SIMULATING SOLID PHASE CHANGES IN DENSE LITHIUM

by

Keaton R. Tate

A senior thesis submitted to the faculty of

Brigham Young University - Idaho

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics

Brigham Young University - Idaho

December 2022

BRIGHAM YOUNG UNIVERSITY - IDAHO

DEPARTMENT APPROVAL

of a senior thesis submitted by

Keaton R. Tate

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

Date	Lance J. Nelson, Advisor
Date	J. Ryan Nielson, Committee Member
Date	Richard A. Hatt, Committee Member
Date	R. Todd Lines, Thesis Coordinator
Date	R. Todd Lines, Department Chair

ABSTRACT

SIMULATING SOLID PHASE CHANGES IN DENSE LITHIUM

Keaton R. Tate

Department of Physics

Bachelor of Science

This project focuses on finding the transition pressures of the ground state (T = 0 K) of dense lithium for solid-solid phase transitions. The Vienna Ab initio Software Package (VASP) was used to run Density Functional Theory (DFT) calculations on each lattice structure. The process was repeated for a range of increased and decreased volumes of the unit cell. Once the resulting total energies of the lattices were found, a common tangent construction between structures on a free energy vs. volume graph visually represented the transition pressures. The transition pressures at the ground state correlate well with experiments at slightly higher temperatures and the predictions made by Guillame et al. in 2011. Additional work should focus on higher pressure solid phases to determine if DFT continues to give good results.

ACKNOWLEDGMENTS

I am grateful to thank the many people who have helped me with this project. Thanks to Jason Meziere and Lydia H. Serafin for helping me get comfortable with submitting jobs on the Fulton supercomputer. Thanks to Dr. Lance Nelson for mentoring my research and igniting a love of materials science in me. Thanks to Dr. Gus Hart for providing valuable guidance and feedback during the project. Thanks to the staff at the BYU Office of Research Computing for helping me, over many hours, optimize my job submissions and get to the root of some software issues.

Thank you to Professors Richard Hatt, Ryan Nielson, Todd Lines, and my peers in PH 412 for giving feedback on drafts of this thesis and for encouraging me throughout my physics degree.

I would also like to thank those who encourage and support me outside school. To my junior high science teacher Jerry Carpenter, thank you for supporting my curiosity and opening new worlds of thought to me. To my parents and family, thank you for all your love and for supporting me through hard times. And to my wonderful wife Lilly, thank you for always encouraging me to learn more and be better. I love you.

Contents

List of Figures

Chapter 1

Introduction/Background

This project focuses on simulating how and when pressurized lithium changes. Lithium exhibits interesting properties when compressed such as a predicted superconductive phase. Simulating the pressures between solid phases at the ground state is a necessary step towards constructing a model to better study the phase space of lithium.

When solid lithium is sufficiently compressed, different lattice shapes become more energetically favorable. This results in a solid to solid phase transition, one crystal transitioning to another. The goal of this work is to simulate this compression using Density Functional Theory (DFT) and determine if the simulated transition pressures agree with experimental results.

1.1 Physics

The physical concepts required to understand this project are how solids and the atoms in them behave, how solids change phase to other solids, and a few thermodynamic concepts. The following sections address each of these.

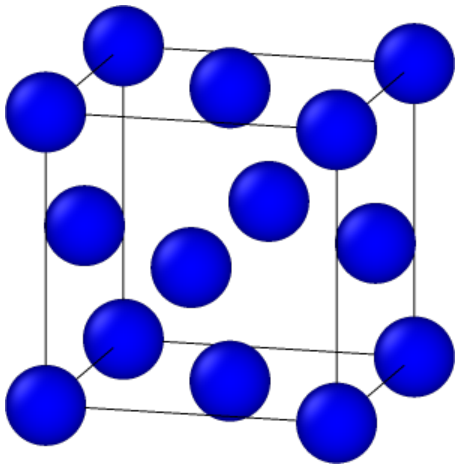

Figure 1.1 The Face-Centered Cubic (FCC) lattice structure. All the sides have equal length with atoms in the middle of each of the faces. These structure figures were made using the software OVITO[1].

1.1.1 Solids

Many solids are crystalline; which means that their atoms form repeating patterns. The shape depends on the atomic makeup, and the way atoms "fit" together determines the various macroscopic properties of materials. These properties include transparency, electrical and thermal conductivity, strength, and how fast sound waves propagate through the material.

The crystal structure that nature chooses is determined by the energy of the atoms. The atoms arrange in a crystal lattice because it is energetically favorable. External factors like pressure, volume, and forces affect the lattice in different ways. As an example, pressure can deform the lattice or cause electrical conductivity to rise or fall. Some common types of lattice include BCC (body-centered cubic) and FCC (face-centered cubic). See Figures 1.1 and 1.2.

How the environment influences the lattice is important to researchers. As more of these effects are documented, it becomes easier to predict how materials will react in a wider range of situations. This leads to improved materials in the future.

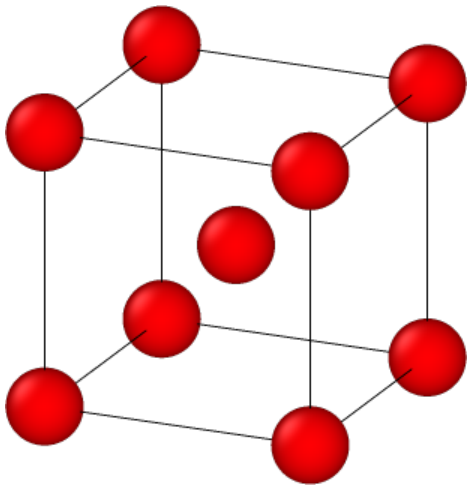

Figure 1.2 The Body-Centered Cubic (BCC) lattice structure. Note the single atom in the center of the unit cell. A unit cell is a unit of repeating structure from the lattice.

1.1.2 Phase Changes

Sometimes when a lattice is put under certain conditions it transitions from one crystal or phase of matter to another. The most frequently encountered phase changes in daily life are liquids changing to gases or liquids changing to solids. For example, when liquid water is heated the molecules gain energy. The water molecules separate and form a gas. When liquid water releases energy in the form of heat to its surroundings it starts freezing. The total energy of the molecules in a solid lattice is now less than the energy of the molecules as a liquid, so it solidifies.

Solid to liquid or liquid to gas phase changes are familiar and experienced daily while cooking or watching the seasons change. Solid to solid phase changes also occur. When Lithium is compressed the energy can rise until another solid phase becomes energetically favorable[2]. The goal of this project is to verify the predicted pressures where these phase transitions occur through simulation at constant temperature.

1.1.3 Thermal Consequences

The pressure at which a phase change occurs can be calculated through a process called common tangent construction. One approach is to take multiple "snapshots" of different volumes holding the atoms' positions fixed. By varying the volume of the cell, the distance between atoms in the lattice is reduced or expanded. This has the effect of increasing or decreasing the pressure within the lattice. These snapshot simulations calculate the total energy in the lattice. These energies are plotted against their associated volume. If the curves formed by these data points overlap with each other, the curves have a common tangent line between them. Another example of the common tangent construction is Figure 5.29 on page 191 of Schroeder[3].

On these diagrams, the slope of the common tangent line is used to find the transition pressure. Moving from the right to the left of the graph and following the curves and tangent lines with the lowest energy plots the energy change across the phase transitions as the volume decreases (conversely, as the pressure increases). These pressures are compared to the predictions based on experiments at various temperatures higher than the ground state temperature. All the simulations are carried out at the ground state energies (0 Kelvin).

1.2 Math

This project requires a few math concepts. The Schrödinger equation is solved using linear algebra on large matrices. The symmetry of various lattice structures is enumerated using group theory and symmetry operators. There is also the thermodynamics side, where equations of state like the Birch-Murnaghan[4] equation describe how various quantities relate. These are the main equations of focus for this project.

1.2.1 Density Functional Theory (DFT)

Density Functional Theory (DFT) is a way to solve Schrödinger's equation for a system of interacting particles. The probability density of the electron wavefunctions describes where electrons are most likely to be. From the electron configuration all the inter-atomic forces can be generated. These forces determine the energy held in the lattice.

The Vienna Ab initio Simulation Package (VASP)[5][6] calculates these orbitals numerically over a set of points in the reciprocal space of the lattice. The reciprocal space is the Fourier transform of the physical space. These points are chosen for efficiency and because it simplifies the math. These points are called k-points because they reside in k-space, k being the wavenumber.

1.2.2 Space Groups and Wyckoff Positions

Lattices can be described based on what unique actions can be done to them. If you apply some transformation to a lattice and it looks the same as before, that transformation is said to be a symmetry operator (it is an operator that preserves symmetry). The more symmetric a lattice, the more symmetry operators there are for it. Figure 1.3 shows a simple 90 degree rotation operator for a square lattice. Because the atoms have the same orientation as before the rotation, it is a symmetry operator.

Applying the same operator to a rectangle changes it from its original orientation. (see Figure 1.4) This means that the 90 degree rotation is not a symmetry operator for the rectangular lattice. There are other symmetry operators common to both lattices such as the 180 and 360 degree rotations. As a result, the rectangular lattice has less symmetry operators and is less symmetric than the square lattice.

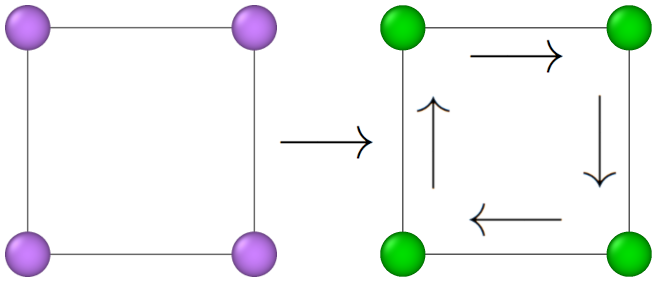

Figure 1.3 The square lattice looks how it did before the rotation.

