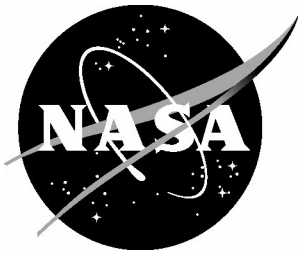
NASA/TM–2018-



# Aladyn – Adaptive Neural Network Molecular Dynamics Simulation Code: Computational Materials Mini-Application

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

*Edward H. Glaessgen*
*Langley Research Center, Hampton, Virginia*

November 2018

# 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:

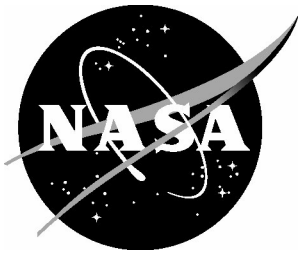
- Access the NASA STI program home page at http://www.sti.nasa.gov

- 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–2018-



# Aladyn – Adaptive Neural Network Molecular Dynamics Simulation Code: 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

November 2018

Acknowledgements

The development of the Aladyn 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 in depth discussions throughout this project.

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 commands description of the Computational Materials mini-application, Aladyn. Aladyn is a simple molecular dynamics code written in FORTRAN 2008, which is designed to demonstrate the use of adaptive neural networks (ANNs) in atomistic simulations. The role of ANNs is to reproduce the very complex energy landscape resulting from the atomic interactions in materials with the accuracy of quantum mechanics-based energy calculations. The ANN is trained on a large set of atomic structures calculated using the density functional theory (DFT) method. The Aladyn 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 related to the High Performance Computing Incubator (HPCI) project at NASA Langley Research Center.

## 1. Introduction

The use of Adaptive Neural Network (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. Use of heuristic machine learning methods to predict atomic energies based on a limited knowledge of the closest atomic surrounding, rather than the whole system, reduces the computational scaling to being proportional to $N$, which allows simulations of orders of magnitude larger systems without compromising accuracy.

Classical MD methods 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, such as 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, and applicable to very specific atomic configurations or predefined crystallographic phases. Another drawback of the empirical potentials, that has become apparent recently, is their unsuitability for massive parallelism in the latest generation of manycore computing platforms based on Graphic-Processing-Units (GPUs). Every functional form (of EAM, or Tersoff type, etc.) requires a different parallelization approach and has variable efficiency

on those platforms.

Recent studies have shown that ANNs can be successfully used to predict the complex atomic energy landscape in a given material [6,7] for simulating the atomic motions through the method of molecular dynamics (MD) [8]. 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.

The Aladyn software employs a trained ANN to calculate internal potential energy of an Al crystal and to perform a simple molecular dynamics simulation to test the efficiency and accuracy of the computation. The mini-application code presented here simulates only constant particle, volume, and energy (NVE) ensemble on a set of predefined single crystal structure of 500, 4000, and 16,000 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 is kindly given for this demonstration code by G. P. Pun and Y. Mishin from George Mason University [9]. The Aladyn 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 allow lessons learned to 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.

## 2. Code Description and Algorithm

Aladyn demonstrates the use of ANNs in calculating atomic energy and forces in a crystalline atomic structure and performing a step integration of the equations of motion of all atoms to simulate structure evolution. In this approach, the ANN predicts the energy of an atom based on its local environment. Interatomic forces are calculated based on the spatial gradient of the energy and used to solve the Newtonian equations of motion to evolve the system in a classical molecular dynamics algorithm.

The mathematical algorithm in Aladyn follows the work by Behler and Parrinello [6]. The local environment is described through a set of Local Structure Parameters (LSPs) [6,7] defined for each atom as functions of the relative positions of its neighbors contained in a sphere of radius, $r_{cut}$, the cut-off radius. A fast search for neighbors in the vicinity of $r_{cut}$ is performed by applying the link-cell method [8] where the system box is divided into approximately cubic shape cells of size slightly larger than $r_{cut}$ (Figure 1). A list of atoms is maintained for each cell. Hence, the search for a neighbor in the interaction range of an atom does not have to exceed the nearest neighbor link-cells (marked in gray in Figure 1). In a 3D system, the search range includes a cube of $3^3 = 27$ link-cells, rather than the whole system.

