Title
A Robust Approach to Lattice Thermal Conductivity

Author
Nielson, Weston Graham

Publication Date
2015

Peer reviewed|Thesis/dissertation

University of California
Los Angeles

A Robust Approach to Lattice Thermal Conductivity

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Materials Science and Engineering

by

Weston Graham Nielson

2015

Abstract of the Dissertation

A Robust Approach to Lattice Thermal Conductivity

by

Weston Graham Nielson

Doctor of Philosophy in Materials Science and Engineering

University of California, Los Angeles, 2015

Professor Vidvuds Ozoliņš, Chair

Thermal conductivity is a key parameter in designing high performance thermoelectric materials. A multitude of computational methods have been developed to calculate lattice thermal conductivity. Molecular dynamics (MD) based techniques, including equilibrium and non-equilibrium methods, in addition to non MD-based solutions, such as the Boltzmann Transport Equation (BTE), are all capable of calculating thermal conductivity, but each comes with different sets of limitations and difficulties. After extensive use of these different methods, we have developed a robust set of tools for obtaining high-quality lattice thermal conductivity values of crystalline solids. The crux of our method involves a novel compressive sensing (CS) based approach for efficiently calculating high quality force constants for crystalline materials. The result is a technique for building lattice dynamical models that can treat compounds with large, complex unit cells and strong anharmonicity, including those with harmonically unstable phonon modes.

The dissertation of Weston Graham Nielson is approved.

Dwight Streit

Sungtaek Ju

Bruce Dunn

Vidvuds Ozoliņš, Committee Chair

University of California, Los Angeles

2015

To my parents.

Table of Contents

1 Introduction 1

1 Thermoelectrics 2

1.1 History 2

1.2 Fundamentals 3

1.3 Applications 5

1.4 Challenges 7

1.5 Optimization 10

2 Motivation & Goals 14

2 Background 21

1 Atomic Interactions 21

1.1 Empirical Potentials 25

1.2 First Principles 28

2 Molecular Dynamics 28

2.1 Steps 29

2.2 Integration Schemes 34

2.3 Ensembles 35

3 Lattice Dynamics 38

4 Heat Transport & Phonons 41

5 Compressive Sensing 43

3 Methodology 49

1 Interatomic Force Constants 50
1.1 Density Functional Perturbation Theory 50
1.2 Compressivley Sensed Lattice Dynamics 51
2 Lattice Thermal Conductivity 56
2.1 Boltzmann Transport Equation 56
2.2 Lattice Molecular Dynamics 60
3 Example Systems 73
3.1 Si 74
3.2 NaCl 75
3.3 $Cu_{12}Sb_4S_{13}$ 78

4 Performance 86
1 Strategies & Hardware 86
2 Notation 91
3 Implementations 96
3.1 Atom Decomposition 96
3.2 Potential Decomposition 99
4 Results 104

5 Future Work 110
1 Automatic HNEMD 110

6 Conclusion 114

A LMD Package 115
1 Overview 115

2 Installation . . . 115
2.1 Optional Libraries . . . 116
3 Usage . . . 118
3.1 Commands . . . 119
3.2 Plugins . . . 122
4 Extending . . . 128
4.1 Commands . . . 128
4.2 Plugins . . . 131
5 Input Files . . . 135
5.1 `in.sim` . . . 135
5.2 `in.lat` . . . 137
5.3 `in.pot` . . . 138

B Time Correlation Functions . . . 139

C Sets & Lists . . . 143
1 Sets . . . 143
2 Lists . . . 145

Acknowledgments

First and foremost, I have to thank my advisor, Professor Vidvuds Ozoliņš. Without his guidance and patience this work would not have been possible. He always found a way to make time when I needed help and was consistently supportive of me exploring new ideas. He has been a fantastic mentor and working with him the past few years has truly been a pleasure.

I must also thank Dr. Fei Zhou for his tireless commitment to this project. Without his hard work, much of this would not have been possible. And to all of my group members, Chi-Ping Liu, Kyle Michel, Yi Xia, and Biljana Rolih—it has been a pleasure to work alongside all of you.

Thank you as well to my additional committee members, Professor Bruce Dunn, Professor Sungtaek Ju and Professor Dwight Streit.

To my girlfriend Mackenzie Kolling, thanks for always being supportive (and for proofreading!). Last but not least, thank you to my parents, Martie and Nancy Nielson. I would not have gotten this far without their support and continual encouragement.

Financial support for general method development were from the National Science Foundation under Award No. DMR-1106024. CSLD and LMD code development and calculations related to $Cu_{12}Sb_4S_{13}$ were performed as part of the Center for Revolutionary Materials for Solid State Energy Conversion, an Energy Frontier Research Center funded by the U.S. DOE, Office of Science, Basic Energy Sciences under Award No. DE-SC0001054. Additional funding was also provided by Office of Naval Research Award No. N00014-14-1-0444.

Vita

2009	B.S. (Major: Materials Science and Engineering.), University of California, Los Angeles.
2009-present	Graduate student researcher for Professor Vidvuds Ozoliņš, University of California, Los Angeles.

Publications

Zhou, F., Nielson, W., Xia, Y., and Ozoliņš, V. (2014). Lattice Anharmonicity and Thermal Conductivity from Compressive Sensing of First-Principles Calculations. Phys. Rev. Lett., 113, 185501.

Nielson, W., Zhou, F., and Ozoliņš, V. (2015). Optimizing Lattice Dynamics for Massively Parallel Heterogeneous Systems. Manuscript in preparation.

Zhou, F., Nielson, W., Xia, Y., and Ozoliņš, V. (2015). Compressive sensing lattice dynamics. I. General theory. Manuscript in preparation.

Nielson, W., and Ozoliņš, V. (2011, March). First-principles calculations of lattice stabilities in Mo. Presented at March meeting of the American Physical Society, Dallas, TX.

Nielson, W., Zhou, F., and Ozoliņš, V. (2013, March). Calculating Lattice Thermal Conductivity via Compressive Sensing Lattice Dynamics. Presented at March meeting of the American Physcial Society, Baltimore, MD.

Nielson, W., Zhou, F., and Ozoliņš, V. (2014, March). Beyond the Harmonic Approximation: Lattice Dynamics and Thermal Conductivity on Massively Parallel Heterogenous Systems. Presented at March meeting of the American Physcial Society, Denver, CO.

Nielson, W., Zhou, F., Xia, Y., and Ozoliņš, V. (2015, March). A Robust Approach to Lattice Thermal Conductivity. Presented at March meeting of the American Physcial Society, San Antonio, TX.

CHAPTER 1

Introduction

Roughly 70 percent of the world's electricity is generated from the combustion of fossil fuels [3]. Typically, this processes involves burning the fuel source to heat water, which is then converted to steam. This steam is then used to power a turbine which generates electricity. The most efficient turbines are able to achieve an efficiency of roughly 60% [13]–the other 40% is lost as heat. This results in roughly 15 terawatts of energy lost as wasted heat every year [10].

These inefficiencies are present in more places than just power plants. Many everyday devices that we rely on treat heat as an unwanted byproduct. Radiators in cars remove heat from the engine to keep it running cool while the exhaust system vents hot gasses out of the tail pipe. Fans in computers keep components from overheating by dissipating the undesirable heat. Air conditioning units keep the temperature inside our homes and buildings at just the right temperature by venting the heat outside. These are but a few examples of places in which heat is considered a nuisance.

If this waste heat could be recaptured and turned back into a useful product, we could not only increase the efficiency of many devices we already rely on, but a path would be paved for creating exciting new applications. This is where a relatively unknown class of materials, called *thermoelectrics*, come in. A thermoelectic material is capable of converting heat into electricity without any moving parts. While they were originally discovered almost 200 years, low conversion efficiencies have limited their use to niche applications. Within the last decade or so, a surge of interest in thermoelectrics have brought about the discovery of

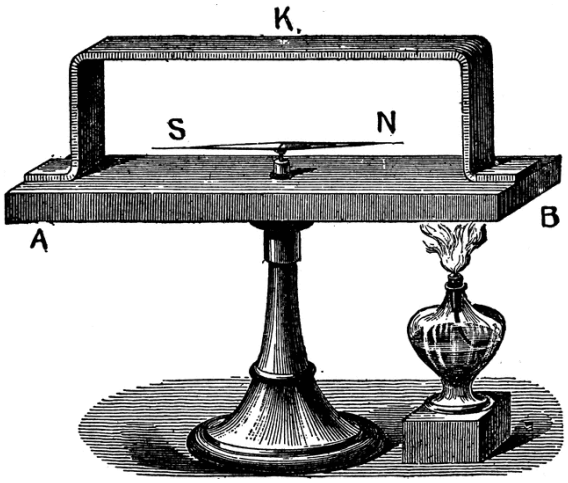


Figure 1.1: The original device created by Seebeck used to discover what would become known as thermoelectricity. The device consisted of two different metals (A-B and K), connected to form a loop. When one junction was heated (B), a magnetic field was induced, causing the compass (S-N) to orient itself accordingly. This phenomena, originally termed *thermomagnetism*, would later become known as *thermoelectricity*. (Image courtesy FCIT)

much more efficient materials, but the search is still on for even better materials.

1 Thermoelectrics

1.1 History

In 1820 a physicist by the name of Thomas Seebeck created a device that was capable of generating a magnetic field simply through the application of heat. His contraption was quite simple, involving just two different metals connected to form a loop (Figure 1.1). When heat was applied to one of the junctions between the two metals, a magnetic field was created. He observed this magnetic field by placing a compass in the middle of the loop which, upon application of heat, would orient the compass needle. Seebeck named this phenomena the thermomagnetic effect [22].

What Seebeck didn't fully realize at the time, however, was how the magnetic field was actually being created. Earlier that same year, a Danish scientist named Hans Christian

Ørsted (famous for *Oersted's Law*), had shown that an electric current could generate a magnetic field. Ørsted clarified how Seebeck's device worked and termed the phenomena thermoelectricity [22]. Today this effect it is still often referred to simply as the Seebeck effect.

A related thermoelectric effect, called the Peltier effect, was discovered a few years later by a Charles Peltier. Instead of applying heat to the junction of two dissimilar metals, Peltier applied an electric current. He found that, depending on the direction of the current, heat could be either absorbed or rejected by the junction.

1.2 Fundamentals

While Seebeck and Peltier discovered the occurence of these thermoelectric effects in their experiments with dissimilar conductors, the effects are also present in single conductors. If a metal wire, for example, is subjected to a temperature gradient by heating and cooling opposite ends, a small electric current will develop along the length of the wire. This current is produced by the flow of hotter electrons in one direction and cooler electrons in the other. The amount of hot and cold electrons flowing in either direction is not perfectly equal, resulting in a net flow of charge, and thus a potetial difference across the length of the wire [7]. This effect also occurs in the case when charge carries are positive (known as 'holes'). The ratio of induced potential (voltage) difference to temperature difference is known as the Seebeck coefficient, S, and is sometimes also referred to as *thermopower* (although modern literature typically refers to thermopower as σS^2). Mathematically, the Seebeck coefficient can be represented as

$$S = -\frac{\Delta V}{\Delta T}, \tag{1.1}$$

where ΔV is the voltage difference and ΔT is the temperature difference. While the term thermopower is often used, it is actually a misnomer, as the value S denotes is a difference in

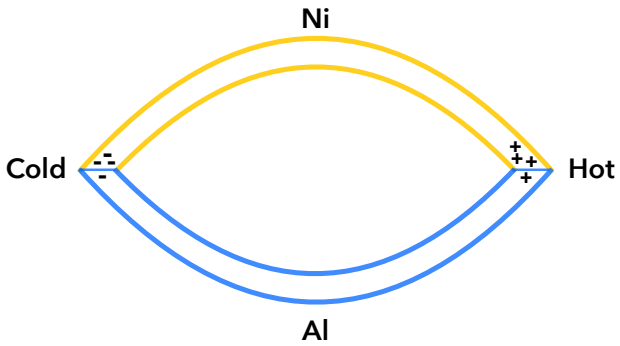


Figure 1.2: Junction between two dissimilar conductors, subjected to a temperature gradient.

voltage not power. Additionally, S is a function of temperature and thus should technically be written as $S = S(T)$. In order to calculate the voltage difference over a temperature gradient, one must perform the following integral

$$\Delta V = \int_{T_0}^{T_1} S(T) dT. \tag{1.2}$$

If we now consider an example along the lines of Seebeck's original experiment, we can see how the junction between dissimilar materials becomes important. An example junction between aluminum (Al) and nickel (Ni) is shown in Figure 1.2. Aluminum and nickel have different Seebeck coefficients, leading to a variation of charge build-up at both junctions. This means that if we want to calculate the voltage difference we cannot simply use Equation 1.2. The voltage between the two junctions is given as $\Delta V_{AB} = \Delta V_A - \Delta V_B$, which leads to a modified form of Equation 1.2

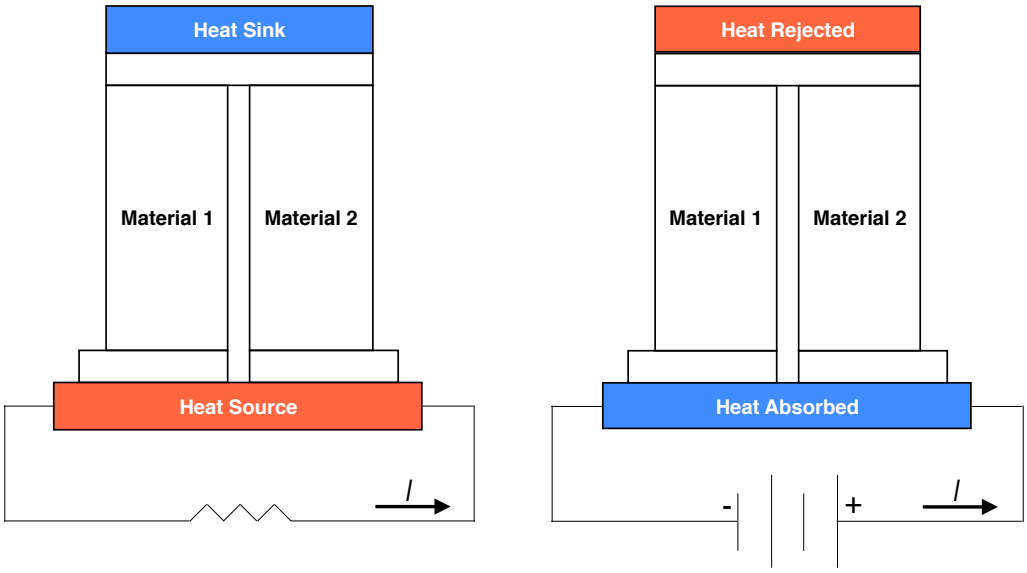
$$\Delta V_{AB} = \int_{T_0}^{T_1} \left(S_A(T) - S_B(T) \right) dT = \int_{T_0}^{T_1} S_{AB}(T) dT. \tag{1.3}$$

If the junction is instead operated as a Peltier device, with an applied current instead of a temperature gradient, the rate at which heat can be absorbed or rejected by the junction is

given by

$$Q_p = \left(\Pi_A - \Pi_B\right) IT, \tag{1.4}$$

where Π_A and Π_B are the Peltier coefficients for materials A and B, respectively, and I is the current from A to B. The Peltier coefficient is related to the Seebeck coefficient via $\Pi = TS$.



