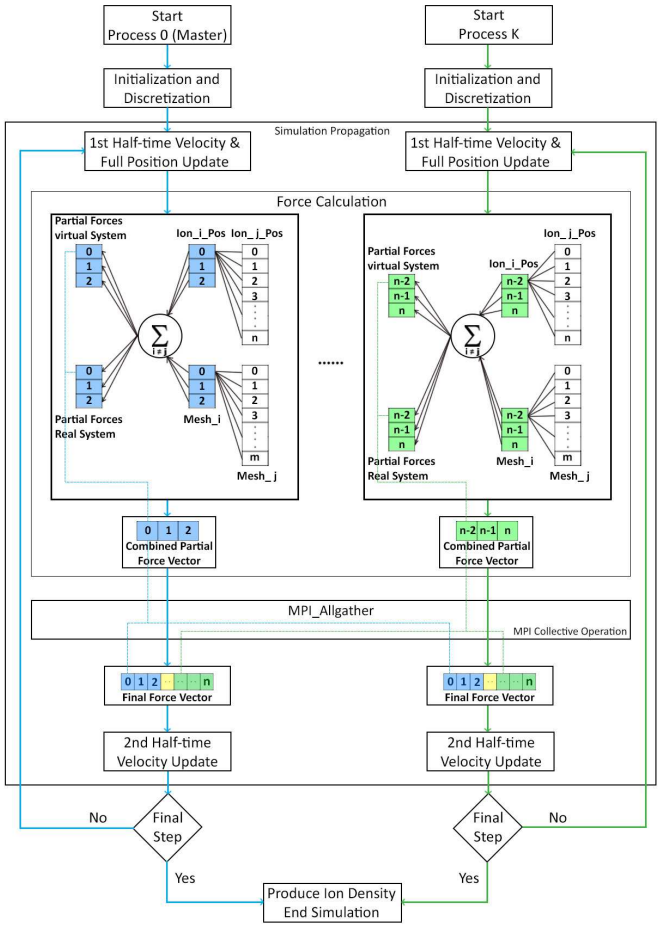

Fig. 5. Distributed memory parallelization approach with MPI. Hybrid parallelization is implemented inside the Force Calculation block.

memory allocations inside the force calculation procedure were moved to the outside of the routine. A force matrix calculation approach was also tried in order to reduce the time complexity to $O(n^2/2)$. Instead of accumulating all the force elements for one outer loop iteration, the force matrix was calculated only for the upper diagonal element of the matrix.

3) Hybrid MPI/OpenMP parallelization: The hybrid masteronly model was tested by combining the distributed memory MPI approach and the shared memory OpenMP approach [46]. The hybrid masteronly model uses one MPI process per node and OpenMP on the cores of the node, with no MPI calls inside the parallel regions. This hybrid model enables the domain decomposition under a two-level mechanism. This approach is applied for the force calculation and energy calculation subroutines in the dynamical optimization framework. On the MPI level, a coarse-grained domain decomposition is performed using boundary conditions as explained in figure 5. The second level of domain decomposition is achieved through OpenMP loop level parallelization inside each MPI process. This multilevel domain decomposition has advantages over pure MPI or pure OpenMP, when cache performance is taken into consideration. This strategy also provides the maximum access locality, a minimum of cache misses, non-uniform memory access (NUMA) traffic and inter-node communication [46].

V. Results and Discussion

A. Parameter Prediction using ML

Multilayer perceptron-based parameter prediction algorithm was tested with 1500 sets of input parameters. The ML algorithm predicted the virtual system parameters correctly with 95.6% accuracy. Table I shows the predicted virtual system parameters (μ and T_v) for some systems along with the quantities R, R_v, and f_d that characterize the two key features identified in our framework. As evidenced by the values R, R_v, and f_d within allowable ranges ($R < 0.05$, $R_v < 0.15$, $|f_d| < 1\%$), the predicted virtual system parameters produced stable dynamics of the system.