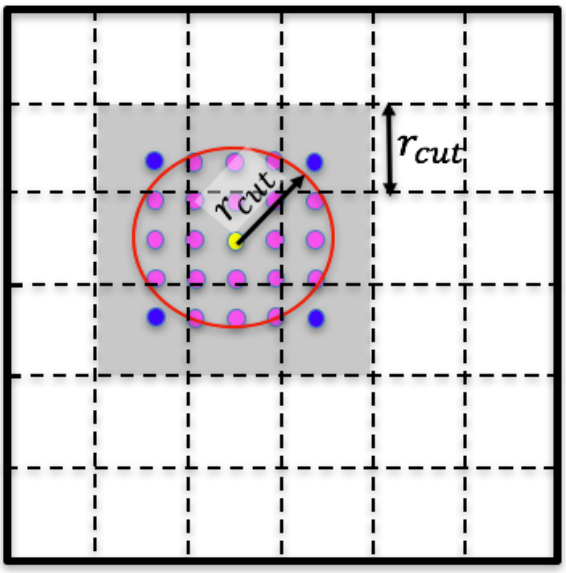



**Figure 1.** Schematic representation of the link-cell volume decomposition. Bold lines indicate the simulated system box boundaries. Dotted lines indicate the link-cell mesh with the cells in grey indicating the nearest cells, among which a search for neighbors of the central atom (in yellow) is performed.

After identifying all cut-off neighbors ($j$) for each atom ($i$) in the system, the code calculates individual LSPs for this atom. The LSPs coefficients, $G_i^{(n)}$, of atom ($i$) are expressed [9] as a product of radial functions of the interatomic distances, $r_{ij}$, between atom ($i$) and its neighbors ($j$)

$$f_R(r_{ij}) = \frac{1}{\sigma} e^{-(r_{ij}-r_0)^2/\sigma^2}, \quad (1)$$

where $\sigma$ and $r_0$ are model defined parameters, and Legendre polynomials of order ($l$), $P^{(l)}(\cos\theta_{ijk})$, of the cosine of the bond angle between atoms ($i$), ($j$), and ($k$)

$$P^{(l+1)}(x) = \left[(2l+1)xP^{(l)} - lP^{(l-1)}\right]/(l+1); \quad P^{(0)}(x) = 1; \quad P^{(1)}(x) = x. \quad (2)$$

The LSPs coefficients, $G_i^{(n)}$, are supplied as an input vector to the first input layer of the ANN. The implemented ANN is a straightforward propagating 4-layer neural network [6] with an input layer, two hidden layers, and an output layer. The mathematical form of the ANN is expressed through Equation (3) as

$$E_{p,i}, = \sum_{l=1}^{N^{(3)}} f\left(\sum_{k=1}^{N^{(2)}} f\left(\sum_{n=1}^{N^{(1)}} G_i^{(n)} w_{n,k}^{(1)} + b_k^{(1)}\right) w_{k,l}^{(2)} + b_l^{(2)}\right) + b_1^{(3)}, \quad (3)$$

where $N^{(s)}$ are the number of elements of the $s$-th layer ($s = 1,2,3$), $w_{p,q}^{(s)}$ and $b_p^{(s)}$ are the matrix and vector elements of the weights and biases of the $s$-th layer. The final output layer, formally considered as the 4-th layer, gives the predicted potential energy, $E_{p,i}$, of atom ($i$). The total system potential energy, $E_p$, is obtained as a sum of the potential energies of all atoms.

The forces, acted on atom ($i$) are calculated as first derivatives of $E_p$ using the finite differences method by virtually displacing atom ($i$) at a small distance, $\Delta\varepsilon$, in the $x$-, $y$-, and $z$- directions backward and forward around its initial $(x, y, z)_i$ – position, as given by Eq. (1) for the $x$-component of the force,

$$\frac{dF}{dx_i} = \lim_{\Delta\varepsilon \to 0} \frac{E_p(x_i+\Delta\varepsilon) - E_p(x_i-\Delta\varepsilon)}{2\Delta\varepsilon} \quad . \tag{4}$$

Once the forces are known, a high precision 5-th order predictor-corrector scheme [10] is used to integrate the Newtonian equations of motion for each particle. The use of a high-order predictor-corrector integrator allows for accurate monitoring of the energy of the system [11] to identify any erroneous deviations from the energy conservation law as the system evolves.

The block scheme of the algorithm is given in Figure 2. At the beginning of the simulation, Aladyn 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. Based on the input system geometry, the algorithm creates a link-cell mesh over the whole system box. An MD step starts with identifying all neighbors in a cut-off radius, $r_{cut}$, around each atom of the system, using the link-cell list. Using the list of neighbors, the LSPs are calculated for each atom, and supplied as an input to the ANN for potential energy and force calculation. The calculated forces are used to integrate the equations of motion for each atom and evolve the system by one MD step. The updated atomic velocities, resulting from the integration, are used to calculate the kinetic energy of each atom and the overall temperature of the system. The total system energy,

$$E_{tot} = \sum_{i=1}^{N} \left(E_{p,i} + E_{k,i}\right) = const. \tag{5}$$

is calculated and reported. $E_{tot}$ is used as a verification test of the simulation since it must remain constant during the simulation. The updated atomic positions are used to calculate new LSPs, and the next steps repeat until the end of the simulation.

