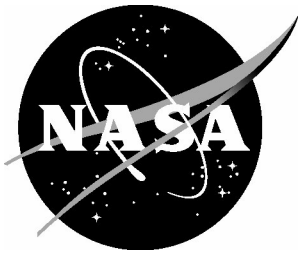
NASA/TM–2019-220431



AladynPi – Adaptive Neural Network Molecular Dynamics Simulation Code with Physically Informed Potential: Computational Materials Mini-Application

Vesselin I. Yamakov
National Institute of Aerospace, Hampton, Virginia

Edward H. Glaessgen
Langley Research Center, Hampton, Virginia

December 2019

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NTRS Registered and its public interface, the NASA Technical Reports Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA Programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counter-part of peer-reviewed formal professional papers but has less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI program, see the following:

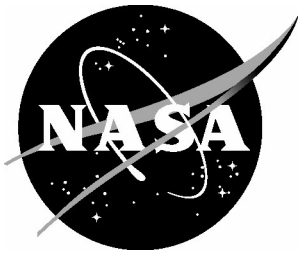
- Access the NASA STI program home page at http://www.sti.nasa.gov

- E-mail your question to help@sti.nasa.gov

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA/TM–2019-220431



AladynPi – Adaptive Neural Network Molecular Dynamics Simulation Code with Physically Informed Potential: Computational Materials Mini-Application

Vesselin I. Yamakov
National Institute of Aerospace, Hampton, Virginia

Edward H. Glaessgen
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

December 2019

Acknowledgments

The development of the AladynPi mini-application software was initiated through funding from the NASA High-Performance Computing Incubator project. V. Yamakov is sponsored through cooperative agreement NNL09AA00A with the National Institute of Aerospace. The authors are especially grateful to Yuri Mishin from George Mason University for providing the mathematical algorithm implemented in the code, and for indepth discussions throughout this project. The authors are also especially grateful to James Hickman from the National Institute of Standards and Technology for providing the trained neural network for silicon used in this package, and for actively assisting with its implementation.

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

This report provides an overview and description of commands used in the Computational Materials mini-application, AladynPi. AladynPi is an extension of a previously released mini-application, Aladyn (https://github.com/nasa/aladyn; Yamakov, V.I., and Glaessgen, E.H., NASA/TM-2018-220104). Aladyn and AladynPi are basic molecular dynamics codes written in FORTRAN 2003, which are designed to demonstrate the use of adaptive neural networks (ANNs) in atomistic simulations. The role of ANNs is to efficiently reproduce the very complex energy landscape resulting from the atomic interactions in materials with the accuracy of the more expensive quantum mechanics-based calculations. The ANN is trained on a large set of atomic structures calculated using the density functional theory method. An input for the ANN is a set of structure coefficients, characterizing the local atomic environment of each atom, for which the atomic energy is obtained in the ANN inference process. In Aladyn, the ANN gives directly the energy of interatomic interactions. In AladynPi, the ANN gives optimized parameters for a predefined empirical function, known as bond-order-potential (BOP). The parameterized BOP function is then used to calculate the energy. AladynPi code is being released to serve as a training testbed for students and professors in academia to explore possible optimization algorithms for parallel computing on multicore central processing unit (CPU) computers or computers utilizing manycore architectures based on graphic processing units (GPUs). The effort is supported by the High Performance Computing incubator (HPCi) project at NASA Langley Research Center.

1. Introduction

The use of Adaptive Neural Networks (ANN) in atomistic simulations follows a recent effort in materials science to employ machine-learning methods in reproducing materials properties from physics-based first principles [1]. ANNs, when properly trained, have been proven to successfully emulate very complex functional dependences, which are impossible or computationally very expensive to calculate directly [2]. Atomic energies, defined by quantum mechanics, are an example of such complex calculations. In most cases, approximate methods based on Density Functional Theory (DFT) are used [3]. The computational cost of DFT methods typically scale as $O(N^3)$, with N being the number of atoms in the simulated system. Even if the most modern supercomputers are employed, this cubic scaling makes the method applicable only to relatively very small systems of a few hundred to a few thousand atoms.

A conventional approach to decrease computational cost in atomistic simulations is to use approximate functional forms, empirically fitted through a set of variable parameters, to emulate atomic energies as direct functions of atomic coordinates. These empirical functions are tailored to be relatively simple to compute, while still preserving some of the features of the quantum calculations related to many-body interactions. Examples include the Embedded-Atom-Method (EAM) potential, which is based on the effective medium theory approximation [4], or the 3-body Tersoff-type potentials aimed at reproducing the angular dependence of the covalent chemical bond [5]. Nevertheless, empirical potentials

were shown to be substantially less accurate compared to quantum calculations [6], and only applicable to very specific atomic configurations or predefined crystallographic phases.

Another approach is to use heuristic machine learning (ML) methods to predict atomic energies based on a limited knowledge of the closest atomic surrounding, rather than the whole system. ML methods reduce the computational scaling to being proportional to N, which allows simulations of orders of magnitude larger systems without compromising accuracy. Recent studies have shown that ANNs can be successfully used to predict the complex atomic energy landscape in a given material [7,8] for simulating the atomic motions through the method of molecular dynamics (MD) [9]. In addition, ANNs expressed in the form of a series of matrix operations are highly parallelizable, thus being able to benefit fully from the latest generation of supercomputers, and can potentially become equal or better in computational speed to empirical potentials. In this way, ANNs are proving to be a promising stepping stone between the high accuracy of first principle quantum mechanics methods and the high computational efficiency of the classical empirical potentials in molecular dynamics simulations.