(a) (b)

Figure 1.3: Thermoelectric modules, operating as (a) a generator of electricity and (b) a Peltier cooling device.

1.3 Applications

Utilizing the Seebeck and Peltier effect in real devices leads to some very interesting and exciting applications. The simplest application of the Seebeck effect is in measuring temperature using a device known as a *thermocouple*. The basic design of a thermocouple is illustrated in Figure 1.4 and is actually very similar to that of the junction from the previous section (Figure 1.2). In this case T_S is the temperature that we wish to measure (or 'sense')

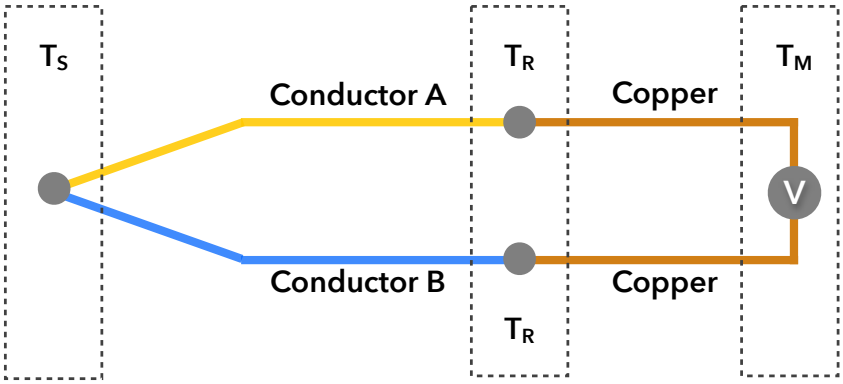


Figure 1.4: A typical thermocouple. T_S is the temperature that we wish to measure (or 'sense') and conductors A and B are composed of different types of materials. A specific combination of A and B will result in a different value of $S_{AB}(T)$.

and conductors A and B are composed of different types of materials. A specific combination of A and B will result in a different value of $S_{AB}(T)$, as we saw in Equation 1.3, the values of which have been tabulated for the most common combinations of A and B [1]. There are a variety of common choices for A and B. For example, a 'J type' thermocouple is constructed with A as iron and B as a copper-nickel alloy. Thermocouples are popular for measuring temperature because they are relatively inexpensive, have good sensitivity and can typically operate over a wide range of temperatures.

Another exciting application of the Seebeck effect is in solid-state energy generation, whereby electricity is generated from nothing more than a temperature difference. An example of a thermoelectric module designed to generate electricity is shown in Figure 1.3a.

This technique has been used in space exploration by NASA for decades in the form of so-called radioisotope thermoelectric generators (RTGs). RTGs operate by essentially encapsulating a radioactive material inside a thermoelectric cylinder. As the radioactive material decays it outputs heat which the thermoelectric modules convert into electricity. The typical half-life for many radioactive isotopes is on the order of decades, making it possible to generate electricity for many years. The RTGs used in the Voyager spacecrafts, launched in 1977, are still operational nearly 40 years later [2].

Thermoelectrics have also begun to find their way into more accessible applications that don't require a trip to space. A company called BioLite has developed a series of portable cooking stoves that harvest some of the excess heat generated by burning wood to generate electricity via integrated thermoelectric modules. This electricity is then used to power an electric fan which acts as a bellows causing the fire to burn hotter. As the fire burns hotter, it also outputs less smoke, making it far safer to use in enclosed environments. Excess electricity from the stove can also be used to charge batteries for devices such as lights and mobile phones. When one considers that over a billion people still rely on wooden stoves for cooking and do not have access to reliable energy sources, it is clear that devices like this have the ability to vastly change lives.

Even more recently, Alphabet Energy has begun producing large, shipping-container size thermoelectric modules capable of producing up to 25 kW of electricity. The material used is an earth abundant material known as tetrahedrite, which was originally identified as a promising thermoelectric material via a collaboration with Michigan State University, UCLA and MSU [14].

The Peltier effect can be put to use in solid-state heating and cooling, offering the potential of replacing cumbersome mechanical cooling techniques such as condensers and compressors. It has been used to create a variety of interesting devices such as light-weight, portable refrigerators and high-performance CPU heat-sinks. An example of a thermoelectric operating as a Peltier cooler is illustraed in Figure 1.3b.

1.4 Challenges

Thermoelectric devices are heat engines and they are bound by the laws of thermodynamics. If we consider a thermoelectric module operating in generation mode without any heat loss

(an ideal case), then determining the efficiency is straight forward

$$\eta = \frac{\text{energy supplied}}{\text{heat absorbed}}. \tag{1.5}$$

Another useful metric, the thermoelectric *figure of merit*, ZT, is a dimensionless parameter the defines how well a material will operate as a thermoelectric. Since it is directly related to the efficiency, as we will see later, the larger the value of ZT, the better a material will perform as a thermoelectric. Mathematically, it is given as

$$ZT = \frac{S^2 \sigma T}{\kappa}, \tag{1.6}$$

where S is the material's Seebeck coefficient, σ is the electrical conductivity and κ is the thermal conductivity. While this equation is relatively simple, it highlights one of the most difficult issues facing the field of thermoelectric materials–conductivity. Many common materials exhibit either conducting or insulating properties. Most metals, for example, are good conductors of both electricity and heat, while other materials, such as wood, plastic, glass and ceramics are typically good insulators. However, what Equation 1.6 tells us is that the best thermoelectrics are good electrical conductors but poor thermal conductors.

The nature behind electrical conductivity is relatively straightforward as it is due primarily to how easily electrons and holes can move through a material. Materials with many free charge carriers, such as metals and heavily doped semiconductors, have good electrical conductivity. Materials like ceramics, however, do not have many free electrons and are thus poor electrical conductors.[1] Thermal conductivity is somewhat more complicated though because heat is transfered through a variety of mechanisms, and typically thermal conductivity occurs via two mechanisms. The first contribution is from electrons and this is why most metals are so

[1]The specifics of bonding in materials is discussed in more detail in the following chapter. For now we keep the discussion relatively high-level.

good at conducting both heat and electricity. The second contribution comes from the way that the atoms in a material vibrate.[2] Mathematically, thermal conductivity is generally written as

$$\kappa = \kappa_e + \kappa_L, \tag{1.7}$$

where κ_e and κ_L are the thermal conductivities due to electrons and atomic vibrations, respectively. We can then rewrite Equation 1.6 as

$$ZT = \frac{S^2 \sigma T}{\kappa_e + \kappa_L}. \tag{1.8}$$

This equation illustrates the key parameters that must be optimized in order to increase ZT and will be discussed later in more detail.

The maximum efficiency of a thermoelectric can also be written in terms of ZT as

$$\eta_{\max} = \frac{\Delta T}{T_H} \frac{\sqrt{1+ZT}-1}{\sqrt{1+ZT}+1}. \tag{1.9}$$

Equation 1.9 is plotted for a range of temperature differences in Figure 1.5, where T_C and T_H are the cold and hot temperatures, respectively, and $\Delta T/T_H$ is the Carnot efficiency. This plot highlights a couple of important issues regarding thermoelectrics. The first is that the efficiency is heavily dictated by the temperature difference across the material, the upper bound of which is set by the Carnot efficiency. The other is that thermoelectric efficiencies are typically quite low. The best turbine power plants attain efficiencies in the range of 40-50% [23], which can only be matched by a thermoelectric with a ZT between 10 to 20–an extremely ambitious value.[3] As a result, thermoelectrics are more immediately attractive

[2]For clarity, we skip mentioning phonons at this point, but will come back to this in more detail later.
[3]Assuming $T_C/T_H = 0.25$

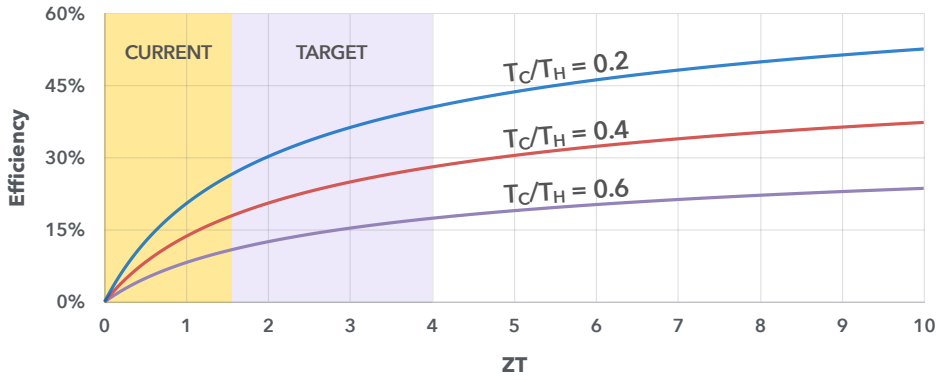


Figure 1.5: Efficiency versus figure of merit, ZT, for a range of temperature differences. The best thermoelectrics currently attain $ZT \sim 1$, but an economically viable material with ZT of 3-4 would be breakthrough.

as a means to increase the overall efficiency of other processes, or in applications in which relatively low conversion efficiencies are acceptable.

1.5 Optimization

As discussed in the previous section, thermoelectrics with high ZT values are extremely desirable. Over the past decades, a variety of different approaches have been taken in an attempt to create high-ZT materials. Recall from Equation 1.8 that ZT can be maximized by increasing the power factor, $S^2\sigma$, and decreasing the thermal conductivity κ. One of the most straightforward approaches to optimizing ZT is to start with a low thermal conductivity material and then increase it's electrical conductivity, σ, through the addition of charge carriers (a process known as 'doping'). There is, however, an interdependence between S, σ and κ that makes this a somewhat tricky process. For metals and heavily doped (degenerate) semiconductors, the Seebeck coefficient can be written as [20]

$$S = \frac{8\pi^2 k_B^2}{3eh^2} m^* T \left(\frac{\pi}{3n}\right)^{2/3}, \tag{1.10}$$

where n is the carrier concentration and m^* is the effective mass of the carrier. The carrier

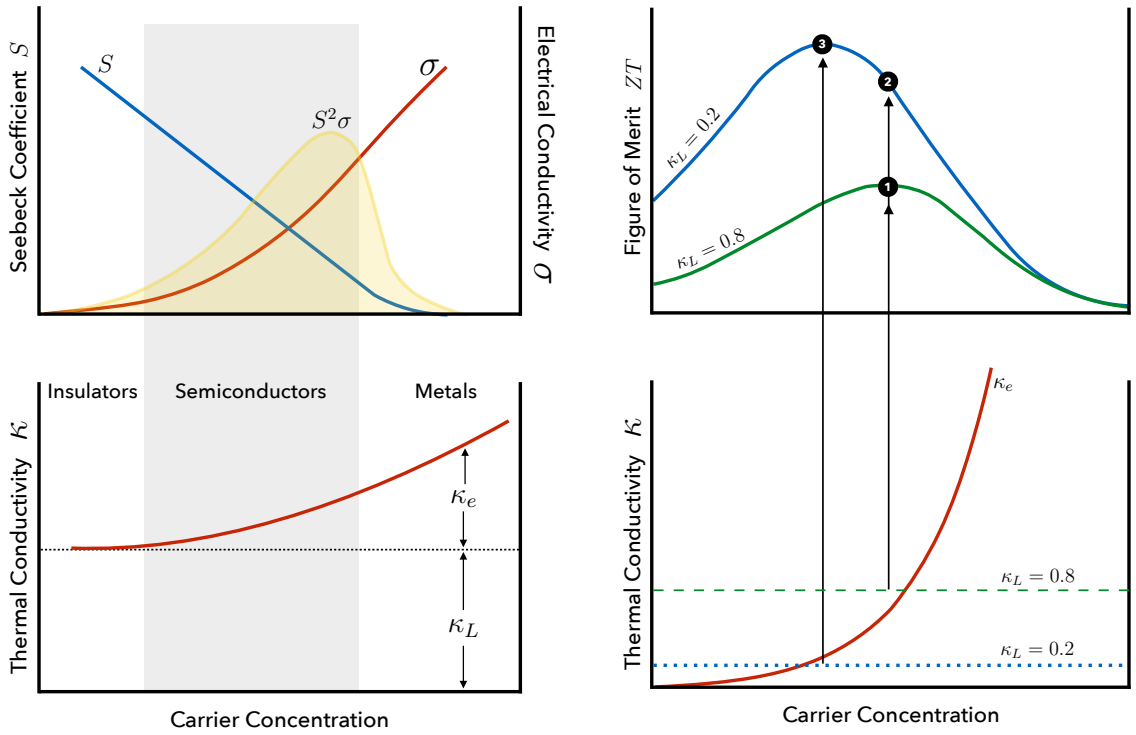


Figure 1.6: Optimization of ZT with respect to carrier concentration. (a) Heavily doped semiconductors typically make good thermoelectric materials. Maximizing ZT requires a compromise between competing properties. Adapted from [19]. (b) Example of ZT versus carrier concentration and thermal conductivity in Be_2Te_3. The electronic thermal conductivity, κ_e, (red line) is a function of carrier concentration. For $\kappa_L = 0.8$ the maximum ZT is at point 1. By decreasing κ_L, ZT is increased to point 2. Further decreasing the carrier concentration leads to additional increase in ZT shown at point 3. Adapted from [20].

concentration is related to the electrical conductivity, σ, via

$$\sigma = ne\mu, \tag{1.11}$$

where μ is the mobility of the carrier. These equations show that a material with low carrier concentration yields a high Seebeck coefficient but also a low electrical conductivity–S and σ have an inverse relationship. The interdependence between these properties is plotted in Figure 1.6. Figure 1.6a highlights the inverse relationship between Seebeck coefficient S and electrical conductivity σ, giving rise to a maximum power factor, $S^2\sigma$, somewhere

in between. Additionally, as the carrier concentration increases, so does the total thermal conductivity, κ, due to the increase in κ_e given by the Wiedemann–Franz law [20]

$$\kappa_e = ne\mu LT = \sigma LT, \tag{1.12}$$

where L is the Lorentz factor.