TABLE I
Predicted Parameters and Simulation Stability

Inputs			**Prediction**		**Results**		
$e_{\mathbf{o}}$, $e_{\mathbf{w}}$	Q, v	g	μ	T_v	R	R_v	f_d
2, 30	-30, 1	132	1	0.002	0.002	0.12	-0.1
2, 78.5	-30, 3	1692	100	0.002	0.003	0.13	-0.6
70, 78.5	-60, 2	752	18	0.001	0.002	0.08	-0.6
80, 160	-90, 3	1272	30	0.002	0.002	0.09	-0.7
100, 120	30, 2	482	36	0.005	0.002	0.1	-0.1
2, 30	-30, 3	1692	30	0.001	0.006	0.1	-0.6

When our ML model was trained utilizing only R and f_d factors, we noticed that there is a tendency to get a prediction with higher μ and lower T_v for any given input parameter pattern. Even though simulations did not fail within the timescales tested for the dynamics of ions, these virtual parameter choices are not always desirable and will not be picked by the experienced, in-domain expert. However, when we included R_v as an additional factor for training our model, the NN started to predict the virtual system parameters (see Table I) that were likely to be selected by in-domain expert and yield a stable simulation for longer times.

B. Benchmarking

The dynamical optimization framework for the simulation of ions near polarizable NPs implemented via CPMD was benchmarked using BigRed2 cluster nodes. These nodes have maximal achieved performance of 596.4 teraFLOPS, and feature a hybrid architecture based on two Cray, Inc., 344 XE6 (CPU-only) compute nodes, providing a total of 1,020 compute nodes, 21,824 processor cores, and 43,648 GB of RAM. Each XE6 node has two AMD Opteron 16-core Abu Dhabi x86_64 CPUs and 64 GB of RAM; each XK7 node has one AMD Opteron 16-core Interlagos x86_64 CPU, and 32 GB of RAM.

The inset in Fig. 6 shows the strong scaling plot of the performance of pure OpenMP $O(n^2)$ for simulating 60 ions and 1082 virtual variables (mesh points) for 10 million time steps. The pure OpenMP approach yielded a maximum speedup of 5.27 with 16 OpenMP threads. The sequential

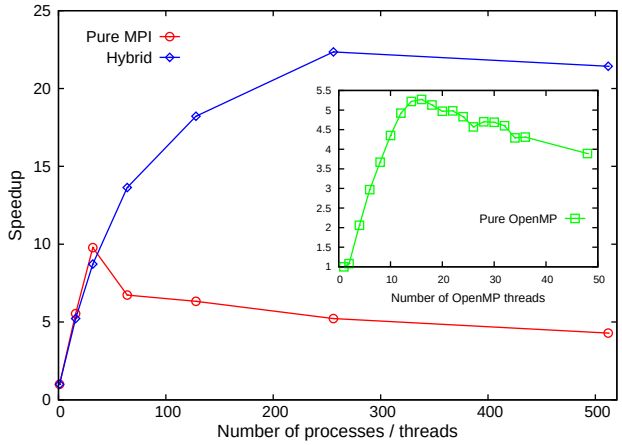

Fig. 6. Strong scaling plot of the performance of MPI vs. MPI/OpenMP hybrid for 60 ions and 1082 grid points. (Inset) Same plot in the case of the pure OpenMP $O(n^2)$ approach for the same system.

program runtime was reduced from 268 hours to 51 hours. The speedup increased only up to 16 threads because BigRed II 676 XK7 compute nodes have 16 internal cores per node. The performance slowdown for higher number of threads can be attributed to oversubscription of the CPU node when more than 16 threads are used. Using the OpenMP $O(n^2/2)$ approach outlined before, the maximum speed up was 6.3 with the sequential program runtime reducing from 268 hours to 42.3 hours. The speedup improvement was not significant in comparison to the $O(n^2)$ approach due to the fact that the force matrix procedure used in the $O(n^2/2)$ approach contained many 2D array accesses and writes compared to the 1D array force vector associated with the $O(n^2)$ approach.