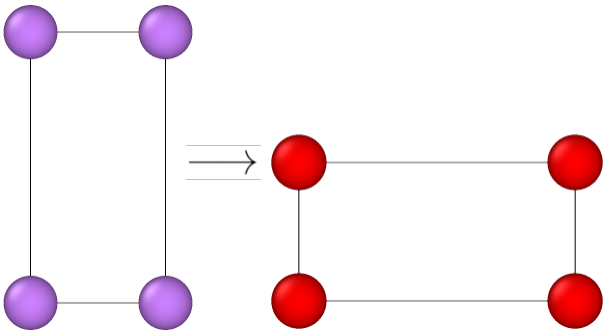

Figure 1.4 The rectangular lattice does not remain the same after rotating 90 degrees, so it is not a symmetry operator for the lattice.

Lattices that have the same symmetry operators are said to be in the same space group. Various physical properties are common to specific space groups. What type of atom and how it is positioned relative to other atoms is the majority of what is required to create any macroscopic property of a material.

An important part of this project is creating a crystal from a space group and a set of Wyckoff positions. These positions are unique because they allow one to create the entire lattice by applying symmetry operators to them. Representing the operators as matrices (usually 3x4 in size) means that the lattice can be created through matrix-vector multiplication, using the vector positions of the atoms. Giving the space group and Wyckoff positions of a material allows other researchers to reconstruct the structure.

1.2.3 The Birch-Murnaghan Equation of State

Equations of state describe the relationship between different quantities at any given moment in time. A common example of an equation of state is the familiar

$$PV = Nk_BT \tag{1.1}$$

This equation shows how various quantities are related. When solving for each quantity, it becomes easier to see the relationships between quantities. For example:

$$P = \frac{Nk_BT}{V}, \quad V = \frac{Nk_BT}{P}, \quad N = \frac{PV}{k_BT}, \quad T = \frac{PV}{Nk_B} \tag{1.2}$$

These are all expressions that relate a quantity as a function of the others. There are many equations of state for different situations. For this project focused on solid phase changes it is necessary to use an equation that takes into account various quantities specific to solids. This is where the Birch-Murnaghan[4] equation of state comes in:

$$P(V) = \frac{3B_0}{2}\left[\left(\frac{V_0}{V}\right)^{7/3} - \left(\frac{V_0}{V}\right)^{5/3}\right]\left[1 + \frac{3}{4}\left(B_0' - 4\right)\left[\left(\frac{V_0}{V}\right)^{2/3} - 1\right]\right] \tag{1.3}$$

where B_0 and B_0' are the bulk modulus and derivative of the bulk modulus with respect to pressure of the material. V_0 and V are the initial and final volumes of the material. The energy of the lattice is found through Density Functional Theory after fixing the volume. Integrating this equation over pressure gives the energy vs̊volume version:

$$E(V) = E_0 + \frac{9V_0B_0}{16}\left[\left[\left(\frac{V_0}{V}\right)^{2/3} - 1\right]^3 B_0' + \left[\left(\frac{V_0}{V}\right)^{2/3} - 1\right]^2\left[6 - 4\left(\frac{V_0}{V}\right)^{2/3}\right]\right] \tag{1.4}$$

This equation of state is the one to be fit to the E vs V data generated in the simulations. After fitting the parameters, functional versions of each curve are produced. These are much easier to work with when applying the common tangent construction.

1.3 Computation

Density Functional Theory (DFT) is relatively fast due to the optimization of linear algebra on computers. Software packages like The Vienna Ab initio Simulation Package[6] (VASP) optimize various quantum mechanics calculations using hardware-optimized linear algebra libraries and efficient programming languages (Fortran in the case of VASP). Even with this optimized software, the calculations still take anywhere from 30 minutes to tens of hours for each data point. The Fulton supercomputer at Brigham Young University (BYU) accelerates the process by running dozens of simulations at once.

1.3.1 The Vienna Ab initio Simulation Package (VASP)

VASP is the software that does the heavy lifting in calculating various quantities of a given structure. The most helpful model in VASP for this project is DFT, which was described earlier. Different types of runs and options can be specified in input files when starting the program. An example INCAR (short for INput CARd, a FORTRAN term for the punch cards used to program older computers) input file for this project is supplied in Appendix B.1.

While the atoms in these simulations are fixed, the wavefunction for each electron needs to be solved for. VASP solves the Schrödinger equation for each electron orbital and relaxes the atoms until forces are down to zero. This is done by finding the difference in the total energy from one numerical relaxation step to another. Once the relaxation is complete the simulation ends and the total free energy is written to an output file.

Various other calculations can be done with VASP to verify results, find out what happens under different processes, and compare various inter-atomic potentials and

quantum mechanics models. The results from VASP can be used to train neural networks to predict what properties future materials may have before simulating them.

1.3.2 Supercomputer Architecture

The Fulton supercomputer at BYU is a large collection of individual machines called nodes. These nodes are grouped based on what hardware they have. These groups are called clusters. There are twelve clusters on the Fulton supercomputer, all managed by software to ensure researchers get their jobs run timely and in order.

The simulations required for this project do not have especially high computer resource requirements. Because of this, the jobs can be put in between other larger ones. No struggles with excessively long queues for jobs to be run were encountered. Because the jobs are very similar with the only changing parameter being the volume, it is easier to generate them all and submit them all at once. Each simulation keeps a processor core busy, so the many thousands of cores on a supercomputer allowed high throughput processing. Access to hundreds of processors was extremely helpful while calculating 50-60+ different volumes at a time.

There are various job statistics in appendix D to show how resources were utilized. The Fulton supercomputer and its team at the Office of Research Computing were invaluable to this research project.

1.3.3 Making the Problem Parallel

Because of the large amount of volumes required for each structure, it became necessary to learn bash scripting. There are many similarities between setting up different jobs where only one parameter is changed by a small amount. Because of this, au-

tomation becomes an attractive option.

Bash scripting is writing a file of commands to be run all at once instead of being input one at a time by an, objectively slower, user. It's called "bash" because the most common Linux shell is the "Bourne Again SHell." In addition to normal command-line commands, control flow statements are also included such as if statements and for loops.

Using bash scripts, a single directory for a run containing all the files common to all runs can be copied for each volume change. This results in 50+ directories and runs being generated and submitted from a single script that takes less than a minute to run. The project would not yet be complete without this extremely useful tool.

Chapter 2

Procedures

This section focuses on running the computations necessary to solve the problem introduced. A supercomputer makes the simulations possible to solve in a reasonable amount of time. Fortunately, Brigham Young University (BYU) has a supercomputer managed by their Office of Research Computing. The Fulton supercomputer allows students and research groups to perform large calculations on equivalently large amounts of computer hardware. Ira and Mary Lou Fulton sponsored BYU and provided the funding necessary to build the first supercomputer on campus.

Once approved for an account, a secure shell login along with two-factor authentication allows access to the login nodes. A node is a physical unit of hardware, usually a motherboard with two processors. After a user account is set up, software and files are loaded and installed. The Vienna Ab initio Simulation Package (VASP) performs all the necessary quantum mechanics from various user-supplied input files. These include configuration files specifying atom positions, inter-atomic potentials, and simulation settings.

After running the simulations, output files are generated. These files are downloaded to a local machine before information is extracted from them. Python and

Julia scripts verify convergence, analyze the data, and perform the necessary calculations to finalize the results. From these results we can compare the simulation to experimental data to see if the simulation correlates with real world results. In this case, the objective is the transition pressures at solid-solid phase transitions of lithium at high pressure.

2.1 Setting Up the Problem

To do simulations with VASP there are specific input files to be prepared. The positions of the atoms, k-points, and simulation input files are all required for VASP to start a run. The following sections provide more details on each of these.

2.1.1 Atom Positions

VASP needs the atomic positions in order to perform its calculations. POSCAR (POSitions CARd) files are long lists of coordinates that tell VASP where atoms are placed at the start of the simulation. The lattice vectors for the crystal are supplied at the top of the file in a matrix. Each atom is specified by a basis vector that specifies the magnitude of the basis vectors needed to form a linear combination that specifies that atom's location.

The positions are specified using either Cartesian or direct coordinates. Cartesian coordinates are specified with the physical distance in Angstroms. Direct coordinates are expressed as fractions of the basis vectors for the unit cell. The lattice parameter's units are in Angstroms.

For simple structures that repeat often and have few unique atoms in the unit cell (the minimum repeating piece of a lattice) creating POSCAR files is generally simple. However, for very large lattices that don't have much symmetry it is better to

generate POSCAR files systemically. This is where space groups, symmetry operators, and Wyckoff positions come into play.

The script to generate the atom positions of the oC88 lithium structure is written in Julia. (see Appendix A.3) It defines the symmetry operators for the space group as matrices. There is a fourth column on the matrices because some of the symmetry operators are non-symmorphic. This means that a fractional translate must be appended to the operation to make it a symmetry operator. The oC88 structure has a glide plane that requires this configuration. The space group information was found in a supplement to the paper by Marques et al.[7] provided by Gregoryanz[8]. The script takes in the Wyckoff positions and applies the necessary symmetry operators via matrix multiplication to create the rest of the unit cell. The supplement has plots of the structures to compare against. Now these coordinates are formatted for POSCAR files which VASP can read. Figure 2.1 shows the output structure.

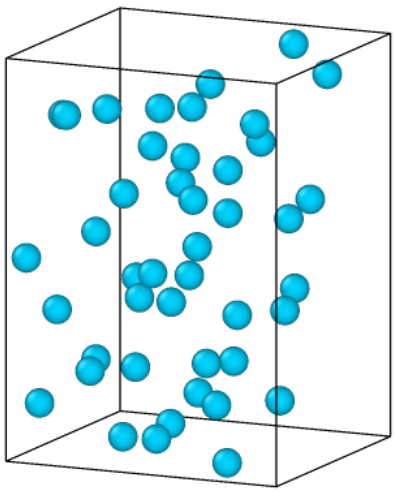

Figure 2.1 Structure of lithium oC88 as output from the Julia script applying symmetry operators to Wyckoff positions and translating atoms back into the unit cell.