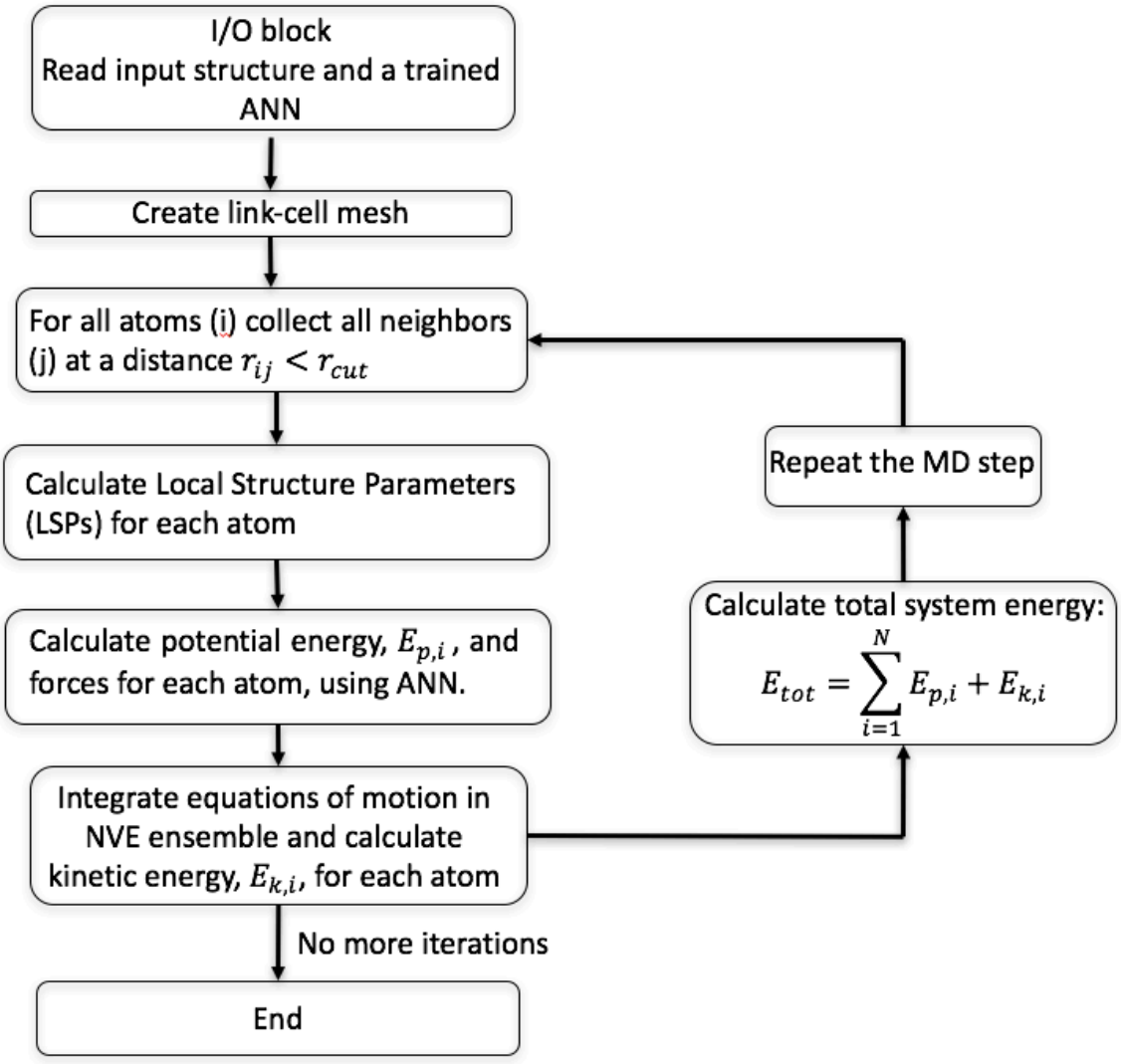



**Figure 2.** Flowchart summarizing the algorithm implemented in Aladyn for performing ANN based molecular dynamics simulation.

## 3. Code Execution

### *3.1. Input files*

For proper execution, Aladyn needs the following input files: an input model structure file, **structure.plt**, and a file defining a trained ANN, **ANN.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 positions in Cartesian coordinates. To preserve compatibility with other 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 Aladyn), 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 Aladyn, and are set equal to *N_atoms* to preserve the compatibility with plt-file format. The next three lines are also not used in Aladyn.

