ARL-TR-8302 ● FEB 2018

US Army Research Laboratory

Expansion and Automation of the Energy Conserving Orientational Force for Calculation of Grain Boundary Mobility

by Matthew Guziewski, Shawn P Coleman, and Mark A Tschopp

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8302 ● FEB 2018

US Army Research Laboratory

Expansion and Automation of the Energy Conserving Orientational Force for Calculation of Grain Boundary Mobility

by Matthew Guziewski
Colorado State University, Fort Collins, CO

by Shawn P Coleman and Mark A Tschopp
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
February 2018	Technical Report	May–August 2017

4. TITLE AND SUBTITLE
Expansion and Automation of the Energy Conserving Orientational Force for Calculation of Grain Boundary Mobility

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Matthew Guziewski, Shawn P Coleman, and Mark A Tschopp

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-WMM-E
Aberdeen Proving Ground, MD 21005-5069

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8302

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
An algorithm is developed that automates the application of the energy conserving orientational (ECO) force bicrystal grain boundaries. This will allow for simpler and more rapid calculations of grain boundary mobility values at varying orientations and temperatures. Additionally, the ECO driving force is expanded from its original implementation for only face-centered cubic crystals to body-centered cubic structures.

15. SUBJECT TERMS
HCP, grain boundary, molecular dynamics, synthetic driving force, microstructure evolution

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Shawn P Coleman
Unclassified	Unclassified	Unclassified	UU	42	19b. TELEPHONE NUMBER (Include area code) 410-306-0697

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

This research is supported through funding from High Performance Computing Modernization Program (HPCMP) Department of Defense (DOD) Next Generation Workforce Development and made possible through an appointment at the US Army Research Laboratory (ARL) administered by the Oak Ridge Institute for Science and Education through an interagency agreement between the US Department of Energy and ARL. This work was supported in part by computer time from the DOD HPCMP at the ARL DOD Supercomputing Resource Center (DSRC), the Navy DSRC, the US Army Corps of Engineers Research and Development Center DSRC, and the US Air Force Research Laboratory DSRC. Also, special thanks to Dr Stephen Foiles (Sandia National Laboratories) for providing the initial Large-scale Atomic/Molecular Massively Parallel Simulator energy conserving orientational code that was used in these simulations.

1. Introduction

In the study of nanoscale systems, grain boundary structure, energy, and mobility can influence a wide range of material properties, such as strength, damage tolerance, and electrical conductivity.[1–3] The grain boundary properties are functions of its geometry, composition, strain state, defect content, and the like. Considering geometry alone, there are 5 macroscopic degrees of freedom to describe the relative misorientation between adjacent grains and the grain boundary plane. Three additional degrees of freedom then describe the nanoscale translation space relative to each grain. A comprehensive experimental study exploring grain boundary geometry effects on structure, energy, and mobility is not possible due to these unbounded degrees of freedom and difficulty in sample processing.[4] However, systematic computational investigations using molecular dynamics simulations have provided insights into particular grain boundary property-structure relationships.[5–7]

Molecular dynamics simulations are ideal methods for calculating grain boundary structure, energy, and mobility due to their capacity to model atomic positions and trajectories over the nanoscale. Using molecular dynamics, it is a relatively straightforward endeavor to create flat grain boundary models to systematically explore varying grain boundary geometries at different temperatures and under boundary conditions. Tracking the atomic structure of the grain boundary over time and under different conditions allows researchers to gain understanding of the specific mechanisms that influence their evolution and properties. These specific data and insights from atomistic models can then be used to generate more realistic mesoscale materials simulations. For example, recent Potts Monte Carlo simulations that incorporated multiple boundary characteristics show how grain growth can stagnate in pure metals.[8] These results point to a need for even more grain boundary structure-property data to better predict other realistic material behavior that can dominate material performance, such as abnormal grain growth.

This technical report discusses algorithmic advancements made to conduct high-throughput grain boundary mobility calculations using flat bicrystal atomistic models. Specifically, this technical note highlights the expansions to the energy conserving orientational (ECO) artificial driving force developed by Ulomek et al.[9] The ECO method applies a synthetic driving force to atoms located at the grain boundary plane to accelerate its motion. In its original form, the ECO artificial driving force was developed for only face-centered cubic (FCC) systems. The current work expands its functionality to body-centered cubic (BCC) materials and makes recommendations for expansion to hexagonal close-packed (HCP) systems.

When possible, results obtained from the ECO driving force are compared directly to the other driving force methods. In order to facilitate high-throughput grain boundary property calculations using the ECO method, a workflow is developed to automate mobility calculations over multiple grain boundaries by applying varying driving force conditions at different temperatures.

2. Background

2.1 Artificial Driving Forces

Natural grain boundary motion often occurs on timescales not traditionally accessible by molecular dynamics simulations. In order for molecular dynamics simulations to model grain boundary motion within the typical nanosecond time span, the motion must be driven by either external or internal forces. The initial methods of producing this driving force mirrored those used experimentally (i.e., shearing the crystals,[10] using curvature,[11] using gradients in dislocation density,[12] or using gradients in temperature[13]). These methods often greatly overestimated grain boundary mobility as compared with the experiments they intended to mimic and were often limited by the specific conditions used to drive grain boundary motion.

More recently, artificial driving force approaches have been created to widen the applicability of grain boundary mobility calculations. The artificial driving force promotes flat grain boundary motion in bicrystal models by making one of the grains more energetically favorable than the other. In molecular dynamics simulations, forces on the atoms are computed using the equation, $F = -\nabla u$, where u is the energy value assigned to the atoms. During artificial driving force simulations, atoms located within the bulk of each grain have a constant added energy; thus, no force is added to these atoms. Atoms located at the boundary, however, have an added energy that smoothly transitions from one grain to the next using an orientation parameter defined by the artificial driving force method. Thus, atoms at the boundary are provided an extra force to drive the motion of the grain boundary.

Two variations of artificial driving force methods have been developed that use different approaches to identify grain orientations. The first, developed by Janssens et al., uses the centrosymmetric positions of the atom neighbors to calculate an orientation parameter for each atom.[14] In select cases, this direct approach is not sufficient for accurately determining the boundary atoms. The second artificial driving force method, developed by Ulomek et al.,[9] calculates the orientations more robustly within a neighboring envelope using reciprocal space vectors. The

following sections provides more details about each method and its implementation in the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) molecular dynamics code.[15]

2.1.1 Janssens et al. Method

In the Janssens et al.[14] artificial driving force method, the desired low-energy crystal orientation is defined as the reference state. For each atom in the system, its orientation parameter is defined as

$$\xi_i = \sum_j \left| r_j - r_j^I \right| . \tag{1}$$

Here r_j is the relative positions of the nearest neighbors (12 for FCC, 8 for BCC) to particle i, and r_j^I is the reference state positions. This value is then normalized by using the positions of the second orientation for r_j. At 0 K, the result will be orientation factors of slightly more than 0 in the bulk of the reference orientation and values of 1 within the bulk of the second orientation. Between the grains, in the grain boundary regions, the orientation factor transitions between 0 and 1. As the simulation temperature increases, thermal noise can become an issue as the atoms are rarely in their ideal lattice positions due to vibrations (Fig. 1a). To address the vibrations, filtering parameters are used, ξ_{lo} and ξ_{hi}, to which orientation parameters between these values are assigned.

In order to promote the motion of the boundary, Janssens et al.[14] add a synthetic energy, u_i, to each atom based off the orientation parameter, using the expression

$$u_i = \begin{cases} 0 & \xi_i < \xi_{lo} \\ \frac{V}{2}(1 - \cos 2\omega_i) \quad with\ \omega_i = \frac{\pi}{2}\frac{\xi_i - \xi_{lo}}{\xi_{hi} - \xi_{lo}} & \xi_{lo} < \xi_i < \xi_{hi} \\ V & \xi_{hi} < \xi_i \end{cases} . \tag{2}$$

Figure 1 shows the resultant energy profile that is added to the simulation. The added energy smoothly transitions from one grain to the next and its form is continuous and differentiable. These properties enable additional forces to be computed and applied to the atoms within the grain boundary region based off the synthetic energy.