2.1.2 Generating k-points

The k-points serve as points for integration and other calculations during a run. They are points spaced throughout reciprocal space. The more k-points there are, the longer the calculation will take. The goal is to find a minimum number of k-points for a maximum amount of computational accuracy. There are many different methods to achieve this. One involves letting VASP automatically generate k-point positions based on the input POSCAR file. However, this can produce more k-points than are needed. The method for generating k-points used in this paper comes from software written by Dr. Gus Hart[9] called "autoGR". This script takes in a KPGEN file (which contains parameters) and POSCAR file, then optimally places k-points based on the symmetries of the lattice.

2.1.3 VASP Input Files

Settings for calculations are defined by tags in a file called INCAR. The VASP Wiki[10] has an explanation for every INCAR tag along with examples. To find the lithium transition pressures, the simulation needs to return the free energy of the crystal. Free energies can be compared to determine thermodynamic stability. Plotting the free energy vs the volume per ion gives a visual representation of the most stable volume for the structure. A static run of VASP keeps all the atoms in the same place rather than letting their positions update. Repeating this kind of run for many different volumes gives the necessary quantities to find the transition pressures. There are more details about how multiple runs are set up in the next section. Other settings tags are related to electronic optimization, ionic relaxation options, and performance options (for multi-core jobs).

Another important file for VASP to run properly is called a POTCAR (POTential

CARd) file. The POTCAR file holds the inter-atomic potential for the elements in the simulation. If a material has multiple chemical elements, as is the case for alloys, the potentials for the individual elements need to be concatenated into one POTCAR file. Since lithium is the only element in this project, the POTCAR file will only have the information for lithium. These potential files are included with a VASP subscription available to research groups. They are called Projector Augmented Wave (PAW) potentials. More information about these can be found in Blochl[11].

2.2 Submitting and Managing Jobs

The first type of job to submit is a relaxation run. In this kind of run, the input crystal atoms are allowed to relax and shift until the forces on those atoms are zero. This is important because optimal calculations from Wyckoff positions and space group symmetry operators may not give the lowest possible energy for the structure. This kind of run has specific options in the INCAR file (such as IBRION=0) found in Appendix B.1. The atom positions from this relaxation run are used to generate the different unit cell volume positions in the static runs.

The graphs and information required to find the transition pressure between phases are volume per ion vs free energy graphs. Many volumes need to be simulated to create a large enough curve to fit the Birch-Murnaghan equation of state. Bash scripts are helpful for organizing all the runs. A directory containing a relaxed structure run acts as a base image for each run. The script (see Appendix A.1) creates a new directory for each run. It copies the base image directory and overwrites parameters in the input files. The POSCAR file is adjusted so that the lattice parameter is increased or decreased by 0.01. This changes the volume of the unit cell by 1 percent each time, effectively reducing or increasing the pressure. This process

repeats until we have folders for each scaling of the unit cell from -40% to +40% the original size. Because the positions of the atoms change, new k-points must be generated as well. The script runs the kpoints.x executable to generate the k-points. A static VASP run (IBRION=-1) fixes the atoms in place and uses Density Functional Theory (DFT) to calculate the desired quantities. These are the volume per ion and the free energy of the structure.

2.2.1 Bash Scripting to Automate Static Runs

Due to the large number of runs required to generate a data set suitable for fitting equations of state, automation becomes key. Each run has the necessary POSCAR (position), INCAR (input), POTCAR (potential), and KPGEN (k-points generation instructions) files. All the files that are common across runs can be put into a "Golden Image" folder that the shell script copies from. The script then generates and names a folder for each run depending on what percentage the volume will be changed by. The script uses a command line text editor called 'sed' to increase the lattice parameter in the POSCAR file depending on what iteration of a for loop it is in. This allows the user to simply provide a structure folder and how much to increase and decrease the volume of that structure in percentages. For example, if a user requests a 20% increase of the volume and a 35% decrease, the script will make 55 folders each with the correct volumes.

After updating the lattice parameter in the POSCAR file, the script also updates the k-points by running the kpoints.x program. After this has been done for all the runs, the script loops through all the directories and submits the jobs by calling a SLURM job submission script. (Yes, bash scripts can call other bash scripts!) Following are additional details on the SLURM job scheduler and how jobs actually get handed off to the supercomputer.

2.2.2 Analyzing Job Statistics

The SLURM job scheduler provides numerous useful outputs to the administrators of the supercomputer as well as to the researchers. There is an online dashboard each researcher can access to see how their jobs are performing and where they can make improvements. The dashboard summarized the processing status, how much processor and RAM usage occurred for a job, and the reasons for jobs failing. Some of these reasons include using up all allocated run time, RAM, or a physical hardware failure. These statistics are valuable to researchers in determining how much RAM is needed for a specific type of job. If the memory is too little, the recommended procedure is to double the amount requested until the job completes. Fine adjustments can then be made to maximize usage of the supercomputer.

SLURM job scripts even have parameters to send e-mail updates when a job starts, ends, or fails. These are especially useful when running large batches of jobs overnight. E-mail filters can also reduce the process of checking on jobs to a quick glance through an inbox folder.

Failed jobs usually just need to be run with additional RAM. However, when working with multiple cores there are many pitfalls. These are covered in Appendix C.

2.3 Compiling data

Once the runs for all the volumes of a structure are complete, the free energy and volume of each unit cell need to be extracted. After this, they are copied into a numpy array in python before being plotted using the matplotlib library. The graphs are inspected to make sure they form an expected curve shape. After this, the Birch-Murnaghan equation of state is fit to the data using the 'scipy' library. Once these

functions are fit, it becomes possible to perform a common tangent construction between the curves. The slope of this common tangent is the transition pressure.

2.3.1 Searching Output Files

The values of interest from the VASP calculation are output to a file named OUTCAR. This means that for 55 different volume runs there are 55 different OUTCAR files, each in their own subfolder. This takes a long time to sort through by hand. There is a command called 'grep' that can search all file names and contents including subdirectories for a specific string. Luckily, there is only one line in the OUTCAR file that has the total energy in each of the OUTCAR files. The grep command looks like 'grep -r "free energy"'. The final free energy calculation has two spaces between it to make it easier to search for. The same process happens for the volume per ion. The energy is divided by the number of atoms in the unit cell to make a comparison between different lattices.

2.3.2 Wrangling and Graphing with Python

Each run produces a single value for the free energy and volume per ion. These are put into separate numpy arrays so they can be manipulated in pairs. The curve fits in Figure 3.1 show why so many runs are necessary. The most stable lattice configuration is at the lowest energy. As the volume decreases towards the left on the chart, the pressure increases. The structure compresses until a phase change is energetically preferable. The phase change occurs along a common tangent line, the construction of which is discussed in the next section.

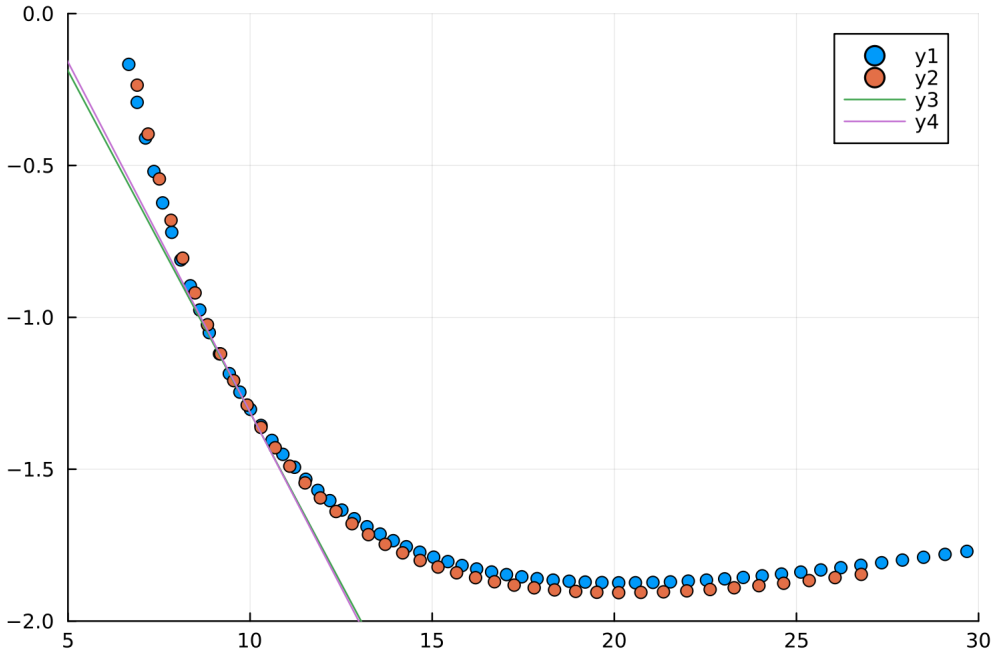

Figure 2.2 An example of common tangent construction in Julia. The data from two structures (y1 and y2) is iterated over to find the closest line match between them.

2.3.3 Common Tangent Construction

A finite difference method was used to find the common tangent line. This is done by taking differences between data points to find the slope and comparing the values between structures. The finite difference method was sufficient to find the common line between the curves within the desired uncertainty. The lines are plotted on the same curve plot to visually confirm they are common between functions. Dr. Nelson[12] provided the code to perform this check, included in Appendix A.3. An example of the output is included in Figure 2.2

For a more accurate method, the Birch-Murnaghan fit equations can be used. The equations are differentiated over a dense grid of points of user-defined size. This allows the user to find the common tangent line to an arbitrary precision, taking into account the error in the fit. These slopes and y-intercepts are compared between

structures to find the common line whose slope is the negative pressure. This process is a good future project for practicing numerical differentiation techniques.