Figure 6 compares the strong scaling plot of the performance of pure MPI and the MPI/OpenMP hybrid model for 60 ions and 1082 grid points with propagation up to 10 million simulation steps. The pure MPI model yielded a maximum speed up of 9.78. Even though the ideal strong scaling should increase the speedup as the number of processes increases in pure MPI parallelization, our simulation framework for the above system strong scale well up to 16 processes. This discrepancy may be because the overhead of running multiple processes on 60 ions (10 million time steps) is higher than the parallelization achieved through the MPI model. In contrast, the hybrid model produced the maximum speedup of 22.16 with 256 processes (16 MPI nodes and 16 OpenMP threads inside each MPI node). This maximum speedup was calculated without considering the execution time reduction gained from the memory optimization techniques. For the hybrid model, we found that the optimal configuration of OpenMP threads is socket bound as noted by other researchers [46]. As a result, the number of optimal OpenMP threads in our experiment was 16 for any number of MPI processes.

Using the hybrid methodology, the runtime for above small system was reduced from 268 hours to 12 hours. When implemented to a large system with 1454 ions and 1082 virtual variables (for 1 million steps), the hybrid model reduced the

TABLE II
STRONG SCALING DATA (SPEEDUP) OF HYBRID PERFORMANCE MODEL

Processes/Threads	Number of Ions		
	60	***1454***	***2908***
1	1	1	1
16	5.2	11.5	12.9
32	8.72	24.44	28.5
64	13.64	38.18	49.4
128	18.22	48.88	63.1
256	***22.35***	83.69	95.4
512	21.43	***95.46***	122.48
1024	19.24	84.86	***179.25***

execution time from 611 hours to 6.0 hours with a speedup of 95.46. Table II shows the strong scaling performance comparison data for the hybrid performance model with systems of different sizes simulated for 10 million time steps. It is clear that optimum number of MPI processes are proportional to the problem size when OpenMP thread affinity is set to the socket resulting a well weak scaling system. The maximum speedup of 179.25 was obtained for 1024 processes/threads when program is executed with 2908 ions and 1082 grid points.

C. Application: Ions near an oil-water nanoemulsion

The enhanced framework was applied to compute the distribution of monovalent electrolyte (Sodium Chloride, NaCl) ions of different concentration c outside an oil-water nanoemulsion droplet [57], [58] at room temperature $T = 298$ K. Ions were modeled as Lennard-Jones (LJ) spheres of diameter $\sigma = 0.357$ nm, and the nanoemulsion (NP) droplet was treated as a charged oil droplet in water forming a spherical dielectric interface with surface charge of $60e$ and radius $a = 7.5\sigma \sim 2.7$ nm. The dielectric permittivity of oil is taken to be $\epsilon_o = 2$, while that of water is $\epsilon_w = 80$. The difference in the polarizable properties of oil and water lead to surface polarization charges. We show results for $c = 0.1, 0.2, 0.3$ M which together with the 60 counterions (associated with the charged nanoemulsion) lead to a total of $1454, 2908, 4362$ ions respectively. The NP surface was discretized with $M = 1082$ points. The whole system of ions and NP is taken to be in a large spherical simulation cell of diameter $b = 40\sigma \sim 14$ nm. The nanoemulsion surface and the simulation cell boundary are modeled as hard LJ walls.

The aforementioned attributes of the physical system supply the input parameters for the enhanced dynamical optimization framework. Following the process elucidated in Fig. 3, we first pass these as inputs to the ML-based procedure that generates the required virtual system parameters. The combined set of physical and virtual system parameters kickstart the parallelized simulation program that produces the dynamics of ions near the polarizable NP with a simulation timestep of 1 femtoseconds.

The dynamics of ions near the polarized nanoemulsion was simulated for over 10 million time steps ($\gtrsim$ 10 ns) with good energy conservation. The inset in Fig. 7 shows that the