A substantial weakness of using ANNs, and ML techniques in general, is that the results are unpredictable, and generally poor, outside the training set. In the case of atomistic simulations, the configurational space of possible atomic structures is enormous, and it is practically impossible to be covered entirely in a training set. Thus, it is essential for the created ANN-based potential to have a good transferability outside the training set to guarantee reliable simulation results. To address this shortcoming, a new approach [10] is to combine the heuristic ANN method with empirical potential functions, that incorporate some of the basic physics behavior of the system, such as metallic or covalent bonding, excluded volume interactions (atomic repulsion at small distances), etc. The result is known as a physically-informed-neural-network (PINN), which combines both heuristic and physics-based knowledge to yield more accurate and reliable reconstruction of interatomic interactions. The approach has been successfully demonstrated to be superior to the traditional pure ANN inference method for the aluminum system [10]. From a computational perspective, the addition of an empirical function substantially increases (two to three times) the complexity of the calculations, which results in slower and smaller simulations. For this reason, the optimized computational implementation of the more complex mathematical algorithm of PINN is of even greater importance than the pure ANN-based approach.

AladynPi is an extension of the previously released mini-application, Aladyn [11,12], which employed a trained ANN to calculate internal potential energy of a crystal. In AladynPi, a trained ANN gives optimized parameters of a predefined empirical function, known as bond-order-potential (BOP) [13], which is then used to calculate the atomic energy and forces. The procedure is described in detail in the literature [10]. The utilized BOP incorporates in itself a pairwise repulsion term for nuclear repulsion, bond-dependent two- and three-body attraction terms to account for the attractive covalent bonding in molecules, a many-body "embedded energy" term to account for the electron density governed metallic bonding in metals, and a screening term to introduce an effective decrease of the chemical bonding due to the presence of a nearby atom. Additional terms, such as long-range Coulomb interactions can also be incorporated if needed.

As its predecessor, Aladyn, AladynPi demonstrates the process in a simple

molecular dynamics simulation of a silicon crystal to test the efficiency and accuracy of the computation. The mini-application code presented here simulates only a constant number of particles, volume, and energy (NVE) ensemble on a set of predefined single crystal structures containing from 8,000 to 1,000,000 atoms. The accuracy of the energy and force calculation is monitored by following the energy conservation law in the system (i.e., the total computed energy must be constant during the simulation). The trained ANN for this demonstration code was provided by J. Hickman from the National Institute of Standards and Technology. The AladynPi code is meant to serve as a test and training case for students and academia for optimization on parallel multicore central processing unit (CPU) computers or massively parallel manycore GPUs architectures. A successful optimization of the Aladyn code will show which optimization strategies work best for PINN-based simulations that can be implemented in some of the MD simulation codes used by NASA to achieve increased computational efficiency and enhanced capability to simulate large-scale atomic structures with DFT precision.

2. Code Description and Algorithm

2.1. General description

AladynPi demonstrates the use of an ANN in combination with a BOP function in calculating atomic energy and forces in a silicon crystal, which are then used to perform a step integration of the equations of motion of all atoms to simulate structure evolution. The block scheme of the algorithm is given in Figure 1. At the beginning of the simulation, AladynPi reads the input structure as a list of atomic coordinates and velocities, together with the parameters of a trained ANN. The atomic velocities define the initial temperature of the system. Following the work by Pun et al. [10], atomic coordinates are used to create a set of Local Structure Parameters (LSPs) [7,8,10] defined for each atom as functions of the relative positions of its neighbors contained in a sphere of cut-off radius, r_c^{LSP}. A fast search for neighbors in the vicinity of r_c^{LSP} is performed by applying the link-cell method [9].

The LSPs are used as an input to the ANN. In the previous mini-application, Aladyn [11,12], the output of the ANN was used to directly predict the atomic energy and forces after differentiation. This option is still retained in AladynPi when an ANN potential is used as an input to the code. In the case of a PINN potential, the ANN is used to predict the coefficients in the BOP function, which is then used to compute the energy of an atom. To emphasize the two different options, the bold-contour boxes in the flowchart reflect the part of the algorithm which is directly inherited from Aladyn, while the shaded boxes are specific only for the PINN algorithm in AladynPi.

Following the energy calculation, interatomic forces are then calculated based on the spatial gradient of the energy and are used to solve the Newtonian equations of motion to evolve the system in a classical molecular dynamics algorithm. For faster performance, AladynPi uses analytically differentiated energy equations, for which the ANN equations have also been differentiated with respect to the atomic coordinates, and are provided as part of the ANN computation.

AladynPi contains two execution kernels. One is built for using Open Multi-Processing (OpenMP), and the other is built for using Open Accelerators (OpenACC) programming interfaces. When compiled for both interfaces (using the PGI compiler, with the "**makefile.pgi**" file and the option ACC=TRUE, provided in the AladynPi package), the code automatically checks if a graphic processing unit (GPU) accelerator is present, and if so, it uses the OpenACC kernel, otherwise, the OpenMP kernel is executed.