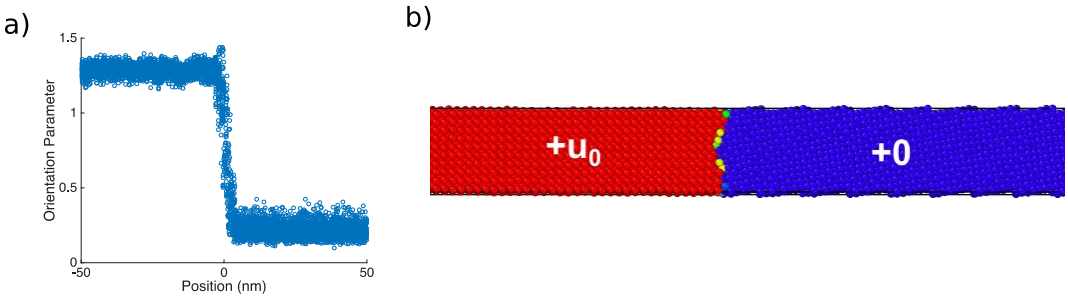

Fig. 1 a) Plot of the orientation parameter vs. position commonly seen with the Janssens artificial driving force. b) Energy to be applied based on the orientation parameter.

2.1.2 Ulomek et al. Method

The ECO artificial driving force method developed by Ulomek et al.[9] uses reciprocal space values for the basis vectors to aid in the calculation of the orientation factors, an approach inspired by diffraction theory. This methodology defines an orientation parameter

$$\chi_j = \frac{1}{N}\left[\sum_\alpha \left|\sum_k \omega\left(R_{jk}\right)e^{iQ_\alpha R_{jk}}\right|^2 - \sum_\beta \left|\sum_k \omega\left(R_{jk}\right)e^{iQ_\beta R_{jk}}\right|^2\right], \tag{3}$$

where N is the number of particles in the local neighborhood, R_{jk} are the displacement vectors between these particles, and Q_α and Q_β are sets of reciprocal space lattice vectors for the 2 orientations. The local neighborhood for this formulation is defined by a cutoff radius, R_{cut}, which is an additional user input for the ECO method. The $\omega\left(R_{jk}\right)$terms are envelope functions that drop smoothly to zero as the distance between particles j and k, respectively, approaches R_{cut}. It is given by

$$\omega\left(R_{jk}\right) = \frac{\left|R_{jk}\right|^4}{R_{cut}^4} - 2\frac{\left|R_{jk}\right|^2}{R_{cut}^2} + 1 \quad \text{for} \quad R_{jk} < R_{cut} \tag{4}$$

for $R_{jk} > R_{cut}$, w_{jk} is defined as zero.

The resulting orientation parameter, χ_j, will vary between –1 and 1 in the bulk of the 2 grains, with values transitioning between the 2 at the grain boundary. Thermal noise also causes variation from these values; however, as seen in Fig. 2a, it is significantly less than that seen with the Janssens et al.[14] method. Nevertheless, a filtering parameter, η, is used to eliminate any noise effects due to vibrations, whereby orientation parameters below $-\eta$ and above η are assigned values of –1 and 1, respectively.

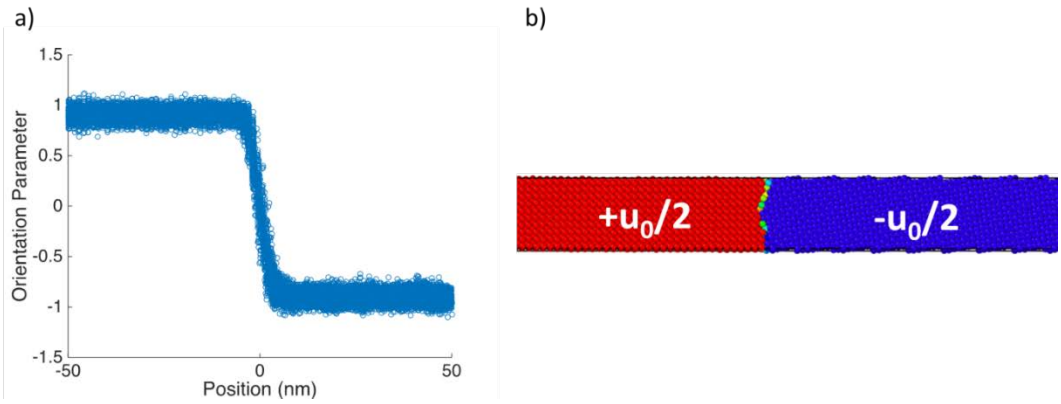

Fig. 2 a) Plot of the orientation parameter vs. position commonly seen with the ECO artificial driving force. b) Energy to be applied based on the orientation parameter.

The ECO artificial driving force method adds a synthetic energy to each atom using the expression

$$u_j(\chi_j) = \frac{u_0}{2}\begin{cases} 1 & \chi_j > \eta \\ \sin\left(\frac{2\pi}{\eta}\chi_j\right) & -\eta < \chi_j < \eta \\ -1 & \chi_j < -\eta \end{cases}. \quad (5)$$

Figure 2 shows the energy profile of the crystal for using the ECO formulation. During implementation within LAMMPS, this energy is never added to the system. Instead, forces are computed for each atom using the gradient of this added energy, and these forces are added to those already present from the interatomic potential.

Although both artificial driving force methods do induce grain boundary motion, the ECO artificial driving force is superior for multiple reasons. Perhaps most importantly is its conservation of energy. The ECO method adds zero net energy to the system due to its symmetric addition and subtraction of energy across the boundary. Its more robust orientation calculation eliminates artificial energy jumps that occurred in special cases with the Janssens method due to the grain boundary structure.

Additionally, the ECO method is less influenced by thermal noise due to its computation of orientation parameters using reciprocal space vectors. As a result, the filtering parameter can remain the same for almost every simulation, a value of η = 0.25 is suggested by Ulomek et al.[9] Conversely, the variability in orientation parameters computed using the method by Janssens et al.[14] often requires a preprocessing step to select appropriate χ values. This additional step hinders the generation of an automatic workflow for high-throughput calculations. Lastly, the Janssens et al.[14] method only considers a set number of nearest neighbors, while the ECO driving force considers all particles in the local neighborhood. This allows

for a better understanding of local structures. For these reasons, the ECO formulation was chosen as the preferred artificial driving force method in this work.

2.2 Grain Boundary Mobility

Grain boundary mobility defines the relationship between the normal grain boundary velocity and the driving force behind the motion. From rate theory, grain boundaries move as a concerted set of atomic jumps happening across the boundary. The grain boundary velocity, *v*, at varying temperatures, *T*, and driving forces *p*, can be described through a traditional Arrhenius relationship of the form

$$v = bv_0 \exp\left(-\frac{H_{GBM}}{k_B T}\right)\left(\exp\left(\frac{pV}{k_B T}\right) - 1\right), \tag{6}$$

where, *b* is the atomic jump distance, v_0 is the attempt frequency, and H_{GBM} is the activation enthalpy for the boundary motion. With the exception of the driving pressure, all the other values in this Arrhenius equation are constants; thus, the boundary velocity can be simplified by the equation

$$v = A(\exp(Bp) - 1), \tag{7}$$

where *A* and *B* are constants. Using the ECO artificial driving force methods, the driving pressure can easily be related to the energy values added to the grains using the expression

$$p = \frac{u_0}{\Omega}, \tag{8}$$

where Ω is the atomic volume.