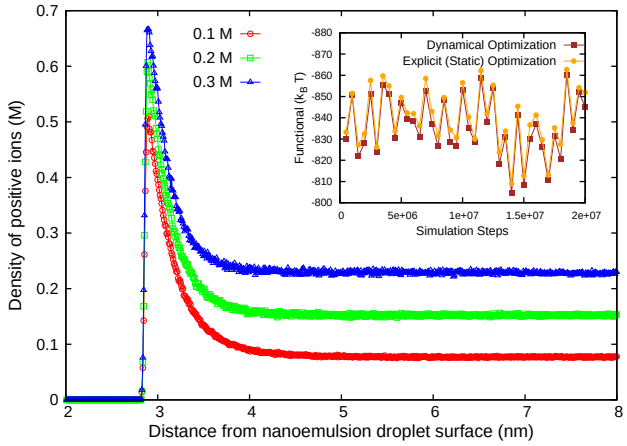

Fig. 7. Density distribution of positive ions at $c_n = 0.1, 0.2, 0.3$ M produced by the dynamical optimization framework. (Inset) Tracking of the induced density via functional matching feature for the $c_n = 0.1$ M system.

induced charges were accurately tracked at all times for up to 20 million time steps (functional matching feature). The stability and accuracy evident from this plot demonstrates the success of the ML-based parameter selection process. The hybrid parallelization scheme yielded converged results for the ion distributions associated with these large systems. For all concentrations, the densities reach a constant value in the bulk (away from the NP surface) but show accumulation near the NP surface. Fig. 7 shows how the preference of positive ions to accumulate near the emulsion droplet changes with c.

D. Potential Issues and Comparison

We showed that the hybrid model increased the performance of a given system in the regime where MPI has started to decrease the speedup. However, a hybrid implementation is not expected to outperform pure MPI in the regime where the scaling of the MPI is still good. We will use roofline model analysis to further identify effects and bottlenecks due to memory traffic. Using this model, we will evaluate the optimization for the peak performance and memory bandwidth as they are available as upper bounds.

We will experiment with other MPI collective approaches such as pipeline for replacing allgather collective operation in hybrid implementation as the performance of different approaches depends on how they are implemented. We also plan to run larger system sizes for a shorter number of steps to better understand the overheads of the parallel implementation.

To support periodic boundary conditions, we will integrate the current parallelization features of the framework with acceleration strategies such as fast particle-mesh Ewald solvers that reduce the $O(n^2)$ scaling in particle system size to $O(n \log n)$ [6], [23]. This process will extend the application of the framework to investigate phenomena involving larger NPs and dense ionic systems.

For very long simulations (over 100 million steps, $O(1)$ microseconds), the virtual system parameters may need to be re-tuned. In this scenario, we will need to adapt our ML procedure to provide dynamical auto-tuning of parameters, which will extend the applicability of the framework to study phenomena involving long ion relaxation times.

We compare this framework with another dynamical optimization framework based on solving for the polarization vector variable with $O(N \log N)$ scaling [6]. They reported simulations for time $t \leq 40$ picoseconds with the computational time per step for single processor of $O(1)$ seconds for a system of $O(100)$ particles. For similar system size, the compute time per step per processor for our framework is $\tau \sim O(0.1)$ seconds despite scaling in particle size as $O(N^2)$. With OpenMP parallelization, we obtain $\tau \sim O(0.01)$ seconds. And with MPI/OpenMP hybrid implementation, $\tau \sim O(0.001)$ seconds. Further, we were able to simulate the system for > 10 nanoseconds (~ 3 orders of magnitude longer). These results support our choice of 2-dimensional induced surface charge density as the variable to optimize for compared to the 3-dimensional vector polarization. Further they validate the acceleration obtained by employing the hybrid parallel computing model to enhance the framework.

VI. Conclusion and Outlook

We enhanced the stability and efficiency of the dynamical simulation framework using a judicious combination of parallel computing and machine learning techniques. The hybrid openMP/MPI parallelization scheme reduced the computational time from thousands of hours to tens of hours yielding a maximum speed up of over 175, and enabled the extraction of ionic distributions for systems with thousands of ions and surface mesh points. Machine learning based parameter prediction procedure predicted the virtual system parameters that provide the desired stability at 95.6% accuracy. This enhanced simulation framework has many applications and we demonstrated its utility by generating stable, accurate dynamics of ions in the presence of a polarized nanoemulsion droplet for over 10 million simulation steps ($t > 10$ ns) that resulted in converged ion density profiles. This framework paves way for investigating phenomena involving ions near polarizable NPs of complex shapes and enable the study of long-time ion dynamics.