Chapter 3

Analysis and Discussion

Now that all the energies have been found for the volumes, it is time to figure out the transition pressures between the various solid phases of the lithium. The first step is to determine what other experimental predictions exist for a comparison. The next step is to graph all the energy vs volume data points from the simulations and fit the Birch-Murnaghan equation of state to them. Last is the common tangent construction, a process that finds the line on this graph that is shared between two phase structure curves. This line gives us the information needed to determine the transition pressure.

3.1 Predicted Transition Pressures

The calculated lithium transition pressure needs to be compared with the predictions made in Figure 1 of the paper by Guillaume et al.[2]. The predicted phase transition pressure from the FCC (Face-Centered Cubic) to cI16 (denoted with a Pearson symbol[13]) structures is around 30 GPa. The predicted phase transition pressure from the cI16 to oC88 structures is around 62 GPa. Table 3.1 are the pressures to compare

with the DFT simulated results of this project:

Table 3.1 The transition pressures of Lithium calculated by DFT using VASP and a PAW potential.

Structure Transition	Simulated Pressure (GPa)
FCC - cI16	36.45 ± 0.78
cI16 - oC88	84.73 ± 0.59

3.2 The Energy vs. Volume Graph

The energy vs volume graph shows the energy of a given crystal. When the lattice is put under pressure its volume decreases. This generally increases the energy stored in the lattice. This also happens when the volume increases. There is a minimum energy in the potential that acts as an equilibrium. The volume that corresponds with this energy is the preferred state of the system.

The tangent line shared by two curves on this graph has a lower energy level than the compressed or expanded volume of either. Because it keeps the energy minimized it is more preferable for the structure to change into the shape of the other one. This is a solid-solid phase change.

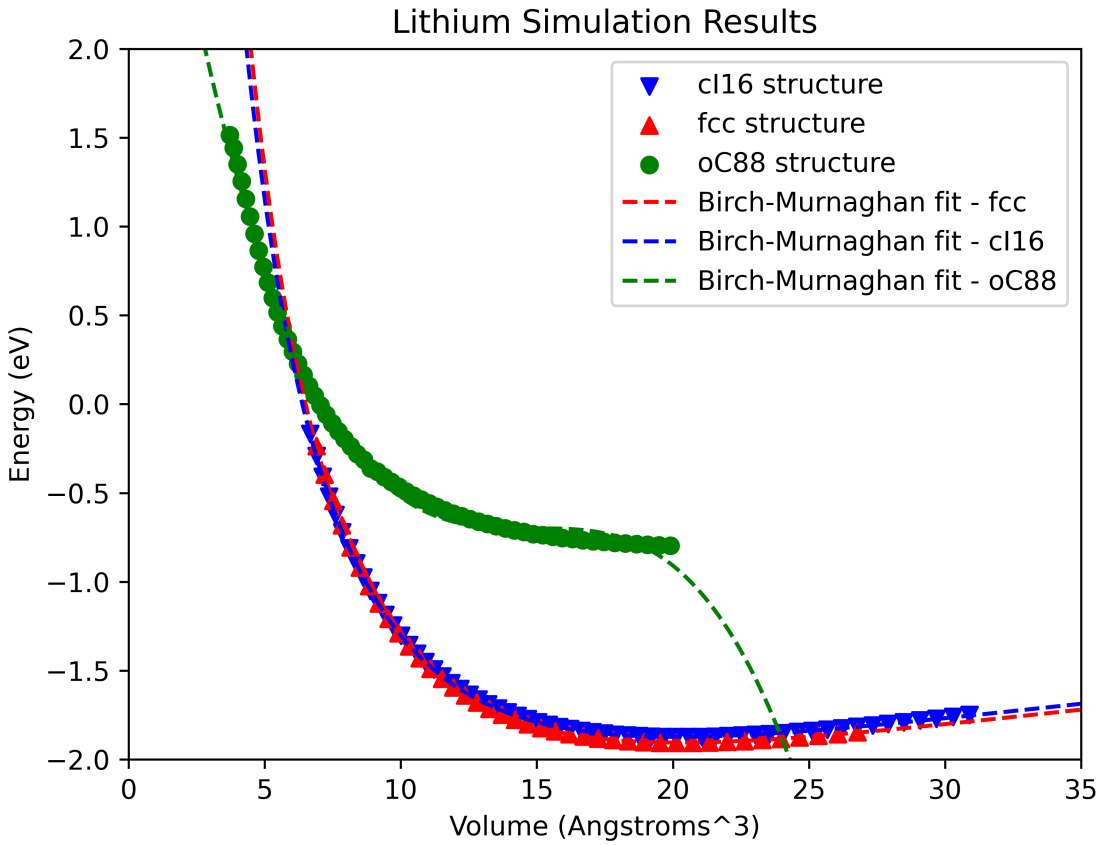

Figure 3.1 Simulated energy vs. volume for three dense structures of lithium. Each data point is the result from a static VASP calculation.

As a reminder from the introduction, and for convenience, the Birch-Murnaghan equation of state integrated over pressure gives the energy-volume version:

$$E(V) = E_0 + \frac{9V_0B_0}{16}\left\{\left[\left(\frac{V_0}{V}\right)^{2/3} - 1\right]^3 B_0' + \left[\left(\frac{V_0}{V}\right)^{2/3} - 1\right]^2 \left[6 - 4\left(\frac{V_0}{V}\right)^{2/3}\right]\right\} \tag{3.1}$$

where V_0 is the starting volume, V is the final volume, B_0 is the bulk modulus of the material, and B_0' is the derivative of the bulk modulus with respect to pressure. The scipy.optimize curve fitting function takes in a list of parameter guesses as an argument along with this function. More accurate guesses for these parameters increases

the speed of the curve fit. They are similar enough for this experiment that the same guesses were used for each structure.

3.3 The Common Tangent Construction

The procedure for finding the common tangent line is as follows. Find the slopes of the curves at each data point by using finite differences, append the slopes to a list, then find the slope values and y-intercept values that are closest to each other for both curves. Use the matching values to make an equation for the line tangent to both curves.

The thermodynamic identity for the free energy of a thermal system is found in Schroeder[3]:

$$dU = TdS - PdV + \mu dN \tag{3.2}$$

Since the number of atoms does not change and the temperature is zero for our calculations, the identity can be rewritten as:

$$dU = TdS - PdV \tag{3.3}$$

Since VASP does not calculate the entropy for the system, we can also remove the temperature dependent term. The TdS term can also be omitted because the contribution from this term is negligible in the ground state.

$$dU = -PdV \tag{3.4}$$

Rearranging gives us a relationship for pressure:

$$-P = \frac{dU}{dV} \tag{3.5}$$

This shows that the derivative of the energy with respect to volume is equal to the negative pressure.

A finite difference method calculates the derivatives at each data point. The derivatives for each curve are calculated and stored. These values are sorted to find the two closest for the structures in question. If necessary the fit Birch-Murnaghan equations provide a continuous function to take derivatives over if the finite difference method has too much variance between the derivatives.

The resulting value is the negative pressure at which the solid-solid phase transition occurs. This is the purpose for the straight connecting lines on figure 2.2 between the curves.

Chapter 4

Conclusions

This project tested the transition pressure predictions of ground state lithium in the paper by Guillaume et al.[2]. The previously discussed VASP calculations are off from the predicted transition pressures at low temperature from the FCC to cI16 and cI16 to oC88 structures. This can be seen in Table 4.1. More testing needs to be done to ensure that Density Functional Theory can be used at higher pressures required for the additional phases not explored in this project.

Table 4.1 The transition pressures of Lithium predicted by DFT using VASP and a PAW potential compared to predictions by Guillaume et al.[2]

Structure Transition	Simulated Pressure (GPa)	Predicted Pressure (GPa)
FCC - cI16	36.45 ± 0.78	31
cI16 - oC88	84.73 ± 0.59	62

The simulated pressures are much higher than the estimates. The results for the FCC to cI16 phase change are closer than the cI16 to oC88 phases. A possible explanation for this skew is insufficient potentials between the atoms due to the extremely small distances between them. More simulations need to be run to test convergence to known transition pressures of lithium in lower pressure conditions.

Chapter 5

Future Work

This project creates a large amount of data in the VASP output files. This makes it easier to start other projects. The process still needs optimizing both in the workflow and the computation. These processes can be applied to other elements and alloys to determine their phase transition pressures as well. A model to predict the phase transition pressures using machine learning techniques could be created. This would require more simulations across additional elements.

5.1 Reduce Computation Time and Complexity

The time to calculate the electronic structure of smaller atomic lattices is short. This is in part because the simulations are using scaled versions of pre-relaxed structures. Because the atoms aren't allowed to move, this increases the speed. More atoms in the unit cell results in a longer relaxation time. The FCC and BCC lattices take 10-30 minutes each. However, the computation time to run the larger structures in this project (like oC88, containing 88 atoms) is around 10 hours. This time can be improved by optimizing VASP for the calculation and the calculation inputs.

5.1.1 Optimizing VASP Settings

The settings for VASP are documented on the VASP wiki[10]. These include settings for how specific parts of the calculation are evaluated. Future projects should involve tuning simulations using these settings.

In addition to these, VASP has the ability to parallelize over various features. This means that the problem can be parted out to multiple processors, increasing the speed of the calculation. The most beneficial of these are the INCAR settings NPAR and KPAR. NPAR determines the number of electronic bands that should be assigned to each processor core for a multi-processor job. KPAR determines how many k-points should be given to each processor core.

Another option is to use different inter-atomic potentials and their assosiated settings. A particularly useful setting is ENCUT, which defines the energy cutoff. Each Projector Augmented Wave (PAW) potential from VASP comes with a default energy cutoff. There are also different versions of potentials for the same element that make different assumptions about the electronic structure. These should be explored in the future to find a potential that gives good results along with optimal speed.

5.1.2 Compile VASP for GPUs

Another option is to compile VASP to run on graphical processing units (GPUs). Graphics processors have an architecture that is more conducive to solving linear algebra problems. They also generally have hundreds to thousands more compute cores than conventional CPUs. This would be a great project to improve times for the simulations future projects may use.