Figure 1.6b illustrates the interdependence between carrier concentration, thermal conductivity and ZT. The red curve is the electronic thermal conductivity, κ_e, which, from Equation 1.12, is a function of carrier concentration. For a given lattice thermal conductivity $\kappa_L = 0.8$, the maximum ZT is shown at point 1. Decreasing κ_L, but maintaining carrier concentration, leads to an increased ZT shown at point 2. The maximum ZT, point 3, is reached by further decreasing the carrier concentration. This highlights a fundamental point surrounding thermoelectrics–large gains in ZT can be realized simply by reducing κ_L, while the electrical properties can be fine tuned through adjustment of the carrier concentration. This understanding has led to much research into ways to reduce κ_L.

1.5.1 Reducing κ_L

To begin to understand how to lower κ_L, we must first briefly look at how heat is transported through crystalline solids. At the most basic level, heat is transported through crystals by the vibrations of the atoms that comprise the solid. The degree to which a given atom vibrates is proportional to the amount of energy it has. As heat is a form of energy, the heating of an atom will increase the amount by which it vibrates. Additionally, the vibration of a single atom affects, and is affected by, the vibrations of it's neighboring atoms. As a result, if heat is applied to one side of a solid, it will propagate through the material via the vibrations and interactions between the atoms. These vibrations are often described as waves, just as waves can be sent along a stretched elastic string by plucking one end [15]. These waves, or lattice vibrations, in solids are typically referred to as *phonons*.

The above description is overly simplistic, but is useful as a starting point because it allows us to discuss heat transport through the context of phonons which, like particles, can be scattered and deflected. In the absence of any scattering or deflection, a phonon will propagate in the same direction with a constant velocity, v. The amount of time that a phonon can travel unimpeded is called the phonon lifetime, τ. We can write the lattice thermal conductivity in terms of the phonon lifetime as [15]

$$\kappa_L = \frac{1}{3} C_v v^2 \tau, \tag{1.13}$$

where C_v is the specific heat, which is a function of temperature. Equation 1.13 is useful because it shows that κ_L is proportional to phonon lifetime. The more phonon scattering that occurs, the smaller the phonon lifetime, and consequently the lower κ_L is. Therefore, many attempts to reduce κ_L focus on ways to increase phonon scattering.

Disorder One approach to reducing κ_L is through alloying and the introduction of impurities, such as interstitials (atoms in off-lattice sites) and vacancies (missing atoms in the lattice), which results in disorder within the lattice. For example, alloys in the binary tellurides, such as Be_2Te_3, Sb_2Te_3, PbTe and GeTe, have attracted much interest and in some cases have resulted in significant reductions of κ_L [20]. Structures with large void spaces, such as skutterudites and clathrates, are also interesting as their voids can be filled with impurity atoms. Cases such as $CoSb_3$ [12] and $Sr_8Ga_{16}Ge_{30}$ [16] have been shown to exhibit extremely low κ_L, approaching that of κ_{min} [4].

Complexity Materials with complex atomic compositions are also typically considered to be good candidates for low κ_L, while simple crystals tend to have relatively large κ_L. Simple systems such as diamond (carbon) and pure silicon, for example, both have room temperature $\kappa_L > 100$ W m^{-1}K^{-1}, yet materials such as Ag_9TlTe_5 [11], $Yb_{14}MnSb_{11}$ [5] and

[4]The theoretical lower limit to thermal conductivity at room temperature is $\kappa_{\text{min}} \sim 0.2$ W m^{-1}K^{-1} [6]

$Cu_{12}Sb_4S_{13}$ [14] all exhibit extremely low thermal conductivities of $\kappa_L < 1$ W m^{-1}K^{-1} at room temperature. Complexity, in this case, does not necessarily require complex symmetry either, as $Cu_{12}Sb_4S_{13}$ exhibits cubic symmetry. While complexity does not guarantee low κ_L, it is generally a good indicator.

Nanostructuring Some of the most recent research on thermoelectrics has focused on the possibility of using thin-films and nano-wires for creating low κ_L, high ZT materials. One proposed topology involves building up a larger material from a series of smaller layers stacked together, forming what is typically referred to as a 'superlattice'. For example, Bi_2Te_3-Sb_2Te_3 thin-film superlattice materials have been shown to exhibit ultralow $\kappa_L \sim 0.3$ W m^{-1}K^{-1}, while retaining the electrical properties of the bulk material [21]. Boundary scattering between the thin-film layers, however, can lead to significant reduction in electrical conductivities resulting in poor ZT [20] and continued research is necessary to better understand the design, preparation and optimization of nanostructured thermoelectrics [8].

2 Motivation & Goals

Ultimately, the motivation of this work is to aid in the discovery of promising new thermoelectric materials. The key parameter in this search, as we've seen, is thermal conductivity. More specifically, materials with *low* κ_L are of great interest. Somewhat surprisingly, however, we know very little about the thermal conductivities of most materials. Consider for example, that of the roughly 100,000 known inorganic compounds, κ_L has only been published for roughly 1%. Calculating the thermal conductivity of the remaining 99% presents a formidable challenge, but also provides hope that better thermoelectric materials may lay yet undiscovered.

Much of the progress made in this field to date has been achieved through experimental

studies, but there are some inherent limitations to this approach going forward. Most importantly, examining materials on a case-by-case basis in a laboratory is costly, both in terms of man-hours and economic investment in materials and machinery. Searching for the ideal thermoelectric this way is a bit like trying to find a needle in a haystack–not something you'd want to do by hand, especially considering how large the haystack is. Indeed, this seems like a task ideally suited for a computer.

Fortunately, there has been recent progress in developing computational techniques to help in the search for low κ_L materials. Molecular dynamics (MD) based methods have proven to be extremely popular due to their relative ease of use and widespread availability in the form of freely available computer codes [18]. The development of mathematically rigorous methods for calculating κ_L from MD simulations has also added to the popularity of this approach. However, the poor predictive capability of *empirical* MD codes has restricted this method to studying only the simplest of systems.[5] While much higher accuracy (i.e. non-empirical) MD is possible,[6] it is currently far too computationally expensive to use for the purposes of calculating κ_L.

An alternative approach to determining κ_L involves solving the Boltzmann transport equation (BTE) [17]. Results published recently have shown that the BTE approach is able to very accurately predict κ_L for a range of systems, including Si and Ge [4], diamond [24] and Bi_2Te_3 [9]. In it's current form, however, this processes is still quite cumbersome and a significant amount of manual involvement is typically required, limiting it's applicability in the search for new low κ_L materials.

The goal of this project is to help fill in the voids where current computational techniques stop. What is presented here is a method for obtaining high quality predictions of κ_L that is both efficient and robust, requiring minimal intervention and no emotional involvement[7].

[5] This issue is discussed in more detail in Chapter 2.

[6] Specifically, density functional (DFT) based *ab initio* MD (AIMD).

[7] By this we mean that the methods presented here do not require continual tweaking of parameters in an attempt to get desirable results. By removing 'emotion' from the equation, one can simply use the tools provided to get consistent, reliable, repeatable results.

The hope is that these techniques can help accelerate the search and aid in the discovery of exciting new thermoelectric materials.

Bibliography

[1] Nist its-90 thermocouple database. URL http://srdata.nist.gov/its90/main/.

[2] Voyager 1 & 2. URL https://solarsystem.nasa.gov/rps/voyager.cfm.

[3] U.S. Energy Information Administration. International energy statistics. URL http://www.eia.gov/.

[4] D. A. Broido, M. Malorny, G. Birner, Natalio Mingo, and D. A. Stewart. Intrinsic lattice thermal conductivity of semiconductors from first principles. *Applied Physics Letters*, 91(23):231922, 2007. doi: http://dx.doi.org/10.1063/1.2822891. URL http://scitation.aip.org/content/aip/journal/apl/91/23/10.1063/1.2822891.

[5] Shawna R. Brown, Susan M. Kauzlarich, Franck Gascoin, and G. Jeffrey Snyder. $Yb_{14}mnsb_{11}$: New high efficiency thermoelectric material for power generation. *Chemistry of Materials*, 18(7):1873–1877, 2006. doi: 10.1021/cm060261t. URL http://dx.doi.org/10.1021/cm060261t.

[6] D G Cahill, S K Watson, and R O Pohl. Lower Limit to the Thermal-Conductivity of Disordered Crystals. *Phys. Rev. B*, 46(10):6131–6140, 1992.

[7] R G Chambers. Thermoelectric effects and contact potentials (for teachers). *Physics Education*, 12(6):374, 1977. URL http://stacks.iop.org/0031-9120/12/i=6/a=006.

[8] Zhi-Gang Chen, Guang Han, Lei Yang, Lina Cheng, and Jin Zou. Nanostructured thermoelectric materials: Current research and future challenge. *Progress in Natural Science: Materials International*, 22(6):535 – 549, 2012. ISSN 1002-0071. doi: http://

dx.doi.org/10.1016/j.pnsc.2012.11.011. URL http://www.sciencedirect.com/science/article/pii/S1002007112001384.

[9] Olle Hellman and David A. Broido. Phonon thermal transport in bi_2te_3 from first principles. *Phys. Rev. B*, 90:134309, Oct 2014. doi: 10.1103/PhysRevB.90.134309. URL http://link.aps.org/doi/10.1103/PhysRevB.90.134309.

[10] Allon I. Hochbaum, Renkun Chen, Raul Diaz Delgado, Wenjie Liang, Erik C. Garnett, Mark Najarian, Arun Majumdar, and Peidong Yang. Enhanced thermoelectric performance of rough silicon nanowires. *Nature*, 451(7175):163–167, 01 2008. URL http://dx.doi.org/10.1038/nature06381.

[11] Ken Kurosaki, Atsuko Kosuga, Hiroaki Muta, Masayoshi Uno, and Shinsuke Yamanaka. Ag9tlte5: A high-performance thermoelectric bulk material with extremely low thermal conductivity. *Applied Physics Letters*, 87(6):061919, 2005. doi: http://dx.doi.org/10.1063/1.2009828. URL http://scitation.aip.org/content/aip/journal/apl/87/6/10.1063/1.2009828.

[12] Ken Kurosaki, Guanghe Li, Yuji Ohishi, Hiroaki Muta, and Shinsuke Yamanaka. Enhancement of thermoelectric efficiency of cosb(3)-based skutterudites by double filling with k and tl. *Frontiers in Chemistry*, 2:84, 2014. doi: 10.3389/fchem.2014.00084. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4195357/.

[13] Lee S. Langston. Efficiency by the numbers. *MEMagazine*, 2009.

[14] Xu Lu, Donald T. Morelli, Yi Xia, Fei Zhou, Vidvuds Ozolins, Hang Chi, Xiaoyuan Zhou, and Ctirad Uher. High performance thermoelectricity in earth-abundant compounds based on natural mineral tetrahedrites. *Advanced Energy Materials*, 3(3):342–348, 2013. ISSN 1614-6840. doi: 10.1002/aenm.201200650. URL http://dx.doi.org/10.1002/aenm.201200650.

[15] N. Mermin N. Ashcroft. *Solid State Physics*. Harcourt, 1976.

[16] George S. Nolas, Joe Poon, and Mercouri Kanatzidis. Recent developments in bulk thermoelectric materials. *MRS Bulletin*, 31:199–205, 3 2006. ISSN 1938-1425. doi: 10.1557/mrs2006.45. URL http://journals.cambridge.org/article_S0883769400009921.

[17] M Omini and A Sparavigna. An iterative approach to the phonon boltzmann equation in the theory of thermal conductivity. *Physica B: Condensed Matter*, 212(2): 101 – 112, 1995. ISSN 0921-4526. doi: http://dx.doi.org/10.1016/0921-4526(95)00016-3. URL http://www.sciencedirect.com/science/article/pii/0921452695000163.

[18] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1 – 19, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1039. URL http://www.sciencedirect.com/science/article/pii/S002199918571039X.

[19] D.M. Rowe, editor. *Thermoelectrics Handbook: Macro to Nano*. CRC Press, 2005.

[20] G. Jeffrey Snyder and Eric S. Toberer. Complex thermoelectric materials. *Nat Mater*, 7(2):105–114, 02 2008. URL http://dx.doi.org/10.1038/nmat2090.