The unique combination of parallel computing and machine learning to enhance the simulation framework enables users across the globe, with a diversity of in-domain experience, to simulate ions near polarizable NPs via the use of web-based applications hosted on services like nanoHUB. A tool powered by this enhanced framework is currently being developed for deployment on nanoHUB, and we expect to publish it this year. We will extend our framework to enable the process of using the data generated by this nanoHUB application for continuous training (self-learning) of the ML-based parameter prediction procedure. We also note that this ML method can be extended to other domains where problems can be framed as optimization problem of similar nature.

References

[1] B. Honig and A. Nicholls, "Classical electrostatics in biology and chemistry," *Science*, vol. 268, no. 5214, pp. 1144–1149, 1995.

[2] E. Raspaud, M. Olvera de la Cruz, J. Sikorav, and F. Livolant, "Precipitation of dna by polyamines: a polyelectrolyte behavior." *Biophys J*, vol. 74, no. 1, pp. 381–93, 1998.

[3] Y. Levin, "Strange electrostatics in physics, chemistry, and biology," *Physica A: Statistical Mechanics and its Applications*, vol. 352, no. 1, pp. 43 – 52, 2005.

[4] S. Sacanna, W. K. Kegel, and A. P. Philipse, "Thermodynamically stable pickering emulsions," *Phys. Rev. Lett.*, vol. 98, p. 158301, Apr 2007.

[5] H. D. Abrua, Y. Kiya, and J. C. Henderson, "Batteries and electrochemical capacitors." *Physics Today*, vol. 61, no. 12, p. 43, 2008. [Online]. Available: http://search.ebscohost.com.turing.library.northwestern.edu/login.aspx?direct=true&db=ulh&AN=35655115&site=ehost-live

[6] M. Marchi, D. Borgis, N. Levy, and P. Ballone, "A dielectric continuum molecular dynamics method," *The Journal of Chemical Physics*, vol. 114, no. 10, pp. 4377–4385, 2001.

[7] D. Boda, D. Gillespie, W. Nonner, D. Henderson, and B. Eisenberg, "Computing induced charges in inhomogeneous dielectric media: Application in a monte carlo simulation of complex ionic systems," *Phys. Rev. E*, vol. 69, no. 4, p. 046702, Apr 2004.

[8] R. Allen, J.-P. Hansen, and S. Melchionna, "Electrostatic potential inside ionic solutions confined by dielectrics: a variational approach," *Phys. Chem. Chem. Phys.*, vol. 3, pp. 4177–4186, 2001.

[9] A. P. dos Santos, A. Bakhshandeh, and Y. Levin, "Effects of the dielectric discontinuity on the counterion distribution in a colloidal suspension," *The Journal of Chemical Physics*, vol. 135, no. 4, p. 044124, 2011.

[10] Z. Gan, H. Wu, K. Barros, Z. Xu, and E. Luijten, "Comparison of efficient techniques for the simulation of dielectric objects in electrolytes," *Journal of Computational Physics*, vol. 291, pp. 317 – 333, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999115001667

[11] K. Grass and C. Holm, "Polyelectrolytes in electric fields: measuring the dynamical effective charge and effective friction," *Soft Matter*, vol. 5, pp. 2079–2092, 2009.

[12] V. Jadhao, F. J. Solis, and M. Olvera de la Cruz, "Simulation of charged systems in heterogeneous dielectric media via a true energy functional," *Phys. Rev. Lett.*, vol. 109, p. 223905, Nov 2012. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.109.223905

[13] ——, "A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity," *The Journal of Chemical Physics*, vol. 138, no. 5, p. 054119, 2013. [Online]. Available: http://link.aip.org/link/?JCP/138/054119/1

[14] Y. Jing, V. Jadhao, J. W. Zwanikken, and M. O. de la Cruz, "Ionic structure in liquids confined by dielectric interfaces," *The Journal of Chemical Physics*, vol. 143, no. 19, p. 194508, 2015.

[15] R. Messina, "Image charges in spherical geometry: Application to colloidal systems," *The Journal of Chemical Physics*, vol. 117, no. 24, pp. 11 062–11 074, 2002.