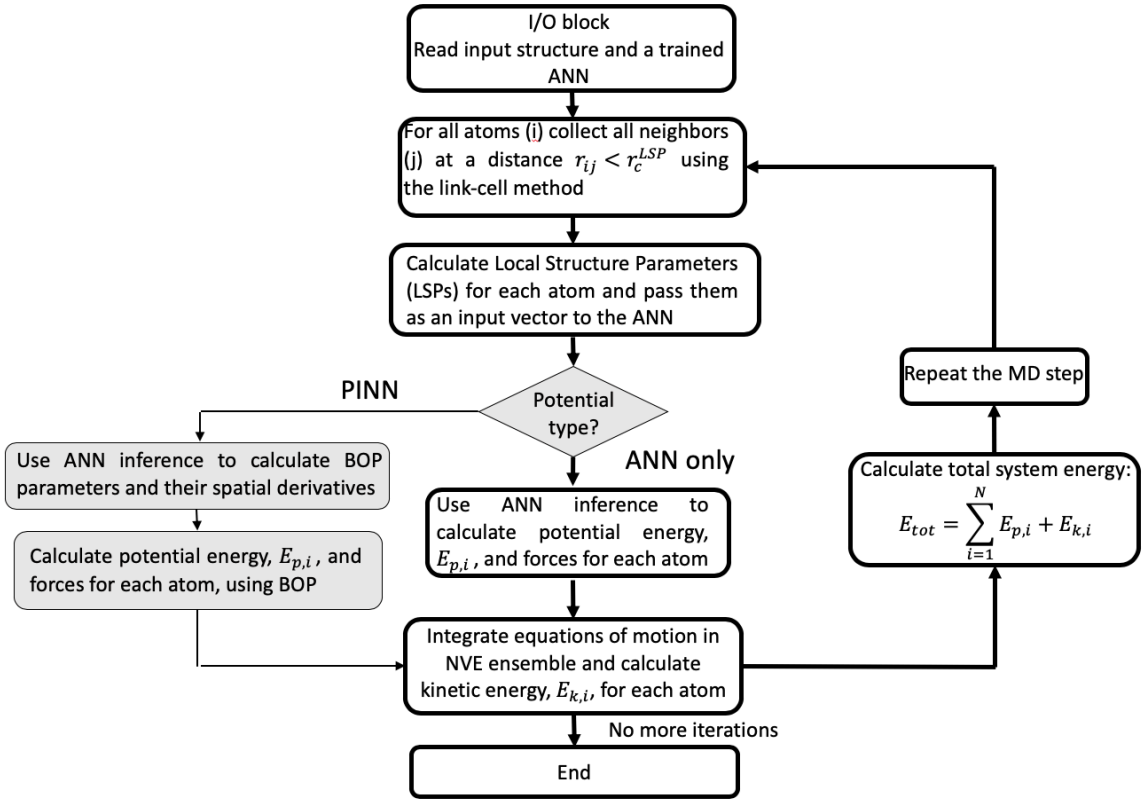


Figure 1. Flowchart summarizing the algorithm implemented in AladynPi for performing ANN-based molecular dynamics simulation.

2.2. Local structure parameters

After identifying all neighbors (*j*) in the cut-off radius for each atom (*i*), the code calculates individual LSPs for this atom. The full set of equations, describing the LSP coefficients, $G_i^{(l,s)}$, are given as follows:

$$G_i^{(l,s)} = sinh^{-1}\,\Gamma_i^{(l,s)} = ln\left(\Gamma_i^{(l,s)} + \sqrt{{\Gamma_i^{(l,s)}}^2 + 1}\right), \qquad (1)$$

with

$$\Gamma_i^{(l,s)}(r_{ij}) = \sum_{j,k\neq i}^{r_{ij}\leq r_c} P_l(\cos\theta_{ijk}) f_s(r_{ij}) f_s(r_{ik}) \quad (l = 0,1,2,4,6;\ \ s = 1,2,..\ 8), \qquad (2)$$

where $P_l(\cos\theta_{ijk})$ are Legendre polynomials of order ($l = 0,1,2,4,6$) defined through the iteration:

$$P_{l+1}(x) = [(2l+1)xP_l - lP_{l-1}]/(l+1)\,; \quad P_0(x) = 1; \quad P_1(x) = x, \tag{3}$$

and θ_{ijk} is the bond angle between the (*i*-*j*) and (*i*-*k*) bonds of atom (*i*), which expressed through the relative cartesian interatomic coordinates $(x_{ij} = x_j - x_i,\ y_{ij} = y_j - y_i,\ z_{ij} = z_j - z_i)$, and $(x_{ik} = x_k - x_i,\ y_{ik} = y_k - y_i,\ z_{ik} = z_k - z_i)$, is:

$$\cos\theta_{ijk} = (x_{ij}x_{ij} + y_{ij}y_{ij} + z_{ij}z_{ij})/(r_{ij}r_{ik}). \tag{4}$$

The functions

$$f_s(r_{ij}) = \frac{1}{r_s} e^{-(r_{ij}-r_s)^2/\sigma^2} f_c(r_{ij}), \tag{5}$$

are Gaussians with a set of expectation values, r_s ($s = 1,2,..\ 8$), defined to probe the surrounding atomic environment at eight distances from the central atom. To limit the range of interactions, these Gaussians are multiplied by a cut-off function,

$$f_c(r_{ij}) = \begin{cases} \frac{(r_{ij}-r_c)^4}{d_c^4+(r_{ij}-r_c)^4} & : \quad r_{ij} \le r_c^{LSP} \\ 0 & : \quad r_{ij} > r_c^{LSP} \end{cases}, \quad \text{and} \quad d_c = 0.5 = const, \tag{6}$$

which provides a smooth transition to 0 when r_{ij} approaches r_c^{LSP}.