[21] M. N. Touzelbaev, P. Zhou, R. Venkatasubramanian, and K. E. Goodson. Thermal characterization of bi2te3/sb2te3 superlattices. *Journal of Applied Physics*, 90(2):763–767, 2001. doi: http://dx.doi.org/10.1063/1.1374458. URL http://scitation.aip.org/content/aip/journal/jap/90/2/10.1063/1.1374458.

[22] Enn Velmre. Thomas johann seebeck (1770–1831). *Proc. Estonian Acad. Sci. Eng.*, 2007.

[23] Cronin B. Vining. An inconvenient truth about thermoelectrics. *Nat Mater*, 8(2):83–85, 02 2009. URL http://dx.doi.org/10.1038/nmat2361.

[24] A. Ward, D. A. Broido, Derek A. Stewart, and G. Deinzer. *Ab initio* theory of the lattice thermal conductivity in diamond. *Phys. Rev. B*, 80:125203, Sep 2009. doi: 10.1103/PhysRevB.80.125203. URL http://link.aps.org/doi/10.1103/PhysRevB.80.125203.

CHAPTER 2

Background

A variety of computational techniques already exist for calculating thermal conductivity. Before jumping straight into how our approach is different, there are a number of topics that must be, at least partially, covered. In particular, there are a few questions that must be answered, namely

1. How are forces between atoms calculated?
2. How can a system of many atoms be evolved through time?
3. How is heat transported through solids?

We will begin answering these questions by looking at various ways in which atomic interactions are typically modeled. By understanding how this is done, we can highlight some of the strengths and weaknesses of the current techniques, which will provide some insight into ways in which improvements can be made. From here we will talk about how to *simulate* interactions between atoms using a technique called molecular dynamics (MD). Additionally, we will touch on some of the basic physics and mathematical concepts that are key to the approaches presented later on.

1 Atomic Interactions

Inorganic solids can generally be classified as either amorphous or crystalline materials. The difference between these two distinctions lays with the way in which the material's atoms

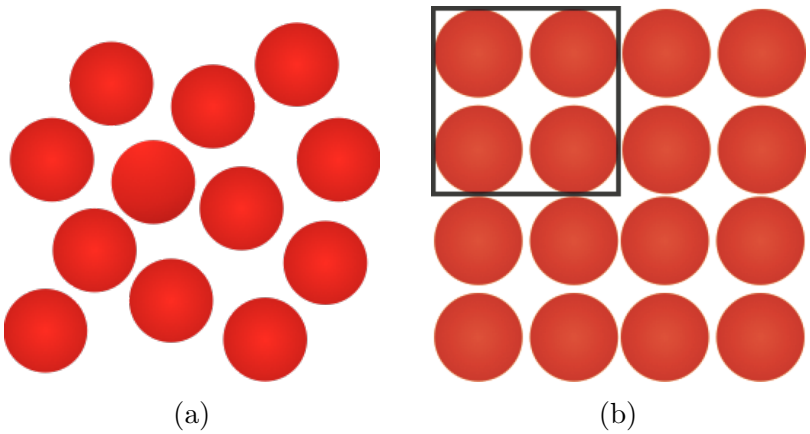


Figure 2.1: Two types of microstructures found in solids, (a) amorphous and (b) crystalline. The black box in the crystalline example denotes the unit cell.

are arranged. In amorphous materials, the atoms are arranged in such a way that there is no discernible repeating pattern, often referred to as a lack of long-range order, and if one could see the individual atoms they would appear randomly distributed. Typical examples of amorphous materials include glasses and polymers. Unlike amorphous materials, crystalline solids contain highly-structured, repeating patterns that extend in all three spatial dimensions. Using the same terminology as before, crystalline solids exhibit long-range order. The smallest repeating unit in crystalline materials is referred to as the unit cell. Common crystalline materials include metals and ceramics. Figure 2.1 illustrates the difference between the two types of materials.

While amorphous materials can exhibit very interesting properties, their lack of periodic, ordered structure makes them difficult to study from a theoretical standpoint. Conversely, the predictable, repeated structure of crystalline materials means that there is a great deal that can be understood simply by knowing the static positions of the atoms. For example, bulk modulus, elastic constants, optical properties and electronic structure are a few examples of properties that can be determined from static crystal analysis [5]. It is this predictability that makes crystalline materials particularly attractive and from here on out we will be concerned solely with crystalline solids.

While a range of useful material properties can be determined by the static position of atoms, there are many important properties that cannot be understood without taking into account the movement of the individual atoms. These dynamic properties include thermal properties, such as heat capacity and thermal expansion, phase transitions and transport properties, such as diffusion and thermal conductivity, to name but a few [5]. Therefore, to be able to determine such properties, it is important to understand how atoms move and interact with each other. The simplest way to model the movement of an atom is via a classical approach of Newton's second law, namely $\mathbf{F} = m\mathbf{a}$; force equals mass times acceleration. If the position of atom i at time t is given by $\mathbf{r}_i(t)$, then we can write the atom's acceleration as

$$\mathbf{a}_i(t) = \frac{\partial^2 \mathbf{r}_i(t)}{\partial t^2} = \frac{\mathbf{F}_i}{m_i}, \tag{2.1}$$

where m_i is the mass of the atom. It is also necessary to determine the force acting on the atom, which we can write in terms of the atom's potential energy, ϕ_i, thusly

$$\mathbf{F}_i = \nabla \phi_i \left(\mathbf{r}_i \left(t \right) \right). \tag{2.2}$$

For a system of N atoms, the potential energy can then be written as a function of the separation between atom i and all other atoms in the system

$$\phi_i = \sum_{j}^{N} \phi_{ij}(\mathbf{r}_{ij}), \tag{2.3}$$

where $\mathbf{r}_{ij}$ is the separation between atoms i and j, and ϕ_{ij} is the potential energy between

the two atoms.[1] The question now becomes, what is the form of ϕ_{ij}? As it turns out, this is not a simple question to answer and is one that has been the focus of scientists for the past century.

The simplest way to begin answering this question is to consider the interaction between two atoms. If the atoms are sufficiently far away from one another, their interaction is negligible. However, as they are brought closer together, they will feel two different types of forces, namely an attractive force and a repulsive force. In ionically bonded materials, this attraction is due to the positive and negative ions, in covalent solids the attraction is due to sharing of electrons and in metals the negative "sea of electrons" attracts the positive ion cores. In addition to these primary types of bonding, there is also the secondary (weaker) "Van der Waals" interactions which occurs in all materials and is the prevalent bonding mechanism in atoms with filled shells. Whatever the bonding mechanism, the attractive force that keeps the material together is also coupled with a repulsive force that stops the atoms from getting too close to one another. This repulsive force arises from the overlapping of core electrons as two atoms get closer together. We can write this attractive-repulsive relationship mathematically as

$$F = F_A + F_R, \tag{2.4}$$

where F is the total force and F_A and F_R are the attractive and repulsive forces, respectively. To continue keeping the discussion simple, for now, let us consider the case of electrically neutral Argon. Here, the attractive bonding will be influenced almost exclusively by "Van der Waals" interactions. The mathematical form for this interaction, the derivation of which can be found elsewhere [5], is typically written as Ar^{-6}, where A is a material constant and r is the atomic separation. The repulsive force is also often modeled as Br^{-12}, where again B

[1] It should be noted that Equation 2.3 is a simple pair-potential that excludes many-body interactions. Later on we will discuss the importance of many-body interactions.

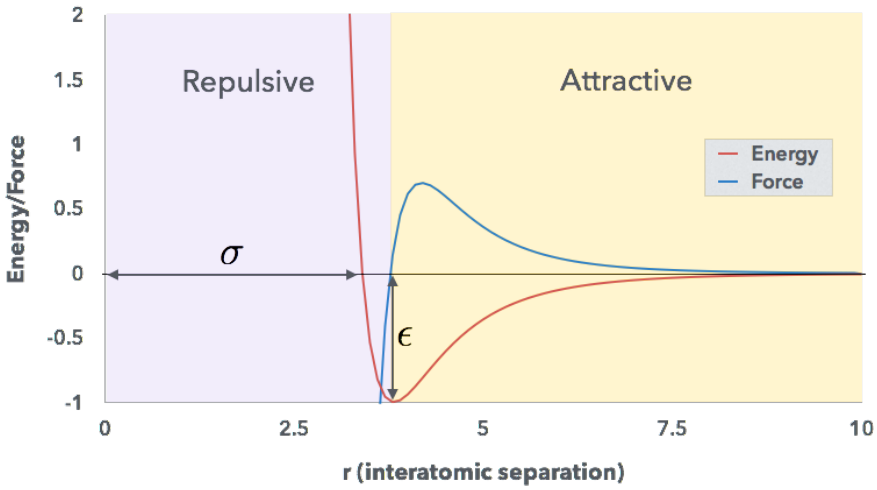


Figure 2.2: Lennard-Jones potential plotted as a function of atomic separation. At large separations there is an attractive force, while at short separations there is a respulsive force. The values ϵ and σ are used to control the equilibrium separation distance.

is a constant. The combination of these two yields the canonical "Lennard-Jones potential"

$$\phi_{ij}(r_{ij}) = -4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{6} - \left(\frac{\sigma}{r_{ij}} \right)^{12} \right], \tag{2.5}$$

where ϵ is the potential energy when the two atoms are at an equilibrium separation and σ is the distance between the two when the energy is equal to zero. This relationship is illustrated graphically in Figure 2.2. The final task to complete before one can actually use Equation 2.5 is to determine appropriate values for ϵ and σ. The typical approach is to fit these parameters with experimental data and it is for this reason that equations of this type are generally referred to as *empirical potentials*.

1.1 Empirical Potentials

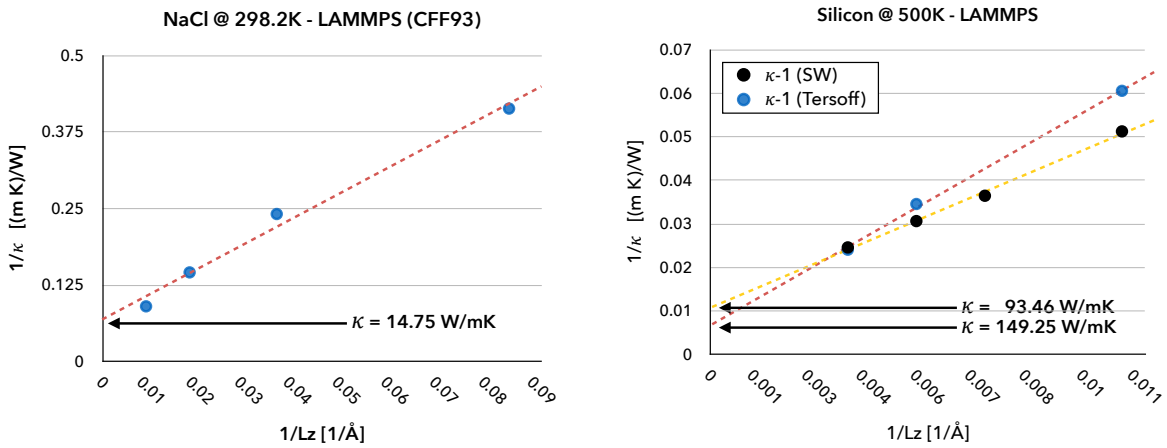
Lennard-Jones is but one example of an empirical potential and many more have been developed over the years. For example, one of the most widely used classical MD programs,

LAMMPS [9], has implemented over 30 different potential functions. By far the most perplexing issue for newcomers is how to choose the correct potential. The answer to this question is, unfortunately, often not very straightforward as it typically depends on the specific system being studied and the properties one wishes to calculate. For example, if one wishes to study NaCl, the embedded-ion method (EIM) potential would likely be a good choice as it was developed to compute pairwise interactions for ionic compounds. Mathematically, it is defined as

$$E = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=i_1}^{i_N}\phi_{ij}(r_{ij}) + \sum_{i=1}^{N}E_i(q_i,\sigma_i), \tag{2.6}$$

where the first term is a double pairwise sum over the J neighbors of I atoms, ϕ_{ij} is the pair potential and the second term is a sum over the embedding energy E_i, which itself is a function of charge q_i and electric potential σ_i. The full expression for Equation 2.6 is not presented here, but when the expressions for q_i, σ_i and E_i have been included, 13 system-specific parameters must be provided. Calculating these parameters is not trivial, thus LAMMPS ships with values for nine elements. To simulate a system containing an element that is not included in the nine provided, one must either calculate the values (very difficult) or choose a different potential.

For metallic systems there is the embedded-atom method (EAM), for Si-based systems (Si, SiC, SiCGe, SiO) there is the Tersoff potential, for III-V and II-VI systems there is the bond order potential (BOP), etc. How, though, does one simulate a material that does not fit nicely into one of these categories *and* has never been studied experimentally? Without previous experimental data, it is extremely difficult to choose an empirical potential that will correctly model the desired system.



(a) (b)

Figure 2.3: Thermal conductivity vs simulation cell length and extrapolation to infinite size for (a) NaCl at 298.2 K and (b) Silicon at 500 K. In the case of Silicon, two different empirical potentials were used; Stillinger-Weber (SW) and Tersoff.

1.1.1 Predictive Capability

Assuming that an empirical potential exists for the system one wishes to study, the next question is how good is the predictive capability? The results from Leblanc et. al[2], are shown in Figure 2.3 for both sodium chloride and silicon obtained via LAMMPS. The NaCl results were obtained with the use of the CFF93 potential[10] and are plotted in Figure 2.3a as inverse thermal conductivity versus inverse cell length. This is a common technique used in NEMD simulations and is discussed in further detail in Section 2.2.2 of Chapter 3. The NaCl results give a value of 14.75 W m^{-1} K^{-1}, which exceeds the experimental value of 5.854 W m^{-1} K^{-1} by almost a factor of three.

In the case of silicon, two potentials were used. The first, Stillinger-Weber, yielded a value of 93.46 W m^{-1} K^{-1} while the Tersoff potential gave a value of 149.25 W m^{-1} K^{-1}. These differ by 20% and 60% from the experimental value of 80 W m^{-1} K^{-1}. These results show that, even for relatively simple systems such as NaCl and silicon, empirical potentials struggle to provide high-accuracy thermal conductivity results.