[16] M. M. Hatlo and L. Lue, "The role of image charges in the interactions between colloidal particles," *Soft Matter*, vol. 4, pp. 1582–1596, 2008.

[17] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, "Machine learning phases of strongly correlated fermions," *Phys. Rev. X*, vol. 7, p. 031038, Aug 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.7.031038

[18] J. Liu, Y. Qi, Z. Y. Meng, and L. Fu, "Self-learning monte carlo method," *Phys. Rev. B*, vol. 95, p. 041101, Jan 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.95.041101

[19] M. Spellings and S. C. Glotzer, "Machine learning for crystal identification and discovery," *ArXiv e-prints*, Oct. 2017.

[20] G. Klimeck, M. McLennan, S. P. Brophy, G. B. A. III, and M. S. Lundstrom, "nanohub.org: Advancing education and research in nanotechnology," *Computing in Science Engineering*, vol. 10, no. 5, pp. 17–23, Sept 2008.

[21] K. Kadupitiya, S. Marru, G. C. Fox, and V. Jadhao, "Ions in nanoconfinement," https://nanohub.org/resources/nanoconfinement, Dec 2017, online on nanoHUB. [Online]. Available: https://nanohub.org/resources/nanoconfinement

[22] S. Tyagi, M. Suzen, M. Sega, M. Barbosa, S. S. Kantorovich, and C. Holm, "An iterative, fast, linear-scaling method for computing induced charges on arbitrary dielectric boundaries," *The Journal of Chemical Physics*, vol. 132, no. 15, p. 154112, 2010.

[23] K. Barros, D. Sinkovits, and E. Luijten, "Efficient and accurate simulation of dynamic dielectric objects," *The Journal of Chemical Physics*, vol. 140, no. 6, p. 064903, 2014. [Online]. Available: https://doi.org/10.1063/1.4863451

[24] Z. Gan and Z. Xu, "Multiple-image treatment of induced charges in monte carlo simulations of electrolytes near a spherical dielectric interface," *Phys. Rev. E*, vol. 84, p. 016705, Jul 2011.

[25] A. Wynveen and F. Bresme, "Interactions of polarizable media in water: A molecular dynamics approach," *The Journal of Chemical Physics*, vol. 124, no. 10, p. 104502, 2006. [Online]. Available: https://doi.org/10.1063/1.2177244

[26] J. Rottler and A. C. Maggs, "Local molecular dynamics with coulombic interactions," *Phys. Rev. Lett.*, vol. 93, no. 17, p. 170201, Oct 2004.

[27] Z. Xu, "Electrostatic interaction in the presence of dielectric interfaces and polarization-induced like-charge attraction," *Phys. Rev. E*, vol. 87, p. 013307, Jan 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.87.013307

[28] K. Barros and E. Luijten, "Dielectric effects in the self-assembly of binary colloidal aggregates," *Phys. Rev. Lett.*, vol. 113, p. 017801, Jul 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.113.017801

[29] J. D. Jackson, *Classical Electrodynamics*, 3rd ed. Wiley, New York, 1999.

[30] R. A. Marcus, "On the theory of oxidation-reduction reactions involving electron transfer. i," *The Journal of Chemical Physics*, vol. 24, no. 5, pp. 966–978, 1956.

[31] B. U. Felderhof, "Fluctuations of polarization and magnetization in dielectric and magnetic media," *The Journal of Chemical Physics*, vol. 67, no. 2, pp. 493–500, 1977.

[32] E. S. Reiner and C. J. Radke, "Variational approach to the electrostatic free energy in charged colloidal suspensions: general theory for open systems," *J. Chem. Soc., Faraday Trans.*, vol. 86, pp. 3901–3912, 1990.

[33] D. M. York and M. Karplus, "A smooth solvation potential based on the conductor-like screening model," *The Journal of Physical Chemistry A*, vol. 103, no. 50, pp. 11 060–11 079, 1999.

[34] P. Attard, "Variational formulation for the electrostatic potential in dielectric continua," *The Journal of Chemical Physics*, vol. 119, no. 3, pp. 1365–1372, 2003.