Grain boundary mobility, *m*, is calculated from the derivative of the boundary velocity with respect to the driving force, in the low driving force limit:

$$m = \left.\frac{\delta v}{\delta p}\right|_{p\to 0}. \tag{9}$$

To calculate grain boundary mobility at a specific temperature, multiple simulations are run using varying energies. For each simulation, the boundary velocities are calculated by tracking the grain boundary position over time. A curve of the form of Eq. 7 is then fit to the driving force and velocity data. Finally, grain boundary mobility is computed as the product $m = AB$ from this fit.

3. Workflow

This section summarizes the various scripts that are used to perform and analyze artificial driving force simulations. These scripts are constructed with high-throughput studies in mind. While all scripts are made with the ECO driving force in mind, they could be altered with minimal effort to produce simulations that use other driving force methods as well.

3.1 Required Software

All scripts presented were tested and developed using Python 2.7.9 with NumPy, SciPy, and matplotlib packages enabled. The molecular dynamics simulations are conducted using the LAMMPS molecular dynamic code and require the artificial driving force commands be compiled. The ECO artificial driving force method uses the commands fix_eco_force.cpp and fix_eco_force.h, whereas the original Janssens et al.[14] method requires the commands fix_orient_fcc.cpp, fix_orient_fcc.h, fix_orient_bcc.cpp, and fix_orient_bcc.h (available through the MISC package).

3.2 Initialization Steps

Two inputs are required before the grain boundary mobility simulations can begin: the initial bicrystal structure and the temperature-dependent lattice constant for the given material. Grain boundaries can being generated using LAMMPS itself; however, for any survey of a large number of differing grain boundary geometries, it is recommended that a grain boundary builder be used to streamline the process. The mobility calculation workflow uses a LAMMPS data file as the input; however, this can be easily changed to other data types if necessary. The inputted grain boundary structure should be periodic in all directions, but with free surfaces at the ends of the box in the direction normal to the grain boundary plane.

An accurate, temperature-dependent lattice constant is needed as the second input to correctly calculate the orientation parameter. For this work, a simple algorithm is developed (Appendix A.1) to compute the lattice constant at increments of 50 K in the range of 0 to 1000 K. This algorithm generates a bulk equilibration simulation from an inputted material, crystal prototype, approximated 0 K lattice constant, and interatomic potential. The equilibration script uses an isothermal-isobaric (NPT) ensemble to perform 10-ps ramps and 5-ps holds at each of the targeted temperatures. The results are unique for every set of material, crystal prototype, and potential.

3.3 ECO Artificial Driving Force Simulations

The scripts for systematic grain boundary motion studies using ECO artificial driving force are included in Appendix A.2. For simplicity, the discussions of each step in the following subsections only include a single applied energy at a single temperature. The complete workflow repeats this process to generate the full data set mapping velocity as a function of driving force, which is necessary to compute grain boundary mobility.

The required inputs are as follows: Miller indices, material, crystal type, cutoff radius, thermal filtering parameter, interatomic potential, timesteps, temperatures, energies, and grain boundary data file.

3.3.1 Simulation Cell Edges

The free surfaces that lie normal to the grain boundary plane within the flat bicrystal structures allow for the study of more general boundaries without worry of possible force-coupling effects. However, because the local environment at the free surface is different from that of the bulk, spurious nonunity orientation values will emerge. In order to avoid adding improper driving forces to these atoms located at the simulation edges, particles within 1 cutoff radius, R_{cut}, are detected and assigned zero force and velocity values normal to the grain boundary plane. By fixing the force and velocity normal to the grain boundary, the simulation becomes anchored in space while allowing the atoms to rearrange as necessary along the planar directions.

3.3.2 Orientation File

The ECO artificial driving force method requires an input of an orientation file that contains the oriented basis vectors for both crystals in the system. These oriented basis vectors are used within the ECO method to both calculate the reciprocal space values and to determine the normalization value for the orientation parameter. The workflow algorithm generates orientation files with the appropriate oriented basis vectors for the given crystal prototype (FCC and BCC) and rotation matrices for each grain, $R_{1/2}$. For crystal types in which the basis vectors and the primitive unit cells are the same (e.g., FCC [Fig. 3a] and BCC [Fig. 3b]), the orientation file consists simply of the following vectors:

FCC

$b_1 = R_1[a/2, a/2, 0]$
$b_2 = R_1[a/2, 0, a/2]$
$b_3 = R_1[0, a/2, a/2]$
$b_1 = R_2[a/2, a/2, 0]$
$b_2 = R_2[a/2, 0, a/2]$
$b_3 = R_2[0, a/2, a/2]$

BCC

$b_1 = R_1[–a/2, a/2, a/2]$
$b_2 = R_1[a/2, –a/2, a/2]$
$b_3 = R_1[a/2, a/2, –a/2]$
$b_1 = R_2[-a/2, a/2, a/2]$
$b_2 = R_2[a/2, –a/2, a/2]$
$b_3 = R_2[a/2, a/2, -a/2]$

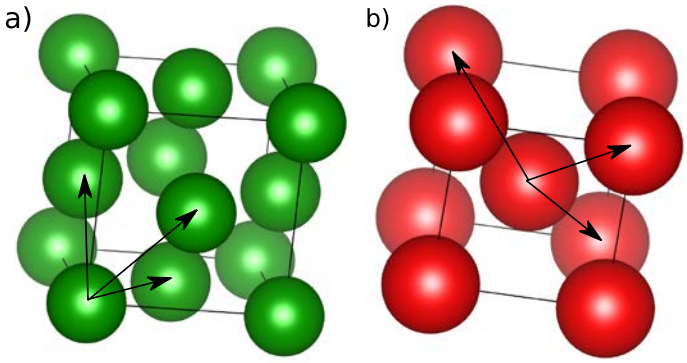

Fig. 3 Basis vectors within the unit cells of a) FCC and b) BCC. For these crystal types, the basis vectors and primitive unit cells are the same.

3.3.3 Temperature Equilibration

In order to calculate grain boundary mobility at a given temperature, multiple simulations at varying driving forces must be explored. To ensure consistent results, all simulations performed at the specified temperature start from the same initial state. This initial state is generated from the inputted bicrystal structure (assumed to be at 0 K). The inputted bicrystal structure is ramped up to the desired temperature using the isobaric-isothermal (NPT) ensemble at a ramp rate of 5 K/ps. At the desired temperature, the simulation is allowed to equilibrate for 5 ps using the NPT thermostat. After this equilibration period, a LAMMPS restart file is written to enable subsequent simulations to be initiated using the same atomic positions and trajectories.

3.3.4 File Submission

To fully automate the process, it is necessary to automate the process of submitting the jobs, as there can often be dozens or more simulations to be run at any given time. An example of the script used to write the submit files for the Excalibur computing cluster at the US Army Research Laboratory is included in Appendix A.3. This should be appropriately adjusted to match the requirements for the cluster in use.

3.4 Data Analysis

The algorithm described previously outputs a large amount of data for each energy simulated, on the scale of hundreds of atomic position files in addition to the input files, submit files, and an output file describing the state of the system. As a result, it is also necessary to automate the postprocessing of the data, specifically determining the velocity of the grain boundary (Appendix A.4). In addition to position and velocity of each atom, the orientation parameter is also outputted in the dump files of the simulation. As at the grain boundary is the only region that the orientation parameter is not –1 or 1 (Fig. 4), by scanning each dump file for values between these values it is possible to determine the particles that form the boundary. This can be done for each timestep, thus allowing for a plot of position against time (Fig. 5). The velocity can then be determined by finding the slope using least-squares linear regression. Additionally, to speed up computation time and allow for more accurate line fits, once the grain boundary is within 2 cutoff radii of the end of the bicrystal, the algorithm ends, as there is no longer any useful data to be gleamed from the simulation. The calculated velocity values are then plotted against driving pressure, and using the procedure outlined in Section 2.2, a curve is fit to the data and a mobility value is determined.