1.2 First Principles

Up to this point we've only looked at solids as collections of interacting atoms. This is, in general, a gross over-simplification. In reality, electrons must be taken into account in order to get the full picture, and as a result quantum mechanical theory is needed. More specifically, the development of density functional theory (DFT), in which the many-body energy and charge density $\rho(\mathbf{r})$ are calculated, has made it possible to achieve unparalleled accuracy in such simulations, solving much of the issues inherent with empirical potentials. In fact, it is possible to do MD simulations using DFT, negating the need for empirical potentials, resulting in parameter-free simulations of very high accuracy. This is often referred to as *ab initio* MD (AIMD). However, the increased accuracy from DFT comes at extreme computational cost.

To give a rough idea of the tremendous cost associated with AIMD, we can compare real-world results from the classical MD package LAMMPS[9] and the DFT package VASP[7]. Running on a four core processor, the average time per MD step in LAMMPS for a 36000 atoms liquid water system was 9 microseconds. VASP, running on a Cray XT3 supercomputer, spread across 320 processors, obtained an average time per MD step for a 192 atom liquid water system of 252 seconds[1]. The main reason for this tremendous difference in performance is that empirical MD techniques typically scale $O(N)$, where N is the number of atoms, while DFT scales as $O(M^{2.5})$,[2] where M is the number of electrons[1]. The steep computational cost of AIMD makes it impractical to use, on current computing hardware, for thermal conductivity calculations.

2 Molecular Dynamics

In the previous section we looked at the basics of atomic interactions, or more specifically, how to calculate the potential energy and forces that act between atoms using an empirical

[2]For very large systems DFT scales as $O(M^3)$ due to orthogonalization.

potential. Our focus now turns to finding a way to evolve such a system of atoms through time. This is exactly where molecular dynamics (MD) comes in. What exactly does "evolve through time" mean? The easiest way to explain this is through an analogy. Consider a video camera running at 30 frames per second. At this rate it means that the camera is capturing successive still images every 1/30th of a second, or every 33 milliseconds. When these images are viewed in the correct order, the video is played back and our eyes perceive continuous motion–the scene captured by the still images is being "evolved through time". In an MD simulation the same technique is employed, but here the images, or snapshots, are not images but instead they are the current state of the system (velocities, positions, accelerations, etc. of every atom) at that point in time. Successive snapshots are separated by a so-called timestep. In the video camera analogy, the timestep is 33 milliseconds, in an MD simulation the timestep is typically on the order of 1 femtosecond (1 quadrillionth of a second).

2.1 Steps

The first MD simulations were done back in the 1950s[4], and since then the basic structure of an MD simulation hasn't changed dramatically. There are essentially four basic steps that every MD program employs, which consist of

1. Initialize: initialize the system
2. Evaluate: calculate the forces on all of the atoms
3. Integrate: integrate the equations of motion to advance the time step and determine new atomic positions
4. Sample: sample the system to calculate properties

2.1.1 Initialize

The first step is to initialize the system by placing the atoms that comprise the system into a three-dimensional coordinate system. All this really involves is simply giving each atom an x, y and z coordinate. These coordinates can then be used to determine one atom's position relative to another. Since we are concerned with crystals, this will involve placing atoms into their respective lattice sites, but in a more general MD simulation there isn't any strict requirement for how atoms should be placed. In terms of units, it is common to use Å(10^{-10} meters), as neighboring atoms are typically separated by at most a few Å.

The discussion of coordinate systems would be incomplete if we didn't stop to consider what happens at the edges, or boundaries, of the system; a term called boundary conditions. There are two types of boundary conditions typically employed in MD simulations. The first, called "fixed boundary conditions", is actually the more "physical" of the two methods in the sense that it is more similar to the behavior of real materials. In a fixed boundary condition simulation, the atoms at the boundaries interact with fewer atoms than those in the interior, or bulk, of the material. This is analogous to surface atoms in a real material. In macroscopic systems, however, the ratio of bulk atoms to surface atoms is usually very large, and as a result many material properties are dictated by the behavior of the atoms in the bulk. If one cares mainly about bulk properties, the surface atoms can typically be neglected. Unfortunately, simulating a bulk material of millions and millions of atoms is, for the most part, impractical with current computational resources. Therefore, one way to make the system appear "infinitely" large is by using so-called "periodic boundary conditions". Under this condition, there are no surface atoms and instead the Cartesian axes are wrapped around. So for a system with dimensions $L \times L \times L$, an atom at position $r_i = (0, 0, 0)$ would

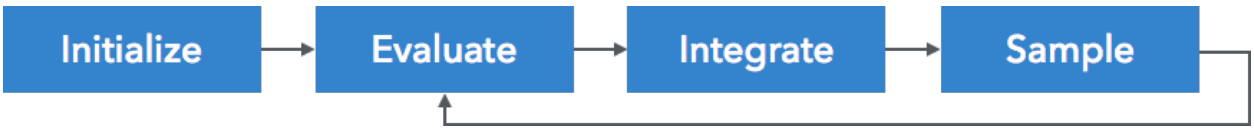


Figure 2.4: Flow chart illustrating the basic steps of an MD program.

simultaneously exist at position $r_i = (L, 0, 0)$.

If we just place atoms on a lattice and start the simulation, nothing will happen and the atoms will remain stationary because they have no velocity. Choosing initial velocities for the atoms is not very intuitive, however, and gives rise to the question of how fast do atoms typically move? A more intuitive approach to answering this question is to consider a property that we are all familiar with–temperature. Rather than starting a simulation with each atom having a random velocity v_i, it is often more convenient to define an initial starting temperature and work backwards to figure out what the initial velocities should be. In order to do so, a relation between temperature and velocity is necessary. Fortunately, kinetic theory provides us with just that via the following relation

$$\frac{1}{2}mv^2 = \frac{3}{2}k_B T, \tag{2.7}$$

where m is mass, v is velocity, k_B is the Boltzmann constant and T is temperature. The left side of this equation is nothing more than the particle's kinetic energy which is equal to the particle's temperature times a constant. From this it is clear that we can define the temperature of a particle, with some given mass m, by setting it's velocity to the correct value. For a system of N particles we can then use Equation 2.7 to get the temperature of the entire system as

$$T = \sum_{i=1}^{N} \frac{m_i v_i^2}{k_B N_f}, \tag{2.8}$$

where N_f is the number of degrees of freedom ($N_f = 3N - 3$). The naïve approach would be to simply give every atom in the simulation a single velocity, in turn yielding the desired temperature. However, this would result in a non-zero momentum for the system,

$$\rho = \sum_{i}^{N} m_i v_i, \tag{2.9}$$

which would cause the system to drift through space (the coordinate system). This is non-ideal for a number of reasons, but in general it is preferred that the system remain stationary in the coordinate system, therefore a better approach for determining initial velocities is necessary.

A better approach is to build off the concept of Maxwell–Boltzmann distribution from statistical mechanics, which describes, in classical physics, the velocity distribution of particles at a given temperature. The functional form of a Maxwell-Boltzmann distribution is

$$f(v) = \sqrt{\left(\frac{m}{2\pi k_B T}\right)^3} 4\pi v^2 e^{-\frac{mv^2}{2k_B T}}. \tag{2.10}$$

In practice, the process then involves picking random velocities from the distribution, with a final scaling of all velocities to ensure that the center of mass of the system is zero.

2.1.2 Evaluate

The evaluation step is responsible for calculating the forces on and the potential energy of every atom in the system. This is also the most computationally expensive part of any molecular dynamics simulation. How expensive this step is depends on the form of the potential being used, or more precisely, how complex ϕ_{ij} is in Equation 2.3. In general though, most MD simulations will spend upwards of 90% of the simulation on this step. As a result, it is extremely important to make this step as computationally efficient as possible. Optimization of this step is the entire focus of Chapter 4.

2.1.3 Integrate

Once the forces acting on each atom have been calculated, the equations of motion must be integrated, resulting in the advancement of the time from time t to time $t + \Delta t$, where Δt is called the *timestep*. This also has the effect of updating atomic positions and velocities. There are various ways to achieve this, but the simplest version is the so-called Verlet algorithm, the derivation of which is quite simple. To derive the equations, we start with a Taylor expansion of a particle's position, $\mathbf{r}$, at time $t + \Delta t$ and $t - \Delta t$

$$\begin{aligned} \mathbf{r}(t + \Delta t) &= \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2!}\mathbf{a}(t)\Delta t^2 + \frac{1}{3!}\mathbf{a}'(t)\Delta t^3 + O(\Delta t^4), \\ \mathbf{r}(t - \Delta t) &= \mathbf{r}(t) - \mathbf{v}(t)\Delta t + \frac{1}{2!}\mathbf{a}(t)\Delta t^2 - \frac{1}{3!}\mathbf{a}'(t)\Delta t^3 + O(\Delta t^4). \end{aligned} \tag{2.11}$$

Adding these two equations together gives

$$\mathbf{r}(t + \Delta t) + \mathbf{r}(t - \Delta t) = 2\mathbf{r}(t) + \mathbf{a}(t) + O(\Delta t^4), \tag{2.12}$$

which can be rearranged to give the next position as a function of the current postition, previous position and acceleration,

$$\mathbf{r}(t + \Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \Delta t) + \mathbf{a}(t). \tag{2.13}$$

The acceleration is given from the interatomic potential as

$$\mathbf{a}(t) = \frac{1}{m}\mathbf{F}(t), \tag{2.14}$$

where m is the particle's mass and $\mathbf{F}$ is determined according to Equation 2.2. In Equation 2.13 we have dropped the fourth order term, which results in an error on the order of Δt^4. As a result of this it is important to choose a very small Δt so as to minimize the error. Typically, Δt is chosen on the order of femtoseconds (10^{-15} seconds).

The equations in 2.11 can also be subtracted to give the particle's new velocity, thusly

$$\mathbf{v}(t) = \frac{\mathbf{r}(t + \Delta t) - \mathbf{r}(t - \Delta t)}{2\Delta t}. \tag{2.15}$$

The main problem with this method is that the velocities and positions are off by a timestep– that is $\mathbf{v}$ at time t is only known once $\mathbf{r}$ has been calculated at time $t + \Delta t$. Often it is convenient to know positions and velocities at the same timestep, thus various other integration schemes have been developed to solve this problem, some of which will be discussed later in Section 2.2.

2.1.4 Sample

The final step is to sample the system before the time step is fully advanced to time $t + \Delta t$. Quantities such as energy, temperature, velocities and forces can be recorded so that they may be analyzed later.

2.2 Integration Schemes

The integration scheme presented in Section 2.1.3 is referred to as the Verlet algorithm and represents one of the simplest ways to evolve a system of particles through time. However, the algorithm is not very convenient, as it results in positions and velocities that differ by a whole time step. As a result, numerous other integration schemes have been developed. The most widely used is called the Velocity Verlet algorithm, which solves the main issue

with the simple Verlet algorithm. By re-expanding Equation 2.11, but taking velocity into account explicitly, one gets the following:

$$\mathbf{r}\left(t+\Delta t\right)=\mathbf{r}\left(t\right)+\mathbf{v}\left(t\right)\Delta t+\frac{1}{2}\mathbf{a}\left(t\right)\Delta t^{2}, \tag{2.16}$$

$$\mathbf{v}\left(t+\Delta t\right)=\mathbf{v}\left(t\right)+\frac{1}{2}\left[\mathbf{a}\left(t+\Delta t\right)+\mathbf{a}\left(t\right)\right]\Delta t. \tag{2.17}$$

The typical approach to implement this algorithm is as follows:

1. $r(t)$ and $v(t)$ are used to compute $a(t)$, at time t, using Equation 2.14.
2. Calculate $r(t+\Delta t)$ using Equation 2.16.
3. Calculate $v\left(t+\frac{\Delta t}{2}\right)$ using Equation 2.17.
4. Calculate new forces using $r(t+\Delta t)$ to get $a(t+\Delta t)$.
5. Calculate final velocities $v\left(t+\Delta t\right)$ using Equation 2.17.

There are a number of other integration schemes, but given the simplicity of the Velocity Verlet algorithm, this one tends to be the most popular. Additionally, as long as a small enough time step is used, the Velocity Verlet is capable of obtaining the same level of accuracy as more costly, higher-order integration schemes.

2.3 Ensembles

Applying the steps discussed in Section 2.1 to create an MD program will result in simulations that contains constant number of particles (N), constant volume (V) and energy (E)–often referred to as the NVE, or microcanonical, ensemble. However, it is often desired to perform MD within different parameters, such as constant temperature or pressure. In particular, we are concerned with the constant temperature (NVT), or canonical, ensemble.

Recall from Equation 2.8 that in classical MD, temperature is determined directly from the kinetic energy of the system, thus in order to control temperature, energy must be added and removed as necessary. The simplest way to do this is through direct velocity rescaling. While this approach works it is not the most ideal for a number of reasons, but the main issue is that simple velocity rescaling techniques result in non-time-reversible, non-deterministic simulations. Because of this, a number of methods have been developed to solve these issues.