There are various ways to optimize how VASP performs during the compiling process. Some of these improvements come from using software libraries optimized

for the hardware. There is opportunity to find more efficient software on hardware configurations.

5.1.3 Minimize required k-points while maintaining accuracy

The k-points define a sort of computational "resolution." The approximation gets better the more there are, but it also gets longer. Therefore, an option is to minimize the number of k-points required to get good results.

Normal k-point generation done by VASP doesn't take into account lattice symmetry. If a lattice is symmetric the associated symmetric k-points should return similar values when evaluated. Using this knowledge, we can select the k-points that are most unique for a given lattice. This is achieved using Dr. Gus Hart's[9] autoGR code. There are many settings for the KPGEN input file. A future project would tweak these settings to find a configuration that gives the smallest amount of k-points while maintaining good results.

5.2 Test Other Elements and Structures

The entire process of this project can be applied to other elements and alloys as well. More simulations across additional elements would provide more insight into the phase transitions of various structures. More work to compare how well DFT correlates with experimental pressure transitions is required.

5.2.1 Larger sample size

Phase transition pressures found experimentally are currently not measured at 0 Kelvin. This simulation process finds these pressures if the structures are in their

ground state. If these simulations were run for other elements, it would provide a good base profile of roughly where to expect transition pressures during experiments.

The phase transition pressures of alloys could also be useful in determining how stable various structures and materials are at a given temperature. It would be interesting to see how the transition pressure of an alloy compares to its constituent elements' transition pressures.

5.2.2 Confirm good results for DFT on other elements

This project compared the phase transition pressures predicted by Density Functional Theory (DFT). The accuracy of DFT for this calculation is unknown and likely to change based on each element. There may be some elements that are more accurately modeled by DFT than others. More work needs to be done to determine which elements these are.

5.3 Build a Predictive Model

If enough data points are gathered, the results can be used to train a neural network. This neural net could then predict the phase transition pressure of elements that haven't been simulated. If the neural net is accurate enough, it could encode the patterns that determine phase transition pressures common to all elements.

This predictive model could be analyzed to see agreement to experimental results. The predictions may also uncover interesting candidates for further study and simulation. The positions and types of the atoms would form the input data, with the transition pressure being the output. This training set needs to be large enough to cover unseen cases. To accomplish this, elements should be picked that have a large variance in the atomic number.

Bibliography

[1] A. Stukowski, "Visualization and analysis of atomistic simulation data with OVITO–the Open Visualization Tool," Modelling and Simulation in Materials Science and Engineering **18,** 015012 (2009), publisher: IOP Publishing.

[2] C. L. Guillaume, E. Gregoryanz, O. Degtyareva, M. I. McMahon, M. Hanfland, S. Evans, M. Guthrie, S. V. Sinogeikin, and H.-K. Mao, "Cold melting and solid structures of dense lithium," Nature Physics **7,** 211–214 (2011), number: 3 Publisher: Nature Publishing Group.

[3] D. Schroeder, *An Introduction to Thermal Physics* (Oxford University Press, 2021).

[4] "Birch–Murnaghan equation of state," Wikipedia (2021), https://en.wikipedia.org/w/index.php?title=Birch%E2%80%93Murnaghan_equation_of_state&oldid=1040002492.

[5] G. Kresse and J. Hafner, "Ab initio molecular dynamics for liquid metals," Physical Review B **47,** 558–561 (1993), publisher: American Physical Society.

[6] G. Kresse and D. Joubert, "From ultrasoft pseudopotentials to the projector augmented-wave method," Physical Review B **59,** 1758–1775 (1999).

[7] M. Marqués, M. I. McMahon, E. Gregoryanz, M. Hanfland, C. L. Guillaume, C. J. Pickard, G. J. Ackland, and R. J. Nelmes, "Crystal Structures of Dense Lithium: A Metal-Semiconductor-Metal Transition," Physical Review Letters **106,** 095502 (2011), publisher: American Physical Society.

[8] E. Gregoryanz, personal communication.

[9] G. L. W. Hart, J. J. Jorgensen, W. S. Morgan, and R. W. Forcade, "A robust algorithm for k-point grid generation and symmetry reduction," Journal of Physics Communications **3,** 065009 (2019).

[10] "The VASP Manual - Vaspwiki,".

[11] P. E. Blöchl, "Projector augmented-wave method," Physical Review B **50,** 17953–17979 (1994).

[12] L. J. Nelson, personal communication.

[13] "Pearson symbol," Wikipedia (2022), https://en.wikipedia.org/w/index.php?title=Pearson_symbol&oldid=1086617249.

[14] J. B. Neaton and N. W. Ashcroft, "Pairing in dense lithium," Nature **400,** 141–144 (1999), number: 6740 Publisher: Nature Publishing Group.

[15] M. Hanfland, K. Syassen, N. E. Christensen, and D. L. Novikov, "New high-pressure phases of lithium," Nature **408,** 174–178 (2000), number: 6809 Publisher: Nature Publishing Group.

[16] K. Shimizu, H. Ishikawa, D. Takao, T. Yagi, and K. Amaya, "Superconductivity in compressed lithium at 20 K," Nature **419,** 597–599 (2002), number: 6907 Publisher: Nature Publishing Group.

[17] I. Tamblyn, J.-Y. Raty, and S. A. Bonev, "Tetrahedral Clustering in Molten Lithium under Pressure," Physical Review Letters **101,** 075703 (2008), publisher: American Physical Society.

[18] V. G. Vaks, M. I. Katsnelson, V. G. Koreshkov, A. I. Likhtenstein, O. E. Parfenov, V. F. Skok, V. A. Sukhoparov, A. V. Trefilov, and A. A. Chernyshov, "An experimental and theoretical study of martensitic phase transitions in Li and Na under pressure," Journal of Physics: Condensed Matter **1,** 5319 (1989), publisher: IOP Publishing.

[19] "Li cI16 (Li hp3, T = 180 K, p = 44.5 GPa) Crystal Structure - SpringerMaterials,".

[20] E. Babaev, A. Sudbø, and N. W. Ashcroft, "A superconductor to superfluid phase transition in liquid metallic hydrogen," Nature **431,** 666–668 (2004), number: 7009 Publisher: Nature Publishing Group.

[21] M. J. Mehl, D. Hicks, C. Toher, O. Levy, R. M. Hanson, G. Hart, and S. Curtarolo, "The AFLOW Library of Crystallographic Prototypes: Part 1," Computational Materials Science **136,** S1–S828 (2017).

[22] G. J. Ackland, M. Dunuwille, M. Martinez-Canales, I. Loa, R. Zhang, S. Sinogeikin, W. Cai, and S. Deemyad, "Quantum and isotope effects in lithium metal," Science **356,** 1254–1259 (2017), publisher: American Association for the Advancement of Science.

[23] V. G. Vaksi, M. I. Katsnelson, G. KoreshkovJ, I. Likhtenstein, E. Parfenovi, V. F. Skok, and V. Trefilovt, "An experimental and theoretical study of martensitic phase transitions in Li and Na under pressure," p. 18 .

[24] S. Limpijumnong and S. Jungthawan, "First-principles study of the wurtzite-to-rocksalt homogeneous transformation in ZnO:A case of a low-transformation barrier," Physical Review B **70,** 054104 (2004).

[25] "Legendre transformation," Wikipedia (2022), https://en.wikipedia.org/w/index.php?title=Legendre_transformation&oldid=1077873569.

[26] Chloe, "Common tangent construction,", 2017.

[27] a-cyclohexane molecule, "Answer to "Common tangent construction",", 2017, https://chemistry.stackexchange.com/a/75045.

[28] J. Custer, "Answer to "Common tangent construction",", 2017, https://chemistry.stackexchange.com/a/75038.

[29] F. D. Murnaghan, "The Compressibility of Media under Extreme Pressures," Proceedings of the National Academy of Sciences of the United States of America **30,** 244–247 (1944).

[30] user1349763, "Common tangent function,", 2012, https://stackoverflow.com/q/10269930/12922352.

[31] S. Byrnes, "Answer to "Common tangent function",", 2012, https://stackoverflow.com/a/10271179/12922352.

[32] J. Dagdelen, "Answer to "Common tangent function",", 2017.

[33] "Conjugate variables (thermodynamics)," Wikipedia (2021), https://en.wikipedia.org/w/index.php?title=Conjugate_variables_(thermodynamics)&oldid=1050973718.

[34] V. Hirvonen, "Legendre Transformations For Dummies: Intuition & Examples,".

[35] D. A. D. A. . Young, *Phase diagrams of the elements* (Berkeley : University of California Press, 1991).

[36] Z. Yu, H. Y. Geng, Y. Sun, and Y. Chen, "Optical properties of dense lithium in electride phases by first-principles calculations," Scientific Reports **8,** 3868 (2018), number: 1 Publisher: Nature Publishing Group.

[37] J. Lv, Y. Wang, L. Zhu, and Y. Ma, "Predicted Novel High-Pressure Phases of Lithium," Physical Review Letters **106,** 015503 (2011), publisher: American Physical Society.

[38] M. I. Aroyo, "SPACE-GROUP SYMMETRY," p. 33 .

[39] "(International Tables) Volume A home page,", publisher: International Union of Crystallography.

[40] "Point group," Wikipedia (2022), https://en.wikipedia.org/w/index.php?title=Point_group&oldid=1090267873.

[41] "List of space groups," Wikipedia (2022), https://en.wikipedia.org/w/index.php?title=List_of_space_groups&oldid=1080561709.

[42] "Symmetry operation," Wikipedia (2022), https://en.wikipedia.org/w/index.php?title=Symmetry_operation&oldid=1085966031.

[43] "Crystallographic Space Group Diagrams and Tables,".

[44] "Bilbao Crystallographic Server,".

[45] H. T. Stokes, *Solid State Physics for Advanced Undergraduate Students* (Brigham Young University, 2000), google-Books-ID: wMZBAAAACAAJ.

[46] M. Glazer and G. Burns, *Space Groups for Solid State Scientists* (Elsevier Science, 2013).