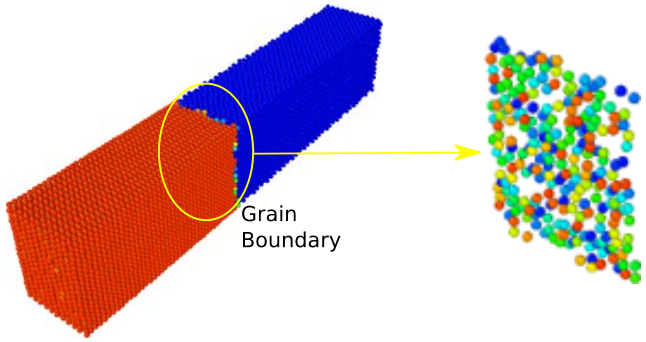

Fig. 4 Orientation parameter generated by the fix ECO command. Blue represents –1 and red 1 with the region between containing intermediate values.

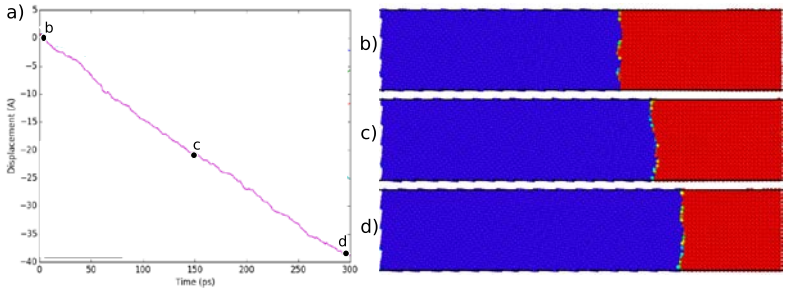

Fig. 5 a) Position of the grain boundary as determined by the tracking algorithm. Output of the bicrystal at times b) 0 ps, c) 150 ps, and d) 300 ps.

4. Examples

4.1 FCC Crystal

Mobility calculations were made for a Σ5 tilt boundary in FCC copper (Cu) from literature[16] using both the ECO driving force and Janssens driving force methods. Simulations were run at 200 and 500 K with u_0 driving energy values ranging from 0.025 to 0.05 eV/atom using the Mishin embedded atom method (EAM) interatomic potential.[17] Figure 6 shows the results of the 200 K ECO simulations, which yielded a mobility value of 181.1 m/s · GPa.

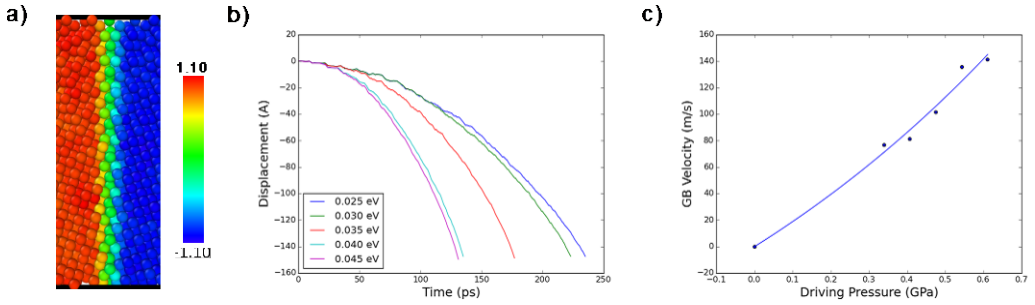

Fig. 6 a) Orientation parameter values at the Σ5 Cu interface at 200 K. b) Comparison of displacement vs. time plots for the various driving energies. c) Plot of velocities at the various driving pressures and the curve fit using Eq. 7.

A comparison of the mobility results derived from the different driving force methods are shown in Table 1. Two trends can be discerned from these values. First, mobility increases with temperature. Second, the values for the ECO driving force are lower than those of the Janssens driving force method. The first is to be expected as, with the exception of more exotic athermal grain boundaries, mobility should increase with more energy in the system. The differences in calculated mobility between the 2 driving forces are consistent with the range found in Ulomek's original work, which is generally attributed to nonconservation of energy with Janssens method.

Table 1 Calculated mobility values for an FCC Cu Σ5 tilt boundary at different temperatures and using different driving forces

Temperature	ECO (m/s·GPa)	Janssens (m/s · GPa)
200 K	181	196
500 K	231	252

4.2 BCC Crystal

A bicrystal with the same orientations as those used in Section 4.1 was constructed for BCC iron (Fe), and simulations were run using the Hepburn EAM interatomic potential.[18] Thermal noise at 500 K was too large for the Janssens method, thus the evaluated temperatures were reduced to 50 and 200 K. Due to difference in material and crystal structure, the driving energies necessary to initiate grain growth differ from that of the FCC simulations, so here values ranging from 0.30 to 0.50 eV/atom were used. Table 2 results show that the same trends emerge here as observed in the FCC simulations. Specifically, increased mobility as temperature increases and finding larger mobility values using the Janssens method. Despite having the same orientation relationship and temperature, the Cu and Fe simulations at 200 K have different mobility values. This suggests additional variables are at play (e.g., grain boundary energy, cohesive energy, and mass).

Table 2 Calculated mobility values for a BCC Fe Σ5 tilt boundary at different temperatures and using different driving

Temperature	ECO (m/s·GPa)	Janssens (m/s·GPa)
50 K	10.9	12.4
200 K	15.4	17.6

5. Future Applications: Mobility in HCP Crystals

As stated in Section 3.3.2, creating the orientation files for crystal structures in which the primitive unit cells and basis vectors are the same is a relatively simple endeavor. In HCP, however, the basis and the primitive unit cell are different. In order to define all the nearest neighbors, a fourth basis vector is needed (Fig. 7):

HCP
$b_1=[a, 0, 0]$
$b_2=[-a/2, a\sqrt{3}/2, 0]$
$b_3=[0, -a\sqrt{3}/3, c/2]$
$b_4=[0, -a\sqrt{3}/3, -c/2]$

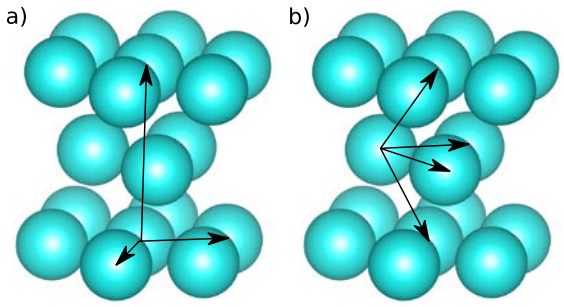

Fig. 7 a) Primitive unit cell for the HCP crystal structure. b) Basis vector required to be used in the fix ECO command for HCP

This also requires a reformulation of the ECO LAMMPS command to allow an extra basis vector to be inputted. As a result of the new formulation of reciprocal space volume, the number of reciprocal basis vectors for each crystal orientation increases from 3 in FCC and BCC materials to 11 in HCP. Furthermore, this approach is only valid if the cutoff radius is less than the c lattice constant, as at values greater than c, this basis no longer defines the atomic positions of the system. A cutoff radius of 0.75c is suggested to allow the maximum amount of thermal noise without influencing the results. This cutoff radius produces a local neighborhood of 12 atoms, which is more than sufficient to describe the orientation parameter. This formulation has shown the ability to calculate appropriate orientation factor values and to drive grain growth. However, further testing and comparison to other driving force methods is required to validate this method.

6. Conclusion

The algorithm outlined here allows for the automated and efficient calculation of mobility values for a wide variety of crystallographic systems. Through the use of the ECO force, grain boundary motion is initiated. This motion is then tracked with the resulting velocities used to calculate the grain boundary mobility. This process can then be repeated for as many simulations as desired. Grain boundary mobility is an extremely important value with regard to the behavior of grain structure in metals. The algorithm developed and discussed in this work can be a very useful tool for researchers studying mobility and how it relates to the character of various grain boundaries. Additionally, the ability to more quickly calculate various mobility values allows them to be used as inputs to mesoscale models that may better model grain growth.