2.3.1 Gaussian Thermostat

The so-called Gaussian Thermostat is one such way of controlling temperature in an MD simulation. A nonholonomic system is one in which the system's state depends on the path taken. In the case of MD simulations, Gauss' principle of least constraint leads to a constraint function $g\left(\mathbf{r}_i, \mathbf{v}_i, t\right) = 0$[6]. To find the relation restricting acceleration, one must perform a single differentiation of the constraint function, resulting in

$$\frac{\partial g\left(\mathbf{r}_i, \mathbf{v}_i, t\right)}{\partial t} = \frac{\partial g}{\partial \mathbf{r}_i}\frac{\partial \mathbf{r}_i}{\partial t} + \frac{\partial g}{\partial \mathbf{v}_i}\frac{\partial \mathbf{v}_i}{\partial t} + \frac{\partial g}{\partial t} \tag{2.18}$$

$$= \frac{\partial g}{\partial \mathbf{r}_i}\mathbf{v}_i + \frac{\partial g}{\partial \mathbf{v}_i}\mathbf{a}_i + \frac{\partial g}{\partial t} = 0. \tag{2.19}$$

Since temperature is proportional to kinetic energy (Equation 2.8), the constraint function can be written as

$$g\left(\mathbf{r}_i, \mathbf{v}_i, t\right) = \sum \frac{1}{2} m_i \mathbf{v}_i^2 - \frac{1}{2} N_f k_B T_{\text{set}} = 0 \tag{2.20}$$

Applying Equation 2.20 to Equation 2.19 gives

$$\sum m_i \mathbf{v}_i \mathbf{a}_i = \sum \mathbf{F}_i \mathbf{v}_i = 0. \tag{2.21}$$

In the unconstrained case, the motion of the system is governed by Equation 2.14. However, this equation of motion can no longer be used directly in the constrained case. Therefore, a friction coefficient, ξ, is added to constrain the equation of motion

$$\mathbf{a}_{ic} = \mathbf{F}_i/m_i - \xi \mathbf{a}_i. \tag{2.22}$$

Combining Equation 2.22 and Equation 2.21 yields a definition for the friction coefficient,

$$\xi = \frac{\sum \mathbf{F}_i \cdot \mathbf{v}_i}{\sum m_i \mathbf{v}_i^2}. \tag{2.23}$$

The Gaussian thermostat can then be easily combined into the Velocity Verlet algorithm as

1. $r(t)$ and $v(t)$ are used to compute $a(t)$, at time t, using Equation 2.14.
2. Calculate thermostat variable $\xi(t)$ using Equation 2.23.
3. Calculate $r(t + \Delta t)$ using Equation 2.16.
4. Calculate $v(t + \frac{\Delta t}{2}) = v(t) + \{a(t) - v(t)\xi\}\frac{\Delta t}{2}$.
5. Calculate new forces using $r(t + \Delta t)$ to get $a(t + \Delta t)$.
6. Calculate $v(t + \Delta t) = v(t + \frac{\Delta t}{2}) + \{a(t + \Delta t) - v(t + \Delta t)\xi(t + \Delta t)\}\frac{\Delta t}{2}$.

Notice in the last step that $v(t + \Delta t)$ is on both sides of the equation. Therefore, the final velocities, $v(t + \Delta t)$, have to be calculated iteratively. This is done using a Newton-Raphson

procedure to solve

$$\mathbf{h}_i(\mathbf{v}_i, \xi) = \mathbf{v}_i' + \left(\mathbf{a}_i - \xi \mathbf{v}_i\right) \frac{\Delta t}{2} - \mathbf{v}_i = 0, \tag{2.24}$$

where $\mathbf{v}_i' = \mathbf{v}_i(t + \frac{\Delta t}{2})$. The Jacobian matrix elements are

$$J_j = -\left\{ \frac{F_j v_j - 2\xi v_j^2 m_j}{\sum \mathbf{v}_j^2 m_j} + \xi \right\} \frac{\Delta t}{2} - 1. \tag{2.25}$$

The Newton-Raphson procedure can be done using the following algorithm

1. Compute initial guess for $\mathbf{v}_i(t + \Delta t) = \left\{ \mathbf{v}_i' + \mathbf{a}_i \frac{\Delta t}{2} \right\} m_i^{-1}$.
2. Calculate ξ' using Equation 2.23 and velocities $\mathbf{v}_i(t + \Delta t)$.
3. Calculate $\mathbf{h}$ using Equation 2.24.
4. Calculate Jacobian using Equation 2.25.
5. Update velocities $v_j(t + \Delta t) \leftarrow v_j(t + \Delta t) - h_j / J_j$.
6. If $\max(\text{abs}(\mathbf{h})) < M_{\text{thresh}}$ stop, otherwise go to step 2.

Where M_{thresh} is some small number to ensure that values are properly converged.

3 Lattice Dynamics

The largest drawback to empirical potentials is that they are, in fact, inexact. They are approximate functions and their reliance on a multitude of experimentally-determined parameters makes them impractical for complex systems and often yield mediocre results even for simple ones. Assuming, however, that one is only interested in crystalline solids, in

which atoms reside on a *lattice*, then the theory of *lattice dynamics* provides the mathematical framework to describe, *exactly*, the interatomic forces. For this reason, lattice dynamical theory is quite an attractive alternative to traditional empirical potentials.

In its most basic form, lattice dynamics describes the crystal's potential energy as a function of separation between the atoms;

$$V = \frac{1}{2}\sum_{i,j}^{N} \phi_{ij}(r_{ij}), \tag{2.26}$$

where $\phi_{ij}(r_{ij})$ is some function that depends on the distance between atoms i and j, such as any of the empirical potentials discussed in Section 1.1. For real (large) crystals, this sum becomes computationally infeasible. Fortunately, many forms of $\phi(r_{ij})$ fall off very rapidly with increasing separation distance (see Figure 2.2), resulting in a more manageable calculation. Many MD codes take advantage of this fact by implementing so-called 'nearest-neighbors' lists which describe, for a given atom i, only those atoms that should be considered in the calculation of $\phi_{ij}(r_{ij})$.

If we consider momentarily a simple one dimensional chain of N atoms, in which only neighboring atoms interact, and that all atoms are separated by a distance r, the total potential energy can be written as

$$V = N\phi(r). \tag{2.27}$$

To simplify things further we have also assumed that the chain is wrapped back onto itself (just like periodic boundary conditions from Section 2.1.1) and that all atoms have the same mass. Atoms in a system at a finite temperature will vibrate at any given moment in time and we represent an atom's displacement from it's initial position with u_i. By performing a

Taylor expansion on Equation 2.27 we get[5]

$$V = N\phi + \sum_{s\geq 1} \frac{1}{s!} \frac{\partial^s \phi}{\partial u^2} \sum_n \left(u_n - u_{n+1}\right)^s . \tag{2.28}$$

The variable s determines when to stop the summation. When $s = 2$ we have what is called the 'harmonic approximation' and only interactions between pairs of atoms are considered. In the case of $s > 2$, anharmonic terms are included. Depending on the system and properties of interest, it can sometimes be acceptable to ignore the higher order anharmonic terms as this greatly simplifies the problem (and computational cost). However, for certain properties (such as thermal conductivity), the anharmonic terms play a vital role and therefore cannot be neglected.

Equation 2.28 can be written in a more general form, taking in to account more than just nearest-neighbors and also including anharmonic terms as

$$V = \underbrace{\Phi_0 + \sum_{\mathbf{a}} \mathbf{\Phi_a} u_{\mathbf{a}} + \frac{1}{2!} \sum_{\mathbf{a,b}} \mathbf{\Phi_{ab}} u_{\mathbf{a}} u_{\mathbf{b}}}_{\text{harmonic}} + \underbrace{\frac{1}{3!} \sum_{\mathbf{a,b,c}} \mathbf{\Phi_{abc}} u_{\mathbf{a}} u_{\mathbf{b}} u_{\mathbf{c}} + \cdots}_{\text{anharmonic}} . \tag{2.29}$$

The term $\mathbf{\Phi_{ab}} u_{\mathbf{a}} u_{\mathbf{b}}$ describes how the atoms in a given cluster interact, atoms a and b in this case, where $\mathbf{\Phi}_{ab}$ is called a *force constant* and is the strength of interaction between the two atoms. Once again, we see that the one and two body clusters describe the harmonic terms, while three body and higher clusters include the anharmonic interactions.

In MD simulations it is necessary to know the forces acting on the atoms, which can simply be done by taking the derivative of Equation 2.29 to obtain an expression for force

$$F_{\mathbf{a}} = -\frac{\partial V}{\partial u_{\mathbf{a}}} = -\left(\mathbf{\Phi}_a + \sum_{\mathbf{b}} \mathbf{\Phi}_{ab} u_{\mathbf{b}} + \frac{1}{2} \sum_{\mathbf{b},\mathbf{c}} \mathbf{\Phi}_{ab} u_{\mathbf{b}} u_{\mathbf{c}} + \cdots \right). \tag{2.30}$$

Equation 2.30 is convenient because it describes a way to model the forces between atoms as a function of only two things–namely displacements and force constants. The final challenge is determining what the force constants are and is, in fact, no simple task. The solution to this problem is discussed further in Chapter 3 Section 1.

4 Heat Transport & Phonons

Heat transport through solids can be described as a result of the interactions between electrons and phonons. Thermal conductivity, κ, is the measure of how easily heat can be conducted though a material. Typically, good conductors, such as metals, possess both very large thermal and electrical conductivity values. Conversely, strong insulators often have low values for both types of conductivity.[3] Mathematically, we can write thermal conductivity as the result of two processes

$$\kappa = \kappa_l + \kappa_e, \tag{2.31}$$

where κ_l is the contribution from the phonons (lattice vibrations) and κ_e is due to conduction via electrons. For now, let us focus on the lattice contribution, κ_l, due to phonons. If we imagine a perfectly harmonic system in which a large amount of energy is imparted to a single phonon, this energy will propagate through the crystal, without any loss. Given enough time this energy will be transported all the way through the material. Without any energy loss,

[3] This is not strictly true, however. For example, diamond is an electrical insulator but a fantastic thermal conductor.

the thermal conductivity would be infinite. However, we know that this is not the case and that real materials have finite thermal conductivity. This can be explained by the fact that real materials are not perfectly harmonic and that phonons actually scatter via a variety of mechanisms, including

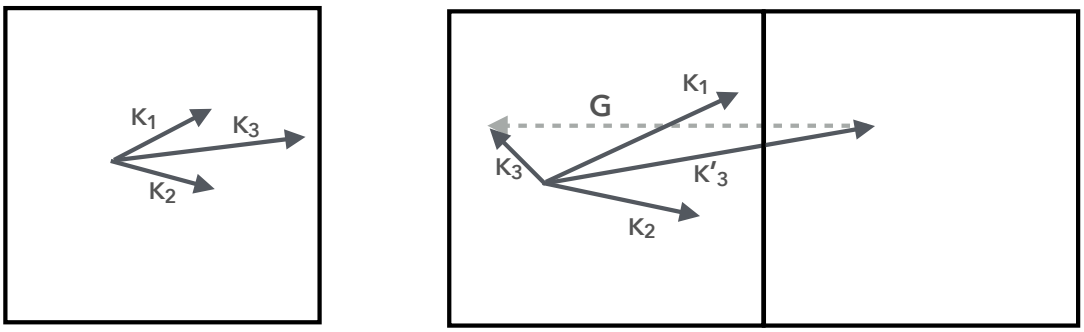
1. phonon-phonon interactions
2. defect scattering
3. surface scattering
4. electron-phonon interactions.

While all of these effects are important, the degree of their impact on the overall thermal conductivity will vary depending on the particular material in question. For now, let us make a few assumptions to simplify things. By assuming that the samples we wish to study are perfect crystals, we can ignore the effects due to defects. Additionally, in bulk materials the ratio of surface to bulk atoms is very small, minimizing the importance of surface scattering. Electron-phonon interactions are really only dominant in metals, particularly at higher temperatures. In non-metals, however, this effect can largely be neglected. That leaves phonon-phonon interactions as the dominant effect in determining heat conduction.

Phonon-phonon interactions can be classified into two distinct type of interactions. The first, called a *normal* process (N-process), involves the collision of two phonons, k_1 and k_2, that combine to form a third phonon, k_3, via the following relation

$$\omega_{\mathbf{3}} = \omega_{\mathbf{1}} + \omega_{\mathbf{2}}. \tag{2.32}$$

In this type of interaction, crystal momentum is conserved. This is shown graphically in Figure 2.5a. If this was the only type of phonon interaction, then there would be nothing impeding the transport of energy, resulting in a very large thermal conductivity. The second



(a) (b)

Figure 2.5: Types of phonon scattering, (a) normal process (N-process) in which crystal momentum is conserved and (b) Umklammp scattering (U-process) in which crystal momentum is not conserved.

type of phonon interaction, called *Umklapp* process (U-process), involves combining two phonons in such a way that crystal momentum is not conserved. Mathematically, a U-process is given as

$$\omega_{\mathbf{1}} + \omega_{\mathbf{2}} = \omega_{\mathbf{3}} + \mathbf{G}, \tag{2.33}$$

and is shown schematically in Figure 2.5b. U-processes lead to finite thermal conductivites and are described by the anharmonic terms. Therefore, without the anharmonic terms, the thermal conductivity will be overestimated.

5 Compressive Sensing

Compressive sensing (CS) is a technique recently developed in the field of information science that is capable of recovering sparse solutions from incomplete data[3]. While CS has found wide use in many fields, such as signal processing, it has, until recently, gone largely unnoticed by the physical sciences community. Nelson et. al recently applied CS to the problem of cluster expansion, a technique popular for alloy modeling, with exceptionally good results[8].

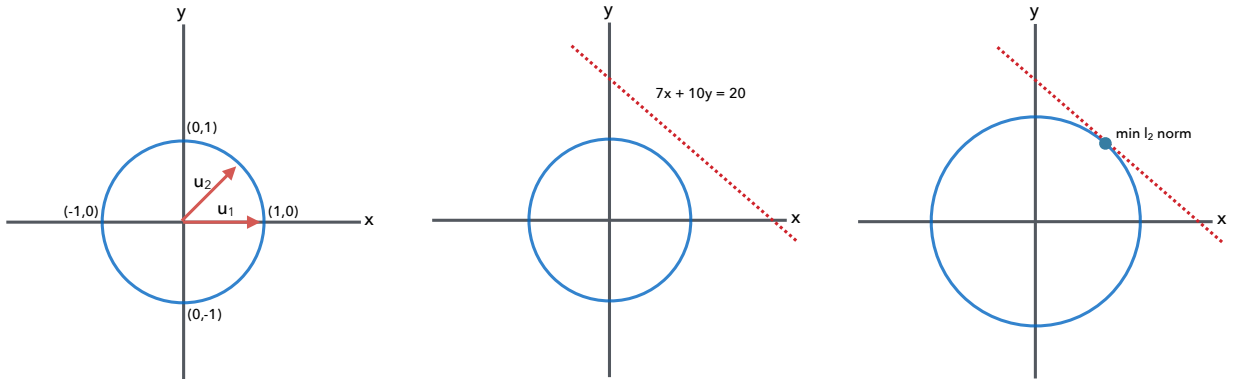
Here, however, we apply CS to the problem of determining unknown force constants of a lattice. Before diving into how this is done, it is more instructive to start with a simple example of how CS works.