The specific choice of (l,s)-set in Eq. (2), including the values of $\sigma, r_{s=1,2,..8}$, r_c^{LSP}, and d_c, are determined on a case-by-case basis during training of the ANN for a given system. These parameters are provided in the neural network potential file, "**PINN.dat**". The resulting set of LSPs coefficients, $G_i^{(M_0=1,..40)}$ from Eq. (1), where M_0 counts all (l,s)-combinations, 40 in total, as given in Eq. (2), are supplied as an input vector to the first input layer of the ANN.

2.3. Artificial neural network

The implemented ANN is a forward-propagating neural network [7], consisting of an input first layer, one or more hidden layers, and an output layer. Each layer n of atom (i) is represented as a vector $\vec{\boldsymbol{u}}^{(n)}(i) = \left(u_1^{(n)}(i), u_2^{(n)}(i), \ldots\ u_{M_n}^{(n)}(i)\right)$ of length M_n, with $M_0 = 40$ set as the length of the $\vec{\boldsymbol{G}}_i$ vector. The mathematical form of the ANN is expressed in matrix form through the iterations

$$\vec{\boldsymbol{u}}^{(1)}(i) = \vec{\boldsymbol{G}}_i * \hat{\boldsymbol{w}}^{(0,1)} + \vec{\boldsymbol{b}}^{(1)} \tag{7a}$$

$$\vec{\boldsymbol{u}}^{(n)}(i) = \vec{\boldsymbol{f}}\left(\vec{\boldsymbol{u}}^{(n-1)}(i)\right) * \hat{\boldsymbol{w}}^{(n-1,n)} + \vec{\boldsymbol{b}}^{(n)}; \quad n > 1. \tag{7b}$$

The LSPs, $\vec{\boldsymbol{G}}_i$, of atom (i), are used as an input to the first layer, $\boldsymbol{u}^{(1)}(i)$ in Eq. (7a), where they are weighted by the dot product $(*)$ with the weight matrix $\widehat{\boldsymbol{w}}^{(0,1)}$ of size $(M_0 \times M_1)$. Next, layers, $\vec{\boldsymbol{u}}^{(n)}(i)$, are calculated using Eq. (7b), where the input from the previous layer, $\vec{\boldsymbol{u}}^{(n-1)}(i)$, is modified through a transfer function

$$f(\boldsymbol{u}) = \frac{1}{1+e^{-\boldsymbol{u}}}. \tag{8}$$

The last layer gives a set of coefficients, which become parameters, $(\chi_1, \chi_2, \dots)_i$, in the BOP function, used to calculate the energy of atom (i),

$$(\chi_1, \chi_2, \dots)_i = \vec{\boldsymbol{u}}^{(last)}(i). \tag{9}$$

2.4. Bond order potential

The BOP function, giving the potential energy, E_i, of atom (i) is defined as:

$$E_i = \frac{1}{2}\sum_{j\neq i}\left[e^{(A-\alpha r_{ij})} - S_{ij}b_{ij}e^{(B-\beta r_{ij})}\right]f_c(r_{ij}) + W_i \tag{10}$$

where the coefficients, A and α, in the first term define repulsion, B and β, in the second term define attraction. The attraction term also includes a multiplication with a screening coefficient, S_{ij}, and with a bond-order parameter, b_{ij}.

The screening coefficient is calculated as

$$S_{ij} = \prod_{k\neq i,j}^{r_{ik},\ r_{jk}<1.5R_c} S_{ijk}, \tag{11a}$$

where

$$S_{ijk} = 1 - f_c(r_{ik} + r_{jk} - r_{ij})e^{-\lambda(r_{ik}+r_{jk}-r_{ij})}. \tag{11b}$$

From Eq. (11a) and Eq. (11b), it follows that $S_{ij} \in (0, 1)$, with 0 representing full screening, and 1 representing no screening.

The cut-off function, $f_c(x)$ in Eq. (11b), is the same as given through Eq. (6) with the only difference that the cut-off radius is r_c^{BOP} instead of r_c^{LSP}. There is a fixed relation between r_c^{BOP} and r_c^{LSP}, which is defined by the screening function as (see Appendix)

$$r_c^{LSP} = \frac{3}{2}r_c^{BOP}. \tag{11}$$

The bond-order parameter, b_{ij}, is defined as

$$b_{ij} = (1 + z_{ij})^{-1/2}, \tag{12a}$$

where

$$z_{ij} = \sum_{k\neq i,j}^{r_{ij},\ r_{ik}<R_c} a f_c(r_{ik}) S_{ik} \left(\cos\theta_{ijk} - h\right)^2. \quad (12b)$$

In addition, an embedded term, W_i, is added to represent metallic bonding, when present. The definition of W_i is given as:

$$W_i = -\sigma \psi_i^{1/2} \quad (13a)$$

With

$$\psi_i = \sum_{j\neq i} f_c(r_{ij}) S_{ij} b_{ij}. \quad (13b)$$

Finally, the total system potential energy, E, is obtained as a sum of the potential energies of all atoms

$$E = \sum_i E_i. \quad (14)$$

Equations (1) through (14) contain 8 fitting parameters: $(A, \alpha, B, \beta, \lambda, a, h, \sigma)$ which are given by the 8-component output vector, $(\chi_1, \chi_2, \ldots, \chi_8)_i$, of the ANN (Eq. 9), produced for atom (i). In this way, the BOP equations are customized by the ANN for each individual atom according to its specific atomic surroundings, producing an accurate estimate of its potential energy, E_i.