7. References

1. Greuter F, Blatter G. Electrical properties of grain boundaries in polycrystalline compound semiconductors. Semiconductor Science and Technology. 1990;5(2):111.

2. Schiøtz J, Jacobsen KW. A maximum in the strength of nanocrystalline copper. Science. 2003;301(5638):1357–1359.

3. Beyerlein IJ, Caro A, Demkowicz MJ, Mara NA, Misra A, Uberuaga BP. Radiation damage tolerant nanomaterials. Materials Today. 2013;16(11):443–449.

4. Yonehara T, Nishigaki Y, Mizutani H, Kondoh S, Yamagata K, Ichikawa T. Control of grain boundary location by selective nucleation over amorphous substrates. Cambridge University Press. 1987;106:21.

5. Van Swygenhoven H, Caro A, and Farkas F. A molecular dynamics study of polycrystalline FCC metals at the nanoscale: grain boundary structure and its influence on plastic deformation. Materials Science and Engineering: A. 2001;309:440–444.

6. Hasnaoui A, Van Swygenhoven H, Derlet PM. On non-equilibrium grain boundaries and their effect on thermal and mechanical behaviour: a molecular dynamics computer simulation. Acta Materialia. 2002;50(15):3927–3939.

7. Cao A, Wei Y, Ma E. Grain boundary effects on plastic deformation and fracture mechanisms in Cu nanowires: molecular dynamics simulations. Physical Review B. 2008;77(19):195429.

8. Holm EA, Foiles SM. How grain growth stops: a mechanism for grain-growth stagnation in pure materials. Science. 2010;328(5982):1138–1141.

9. Ulomek F, O'Brien CJ, Foiles SM, Mohles V. Energy conserving orientational force for determining grain boundary mobility. Modelling and Simulation in Materials Science and Engineering. 2015;23(2):025007.

10. Schönfelder B, Wolf D, Phillpot SR, Furtkamp M. Molecular-dynamics method for the simulation of grain-boundary migration. Interface Science. 1997;5(4):245–262.

11. Zhang H, Upmanyu M, Srolovitz DJ. Curvature driven grain boundary migration in aluminum: molecular dynamics simulations. Acta Materialia. 2005;53(1):79–86.

12. Trautt Z, Upmanyu M. Atomic-scale simulation of grain boundary kinetics during recrystallization. Cambridge University Press. 2004;819:N6.7.

13. Bai X-M, Zhang Y, Tonks MR. Testing thermal gradient driving force for grain boundary migration using molecular dynamics simulations. Acta Materialia. 2015;85:95–106.

14. Janssens KGF, Olmsted D, Holm EA, Foiles SM, Plimpton SJ, Derlet PM. Computing the mobility of grain boundaries. Nature Materials. 2006;5(2):124–127.

15. Plimpton S. Fast parallel algorithms for short-range molecular dynamics. Journal of Computational Physics. 1995;117(1):1–19.

16. Tschopp MA, McDowell DL. Asymmetric tilt grain boundary structure and energy in copper and aluminum. Philosophical Magazine. 2007;87(25):3871–3892.

17. Mishin Y, Mehl MJ, Papaconstantopoulos DA, Voter AF, Kress JD. Structural stability and lattice defects in copper: ab initio, tight-binding, and embedded-atom calculations. Physical Review B. 2001;63(22):224106.

18. Hepburn DJ, Ackland GJ. Metallic-covalent interatomic potential for carbon in iron. Physical Review B. 2008;78(16):165115.

INTENTIONALLY LEFT BLANK.

Appendix. Grain Boundary Mobility Scripts

This appendix appears in its original form, without editorial change.

A.1 ThermalData.py

```
import numpy
import math
import os
from submit_maker import *
import time



#Crystalography
material='Fe'
crystaltype='bcc'       #bcc, fcc, or hcp
a=2.86                  #Lattice constant approximation
                    #c will be assumed to be ideal, relaxation will find actual

unitcell_dump=''          #Only if not bcc, fcc, or hcp
unit_timestep=0


#Potential
pair_style='eam/fs'
pair_coeff='* * Potentials/Fe-C_Hepburn_Ackland.eam.fs '+material
mass=24.305

os.system('mkdir ThermalData')
os.system('mkdir ThermalData/'+material+'_'+crystaltype)

#Calculate temperature dependent lattice constants
#Create LAMMPS input script ramping up temperature
f = open('ThermalData/'+material+'_'+crystaltype+'/temp_'+material+'.in','w')
f.write('#LAMMPS input file to calculate temperature dependent lattice constants of
'+material+' \n')
f.write('dimension 3 \n')
f.write('boundary p p p \n')
f.write('units metal \n')
f.write('atom_style atomic \n')
if crystaltype=='fcc' or crystaltype=='bcc' or crystaltype=='hcp':
    f.write('lattice '+crystaltype+' '+repr(a)+' origin .01 .01 .01 \n')
    f.write('region box block 0 8 0 8 0 8 units lattice \n')
    f.write('create_box 1 box \n')
    f.write('create_atoms 1 box \n')
else:
    f.write('lattice bcc '+repr(a)+' origin .01 .01 .01 \n')
    f.write('region box block 0 8 0 8 0 8 units lattice \n')
    f.write('create_box 1 box \n')
    f.write('read_dump '+unitcell_dump+' '+repr(unit_timestep)+' box yes replace yes \n')
    f.write('replicate 8 8 8 \n')
    f.write('reset_timestep 0 \n')
    f.write('pair_style '+pair_style+'\n')
    f.write('pair_coeff '+pair_coeff+'\n')
```

```
f.write('mass 1 '+repr(mass)+' \n')
f.write('variable Lx equal lx/8 \n')
f.write('variable Ly equal ly/8 \n')
f.write('variable Lz equal lz/8 \n')
f.write('variable T equal temp \n')
f.write('thermo 1000 \n')
f.write('thermo_style custom step temp etotal press \n')
f.write('fix zeroK all box/relax x 0.0 y 0.0 z 0.0 \n')
f.write('minimize 1e-12 1e-12  100000 100000 \n')
f.write('fix 2 all print 1 "${T} ${Lx} ${Ly} ${Lz}" file
ThermalData/'+material+'_'+crystaltype+'/LatticeConst title "Temperature dependent
lattice constants (T a b c)" \n')
f.write('run 1 \n')
f.write('reset_timestep 0 \n')
f.write('unfix zeroK \n')
f.write('unfix 2 \n')
f.write('timestep 0.001 \n')
f.write('velocity all create 5 1 \n')
for j in range(0,20):
    X=int(j*50)
    Y=int((j+1)*50)
    if j==0:
        f.write('fix integrator all npt iso 0 0 1 temp 1 '+repr(Y)+' .1 \n')
    else:
        f.write('fix integrator all npt iso 0 0 1 temp '+repr(X)+' '+repr(Y)+' .1 \n')
    f.write('run 10000 \n')
    f.write('unfix integrator \n')
    f.write('fix integrator all npt iso 0 0 1 temp '+repr(Y)+' '+repr(Y)+' .1 \n')
    f.write('fix 3 all print 1 "${T} ${Lx} ${Ly} ${Lz}" file
ThermalData/'+material+'_'+crystaltype+'/Temp'+repr(Y)+'.out title "" screen no \n')
    f.write('run 5000 \n')
    f.write('unfix 3 \n')
    f.write('unfix integrator \n')
f.write('dump check all atom 1 SimDone \n')
f.write('run 0')
f.close()


write_submit_excalibur('LatticeCompute.bash',material+'_lattice',16,'00:30:00','debug
','/work/mcg84/test','ThermalData/'+material+'_'+crystaltype+'/temp_'+material+'.in','T
hermalData/'+material+'_'+crystaltype+'/Temp.out')
os.system('qsub LatticeCompute.bash')
while not os.path.exists('SimDone'):
        time.sleep(10)
        print('Waiting for restart file')

os.system('rm SimDone')
os.system('rm LatticeCompute.bash')


#Average out data from constant temperature holds
```

```
for j in range(0,20):
    Y=int((j+1)*50)
    f = open('ThermalData/'+material+'_'+crystaltype+'/Temp'+repr(Y)+'.out','r')
    f.readline()
    T=0; Lx=0; Ly=0; Lz=0;
    for k in range(0,5000):
        line=f.readline()
        data=line.split(' ')
        T=T+float(data[0]); Lx=Lx+float(data[1]); Ly=Ly+float(data[2]);
Lz=Lz+float(data[3]);
    os.system('rm ThermalData/'+material+'_'+crystaltype+'/Temp'+repr(Y)+'.out')
    T=T/5000; Lx=Lx/5000; Ly=Ly/5000; Lz=Lz/5000;
    g = open('ThermalData/'+material+'_'+crystaltype+'/LatticeConst','a')
    g.write(repr(T)+' '+repr(Lx)+' '+repr(Ly)+' '+repr(Lz)+'\n')
    g.close()
f.close()
```