The ninth line gives the average potential energy per atom of the system, expressed in electron-volts (eV), and the system temperature, 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. The temperature value, *T*, given in (K), 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 an aluminum system described through a straight ANN is the following:

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

 1 - number of chemical species in the system
'Al'  26.982   ! element symbol and atomic mass
 100           ! straight neural network potential
'./ann.dat'    ! filename containing the neural network parameters
```

The **ann.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 was provided by Y. Mishin at George Mason University. The file format is given in the example below.

```
--- ann.dat ---

0 0.100000 6.000000 1.500000 1.000000 1
 12 2.00 2.20 2.40 2.60 2.80 3.00 3.40 3.80 4.20 4.60 5.00 5.40
 4 60 20 20 1
  4.07328794e-01  0.0000
  3.66106892e-01  0.0000
 -8.54926592e-02  0.0000
 -5.54124400e-01  0.0000
```

The first two lines give the parameters for the LSP function. The third line gives the structure of the ANN, which for the provided Al potential, consists of 4 layers with 60 terms in the first layer, 20 terms in the 2-nd and 3-rd layers, and 1 term (the output energy) in the 4-th layer. 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 [9]. The rest of the **ann.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, $p = (1,..60)$, $q = (1,..20)$ in $w_{pq}^{(1)}$ and $b_p^{(1)}$; $p = (1,..20)$, $q = (1,..20)$ in $w_{pq}^{(2,3)}$ and $b_p^{(2,3)}$; and $p = (1,..20)$, $q = 1$ in $w_{pq}^{(4)}$ and $b_p^{(4)}$.

*3.2. Output files*

As a result of the simulation, Aladyn 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 directly used 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 output data file has the name **results.dat**. This file stores measurements in columns at each reporting step. The following system parameters are measured and stored: kinetic and potential energy, total system energy, and temperature. The file starts with a header which identifies the meaning of each column. An example of a **results.dat** file is given below.

--- results.dat ---

| 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, Aladyn accepts the following online options as commands:

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

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

**-v #** – Selects the version number [# = 0, 1, ..] of the code, when several versions of one or more subroutines are implemented for comparison. What each version does has to be specified by the programmer in the code. In the current release, version 0 (-v 0) uses non-optimized subroutines, while in version 1 (-v 1) the subroutines energy_ANN (energy calculation) and force_ANN (force calculation) are being optimized for multithreading, when using OpenMP at compilation.
*Default value:* -v 0
*Example:* aladyn –v 1  ! executes version 1 of the code.

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 Aladyn, written in FORTRAN 2008, consists of several files. The main program with subroutines global for the entire code are in aladyn.f. The rest of the files contain modules as follows: aladyn_mods.f contains general purpose modules, such as:

MODULE constants – contains some constants used throughout the code;

MODULE sim_box – contains variables and subroutines defining the system box;

MODULE atoms – contains variables and subroutines related to atomic structure;

MODULE pot_module – contains variables and subroutines related to the form of the interatomic potential, such as cut-off distance, potential file type (ANN in this case), etc.

MODULE string_mod – contains variables and subroutines related to string operations.

File aladyn_IO.f contains MODULE IO consisted of the input/output procedures and functions.

File aladyn_MSR.f contains MODULE MEASURE consisted of the data reporting procedures and functions.

File aladyn_MD.f contains MODULE MD consisted of the procedures and functions related to the MD simulation, such as the predictor-corrector integrator of the equations of motion subroutines.

File aladyn_ANN.f contains MODULE ANN consisted of the procedures and functions which calculate the LSPs of each atom and perform the ANN computation. This is the module where the optimization efforts should be mainly focused. Currently, the code uses OpenMP calls to run on multiprocessor platforms.

## 5. Summary

This report presents the basic algorithm and software description of the Aladyn code. The code is 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. The code demonstrates the use of adaptive neural networks in atomistic simulations. The neural network, provided by Yuri Mishin's group at George Mason University, is trained to reproduce the interatomic energy of a variety of Al 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 and manycore systems in performing a basic molecular dynamics simulation on an Al 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.

## 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] Behler, J., Parrinello, M., "Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces", Phys. Rev. Lett. 98 (2007) 146401-1-4.

[7] Behler, J., "Perspective: Machine Learning Potentials for Atomistic Simulations", J. Chem. Phys. 145 (2016) 170901-1-9.
[8] Frenkel, B., Smit, B., "Understanding Molecular Simulation", Academic Press, London, (2001).
[9] Pun, G. P. P., Batra, R., Ramprasad, R., Mishin, Y., "Physically-Informed Artificial Neural Networks for Atomistic Modeling of Materials", arXiv:1808.01696v2 [cond-mat.mtrl-sci].
[10] Gear, C. W., "The Numerical Integration of Ordinary Differential Equations of Various Orders", Technical Report ANL 7126 (1966) Argonne National Laboratory, Argonne, IL.
[11] 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.