The forces acting on atom (i) are calculated as the spatial derivatives of E. The overall procedure of obtaining E, starting from Eq. (1) through Eq. (14), form a complex function:

$$E = E(\{r_{ij}\}) = E\left(\left\{\left(\chi_1\left(\vec{G}_i(\{r_{ij}\})\right), \chi_2, \ldots, \chi_8\right)_i\right\}, \{r_{ij}\}\right). \quad (15)$$

Analytical differentiation of Eq. (15), using the chain rule for complex function differentiation, is encoded in AladynPi, allowing for fast and efficient force calculations. Once the forces are known, a high precision 5-th order predictor-corrector scheme [14] is used to integrate the Newtonian equations of motion for each atom. The use of a high-order predictor-corrector integrator allows for accurate monitoring of the energy of the system [15] to identify any erroneous deviations from the energy conservation law as the system evolves.

3. Code Execution

3.1. Input files

For proper execution, AladynPi needs the following input files: an input model structure

file, "**structure.plt**", and a file defining a trained ANN, "**PINN.dat**".

3.1.1. Input structure file: structure.plt

The input atomic structure is given in a text file format, named "**structure.plt**", which lists all of the atoms in the structure with their chemical type and position in Cartesian coordinates. To preserve compatibility with other MD codes, the file format follows a simplified version of the format, called "plt", where some of the header information is preserved, but not used. The first nine lines in the file form the file header describing the dimensions of the system and the number of atoms it contains.

Example:

--- structure.plt ---

```
-0.1008890500E+02 -0.1008890500E+02 -0.1008890500E+02   ! -h11/2 -h22/2 -h33/2 initial
 0.1008890500E+02  0.1008890500E+02   0.1008890500E+02   !  h11/2  h22/2  h33/2 initial
-0.1008890500E+02 -0.1008890500E+02 -0.1008890500E+02   ! -h11/2 -h22/2 -h33/2 current
 0.1008890500E+02  0.1008890500E+02   0.1008890500E+02   !  h11/2  h22/2  h33/2 current
    1    500    500    500                              ! N_elements N_atoms n/a n/a
 0.67248840E+01      1      1      1                     ! n/a
   -1     -1     -1                                     ! n/a
    0      0                                            ! n/a
-0.3346355136E+01     95.2                              ! Pot.energy/atom, T of the system
    1  0.9950327408E+01  0.1000432080E+02 -0.9896154520E+01  1  0  ! id X Y Z chem.type constraint
    2 -0.8166704053E+01 -0.8205995631E+01  0.1005817294E+02  1  0  ! id X Y Z chem.type constraint
     .
  500  0.5821800387E+01  0.8042023494E+01  0.8000757656E+01  1  0
1                                                ! a separation line between coordinates and velocities
    1 -0.1393116149E+01 -0.2181636385E+01  0.1421376560E+01    ! id Vx Vy Vz
    2 -0.1969236850E+01  0.1003207253E+01  0.2951624507E+00    ! id Vx Vy Vz
     .
  500  0.2141071866E+01 -0.5388138108E+00  0.1865096587E+01
0                                                            ! end of file
```

In the above example, h_{11}, h_{22}, and h_{33} are the system dimensions in the *x*-, *y*-, and *z*- directions, given in (Å). The first two lines give the initial system dimensions (not used in AladynPi), while the third and the forth lines give the current dimensions, which are used at the start of the simulation. The fifth line describes the structure content: *N_elements* gives the number of chemical elements present in the system, *N_atoms* gives the number of all the atoms in the system. The last two numbers are not used in AladynPi, and are set equal to *N_atoms* to preserve compatibility with plt-file format. The next three lines are also not used in AladynPi.

The ninth line gives the average potential energy per atom of the system, expressed in electron-volts (eV), and the system temperature, *T*, expressed in Kelvin (K). The potential energy value is used for verification when compared with the calculated energy

at the start of the simulation. Deviations larger than 0.1% from that value are reported as a warning for the user to verify the implemented potential or if there were changes in the file. *T* is the system temperature of the last simulation and is derived from the average kinetic energy of all the atoms.

The atoms are listed after the header. Each atom is described by its identification (ID) number, atomic position given in Cartesian (x, y, z) coordinates in Å, a number identifying the associated chemical element, and a code number for the constraint degrees of freedom for this atom, if any. The atomic velocities, if available from a previous MD simulation, are listed after the atomic coordinates, separated by a line with a nonzero number ("1"), indicating continuation of the file. The file ends with a line containing a 0 number, indicating "end-of-file".

3.1.2. Input potential and adaptive neural network files: pot.dat and ann.dat

The type of interacting atoms and the source for the interatomic potential are defined in the "**pot.dat**" file. This file gives the number and type of the chemical elements in the structure, the potential functional type number of the interatomic potential implemented, followed by a list of files which define the interatomic potential between these elements.