A.2 ECO_Builder.py

```
import numpy
import math
import os
from submit_maker import *
import time
from decimal import *

#NOTE: ThermalData.py for the given material and crystaltype must have previously
been run


#Dump file from Shawn's GB (Output: id type x y z)
dumpfile='InitialDumpfiles/Fe_tilt.dump'


#Temperatures to be analyzed
Temps=[50,200]

#Driving energies to be considered (+/- u0/2 added to each crystal
u0=[0.10,0.20,0.30,0.40,.50,.75,1.0,1.25,1.5]

#Orientation Title
Orientation='Fe_sig5'

#Computational machine
machine='excalibur'

#Parameters
rcut=4              #ECO cutoff radius
eta=.7               #ECO cutoff parmater (.25 in most cases)
cutlo=.65              #orient cutoff parameters (between 0 and 1)
cuthi=.95
```

```
timesteps=300000


#Crystalography
material='Fe'
crystaltype='bcc'       #bcc, fcc, or hcp
a=2.86                  #Lattice constant approximation


#y is normal direction Orientation 1 will grow
miller1x=numpy.array([0, 0, 1])
miller1y=numpy.array([-1, -3, 0])
miller1z=numpy.array([3, -1, 0])
miller2x=numpy.array([0, 0, 1])
miller2y=numpy.array([3, -1, 0])
miller2z=numpy.array([-1, -3, 0])



unitcell_dump=''          #Only if not bcc, fcc, or hcp
unit_timestep=0


#Potential
pair_style='eam/fs'
pair_coeff='* * Potentials/Fe-C_Hepburn_Ackland.eam.fs Fe Fe Fe Fe'
mass=55.85

os.system('mkdir MobilityData')
os.system('mkdir MobilityData/'+material+'_'+crystaltype)
os.system('mkdir MobilityData/'+material+'_'+crystaltype+'/'+Orientation)
os.system('mkdir MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/InputScripts')
os.system('mkdir MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Dumps')
os.system('mkdir MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Outputfiles')
os.system('mkdir MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Submitfiles')

#Create array of temperature dependent lattice constants
f = open('ThermalData/'+material+'_'+crystaltype+'/LatticeConst','r')
f.readline()
for i in range(0,21):
    line=f.readline()
    data=line.split(' ')
    if i == 0:
        TempData=numpy.array([float(data[0]),float(data[1]),float(data[3])])
    else:

TempData=numpy.vstack([TempData,[float(data[0]),float(data[1]),float(data[3])]])
f.close()
```

```
#Find Particles on free surface
f = open(dumpfile,'r')
f.readline()
line2=f.readline()
data = line2.split(' ')
dump_timestep=int(data[0])
f.readline()
line4 = f.readline()
data=line4.split(' ')
N=int(data[0])
f.readline()
line6 = f.readline()
xdata=line6.split(' ')
Lx=float(xdata[1])-float(xdata[0])
line7 = f.readline()
ydata=line7.split(' ')
Ly=float(ydata[1])-float(ydata[0])
line8 = f.readline()
zdata=line8.split(' ')
Lz=float(zdata[1])-float(zdata[0])
f.readline()

#Use far side of box as starting points
ymin=float(ydata[1])
ymax=float(ydata[0])

#Find min and max values of particles in y-direction
for j in range(0,N):
    line = f.readline()
    data=line.split(' ')
    if float(data[3])<ymin:
        ymin=float(data[3])
    if float(data[3])>ymax:
        ymax=float(data[3])
f.close()

#Calculate regions for vy=0,fy=0 (Necessary to keep box stationary)
yhi=ymax-rcut
ylo=ymin+rcut


#For temperatures to be analyzed, perform linear interpolation to find a and c
for j in range(0,len(Temps)):
    temp=Temps[j]
    for i in range(0,21):
        if temp<TempData[i][0]:
            T1=TempData[i-1][0]
            T2=TempData[i][0]
            a1=TempData[i-1][1]
            a2=TempData[i][1]
            break
```

```
    i=(temp-T1)/(T2-T1)
    a=(1-i)*a1+i*a2


    #Define basis vectors and rotation matrices
    if crystaltype=='fcc':
        a1=numpy.matrix([[a/2], [a/2],[0]])
        a2=numpy.matrix([[a/2], [0],[a/2]])
        a3=numpy.matrix([[0], [a/2],[a/2]])
        ori1x=miller1x
        ori1y=miller1y
        ori1z=miller1z
        ori2x=miller2x
        ori2y=miller2y
        ori2z=miller2z
    if crystaltype=='bcc':
        a1=numpy.matrix([[a/2], [a/2],[-a/2]])
        a2=numpy.matrix([[a/2], [-a/2],[a/2]])
        a3=numpy.matrix([[-a/2], [a/2],[a/2]])
        ori1x=miller1x
        ori1y=miller1y
        ori1z=miller1z
        ori2x=miller2x
        ori2y=miller2y
        ori2z=miller2z


    ori1x=ori1x.T/numpy.linalg.norm(ori1x)
    ori1y=ori1y.T/numpy.linalg.norm(ori1y)
    ori1z=ori1z.T/numpy.linalg.norm(ori1z)
    ori2x=ori2x.T/numpy.linalg.norm(ori2x)
    ori2y=ori2y.T/numpy.linalg.norm(ori2y)
    ori2z=ori2z.T/numpy.linalg.norm(ori2z)

    x=numpy.array([1,0,0])
    y=numpy.array([0,1,0])
    z=numpy.array([0,0,1])


R1=numpy.matrix([[float(numpy.dot(x,ori1x)),float(numpy.dot(y,ori1x)),float(numpy.do
t(z,ori1x))],[float(numpy.dot(x,ori1y)),float(numpy.dot(y,ori1y)),float(numpy.dot(z,ori1y
))],[float(numpy.dot(x,ori1z)),float(numpy.dot(y,ori1z)),float(numpy.dot(z,ori1z))]])

R2=numpy.matrix([[float(numpy.dot(x,ori2x)),float(numpy.dot(y,ori2x)),float(numpy.do
t(z,ori2x))],[float(numpy.dot(x,ori2y)),float(numpy.dot(y,ori2y)),float(numpy.dot(z,ori2y
))],[float(numpy.dot(x,ori2z)),float(numpy.dot(y,ori2z)),float(numpy.dot(z,ori2z))]])

    #Calculate Rotated Basis
    b1_1=R1*a1
    b2_1=R1*a2
    b3_1=R1*a3
```

```
    b1_2=R2*a1
    b2_2=R2*a2
    b3_2=R2*a3

    #Write Orientation file (ECO)

f=open('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/InputScripts/eco_'+r
epr(temp)+'K.ori','w')
    f.write(str(b1_1[0]).strip('[]')+' '+str(b1_1[1]).strip('[]')+' '+str(b1_1[2]).strip('[]')+'\n')
    f.write(str(b2_1[0]).strip('[]')+' '+str(b2_1[1]).strip('[]')+' '+str(b2_1[2]).strip('[]')+'\n')
    f.write(str(b3_1[0]).strip('[]')+' '+str(b3_1[1]).strip('[]')+' '+str(b3_1[2]).strip('[]')+'\n')
    f.write(str(b1_2[0]).strip('[]')+' '+str(b1_2[1]).strip('[]')+' '+str(b1_2[2]).strip('[]')+'\n')
    f.write(str(b2_2[0]).strip('[]')+' '+str(b2_2[1]).strip('[]')+' '+str(b2_2[2]).strip('[]')+'\n')
    f.write(str(b3_2[0]).strip('[]')+' '+str(b3_2[1]).strip('[]')+' '+str(b3_2[2]).strip('[]')+'\n')
    f.close()


    #Calculate necessary size of simulation

    repx=int(math.ceil(50/Lx))
    repz=int(math.ceil(50/Lz))

    cutoff=2*rcut
    ramp=int(math.ceil(temp/50)*10000)

    #Write restart file at temperature to be analyzed

f=open('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/InputScripts/temp_e
quil_'+repr(temp)+'.in','w')
    f.write('#Thermal Equilibration Input File \n')
    f.write('dimension 3 \n')
    f.write('boundary p p p \n')
    f.write('units metal \n')
    f.write('atom_style atomic \n')
    f.write('lattice bcc '+repr(a)+' origin .01 .01 .01 \n')
    f.write('region box block 0 1 0 1 0 1 units lattice \n')
    f.write('create_box 4 box \n')
    f.write('read_dump '+dumpfile+' '+repr(dump_timestep)+' x y z box yes add yes \n')
    f.write('replicate '+repr(repx)+' 1 '+repr(repz)+' \n')
    f.write('reset_timestep 0 \n')
    f.write('pair_style '+pair_style+'\n')
    f.write('pair_coeff '+pair_coeff+' \n')
    f.write('comm_modify cutoff '+repr(cutoff)+'\n')
    f.write('comm_style tiled \n')
    f.write('balance 0.9 rcb \n')
    f.write('timestep 0.001 \n')
    f.write('thermo 1000 \n')
    f.write('velocity all create 5 1 \n')
    f.write('fix integrator all npt iso 0 0 1 temp 1 '+repr(temp)+' .1 \n')
    f.write('thermo_style custom step temp etotal press \n')
    f.write('run '+repr(ramp)+' \n')
```

```
    f.write('unfix integrator \n')
    f.write('fix integrator all npt iso 0 0 1 temp '+repr(temp)+' '+repr(temp)+' .1 \n')
    f.write('run 5000 \n')
    f.write('write_restart
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Dumps/'+repr(temp)+'K.resta
rt \n')
    f.write('run 0')
    f.close()



write_submit_excalibur('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Sub
mitfiles/submit_excal_restartmaker'+repr(temp)+'K.bash',material+'_'+repr(temp)+'K_
restart',16,'00:30:00','debug','/work/mcg84/test','MobilityData/'+material+'_'+crystaltyp
e+'/'+Orientation+'/InputScripts/temp_equil_'+repr(temp)+'.in','MobilityData/'+material
+'_'+crystaltype+'/'+Orientation+'/Outputfiles/temp_equil_'+repr(temp)+'.out')
    os.system('qsub
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Submitfiles/submit_excal_re
startmaker'+repr(temp)+'K.bash')

    for k in range(0,len(u0)):
        u=format(u0[k], '.3f')
        #Write ECO Input file

f=open('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/InputScripts/eco_'+r
epr(temp)+'K_'+u+'eV.in','w')
        f.write('#ECO input '+repr(temp)+'K_'+repr(u)+'eV \n')
        f.write('read_restart
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Dumps/'+repr(temp)+'K.restart \n')
        f.write('reset_timestep 0 \n')
        f.write('pair_style '+pair_style+'\n')
         f.write('pair_coeff '+pair_coeff+' \n')
        f.write('comm_modify cutoff '+repr(cutoff)+'\n')
         f.write('thermo 100 \n')
        f.write('fix integrator all nvt temp '+repr(temp)+' '+repr(temp)+' .1 \n')
        f.write('region ends block INF INF '+repr(ylo)+' '+repr(yhi)+' INF INF units box side out \n')
        f.write('group end region ends \n')
        f.write('fix eco all eco/force '+u+' '+repr(eta)+' '+repr(rcut)+'
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/InputScripts/eco_'+repr(temp
)+'K.ori \n')
        f.write('fix edges end setforce NULL 0.0 NULL \n')
        f.write('velocity end set NULL 0.0 NULL \n')
        f.write('balance 1.1 shift y 5 1.02 \n')
        f.write('thermo_style custom step temp etotal press f_eco \n')
        f.write('dump save all custom 1000
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Dumps/eco_'+repr(temp)+'K
_'+u+'eV.* id x y z f_eco[1] f_eco[2] f_eco[3] f_eco[4] f_eco[5] fx fy fz vx vy vz \n')
        f.write('dump_modify save sort id \n')
        f.write('run '+repr(timesteps))
        f.close()
```

```
write_submit_excalibur('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Sub
mitfiles/submit_excal_'+repr(temp)+'K_'+u+'eV.bash',material+'_'+repr(temp)+'K_'+u
+'eV',28,'11:20:00','standard','/work/mcg84/test','MobilityData/'+material+'_'+crystalty
pe+'/'+Orientation+'/InputScripts/eco_'+repr(temp)+'K_'+u+'eV.in','MobilityData/'+mat
erial+'_'+crystaltype+'/'+Orientation+'/Outputfiles/'+repr(temp)+'K_'+u+'eV.out')

      while not
os.path.exists('MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Dumps/'+repr
(temp)+'K.restart'):
          time.sleep(10)
          print('Waiting for restart file')
      os.system('qsub
MobilityData/'+material+'_'+crystaltype+'/'+Orientation+'/Submitfiles/submit_excal_'+r
epr(temp)+'K_'+repr(u)+'eV.bash')
```