[35] F. Lipparini, G. Scalmani, B. Mennucci, E. Cances, M. Caricato, and M. J. Frisch, "A variational formulation of the polarizable continuum model," *The Journal of Chemical Physics*, vol. 133, no. 1, 2010.

[36] A. P. Bartók, S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and M. Ceriotti, "Machine learning unifies the modeling of materials and molecules," *Science Advances*, vol. 3, no. 12, 2017.

[37] S. S. Schoenholz, "Combining Machine Learning and Physics to Understand Glassy Systems," *ArXiv e-prints*, Sep. 2017.

[38] C. H. Ding and I. Dubchak, "Multi-class protein fold recognition using support vector machines and neural networks," *Bioinformatics*, vol. 17, no. 4, pp. 349–358, 2001.

[39] J. Kunce and S. Chatterjee, "A machine-learning approach to parameter estimation."

[40] D. Balakrishnan and S. Puthusserypady, "Multilayer perceptrons for the classification of brain computer interface data," in *Bioengineering Conference, 2005. Proceedings of the IEEE 31st Annual Northeast*. IEEE, 2005, pp. 118–119.

[41] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[42] A. Ganapathi, K. Datta, A. Fox, and D. Patterson, "A case for machine learning to optimize multicore performance," in *First USENIX Workshop on Hot Topics in Parallelism (HotPar09)*, 2009.

[43] J. Bergstra, N. Pinto, and D. Cox, "Machine learning for predictive auto-tuning with boosted regression trees," in *Innovative Parallel Computing (InPar), 2012*. IEEE, 2012, pp. 1–9.

[44] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on*. IEEE, 2013, pp. 11–20.

[45] G. Liao, K. Datta, and T. L. Willke, "Gunther: Search-based auto-tuning of mapreduce," in *European Conference on Parallel Processing*. Springer, 2013, pp. 406–419.

[46] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*. IEEE, 2009, pp. 427–436.

[47] G. Mahinthakumar and F. Saied, "A hybrid mpi-openmp implementation of an implicit finite-element code on parallel architectures," *the International Journal of High Performance Computing Applications*, vol. 16, no. 4, pp. 371–393, 2002.

[48] G. Mahinthakumar and M. Sayeed, "Hybrid genetic algorithmlocal search methods for solving groundwater source identification inverse problems," *Journal of water resources planning and management*, vol. 131, no. 1, pp. 45–57, 2005.

[49] G. Tang, E. F. DAzevedo, F. Zhang, J. C. Parker, D. B. Watson, and P. M. Jardine, "Application of a hybrid mpi/openmp approach for parallel groundwater model calibration using multi-core computers," *Computers & Geosciences*, vol. 36, no. 11, pp. 1451–1460, 2010.

[50] P. D. Mininni, D. Rosenberg, R. Reddy, and A. Pouquet, "A hybrid mpi–openmp scheme for scalable parallel pseudospectral computations for fluid turbulence," *Parallel Computing*, vol. 37, no. 6-7, pp. 316–326, 2011.

[51] H. J. Limbach, A. Arnold, B. A. Mann, and C. Holm, "ESPResSo – an extensible simulation package for research on soft matter systems," *Comp. Phys. Comm.*, vol. 174, no. 9, pp. 704–727, May 2006.

[52] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S002199918571039X

[53] M. Sprik, "Computer simulation of the dynamics of induced polarization fluctuations in water," *The Journal of Physical Chemistry*, vol. 95, no. 6, pp. 2283–2291, 1991.

[54] P. E. Blöchl and M. Parrinello, "Adiabaticity in first-principles molecular dynamics," *Phys. Rev. B*, vol. 45, pp. 9413–9416, Apr 1992.

[55] E. S. Fois, J. I. Penman, and P. A. Madden, "Control of the adiabatic electronic state in ab initio molecular dynamics," *The Journal of Chemical Physics*, vol. 98, no. 8, pp. 6361–6368, 1993.

[56] E. Acklam, A. Jacobsen, and H. P. Langtangen, "Optimizing c++ code for explicit finite difference schemes," *Oslo Scientific Computing Archive, Report*, vol. 4, 1998.