We start with the trivial problem of $7x + 10y = 20$, in which there are more unknowns than knowns, making this an underdetermined problem. Let us first consider the well known *least squares* approach to solving this problem. The least squares approach involves minimizing the ℓ_2 norm $||\mathbf{u}||_2 = \left(\sum_i |u_i|^2 \right)^{\frac{1}{2}}$ according to the following

$$\min ||\mathbf{f} - A\mathbf{u}||_2^2 \tag{2.34}$$

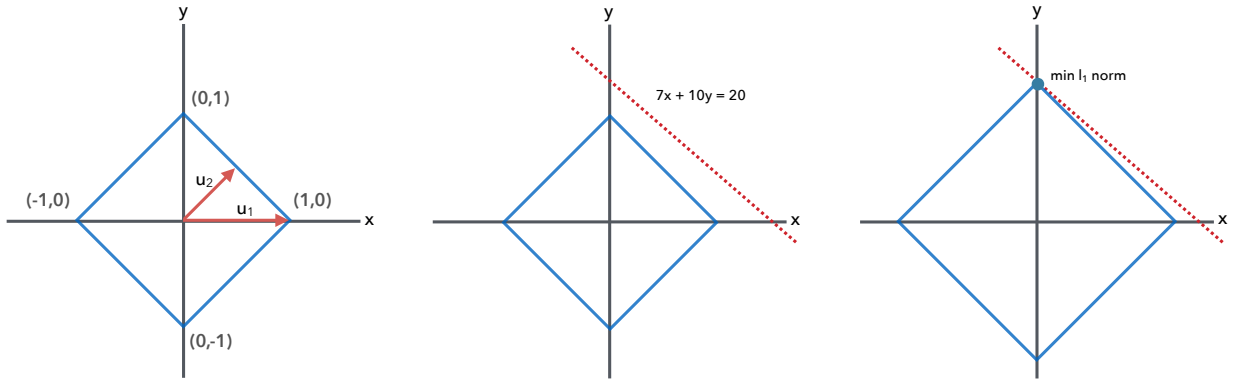
The ℓ_2 norm is shown graphically in Figure 2.6a. In two dimensions the ℓ_2 norm is a circle. In Figure 2.6b the ℓ_2 norm does not satisfy the constraint $\mathbf{f} = A\mathbf{u}$, hence a solution has not yet been found. By growing the ℓ_2 norm, a solution can eventually be found, shown in Figure 2.6c. The result is a dense vector, since both x and y are non-zero (≈ 0.947 and ≈ 1.336 respectively). If an ℓ_1 norm is used in place of the ℓ_2 norm, we get Figure 2.7a. As the ℓ_1 norm is grown, we see that eventually we get a solution as well, but this time it is sparse with only one non-zero value ($x = 0$ and $y = 2$).

One of the key concepts central to CS is sparsity. More specifically, CS cannot solve just *any* underdetermined problem, but rather it can find the solution in cases where there are very few nonzero coefficients. In general, CS is well suited for problems where there are S nonzero coefficients out of a pool of N possible solutions, i.e. when $S \ll N$. In fact, Candes et. al showed that any sparse signal with S nonzero components can be recovered exactly with $M \sim S \ln N$ measurements.



(a) (b) (c)

Figure 2.6: (a) ℓ_2 unit circle, (b) ℓ_2 norm is too small and has not yet found a solution, (c) continuing to grow the ℓ_2 norm gives a solution denoted by the blue dot.



(a) (b) (c)

Figure 2.7: (a) ℓ_1 unit "circle", (b) ℓ_1 norm is too small and has not yet found a solution, (c) continuing to grow the ℓ_1 norm gives a solution denoted by the blue dot.

Mathematically, CS is represented as

$$\min \left\{ ||\mathbf{u}||_1 : A\mathbf{u} = \mathbf{f} \right\}, \tag{2.35}$$

where $||\mathbf{u}||_1$ is the ℓ_1 norm of the unknown coefficients. If this problem were altered to use an ℓ_2 norm like in the previous problem, the result would be a dense solution. This is typically undesirable as the ℓ_2 solution will often be over fitted, leading to an inaccurate solution to the problem.

The application of CS to lattice dynamics will be discussed in Section 1.2 of Chapter 3.

Bibliography

[1] Lammps benchmarks. URL http://lammps.sandia.gov/bench.html.

[2] P. Saxe B. Leblanc and D. Rigby. Validation of thermal conductivity and validation of thermal conductivity and viscosity calculations using lammps. URL http://lammps.sandia.gov/workshops/Feb10/Dave_Rigby/MD_LammpsWorkshop_Feb2010.pdf.

[3] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006. ISSN 1097-0312. doi: 10.1002/cpa.20124. URL http://dx.doi.org/10.1002/cpa.20124.

[4] Berend Smit Daan Frenkel. *Understanding Molecular Simulation*. Academic Press, second edition edition, 2002.

[5] Martin T. Dove. *Introduction to Lattice Dynamics*. Cambridge University Press, 1993.

[6] Denis J. Evans, William G. Hoover, Bruce H. Failor, Bill Moran, and Anthony J. C. Ladd. Nonequilibrium molecular dynamics via gauss's principle of least constraint. *Phys. Rev. A*, 28:1016–1021, Aug 1983. doi: 10.1103/PhysRevA.28.1016. URL http://link.aps.org/doi/10.1103/PhysRevA.28.1016.

[7] G. Kresse and J. Hafne. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47(558), 1993.

[8] Lance J. Nelson, Vidvuds Ozoliņš, C. Shane Reese, Fei Zhou, and Gus L. W. Hart. Cluster expansion made easy with bayesian compressive sensing. *Phys. Rev. B*, 88: 155105, Oct 2013. doi: 10.1103/PhysRevB.88.155105. URL http://link.aps.org/doi/10.1103/PhysRevB.88.155105.

[9] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1 – 19, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1039. URL http://www.sciencedirect.com/science/article/pii/S002199918571039X.

[10] Huai Sun, Stephen J. Mumby, Jon R. Maple, and Arnold T. Hagler. An ab initio cff93 all-atom force field for polycarbonates. *Journal of the American Chemical Society*, 116 (7):2978–2987, 1994. doi: 10.1021/ja00086a030. URL http://dx.doi.org/10.1021/ja00086a030.

CHAPTER 3

Methodology

As the title of this project suggests, the goal is to develop a robust method for obtaining high-quality lattice thermal conductivity values using computational techniques. What is presented in this chapter is a novel technique for calculating lattice thermal conductivity for arbitrary crystals. Figure 3.1 illustrates the basic idea behind this approach. The starting point is a crystal structure for which the lattice thermal conductivity is desired. This crystal structure can be obtained from various online databases, such as ICSD[15], or from experimental results. The interatomic force constants (IFCs) for this crystal structure are then calculated. A number of techniques can be use to determine the IFCs, including density functional perturbation theory (DFPT) or compressively sensed lattice dynamics (CSLD). The latter, CSLD, is a technique that we've developed which solves many of the shortcomings of existing approaches to calculating IFCs. This is the focus of Section 1.

Once the IFCs for the given crystal structure have been determined, they can then be used to calculate the lattice thermal conductivity. There are multiple techniques for doing so as well, including Boltzmann transport equation (BTE) based methods or through the used of

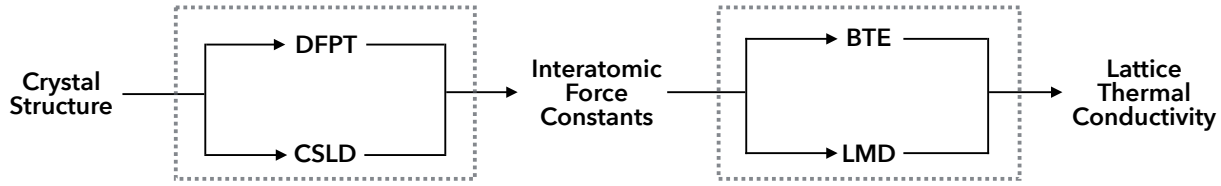


Figure 3.1: Overview of our method for calculating lattice thermal conductivity κ_L. Given a crystal structure, the IFCs are first computed. Once IFCs have been found, one of two methods can be used to calculate κ_L.

molecular dynamics. In particular, we have developed an MD package for simulating systems on a lattice, called lattice molecular dynamics (LMD). Both methods have their strengths and weaknesses, and are the topic of Section 2.

1 Interatomic Force Constants

The first step involves the calculation of interatomic force constants (IFCs). Several *ab initio* methods for studying *harmonic* phonon properties of solids have been proposed, including the frozen phonon approach[13, 29], supercell small displacement method[18, 24] and density-functional perturbation theory (DFPT)[1]. A systematic approach to *anharmonicity*, however, has been more difficult and is the focus of the following sections.

1.1 Density Functional Perturbation Theory

Density functional perturbation theory (DFPT), as the name implies, is a combination of density functional theory (DFT) and perturbation theory (PT). As discussed briefly in Section 1.2, DFT is a very popular technique for calculating ground state properties of electronic systems. More specifically, DFT provides the formalism for calculating *ground state* densities and total energies. DFT is known, however, to have issues representing systems in excited states[28]. Nonetheless, assuming that electrons stay in their ground state, PT can be applied, resulting in DFPT.

Since it's inception in the 80's, DPFT has grown in popularity as a tool for analyzing vibrational and spectroscopic properties of materials, capable of determining phonon frequencies and dielectric properties[28]. Additionally, it has been implemented in a number of popular electronic structure programs, such as Quantum Espresso[10], ABINIT[11], VASP[16] and others.

The theory and specifics of DFPT, however, is beyond the scope of this work. For a detailed review on the current state of the field, the comprehensive review paper by Baroni et al.[2]

is a good starting point. What we wish to highlight here is that, while the '2n + 1' theorem of DFPT can be used to calculate the fourth- and higher-order terms, the computations are cumbersome and require specialized codes which, to the best of our knowledge, are not available for $n > 1$. These issues were the primary motivation for the development of CSLD.

1.2 Compressivley Sensed Lattice Dynamics

Compressively sensed lattice dynamics (CSLD), an alternative to DFPT, is a general and efficient nonperturbative approach that can accurately describe four-phonon and higher-order interactions. This approach to building lattice dynamical models can treat compounds with large, complex unit cells and strong anharmonicity, including those with harmonically unstable phonon modes, and solves many of the issues that currently plague DFPT. CSLD can determine anharmonic force constants from nothing more than standard DFT total energy calculations. Compressive sensing (CS), the basics of which are discussed in Section 5 of Chapter 2, is used to determine which anharmonic terms are important and find their values simultaneously. CSLD does not require any prior physical intuition about a given material, making it simple to use even for the most complex crystal structures.

The starting point for CSLD is with the Taylor expansion of the potential energy in powers of atomic displacements (Equation 2.29). This expansion can be carried out to construct a series of clusters, differing in number of sites and distance between sites. This enumeration is illustrated in Figure 3.2. Clusters in which every site is unique are called *proper clusters*, while those with one or more repeated site are termed *improper clusters*. Figure 3.2 illustrates some of these enumerations for up to 4-body clusters.

To see how CS can help solve the problem of determining the unknown IFCs, we start with the force-displacement relationship for Equation 2.29,

$$F_a = -\Phi_a - \Phi_{ab}u_b - \Phi_{abc}u_b u_c/2 - \cdots . \tag{3.1}$$

These forces can be calculated directly from first-principles using any general-purpose DFT code for a set of L atomic configurations in a supercell. This establishes the linear problem we wish to solve

$$\mathbf{F} = \mathbb{A}\Phi, \tag{3.2}$$

where Φ are the unknown IFCs and $\mathbb{A}$ is referred to as the *sensing matrix*, which is constructed as

$$\mathbb{A} = \begin{pmatrix} 1 & u_b^1 & u_b^1 u_c^1 & \cdots \\ 1 & u_b^2 & u_b^2 u_c^2 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix}. \tag{3.3}$$

The elements in $\mathbb{A}$ are products of atomic displacements, u_i, which correspond to distinct terms in Equation 3.1. Different training configurations are labeled by superscript u_b^j. The rows in $\mathbb{A}$ correspond to a specific force component on a single atom, resulting in a total number of rows $M = 2LN_{\text{at}}$, where N_{at} is the number of atoms in the supercell. The columns in $\mathbb{A}$ correspond to N IFC tensor components, which are arranged in vector Φ. Since N far exceeds M, this makes Equation 3.2 a classically underdetermined problem. Combining

$\Phi_a u_a, \Phi_{aa} u_a^2, \Phi_{aaa} u_a^3, ...$

$\Phi_{ab} u_a u_b, \Phi_{aab} u_a^2 u_b, ...$

$\Phi_{abc} u_a u_b u_c, \Phi_{aabc} u_a^2 u_b u_c, ...$

$\Phi_{abcd} u_a u_b u_c u_d, ...$

Figure 3.2: Example enumeration of clusters, up to 4-site clusters.

everything, we get

$$\begin{pmatrix} 1 & u_b^1 & u_b^1 u_c^1 & \cdots \\ 1 & u_b^2 & u_b^2 u_c^2 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{pmatrix} \begin{pmatrix} \Phi_a \\ \Phi_{ab} \\ \Phi_{abc} \\ \cdots \end{pmatrix} = \begin{pmatrix} -F_a^1 \\ -F_a^2 \\ \cdots \end{pmatrix}. \tag{3.4}$$

CS solves the underdetermined linear problem of Equation 3.2 by minimizing the ℓ_1 norm of the coefficients

$$||\Phi||_1 = \sum_I |\Phi_I| \tag{3.5}$$

while requiring a certain level of accuracy in reproducing the data. The ℓ_1 norm converts this into a solvable convex optimization problem. The final result is

$$\Phi^{\mathrm{CS}} = \arg\min_\Phi ||\Phi||_1 + \frac{\mu}{2}||\mathbf{F} - \mathbb{A}\Phi||_2^2 \tag{3.6}$$

$$= \arg\min_\Phi \sum_I |\Phi_I| + \frac{\mu}{2}\sum_{ai}\left(F_{ai} - \mathbb{A}_{ai,J}\Phi_J\right)^2. \tag{3.7}$$

where the second term is the sum-of-squares ℓ_2 norm. The ℓ_2 norm is the fitting error for the training data, which in this case are DFT forces. The ℓ_1 norm drives the solution towards one that is sparse, i.e, towards a solution with very few non-zero components, while the parameter μ is used to control the relative weights of the ℓ_1 and ℓ_2 terms. By increasing the value of μ, the system is driven towards a least-squares-like fitting, resulting in a denser solution. If μ becomes too large, the results will be prone to overfitting, which is undesirable because this results in decreased predictive accuracy. Conversely, if μ becomes too small, the results will be very sparse and underfitted, which is also bad for predictive accuracy. Therefore, the ideal value of μ lies somewhere in between. The optimal value of μ can be

determined by monitoring the predictive error on a separate set of training data that is not used in the fitting. The predictive accuracy of the model is verified against a third set of data referred to as the 'prediction set'.