A.3 submit_maker.py

```
import math
import os

def write_submit_topaz(fname, jobname, procnum, walltime, qtype, directory, input,
output):
   cores = int(math.ceil(procnum / 36.0))
   f = open(fname, 'w')
   f.write('#!/bin/bash \n')
   f.write('#PBS -A xxxxxxxxxx \n')
   f.write('#PBS -q ' + qtype + ' \n')
   f.write('#PBS -l select=' + repr(cores) + ':ncpus=36:mpiprocs=36 \n')
   f.write('#PBS -l walltime=' + walltime + ' \n')
   f.write('#PBS -l application=LAMMPS \n')
   f.write('#PBS -j oe \n')
   f.write('#PBS -N ' + jobname + ' \n\n')
   f.write('cd ' + directory + ' \n\n')
   f.write('mpiexec_mpt -np ' + repr(procnum) + ' ~/LAMMPS/src/lmp_topaz < ' + input
+ ' > ' + output)
   f.close()


def write_submit_copper(fname, jobname, procnum, walltime, qtype, directory, input,
output):
   cores = int(math.ceil(procnum / 32.0))
   f = open(fname, 'w')
   f.write('#!/bin/bash \n')
   f.write('#PBS -A xxxxxxxxxx \n')
   f.write('#PBS -q ' + qtype + ' \n')
   f.write('#PBS -l select=' + repr(cores) + ':ncpus=36:mpiprocs=36 \n')
   f.write('#PBS -l walltime=' + walltime + ' \n')
   f.write('#PBS -l application=LAMMPS \n')
   f.write('#PBS -j oe \n')
```

```
    f.write('#PBS -N ' + jobname + ' \n\n')
    f.write('cd ' + directory + ' \n\n')
    f.write('aprun -n ' + repr(procnum) + ' ~/Lammps/src/lmp_copper < ' + input + ' > ' +
output)
    f.close()


def write_submit_excalibur(fname, jobname, procnum, walltime, qtype, directory, input, output):
    cores = int(math.ceil(procnum / 32.0))
    f = open(fname, 'w')
    f.write('#!/bin/bash \n')
    f.write('#PBS -A xxxxxxxxxx \n')
    f.write('#PBS -q ' + qtype + ' \n')
    f.write('#PBS -l select=' + repr(cores) + ':ncpus=32:mpiprocs=32 \n')
    f.write('#PBS -l place=scatter:excl \n')
    f.write('#PBS -l walltime=' + walltime + ' \n')
    f.write('#PBS -l application=LAMMPS \n')
    f.write('#PBS -j oe \n')
    f.write('#PBS -N ' + jobname + ' \n\n')
    f.write('cd ' + directory + ' \n\n')
    f.write('aprun -n ' + repr(procnum) + ' ~/lammps-30Jul16/src/lmp_excalibur < ' + input + ' > '
+ output)
    f.close()
```