[47] M. Boas, *Mathematical Methods in the Physical Sciences* (Wiley, 2006).

[48] W. Setyawan and S. Curtarolo, "High-throughput electronic band structure calculations: Challenges and tools," Computational Materials Science **49,** 299–312 (2010).

[49] H. J. Monkhorst and J. D. Pack, "Special points for Brillouin-zone integrations," Physical Review B **13,** 5188–5192 (1976), publisher: American Physical Society.

[50] J. Hafner, "Ab-initio simulations of materials using VASP: Density-functional theory and beyond," Journal of Computational Chemistry **29,** 2044–2078 (2008), _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.21057.

[51] "Slurm Workload Manager - Documentation,".

[52] "NIST ICSD Web Site,".

[53] C. J. Pickard, personal communication.

[54] M. Marques Arias, personal communication.

[55] G. Hart, personal communication.

[56] Unknown, "Ice Structure," https://web.archive.org/web/20161016143124/http://www.uwgb.edu/dutchs/petrology/Ice%20Structure.HTM (2016).

Appendix A

Code and Scripts

The following is a selection of scripts and code used for building plots and automating tasks on the supercomputer. All the code files can also be found on GitHub at the following link: https://github.com/keatontate/senior-thesis

A.1 Bash Scripts

The bash scripts are for automating common tasks while performing simulation calculations. They take care of generating file structures and submitting large amounts of jobs all at once.

This one sets up the decreased and increased volume (increased and decreased pressure) runs:

```
#!/bin/bash

# Get parameters from user

read -p 'How many increased volume runs? (+1% volume ea.) ' numrunsinc
read -p 'How many decreased volume runs? (-1% volume ea.) ' numrunsdec
read -p 'Name of structure? ' structname
read -p 'Name of starting folder containing relaxed structure? ' imagefolder

```

```
10 echo 'Generating increased volume folders...'
11
12 for ((i = 1; i <= $numrunsinc; i++)); do
13         dir=./$structname-$i-percent-larger
14         mkdir $dir
15         cp -r ./$imagefolder/* $dir/
16         latticeparam=$(sed -n '2p' $dir/POSCAR)
17         if (( $i >= 10 )); then
18                 newlatticeparam=$(echo "print($latticeparam * 1.$i)" | python)
19         else
20                 newlatticeparam=$(echo "print($latticeparam * 1.0$i)" | python)
21         fi
22         echo $newlatticeparam
23         sed -i "2d" $dir/POSCAR
24         sed -i "2i $newlatticeparam" $dir/POSCAR
25         # also update the INCAR file IBRION tag
26         sed -i "3d" $dir/INCAR
27         sed -i "3i IBRION=-1" $dir/INCAR
28 done
29
30 echo 'Generating decreased volume folders...'
31
32 for ((j = 1; j <= $numrunsdec; j++)); do
33         dir=./$structname-$j-percent-smaller
34         mkdir $dir
35         cp -r ./$imagefolder/* $dir/
36         latticeparam=$(sed -n '2p' $dir/POSCAR)
37         newlatticeparam=$(echo "print($latticeparam * (1.00 - (0.01 * $j)))" |
           ↪  python)
38         echo $newlatticeparam
39         sed -i "2d" $dir/POSCAR
40         sed -i "2i $newlatticeparam" $dir/POSCAR
41         # also update the INCAR file IBRION tag
42         sed -i "3d" $dir/INCAR
43         sed -i "3i IBRION=-1" $dir/INCAR
44 done
45
46 # generate new kpoints based on updated POSCARs
47 # also running vasp interactively here since these calculations are so fast...
48 for d in $structname-*/; do
49         cd $d
50         ~/bin/kpoints.x
51         # this is for running interactively
52         #~/bin/vasp6_serial
53
54         # this is for running as seperate jobs
55         sbatch runjob.sh
56         echo "Job $d submitted"
57         cd ..
58 done
59
```

```
60 echo "All jobs submitted, wait for completion."
```

This is an example of a SLURM job script for a relaxation run of lithium in a Face Centered Cubic (FCC) lattice. The bash script runs this script for each of the batch jobs.

```
1  #!/bin/bash
2
3  #SBATCH --time=00:15:00   # walltime
4  #SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
5  #SBATCH --nodes=1   # number of nodes
6  #SBATCH --mem-per-cpu=4096M   # memory per CPU core
7
8
9  # Set the max number of threads to use for programs using OpenMP. Should be <=
   ↪  ppn. Does nothing if the program doesn't use OpenMP.
10 export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE
11
12 # LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
13
14 vasp6_serial
```

A.2 Python Scripts

This python script plots the energy vs. volume data received from the simulations:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Thu Jun  9 21:57:00 2022
5
6  @author: keaton
7  """
8  from matplotlib import pyplot as plt
9  import numpy as np
10 import pandas as pd
11 import altair as alt
12 from scipy.optimize import curve_fit
13
14 # atoms in lattice
```

```
cI16_atoms = 8
fcc_atoms = 1
oc88_atoms = 88

### both these quantities need to be per ion ###

# in eV
free_energy_cI16 = np.array([-5.76071198, -14.94661054, -14.04178384,
↪  -14.96733993, -13.07569301, -14.31603415, -8.96118289, -9.96961478,
↪  -14.39283052, -14.83343311, -6.48775741, -1.33737545, -14.59339995,
↪  -11.24107315, -14.77548444, -14.98944339, -13.51464566, -2.33795984,
↪  -12.55685694, -13.99655481, -14.97164714, -14.88768743, -14.08100655,
↪  -4.15864719, -13.70682722, -11.94846513, -14.70888726, -3.27704609,
↪  -14.62741058, -14.24276185, -4.98569255, -14.80771303, -14.92034558,
↪  -11.60804287, -14.65309909, -14.53596819, -14.31928642, -9.48276124,
↪  -14.76049736, -14.95032561, -14.99084672, -13.30445515, -12.26422677,
↪  -14.98504946, -14.85022279, -8.40303394, -14.16334851, -13.88201050,
↪  -7.80624451, -14.43280399, -10.42372953, -14.46304864, -14.88161733,
↪  -14.52999381, -10.84704214, -14.70694693, -14.91991233, -7.16892974,
↪  -14.18606114, -14.98115802, -12.82677793])
free_energy_bcc = np.array([-1.90129703, -1.12543580, -1.71517398, -1.90170208,
↪  -1.61128549, -1.75037602, -1.66427206, -1.88734750, -1.89688569, -1.21307937,
↪  -1.74696577, -1.71855250, -1.84180302, -1.90480916, -1.69869647, -1.87202129,
↪  -1.80310966, -1.29323023, -1.57481236, -1.85272426, -1.77848058, -0.24304845,
↪  -1.51952618, -1.81798768, -1.53829113, -1.36706839, -1.90390034, -1.77699657,
↪  -1.86281918, -0.55074867, -1.82400976, -1.54812169, -1.80491664, -1.55693872,
↪  -1.79124242, -0.81137812, -1.85803247, -1.43305428, -0.40364647, -1.59759530,
↪  -1.84238997, -1.68137575, -1.88021044, -0.68659482, -1.49339194, -1.87108210,
↪  -1.90385723, -1.59316861, -1.88170442, -1.03015280, -1.76222951, -1.64221680,
↪  -1.83012194, -1.73127110, -1.89052865, -1.89816701, -0.92559428, -1.62920445,
↪  -1.68271815, -1.64686780, -1.89334344])
free_energy_fcc = np.array([-1.82218418, -1.02434174, -1.88344029, -1.85706577,
↪  -1.63926991, -1.90551502, -1.86660133, -0.91947316, -1.84103586, -1.67945519,
↪  -1.87051619, -1.88166698, -1.90053114, -1.54504356, -0.54434882, -1.42922659,
↪  -1.89602815, -0.80514342, -1.59443753, -0.39643262, -1.89051212, -1.48995755,
↪  -0.68047808, -1.90215523, -1.80036894, -1.28878564, -1.77541731, -1.89736854,
↪  -0.23560541, -1.90371114, -1.84624746, -1.36235381, -1.89030055, -1.85683752,
↪  -1.12044481, -1.74730799, -1.20839758, -1.90640338, -1.87549594, -1.71536917,
↪  -1.90512431])
free_energy_oc88 = np.array([-323.99678726, 76.03653867, -43.21161243,
↪  -13.45311139, -66.21759986, -49.17646483, -319.47807871, -69.34366403,
↪  -65.57716280, -0.63232304, -24.64216112, -56.90697954, 67.93108541,
↪  -406.69271917, -543.92191157, -64.32428289, -69.91573974, 13.59370482,
↪  9.16771468, -52.57329215, -31.82446810, -68.66423870, 133.30040856,
↪  -38.58146008, -310.95772289, -67.33873947, 60.15504731, 126.08493651,
↪  -36.04827367, -62.42758162, 4.13386926, -55.55571820, -59.34896701,
↪  52.67115547, -528.93714408, 14.46071083, -33.35763977, -61.47402082,
↪  -531.04424424, -2655.35509235, -54.17711831, 118.86613675, -60.45020365,
↪  -471.60656837, -258.43369455, 101.65317169, -140.63364421, 126.89907614,
↪  -50.92877473, -502.40869192, -27.47875458, -68.26602799, 20.07382677,
↪  -3179.33654111, 110.35510273, -451.79950775, -40.96677307, -5.14531147,
↪  -67.82708170, -9.38552557, -462.11468218, -58.16939974, 84.38860900,
↪  38.68363826, -70.04708415, -63.30695183, -69.61953221, -2024.08725176,
↪  25.95599073, 45.51844715, -45.32354028, 92.95461507, -17.26502011,
↪  -66.80327023, -401.91836284, -47.30941650, 32.16914757, -69.02671845,
↪  -64.88115057, -491.13505807, -20.86907310])
```