Example:

An example of a "**pot.dat**" file for a silicon system described is the following:

```
--- pot.dat ---

1 - number of chemical species in the system
'Si'  28.085   ! element symbol and atomic mass
100            ! straight neural network potential
'./PINN.dat'     ! filename containing the neural network parameters
```

The "**PINN.dat**" file contains all of the parameters for the LSP functions, and the weights and biases of all the layers of the trained ANN. The file format is shown in the example below.

```
--- PINN.dat ---

6 0.000000 1 - ANN version, reference LSPs, and type of the activation function.
1 - number of chemical species in the system.
Si 28.085500
0 0.50000 4.5 1.0 0.5
5 0 1 2 4 6
8 2.0000 2.2860 2.5710 2.8570 3.1430 3.4290 3.7140 4.0000
1 10.787010 5.237710 4.040920 1.365000 0.104528 0.979074 0.891061 0.803526
4 40 16 16 8
 3.04159433e-01   0.0000
```

-1.71123564e-01 0.0000
1.28102273e-01 0.0000

The first line specifies the parameters to identify the type of the ANN used. For example, if the first number is 6, then the ANN output is used to set the parameters of the BOP function in the PINN algorithm. If the first number is 5, then the ANN output gives the atomic energy directly. The next two numbers are used to specify some additional variations in the ANN type, such as providing the possibility of using a reference LSPs, in addition to the calculated ones, together with the possibility to apply different forms of activation functions in Eq. (8). The last two numbers are not used in AladynPi, as only one type of the LSPs and of the activation function are used, as described in Sec. 2.

The next two lines give the number and type of the chemical elements for which the ANN has been trained. For demonstration purpose and simplicity, AladynPi works only with monoatomic systems.

Line 4 contains parameters that are fixed (i.e., not predicted by the ANN) in the PINN formulation. Those are, from left to right: a flag allowing for different types of LSP functions (not active in AladynPi); minimum and maximum range of the BOP potential (the later being equivalent to r_c^{BOP}, giving also $r_c^{LSP} = 1.5r_c^{BOP}$ in Eq. 6); σ in Eq. (5); and d_c in Eq. (6).

Line 5 gives the number, and the order, l, of the Legendre polynomials in Eq. (3).

Line 6 gives the number and values of Gaussians expectation points, r_s $(s = 1,2,..\ 8)$ in Eq. (5).

Line 7 gives the number of the BOP parameters predicted by the ANN, together with their baseline values, when this number is not zero. The baseline BOP parameters are used as fixed parameters, and the ANN provides only corrections to them, rather than their absolute values. It was found that, while principally no different from providing the absolute BOP parameters, the use of an ANN which gives only corrections to a baseline set of parameters helps significantly in the training process. It also gives a possibility for using lower floating point (fp) precision (single or even half fp formats) in the LSPs and ANN computations, which can benefit from the use of highly efficient tensor-core GPUs, specifically designed for Artificial Intelligence (AI) training and inference.

Line 8 gives the structure of the ANN which, for the provided Si potential, consists of 4 layers with 40 nodes in the first layer, 16 nodes in the second and third layers, and 8 nodes in the fourth layer. The eight output values of the 4-th layer are the BOP fitting parameters (see Eq. 9) arranged as: $(A, \alpha, B, \beta, h, \sigma, a, \lambda)$. The number of ANN layers and coefficients in each layer are chosen during the preparation of the potential through an optimization procedure for training the ANN [10].

The rest of the **PINN.dat** file gives the weights, $w_{pq}^{(s)}$, and biases, $b_p^{(s)}$, of all of the layers listed in the following order: $w_{pq}^{(1)}$, $b_p^{(1)}$, $w_{pq}^{(2)}$, $b_p^{(2)}$, $w_{pq}^{(3)}$, $b_p^{(3)}$, and $w_{pq}^{(4)}$, $b_p^{(4)}$, where in this specific example, $p = (1,..40)$, $q = (1,..16)$ in $w_{pq}^{(1)}$ and $b_p^{(1)}$; $p = (1,..16)$, $q = (1,..16)$ in $w_{pq}^{(2,3)}$ and $b_p^{(2,3)}$; and $p = (1,..16)$, $q = 8$ in $w_{pq}^{(4)}$ and $b_p^{(4)}$.

3.2. Output files

As a result of the simulation, AladynPi produces the following output files: (i) an output

structure file, (ii) an output data file, and (iii) a log file.

The output structure file has the same format as the input structure file "**structure.plt**", but its name is appended with the number of the performed MD steps as, for example "**structure.00123456.plt**" is the name of an output structure file after 123,456 MD steps. The output file can be used directly as an input file, after renaming to "structure.plt", so that a follow up simulation can be started where the first simulation has been interrupted.

The results are given as a printout on the screen (that can be redirected to a file) in columns, giving the run time-step, the simulated time in femtoseconds (10^{-15} s), kinetic and potential energy, total system energy, and temperature in (K):

Run step	Time(fs)	Ek	Ep	Etot	T(K)
0	0.00	0.01230883	-3.34635515	-3.33404633	95.23
1	1.00	0.01238930	-3.34642943	-3.33404013	95.85
2	2.00	0.01246350	-3.34650749	-3.33404399	96.42
3	3.00	0.01254594	-3.34658884	-3.33404290	97.06
4	4.00	0.01263003	-3.34667294	-3.33404291	97.71
5	5.00	0.01271631	-3.34675922	-3.33404291	98.38
6	6.00	0.01280416	-3.34684707	-3.33404291	99.06
7	7.00	0.01289296	-3.34693587	-3.33404290	99.74
8	8.00	0.01298207	-3.34702497	-3.33404290	100.43
9	9.00	0.01307082	-3.34711372	-3.33404290	101.12
10	10.00	0.01315856	-3.34720146	-3.33404290	101.80

3.3. Command line options

To control the execution, AladynPi accepts the following command-line options:

-n # – Specifies the number [# = 1,2, ..] of iterations (molecular dynamics steps - MDS) of the system evolution. The timestep of each MDS is equal to 1 fs (10^{-15} s).
Default value: -n 10.
Example: aladyn_pi –n 10 ! executes 10 MDS.