[57] J. de Graaf, J. Zwanikken, M. Bier, A. Baarsma, Y. Oloumi, M. Spelt, and R. van Roij, "Spontaneous charging and crystallization of water droplets in oil," *The Journal of Chemical Physics*, vol. 129, no. 19, p. 194701, 2008.

[58] M. Bier, J. Zwanikken, and R. van Roij, "Liquid-liquid interfacial tension of electrolyte solutions," *Phys. Rev. Lett.*, vol. 101, p. 046104, Jul 2008.

APPENDIX: ARTIFACT DESCRIPTION

A. Overview

In this appendix, the relevant information needed to launch/use the dynamical optimization framework is provided. Details for the parallelization approach and the parameter prediction procedure are outlined. The library dependencies for the compilation of simulation program are provided. The installation and the execution of the underlying parallelized code on a supercomputer or in a local computer is explained. This is followed by a brief discussion on the trustworthiness of our results. Our framework is open source and can be downloaded from

- git@github.com:softmaterialslab/np-electrostatics-lab.git

The basic computational framework and the MPI/OpenMP hybrid enhancement is written in C++. The machine learning procedure to evaluate optimal virtual system parameters is written in Python. As envisioned in the paper, this framework will enable the development of a simulation tool on nanoHUB that provides the ionic structure near polarizable NPs. A similar application for simulation the dynamics of ions near unpolarizable nanoparticles was deployed here:

- http://nanohub.org/tools/nanoconfinement/.

Additional documentation and source code for this particular application can be found here:

- https://github.com/softmaterialslab/nanoconfinement-md

B. Library Dependencies

Our parallelized simulations are based on MPI/OpenMP hybrid methodology and use the following libraries to compile and run.

- GNU programming environment
- boost/1.65.0
- gsl

At the beginning of the simulation, the input parameters are fed into a machine learning procedure that requires Python 3.6 programming environment. It uses the following Python libraries:

- numpy 1.14.2
- pandas 0.22.0
- scikit-learn 0.19.1

For all the aforementioned libraries, the environment variables should be set by the user according to their system configuration.

C. Installation Instructions

- Download or git clone np-electrostatics-lab project into a directory.
- "make install" to build and install the project. This will create the executable and install the executable into root directory.

D. Evaluation Workflow

There are two ways to execute the application:

1) Run on a computing cluster using the provided jobscript: here all the input parameters are embedded in the jobscript. The jobscript can be edited to change the input parameters.
2) Run on a local computer using python: here the input parameters are supplied as command line arguments. An example command highlighting key inputs is noted below:

- python np-electrostatics -n 1 -d 4 -a 2.6775 -b 14.28 -e 2 -E 78.5 -V -60 -v 1 -C 0.1 -g 1082 -T 0.001 -S 10000000 -B 0.025

Before running the program, the "OMP_NUM_THREADS" variable needs to be set to the available number of cores in the user computer. The help menu can be accessed via "python np-electrostatics –help" to get more information about the program inputs and outputs. The user can select the number of MPI processes and OpenMP threads per MPI process by changing the -n and -d flags.

In the above command, "a" is the nanoparticle radius, "b" is the simulation box radius, "e" is the permittivity of the NP "E" is the permittivity of the environment "V" is the NP charge, "v" is the counterion valency, "C" is the salt ion concentration, "g" is the mesh size, "T" is the timestep, "S" is the simulation steps, and "B" is the bin width to compute the density distribution.

E. Results Analysis and Discussion

The simulation framework generates information about the ion dynamics including energy profiles, temperature profiles, ion trajectories, density profiles etc. These data files will be stored under outfiles and datafiles folders after the simulation is completed. You can confirm the success of the simulation by checking energy conservation factors: $R < 0.05$ and $R_v < 0.15$ and stability and accuracy factor: $f_d < 1\%$.

To test and replicate the benchmark results we provided in the paper, the MPI processes and OpenMP threads can be changed. We have bechmarked our application in BigRed II computing cluster (which is in the TOP500 list) using CPU nodes which has 16-core per node. It is expected that following the procedure outlined here will replicate the results in the paper.