```

# in Angstroms**3
volume_cI16 = np.array([7.85, 22.03, 14.29, 21.54, 12.52, 15.04, 9.16, 9.72, 27.91, 17.46, 8.10, 6.67, 26.22, 10.60, 17.04, 20.58, 13.21, 6.90, 11.86, 30.90, 19.20, 23.03, 30.29, 7.36, 13.57, 11.22, 25.12, 7.13, 16.22, 29.08, 7.60, 24.06, 18.32, 10.90, 25.67, 15.82, 28.49, 9.43, 24.59, 18.76, 20.12, 12.86, 11.53, 19.66, 23.54, 8.88, 29.68, 13.93, 8.62, 15.43, 10.01, 27.34, 17.88, 26.77, 10.30, 16.63, 22.53, 8.36, 14.66, 21.06, 12.19])
# bcc is not accessible for the ground state of lithium. the structure is hR9 instead. See Guillame et. al.
# volume_bcc = np.array([19.11, 9.27, 34.21, 21.54, 39.65, 13.83, 36.86, 23.50, 18.53, 9.63, 32.51, 13.37, 27.02, 20.30, 35.08, 24.87, 14.80, 10.01, 41.58, 26.29, 14.31, 6.96, 44.60, 28.52, 43.58, 10.39, 19.70, 30.88, 25.57, 7.58, 15.30, 11.61, 29.29, 42.57, 30.08, 8.23, 16.33, 10.79, 7.27, 12.03, 15.81, 35.96, 24.18, 7.90, 11.19, 16.86, 20.92, 40.61, 17.41, 8.91, 31.69, 12.47, 27.76, 33.35, 17.96, 22.18, 8.56, 38.71, 12.91, 37.78, 22.84])
volume_fcc = np.array([15.16, 8.83, 23.97, 16.19, 12.36, 20.73, 25.35, 8.49, 15.67, 12.80, 16.71, 17.25, 21.99, 11.51, 7.51, 10.69, 22.63, 8.15, 11.93, 7.20, 17.80, 11.09, 7.83, 18.94, 14.67, 9.92, 14.19, 18.36, 6.90, 21.35, 26.78, 10.30, 23.29, 26.06, 9.19, 13.71, 9.55, 20.12, 24.65, 13.25, 19.52])
volume_oc88 = np.array([2.28, 4.78, 10.20, 7.70, 15.94, 11.04, 3.44, 18.66, 15.58, 7.04, 8.40, 12.54, 4.95, 2.18, 1.57, 14.87, 19.49, 3.31, 6.62, 11.62, 8.89, 17.86, 3.72, 9.66, 2.49, 16.69, 5.12, 3.58, 9.40, 14.17, 6.83, 12.23, 13.18, 5.29, 2.38, 6.42, 9.14, 13.84, 1.65, 2.82, 11.92, 4.00, 13.50, 1.81, 2.60, 4.30, 2.94, 3.86, 11.33, 1.73, 8.64, 17.46, 6.22, 2.71, 4.15, 1.90, 9.93, 7.26, 17.07, 7.48, 1.99, 12.86, 4.62, 5.65, 19.91, 14.52, 19.07, 3.06, 6.03, 5.47, 10.47, 4.46, 7.93, 16.31, 2.09, 10.75, 5.84, 18.25, 15.22, 3.19, 8.16])

# trim the parts of the oc88 data that diverged when using DFT
oc88_data = pd.DataFrame(np.transpose([free_energy_oc88,volume_oc88]),columns = ["Energy","Volume"])
oc88_data = oc88_data[oc88_data["Volume"] > 3.6]

##################################################

### fit the data to the Birch-Murnaghan equation of state ###

x_vols = np.linspace(1,40,1000)

# first, define the BM eqn...
def BirchM(V, E_0, V_0, B_0, B_1, H):
    return E_0 + ((9*V_0*B_0)/16) * (((V_0/(V+H))**(2.0/3.0) - 1)**3.0 * B_1 + ((V_0/(V+H))**(2.0/3.0) - 1)**2.0 * (6 - 4*(V_0/(V+H))**(2.0/3.0)))

# fit the function for each structure
popt_fcc, pcov_fcc = curve_fit(BirchM, volume_fcc, free_energy_fcc/fcc_atoms, p0=[-1.9,23.9,0.10,3.87,3.24], maxfev=100000)
popt_cI16, pcov_cI16 = curve_fit(BirchM, volume_cI16, free_energy_cI16/cI16_atoms, p0=[-1.9,23.9,0.10,3.87,3.24], maxfev=100000)
# popt_oc88, pcov_oc88 = curve_fit(BirchM, volume_oc88, free_energy_oc88/oc88_atoms, p0=[-1.9,23.9,0.10,3.87,3.24], maxfev=1000000)
```

```
52 popt_oc88, pcov_oc88 = curve_fit(BirchM, oc88_data['Volume'],
   ↪ oc88_data['Energy']/oc88_atoms, p0=[-1.9,23.9,0.10,3.87,3.24],
   ↪ maxfev=1000000000)
53
54 ### Plotting section ###
55 fig, ax1 = plt.subplots(1,1)
56
57 # plot original data
58 ax1.scatter(volume_cI16, free_energy_cI16/cI16_atoms, color="b", label="cI16
   ↪ structure", marker="v")
59 ax1.scatter(volume_fcc, free_energy_fcc/fcc_atoms, color="r", label="fcc
   ↪ structure", marker="^")
60 ax1.scatter(oc88_data['Volume'], oc88_data['Energy']/oc88_atoms, color="g",
   ↪ label="oC88 structure")
61
62 # plot fit curves
63 ax1.plot(x_vols, BirchM(x_vols, *popt_fcc), "r--", label="Birch-Murnaghan fit -
   ↪ fcc")
64 ax1.plot(x_vols, BirchM(x_vols, *popt_cI16), "b--", label="Birch-Murnaghan fit -
   ↪ cI16")
65 ax1.plot(x_vols, BirchM(x_vols, *popt_oc88), "g--", label="Birch-Murnaghan fit -
   ↪ oC88")
66
67 # limits and labels
68 ax1.set_xlim(0,35)
69 ax1.set_ylim(-2,2)
70
71 ax1.set_xlabel("Volume (Angstroms^3)")
72 ax1.set_ylabel("Energy (eV)")
73 ax1.set_title("Lithium Simulation Results")
74 ax1.legend()
75
76 fig.show()
77 fig.savefig("./test.png", dpi='figure')
78
79
```

A.3 Julia Scripts

This script was written by Dr. Nelson. It uses a finite difference method to find the common line between the two structure curves.

```julia
using LinearAlgebra

function getTangentLine(data)
    y = data[begin:3]
    x = data[4:end]
    slope = (y[3] - y[1])/(x[3] - x[1])
    b  = y[2] - slope * x[2]

    return [slope,b]

end


# in eV
free_energy_cI16 = [-5.76071198, -14.94661054, -14.04178384, -14.96733993,
↪  -13.07569301, -14.31603415, -8.96118289, -9.96961478, -14.39283052,
↪  -14.83343311, -6.48775741, -1.33737545, -14.59339995, -11.24107315,
↪  -14.77548444, -14.98944339, -13.51464566, -2.33795984, -12.55685694,
↪  -13.99655481, -14.97164714, -14.88768743, -14.08100655, -4.15864719,
↪  -13.70682722, -11.94846513, -14.70888726, -3.27704609, -14.62741058,
↪  -14.24276185, -4.98569255, -14.80771303, -14.92034558, -11.60804287,
↪  -14.65309909, -14.53596819, -14.31928642, -9.48276124, -14.76049736,
↪  -14.95032561, -14.99084672, -13.30445515, -12.26422677, -14.98504946,
↪  -14.85022279, -8.40303394, -14.16334851, -13.88201050, -7.80624451,
↪  -14.43280399, -10.42372953, -14.46304864, -14.88161733, -14.52999381,
↪  -10.84704214, -14.70694693, -14.91991233, -7.16892974, -14.18606114,
↪  -14.98115802, -12.82677793]
free_energy_fcc = [-1.82218418, -1.02434174, -1.88344029, -1.85706577,
↪  -1.63926991, -1.90551502, -1.86660133, -0.91947316, -1.84103586, -1.67945519,
↪  -1.87051619, -1.88166698, -1.90053114, -1.54504356, -0.54434882, -1.42922659,
↪  -1.89602815, -0.80514342, -1.59443753, -0.39643262, -1.89051212, -1.48995755,
↪  -0.68047808, -1.90215523, -1.80036894, -1.28878564, -1.77541731, -1.89736854,
↪  -0.23560541, -1.90371114, -1.84624746, -1.36235381, -1.89030055, -1.85683752,
↪  -1.12044481, -1.74730799, -1.20839758, -1.90640338, -1.87549594, -1.71536917,
↪  -1.90512431]

# in Angstroms**3
volume_cI16 = [7.85, 22.03, 14.29, 21.54, 12.52, 15.04, 9.16, 9.72, 27.91, 17.46,
↪  8.10, 6.67, 26.22, 10.60, 17.04, 20.58, 13.21, 6.90, 11.86, 30.90, 19.20,
↪  23.03, 30.29, 7.36, 13.57, 11.22, 25.12, 7.13, 16.22, 29.08, 7.60, 24.06,
↪  18.32, 10.90, 25.67, 15.82, 28.49, 9.43, 24.59, 18.76, 20.12, 12.86, 11.53,
↪  19.66, 23.54, 8.88, 29.68, 13.93, 8.62, 15.43, 10.01, 27.34, 17.88, 26.77,
↪  10.30, 16.63, 22.53, 8.36, 14.66, 21.06, 12.19]
volume_fcc = [15.16, 8.83, 23.97, 16.19, 12.36, 20.73, 25.35, 8.49, 15.67, 12.80,
↪  16.71, 17.25, 21.99, 11.51, 7.51, 10.69, 22.63, 8.15, 11.93, 7.20, 17.80,
↪  11.09, 7.83, 18.94, 14.67, 9.92, 14.19, 18.36, 6.90, 21.35, 26.78, 10.30,
↪  23.29, 26.06, 9.19, 13.71, 9.55, 20.12, 24.65, 13.25, 19.52]