-m # – Specifies the measurement period in MDS.
Default value: -m 1.
Example: aladyn_pi –m 5 ! measurements of the system state (energy and temperature) are taken and reported every fifth MDS.

All of the command line options are optional, and if missing, the default value will be used.

4. Source Code Description

The source code of AladynPi, written in FORTRAN 2003, consists of several files. The main program with subroutines global for the entire code are in "**aladyn_pi.f**". The

remaining files contain modules as listed in Table 1.

Module Name	File	Contains
constants	aladyn_pi_mods.f	some constants used throughout the code
sim_box	aladyn_pi_mods.f	variables and subroutines defining the simulated system box
Atoms	aladyn_pi_mods.f	variables and subroutines related to atomic structure
pot_module	aladyn_pi_mods.f	variables and subroutines related to the form of the interatomic potential, such as cut-off distance, potential file type (ANN in this case), etc.
string_mod	aladyn_pi_mods.f	variables and subroutines related to string operations
IO	aladyn_pi_IO.f	input/output procedures and functions
MEASURE	aladyn_pi_MSR.f	data reporting procedures and functions
MD	aladyn_pi_MD.f	procedures and functions related to the MD simulation, such as the predictor-corrector integrator.
ANN_OMP	aladyn_pi_ANN_OMP.f	procedures and functions which calculate the LSPs of each atom and perform the ANN computation using OpenMP programming interface
ANN_ACC	aladyn_pi_ANN_ACC.f	procedures and functions which calculate the LSPs of each atom and perform the ANN computation using OpenACC programming interface

Table 1: Description of the existing modules in the code, with the file where they are placed, and what type of subroutines and functions they contain.

5. Summary

This report presents the basic algorithm and software description of the AladynPi mini-application, which is an extension of the previously released Aladyn mini-application. These are a part of a series of Computational Materials mini-applications developed and released by NASA to assist the high-performance computing effort in increasing the performance of the simulation and modeling tools in materials science. AladynPi

demonstrates a recent new approach in the use of artificial neural networks in atomistic simulations, known as a physically-informed neural network. The mini-application implements a neural network trained to reproduce the interatomic energy of a variety of Si crystalline structures and defects. The code is intended to be used to study the scalability and efficiency of implementing various optimization techniques on different computing platforms, including multicore systems and graphic accelerators to perform a basic molecular dynamics simulation on a Si crystal as a test example. The effort is related to the High Performance Computing Incubator (HPCI) project at NASA Langley Research Center in collaboration with George Mason University.

Appendix

Screening Function for BOP in PINN

The screening function in PINN is defined as follows:

$$S_{ij} = \prod_{k \neq i,j}^{r_{ik},\ r_{jk} < 1.5R_c} S_{ijk} \tag{A1}$$

where

$$S_{ijk} = 1 - f_c\left(r_{ik} + r_{jk} - r_{ij}\right) e^{-\lambda\left(r_{ik} + r_{jk} - r_{ij}\right)} \tag{A2}$$

represents the screening contribution of atom (k) to the (i-j) bond.

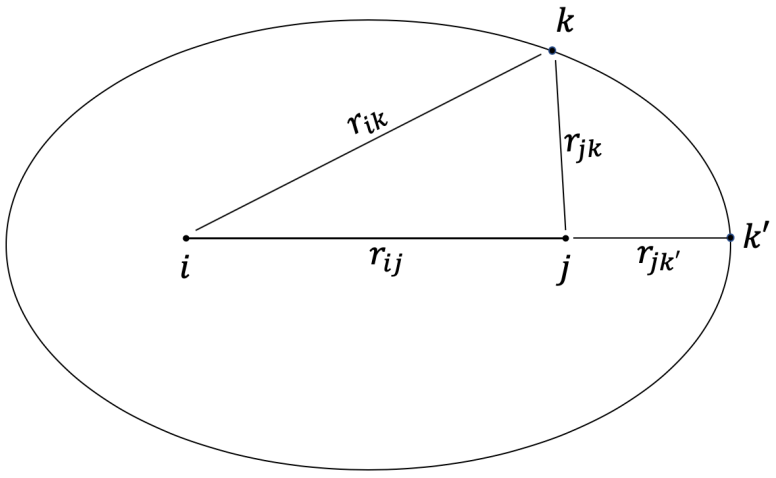


Figure A1. A 2-dimensional cross-section of the screening iso-ellipsoid for atoms (i), (j), and (k).