A.4 BoundaryTrack.py

```
import numpy
import math
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import scipy.optimize


Temp=50


dumpnames=['0.005','0.010','0.015','0.020','0.025']

#Atomic Volume of Element(cc/mol)
V=7.1

figname='ProcessedData/eco_sig5_11_'+repr(Temp)+'K'

v=0
p=0
for k in range(0,len(dumpnames)):
    labelplot=dumpnames[k]
#Dump filename and path
```

```
filename='MobilityData/Cu_fcc/Cu_sigma5_11/Dumps/eco_'+repr(Temp)+'K_'+dump
names[k]+'eV'

    # Number of dump files
    dumps=200
    rcut=4

    #eta range to be examine (ECO spans -1 to 1,orient spans 0 to 1)
    elo=-.25
    ehi=.25

    #Find free surfaces of simulation
    f = open(filename+'.0','r')
    f.readline()
    f.readline()
    f.readline()
    line4 = f.readline()
    data=line4.split(' ')
    N=int(data[0])
    f.readline()
    line6 = f.readline()
    data=line6.split(' ')
    Lx=float(data[1])-float(data[0])
    line7 = f.readline()
    ydata=line7.split(' ')
    Ly=float(ydata[1])-float(ydata[0])
    line8 = f.readline()
    data=line8.split(' ')
    Lz=float(data[1])-float(data[0])
    f.readline()
    ymin=float(ydata[1])
    ymax=float(ydata[0])

    #Find min and max values of particles in y-direction
    for j in range(0,N):
            line = f.readline()
            data=line.split(' ')
            if float(data[2])<ymin:
                ymin=float(data[2])
            if float(data[2])>ymax:
                ymax=float(data[2])
    f.close()

    ylo=ymin+rcut
    yhi=ymax-rcut

    yend_hi=ymax-3*rcut
    yend_lo=ymin+3*rcut

    #For each dump file, find number of atoms and box size
    for i in range(0,dumps):
```

```
        time=int(i)
        if time==0:
            times=time
        else:
            times=numpy.vstack([times,time])

        k=int(i*1000)
        f = open(filename+'.'+repr(k),'r')
        f.readline()
        f.readline()
        f.readline()
        line4 = f.readline()
        data=line4.split(' ')
        N=int(data[0])
        f.readline()
        line6 = f.readline()
        data=line6.split(' ')
        Lx=float(data[1])-float(data[0])
        line7 = f.readline()
        data=line7.split(' ')
        Ly=float(data[1])-float(data[0])
        line8 = f.readline()
        data=line8.split(' ')
        Lz=float(data[1])-float(data[0])
        f.readline()
        n=0
        Fdx=0
        Fdy=0
        Fdz=0

        #Find atoms within the specified f_eco(2) range  Note: this will be a function of
eta input in LAMMPS
        for j in range(0,N):
            line = f.readline()
            data=line.split(' ')
            if float(data[2])<yhi and float(data[2])>ylo:
                if float(data[4])<ehi and float(data[4])>elo:
                    Fdx=Fdx+float(data[6])
                    Fdy=Fdy+float(data[7])
                    Fdz=Fdz+float(data[8])
                    n=n+1
                    if n == 1:
                        A=numpy.array([float(data[1]),float(data[2]),float(data[3])])
                    else:
                        newrow=numpy.array([float(data[1]),float(data[2]),float(data[3])])
                        A=numpy.vstack([A,newrow])
        #Calculate centroids
        sizeA=A.shape[0]
        sum_xA=numpy.sum(A[:,0])
        sum_yA=numpy.sum(A[:,1])
        sum_zA=numpy.sum(A[:,2])
```

```
        xA=sum_xA/sizeA
        yA=sum_yA/sizeA
        zA=sum_zA/sizeA
        if i==0:
            x0=xA
            y0=yA
            z0=zA

        centroid=numpy.array([xA-x0, yA-y0, zA-z0])

    #Create lists of centroids and norms at every timestep
        if int(i)==0:
            centroids=centroid

        else:
            centroids=numpy.vstack([centroids,centroid])
        Fd=numpy.array([Fdx, Fdy, Fdz])

        dim=centroids.shape[0]

        if yA>yend_hi:
            break
        elif yA<yend_lo:
            break

    t=numpy.vstack([times.T,numpy.ones(dim)]).T
    linpart=int(math.floor(dim/4))

    #Least Squares Regression of times vs centroid location in normal (y) direction,
slope yields velocity
    vel, c=numpy.linalg.lstsq(t[linpart:dim,:],centroids[linpart:dim,1])[0]
    print(vel)
    speed=abs(vel)
#Convert v to m/s and p to GPa
    v=numpy.hstack([v,speed*100])
    p=numpy.hstack([p,float(labelplot)*9.647e1/V])

    plt.plot(t[:,0],centroids[:,1],label=labelplot+' eV')
print(p)
print(v)

#Define function to be minimized: A(e^Bp-1)
def func(q,A,B):
    return A*(numpy.exp(B*q)-1)

popt=scipy.optimize.curve_fit(func,p,v)

print(popt)
p_curve=numpy.zeros(101)
v_curve=numpy.zeros(101)
```

```
for k in range(0,101):
    p_curve[k]=p[len(p)-1]/100*k
    v_curve[k]=func(p_curve[k],popt[0][0],popt[0][1])




plt.legend(loc='lower left')
plt.ylabel('Displacement (A)',fontsize=18)
plt.xlabel('Time (ps)',fontsize=18)
plt.savefig(figname+'_displacements')

plt.figure()
plt.scatter(p,v)
plt.plot(p_curve,v_curve)
plt.xlabel('Driving Pressure (GPa)',fontsize=18)
plt.ylabel('GB Velocity (m/s)',fontsize=18)
plt.savefig(figname+'_mobility')
```

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

BCC	body-centered cubic
Cu	copper
DOD	Department of Defense
DSRC	DOD Supercomputing Resource Center
EAM	embedded atom method
ECO	energy conserving orientational
FCC	face-centered cubic
Fe	iron
HCP	hexagonal close-packed
HPCMP	High Performance Computing Modernization Program
LAMMPS	Large-scale Atomic/Molecular Massively Parallel Simulator
NPT	isobaric-isothermal

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIR ARL
IMAL HRA
RECORDS MGMT
RDRL DCL
TECH LIB

1 (PDF) GOVT PRINTG OFC
A MALHOTRA

1 (PDF) COLORADO STATE
M GUZIEWSKI

3 (PDF) ARL
RDRL D
M TSCHOPP
RDRL WMM E
S COLEMAN
S SILTON