# Sort the data, but preserve the pairing
Vfcc = sort(volume_fcc)
```

```
fEfcc = free_energy_fcc[sortperm(volume_fcc)]

VcI16 = sort(volume_cI16)
fEcI16 = free_energy_cI16[sortperm(volume_cI16)]/8


# calculate tangent lines for each data pair.
tangentLinesFcc = [getTangentLine([fEfcc[i:i+2];Vfcc[i:i+2]]) for i=1:38]
tangentLinescI16 = [getTangentLine([fEcI16[i:i+2];VcI16[i:i+2]]) for i=1:57]

# Calculate matrix of norms to see which tangent lines are closest.
diffs = [norm(i - j) for i in tangentLinescI16, j in tangentLinesFcc]


display(argmin(diffs[begin:25,begin:25]))  # Find the tangent lines that are the
↪  closest match. This didn't produce the line I was hoping for so I just
↪  startered toying around with tangent lines and verifying visually.


using Plots

x = collect(0:0.1:30)
#plot(Pfcc,fEfcc[2:end-1],seriestype = :line)
#plot!(PcI16,fEcI16[2:end-1],seriestype = :line)
plot(volume_cI16, free_energy_cI16/8, seriestype = :scatter)
plot!(volume_fcc, free_energy_fcc, seriestype = :scatter)
plot!(x,tangentLinescI16[11][1] .* x .+ tangentLinescI16[11][2],ylim =
↪  (-2,0),xlim = (5,30))
plot!(x,tangentLinesFcc[8][1] .* x .+ tangentLinesFcc[8][2],ylim = (-2,0),xlim =
↪  (5,30))

display(tangentLinescI16[11][1]/(1e-10)^3 * 1.602e-19)
display(tangentLinesFcc[8][1]/(1e-10)^3 * 1.602e-19)
```

This next script generates the POSCAR file for the oC88 crystal structure. It does this by defining symmetry operators as matrices, then applies them to the Wyckoff positions specified in the vecs variable. Each row is a different vector specifying the position of a lithium atom.

```
### define lattice operators, and store glide plane translations as a fourth
↪  column since space group 39 is nonsymmorphic

C2x = [1 0 0 0
       0 -1 0 0
       0 0  -1 0]
```

```
C2z = [-1 0 0 0
       0 -1 0 0
       0 0  1 0]
C2y = [-1 0 0 0
       0 1 0 0
       0 0  -1 0]
σx = [-1 0 0 0
       0 1 0 0.5
       0 0 1 0]
σy = [1 0 0 0
      0 -1 0 0.5
      0 0 1 0]
σz = [1 0 0 0
      0 1 0 0.5
      0 0 -1 0 ]
I = [-1 0 0
      0 -1 0
      0 0 -1]
E = [1 0 0 0
     0 1 0 0
     0 0 1 0]
# conventional unit cell basis vectors
c_lVs = [8.512 0 0
        0 9.242 0
        0 0 8.313]

# primitive unit cell basis vectors (for the fractional atom positions)
p_lVs = [0 4.621 4.621
        0 -4.1565 4.1565
        8.3123 0 0]

# p_lVs = [8.512 0 0
#          0 4.621 4.621
#          0 -4.1565 4.1565]

# the positions of the atoms in fractional coords with primitive lattice vectors
vecs = [0.3832 0.1366 0.1804
        0.6398 0.8862 0.2295
        0.4023 0.6165 0.3411
        0.5352 0 0.5
        0.6958 0.75 0.0613
        0.4633 0.75 0.0266
        0.4351 0.9355 0.1182
        0.4706 0.75 0.6311
        0.2759 0.75 0.1870
        0.1129 0.8972 0.1287
        0.1334 0.1075 0.6443
        0.7434 0 0
        0.3008 0.0604 0.3845
        0.2759 0.75 0.5035
        0.5782 0.75 0.3917]
```

```
57
58  # store all our symmetry operators to loop over later
59  syms = [C2x, σy, σz, E]
60
61  keepvecs = zeros(44,3)
62
63  i = 0
64  for pos in eachrow(vecs)
65      println("new pos")
66      j = 0
67      for sym in syms
68          cart = c_lVs * pos  # Get vector to cartesian coords
69          newPos = sym[1:3,1:3] * cart  # Apply symmetry operator
70          direct = inv(c_lVs) * newPos  # Go back to direct coords
71          inCell = mod.(direct + sym[:,4],1) # Map back into cell
72          #println("symmetry operator")
73          #display(sym)
74          #println("before rotation (fractional)")
75          #display(pos)
76          #println("before rotation (cartesian)")
77          #display(cart)
78          #println("after rotation (cartesian)")
79          #display(newPos)
80          #println("After rotation(fractional)")
81          #display(inCell)
82
83          ### use this one if you want the conventional unit cell, it includes the
            ↪  glide plane translate
84          # inCellTwo = mod.(direct + sym[:,4] + [0.5, 0.5 , 0],1)  # Get the
            ↪  second translate ### Move this outside the loop
85
86          ### this one is for generating the primitive cell positions
87          inCellTwo = mod.(direct + sym[:,4],1)  # Get the second translate ###
            ↪  Move this outside the loop
88          keep = !any([all(isapprox.(inCell,j)) for j in eachrow(keepvecs[1:i,:])])
89
90          #println("should we keep it?")
91          #println(keep)
92          if keep
93              j += 1
94              global i += 1
95              keepvecs[i,:] = inCell
96          else
97              println("throw it out")
98          end
99
100         # keepTwo = !any([all(isapprox.(inCellTwo,j)) for j in
            ↪  eachrow(keepvecs[1:i,:])])
101         # if keepTwo
102         #     i += 1
103         #     keepvecs[i,:] = inCellTwo
```

```
        # else
        #     println("throw it out")
        # end
        # #keepvecs[i,:] = inCellTwo  # Add vector to list
        # #i+=1

    end
    println("kept this many:")
    println(j)
#    println("check")
#    println(i)
    #[]
    #println(size(unique(keepvecs, dims = 1)))
end

println(i)
# @assert i==88
@assert i==44

# keepvecs_cart = keepvecs * c_lVs

# # create a supercell here to compare distances

# using Distances
# R = pairwise(Euclidean(), keepvecs_cart, dims=1)

# R = R[1:i,1:i] # trim R so I don't get more than the atoms we keep

# testR = R[findall(x -> x < 1.77, R)] # this should be empty except for the trace

# ### need to implement some way to find the smallest distance off the trace of this matrix
# smallest_dist, sdist_idx = R[argmin(R)], argmin(R)
# #argmin(R), "\nMin distance: ", R[argmin(R)])

# using Plots
# #plotlyjs()
# heatmap(R, title="Shortest Distance: $smallest_dist A\n $sdist_idx", aspect_ratio=1)

### perform a change of basis to get the lattice in terms of primitive cell vectors
### Ax = By (conventional basis) (conventional positions) = (primitive basis) (primitive positions)
### y = B_inv(Ax)
pvecs = inv(p_lVs) * c_lVs * keepvecs'

# now we translate back into the unit cell again by clever mod like earlier
pvecs = mod.(pvecs,1)

```

```
### write the output to file for use in creating POSCAR
using DelimitedFiles
cd("/home/keaton/***SOME-PATH***/structures/oc88/")
writedlm("oC88.txt",pvecs')
display(pvecs[begin:25,:])
display(size(unique(pvecs,dims = 1)))
# mod(-0.1366,-1)


### these are for testing, showing how unique works and how sensitive isapprox()
↪  is

# A = [1 2 1
#      1 2 1
#      3 2 1]
# B = unique(A,dims = 1)
# display(B)

# b = [0.5352, 0.0, 0.5]
# c = [0.535200001,0.0,0.5]
# all(isapprox.(b,c))

# testx = [0.5;
#          0.5;
#          0.5]

# display(c_lVs * testx)
# display(p_lVs * testx)
```

Appendix B

Input Files

B.1 INCAR

This first INCAR file is for a typical relaxation run. With IBRION = 2 on line 3, this allows the atoms to update their positions due to the forces applied on them. The simulation runs until the positions update less than the default cutoff distance. More information on each of these tags can be found on the VASP wiki[10].

```
1 PREC=Accurate
2 ISIF=3
3 IBRION=2
4 SIGMA=0.1
5 EDIFFG=-.05
```

This next INCAR is for static runs, with IBRION=-1, meaning the positions of the atoms are not allowed to update.

```
1 PREC=Accurate
2 ISIF=3
3 IBRION=-1
4 SIGMA=0.1
5 EDIFFG=-.05
```

B.2 KPGEN

The KPGEN file gives instructions for how the kpoints.x script[9] should produce k-points (if some aren't readily supplied in a KPOINTS file)

```
KPDENSITY=1000
EPS=1e-10
```

KPDENSITY specifies the density of k-point placement.

Appendix C

Troubleshooting and Tips

C.1 Stack Size Issues

The most perplexing errors when setting up VASP came in the form of stack overflows. This is where memory is accessed that is outside what the operating system has allocated for the program. For large VASP jobs, the stack size may need to be increased. Many thanks to the staff at the Office of Research Computing at BYU for helping track down this issue.

Including the command `ulimit` in the job submission script returns what the node's stack size limit is.

`ulimit -s <LIMITSIZE>` sets the size limit for the stack. Be careful using ulimit -s unlimited, especially in a supercomputing environment. It can cause overflow issues that could cause instability in other people's jobs! VASP requires an unusually large stack size to do calculations. The support staff recommended doubling the stack size until the issues subside. These errors disappeared for this project at a 256MB stack size.

Appendix D

Job Statistics

The following linked csv file includes the time it took for each job to complete, along with when the job finished. The first 'Actual Walltime', 1-23:11:18, means the job took 1 day, 23 hours, 11 minutes, and 18 seconds to complete.

https://github.com/keatontate/senior-thesis/blob/main/job-statistics_senior-thesis.csv