For a fixed r_{ij}, the condition, $r_{ik} + r_{jk} - r_{ij} = const$, defines an ellipsoid for atom (k) in the 3-dimensional space with focal points being the positions of atoms (i) and (j), from where atom (k) induces the same amount of screening on the (i-j) bond

(see Fig. A1). The cut-off function, f_c , in Eq. (A1) sets an upper bound for the screening distances as

$$r_{ik} + r_{jk} - r_{ij} \leq r_c^{BOP}. \quad \text{(A3)}$$

From Fig. (A1), it is seen that the furthest position of atom (k) with respect to atom (i) on the defined ellipsoid is the position (k'), which is on the same line with (i) and (j), at a distance

$$r_{ik'} = r_{ij} + r_{jk'}. \quad \text{(A4)}$$

Substituting Eq. (A4) in Eq. (A3) gives

$$2r_{jk'} \leq r_c^{BOP}. \quad \text{(A5)}$$

Using the short-range condition, $r_{ij} \leq r_c^{BOP}$, combined with Eq. (A4) and Eq. (A5) gives the range for the distance, r_{ik}, at which (k) can still screen (i) from (j):

$$r_{ik} = r_{ij} + r_{jk} \leq r_c^{BOP} + \frac{1}{2} r_c^{BOP} = \frac{3}{2} r_c^{BOP}. \quad \text{(A6)}$$

Equation (A6) indicates that all atoms (j) at a distance of $r_{ij} \leq \frac{3}{2} r_c^{BOP}$ from atom (i) are affecting the energy of (i). As such, in order for the ANN to correctly predict the BOP parameters, all atoms in a sphere of radius, $\frac{3}{2} r_c^{BOP}$, need to be counted, which defines the relation:

$$r_c^{LSP} = \frac{3}{2} r_c^{BOP}. \quad \text{(A7)}$$

References

[1] Mueller, T., Kusne. A. G., Ramprasad, R., "Machine Learning in Materials Science: Recent Progress and Emerging Applications", in: Parrill, A. L., Lipkowitz, K.B. (Eds.), Reviews in Computational Chemistry, 29, Wiley (2016) 186-273.

[2] Cheng, B. Titterington, D. M., "Neural Networks: A Review from a Statistical Perspective", Statistical Science 9 (1994) 2-30.

[3] Lejaeghere, K., et al., "Reproducibility in Density Functional Theory Calculations of Solids", Science 351 (2016) aad3000-1-7.

[4] Daw, M. S., Baskes, M. I., "Embedded-Atom Method: Derivation and Application to Impurities, Surfaces, and Other Defects in Metals", Phys. Rev. B 29 (1984) 6443-6453.

[5] Tersoff, J., "Empirical Interatomic Potential for Carbon, with Applications to Amorphous Carbon" Phys. Rev. Lettrs. 61 (1988) 2879-2882.

[6] Brenner, D. W., “The Art and Science of an Analytical Potential”, Phys. Stat. Sol. (b) 217, (2000) 23-40.
[7] Behler, J., Parrinello, M., “Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces”, Phys. Rev. Lett. 98 (2007) 146401-1-4.
[8] Behler, J., “Perspective: Machine Learning Potentials for Atomistic Simulations”, J. Chem. Phys. 145 (2016) 170901-1-9.
[9] Frenkel, B., Smit, B., “Understanding Molecular Simulation”, Academic Press, London, (2001).
[10] Pun, G. P. P., Batra, R., Ramprasad, R., Mishin, Y., “Physically-Informed Artificial Neural Networks for Atomistic Modeling of Materials”, Nature Communications 10 (2019) 2339.
[11] Yamakov, V.I., Glaessgen, E.H., “Aladyn – Adaptive Neural Network Molecular Dynamics Simulation Code: Computational Materials Mini-Application,” NASA/TM-2018-220104.
[12] Yamakov, V.I., Jost, G., Kokron, D.S., Mishin, Y. and Glaessgen, E.H., “High-Performance Computing Optimization for Aladyn – Adaptive Neural Network Molecular Dynamics Mini-Application,” NASA/TM-2019-220409.
[13] Gillespie, B. A. et al. Bond-order potential for silicon. Phys. Rev. B 75 (2007) 155207.
[14] Gear, C. W., “The Numerical Integration of Ordinary Differential Equations of Various Orders”, Technical Report ANL 7126 (1966) Argonne National Laboratory, Argonne, IL.
[15] Schlick, T., Skeel, R. D., Brunger, A. T., Kale, L. V., Hermans, J., Schulten, K., “Algorithmic Challenges in Computational Molecular Biophysics”, J. Comp. Phys. 151 (1999) 9-48.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
1-12-2019

2. REPORT TYPE
Technical Memorandum

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
AladynPi-Adaptive Neural Network Molecular Dynamics Simulation Code with Physically Informed Potential: Computational Materials Mini-Application

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Yamakov, Vesselin I.; Glaessgen, Edward H.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
698259.02.07.07.03.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-21087

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA-TM-2019-220431

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-
Subject Category 24
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report provides an overview and description of commands used in the Computational Materials mini-application, AladynPi. AladynPi is an extension of a previously released mini-application, Aladyn (https://github.com/nasa/aladyn). Aladyn and AladynPi are basic molecular dynamics codes written in FORTRAN 2003, which are designed to demonstrate the use of adaptive neural networks (ANNs) in atomistic simulations. The role of ANNs is to efficiently reproduce the very complex energy landscape resulting from the atomic interactions in materials with the accuracy of the more expensive quantum mechanicsbased calculations.

15. SUBJECT TERMS
High Performance Computing; atomistic simulation; metal alloy; molecular dynamics

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	20	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658