One of the key tenets of CS is that near-perfect signal recovery can be realized by using sensing matrices $\mathbb{A}$ with random entries that are independent and identically distributed (IID)[5]. Therefore, an important aspect of CSLD is the proper construction of the sensing matrix $\mathbb{A}$. While the arrangement of the terms making up each entry in $\mathbb{A}$ is fixed, based on the system, we are free to choose the values of the displacements however we wish. Initially, it might appear appealing to use snapshots from AIMD trajectories, since these represent physically relevant, low-energy configurations. However, the problem with this approach is that it leads to configurations containing strong cross correlations between the columns of $\mathbb{A}$. Put another way, there is high mutual coherence of the sensing matrix[8], which decreases the efficiency of CS due to the difficulty of separating correlated contributions to $\mathbf{F}$ from different IFC tensors. It is difficult, however, to create a truly IID sensing matrix in CSLD since the Taylor expansion employs nonorthogonal and unnormalized basis functions of a continuous variable u^n.

As a solution to this problem, we combine the physically relevant MD trajectories with random displacements (on the order of ~ 0.1 Å) of each atom in well-spaced MD snapshots. Additionally, to ensure that all terms in the ℓ_1 term have the same units of force, Equation 3.1 is scaled by $\Phi \rightarrow \Phi u_0^{n-1}$ and $u \rightarrow u/u_0$ where n is the order of the IFC tensor and u_0 is a 'maximum' displacement chosen to be on the order of the amplitude of thermal vibrations. This procedure significantly decreases cross-correlations between columns of $\mathbb{A}$ and leads to stable fits.

To evaluate the effectiveness of CSLD, we begin by looking at three materials; silicon (Si), sodium chloride (NaCl) and tetrahedrite ($Cu_{12}Sb_4S_{13}$). These examples are good choices for evaluating CSLD for a few reasons. First, they each exhibit different levels of anharmonicity and thermal conductivities. Additionally, they range in structural complexity from simple

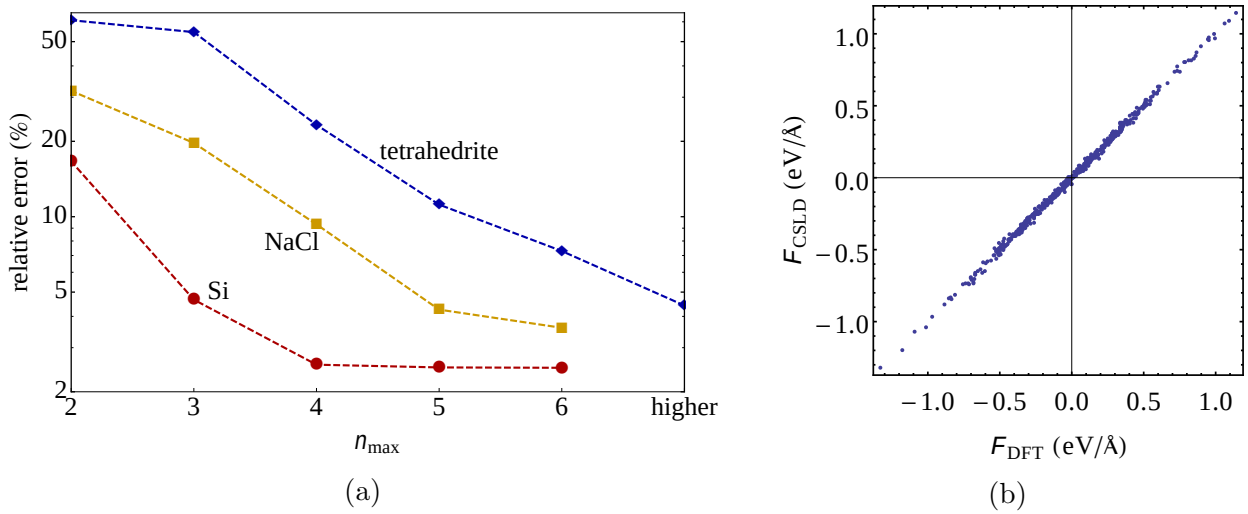


Figure 3.3: Predictive capability of CSLD. (a) shows predictive error versus the maximum number of included higher-order terms. With the inclusion of 6-order terms, all three systems exhibit predictive errors below 10%. (b) demonstrates the predictive error with up to 6-order terms for tetrahedrite compared directly to DFT-derived forces.

silicon to the complex tetrahedrite with 29 atoms in the primitive cell.

DFT calculations were performed using the Perdew-Becke-Ernzerhof functional[26] and projector-augmented wave potentials[3] as implemented in the VASP code[16]. For all systems, up to sixth-order IFC tensors were included, resulting in 712 (Si), 1375 (NaCl) and 3188 (tetrahedrite) symmetrically distinct elements. Using CS, Equation 3.7, 258, 199 and 2101 nonzero IFC tensor elements were found for Si, NaCl and tetrahedrite, respectively. Figure 3.3a illustrates that predictive errors decrease with the inclusion of higher-order IFC tensor elements. Si and NaCl are able to attain force predictions that are $> 95\%$ of those calculated directly via DFT. Figure 3.3b shows the overall accuracy of the model for tetrahedrite over a prediction set from ab initio MD snapshots at 300 K. The root-mean-square error of the predicted force components is 4%.

2 Lattice Thermal Conductivity

Lattice thermal conductivity is well understood from a physical and mathematical standpoint. However, a number of factors have impeded the development of a general-purpose, computational method for calculating κ_L. Some of the earliest approaches to calculating κ_L involved the use of MD simulations, but were typically hindered by relatively low-quality empirical potentials that were incapable of properly describing the interatomic forces between atoms. More recently, first principles-based methods involving the solution to the Boltzmann transport equation (BTE) for phonons have gained in popularity, but have suffered from the inherent difficulty in calculating anharmonic IFCs. CSLD largely solves the issue regarding the calculating (an)harmonic IFCs and, as a result, the BTE approach can now be used with relative ease. While IFCs can also be used to predict interatomic forces, the primary requirement of any MD simulation, no publicly available MD codes currently exists that can directly use such IFCs. To fill this void we have implemented a custom MD code, lattice molecular dynamics (LMD), that requires no other inputs than the calculated IFCs for a given lattice.

2.1 Boltzmann Transport Equation

In 1929, Peierls formulated a microscopic description of the lattice thermal conductivity of semiconductors and insulators, resulting in what has become known as the phonon Boltzmann equation (PBE) or the Boltzman transport equation(BTE)[25]. Since then, a method for an exact iterative solution to the equation has been developed[23], and subsequently implemented as a freely-available code called ShengBTE[19].

In the presence of a temperature gradient, a heat current is induced. In the linear region,

this heat current is given as

$$J^{\alpha} = \sum_{\beta} \kappa^{\alpha\beta} \left(\nabla T\right)^{\beta}, \tag{3.8}$$

where J can be caluclated as

$$\mathbf{J} = \sum_{\lambda} \int f_{\lambda} \hbar \omega_{\lambda} \mathbf{v}_{\lambda} \frac{d\mathbf{q}}{(2\pi)^3}, \tag{3.9}$$

and λ is a phonon mode, of branch index p and a wave vector $\mathbf{q}$, ω_{λ} is the angular frequency, $\mathbf{v}_{\lambda}$ is the group velocity of phonon mode λ and f_{λ} is the phonon distribution function[19]. When a thermal gradient is not present, and the material is in thermal equilibrium, the phonon distribution $f_{0\lambda}$ is given according to Bose–Einstein statistics. Upon the introduction of a thermal gradient, the phonon distribution f_{λ} deviates from the equilibrium $f_{0\lambda}$ and can be calculated via the BTE. The two factors affecting the phonon distribution are the *diffusion* due to the temperature gradient ∇T and the *scattering* between phonons. At steady state, however, the rate of change of the phonon distribution must become zero. Mathematically, this is represented as

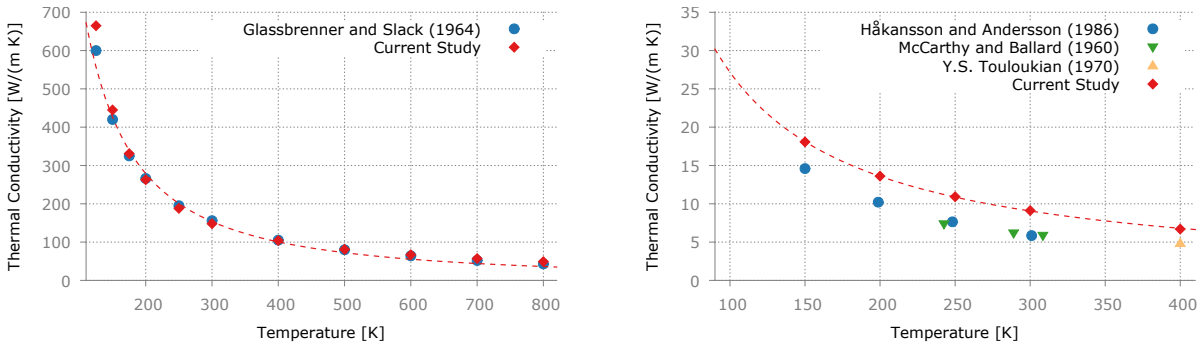
$$\frac{df_{\lambda}}{dt} = \left(\frac{\partial f_{\lambda}}{\partial t}\right)_{\text{diffusion}} + \left(\frac{\partial f_{\lambda}}{\partial t}\right)_{\text{scattering}} = 0 \tag{3.10}$$

where the phonon diffusion is given by

$$\left(\frac{\partial f_{\lambda}}{\partial t}\right)_{\text{diffusion}} = -\nabla T \cdot \mathbf{v}_{\lambda} \frac{\partial f_{\lambda}}{\partial T} \tag{3.11}$$

When the temperature gradient ∇T is small, we can write the phonon distribution as[4]

$$f_{\lambda} = f_{0\lambda} + f_{1\lambda} \tag{3.12}$$



(a) (b)

Figure 3.4: κ_L calculated using BTE as implemented in ShengBTE using IFCs calculated from CLSD, for (a) silicon and (b) sodium chloride.

where $f_{0\lambda} \equiv f_0(\omega_\lambda)$, the equilibrium phonon distribution, and $f_{1\lambda}$ is the non-equilibrium distribution that produces the thermal current and is given as

$$f_{1\lambda} = -\mathbf{F}_\lambda \cdot \nabla T \frac{df_{0\lambda}}{dT} \tag{3.13}$$

Under the assumption that scattering occurs solely via three phonon processes, the linearized BTE is given as

$$\mathbf{F}_\lambda = \tau_\lambda^0 \left(\mathbf{v}_\lambda + \Delta_\lambda\right) \tag{3.14}$$

where τ_λ^0 is the relaxation time of mode λ and Δ_λ is a function of angular frequency ω_λ and $\mathbf{F}_\lambda$[19]. As both sides of Equation 3.14 include $\mathbf{F}_\lambda$, it must be solved iteratively. Finally, κ_L can be written in terms of $\mathbf{F}_\lambda$ as

$$\kappa_L^{\alpha\beta} = \frac{1}{k_B T^2 V N} \sum_\lambda f_{0\lambda}\left(f_{0\lambda}+1\right)\left(\hbar\omega_\lambda\right)^2 v_\lambda^\alpha F_\lambda^\beta \tag{3.15}$$

where V is the volume of the unit cell and N is the number of $\mathbf{k}$-points used in the discretization of the Brillouin zone.

The only inputs necessary are second and third-order IFCs, calculated via either DFPT or CSLD. The results shown in Figure 3.4 were calculated using IFCs calculated with CSLD which were then used in ShengBTE to determine κ_L across a range of temperatures. In the case of silicon, Figure 3.4a, extremely good agreement is seen between experimental values and calculated values, with an average error of $\approx 5\%$. In the case of sodium chloride, Figure 3.4b, the agreement is not as good, with an average error of $\approx 40\%$. The difference in BTE-calculated κ_L in Si and NaCl can be explained largely by looking at the difference in anharmonicity between the two materials. From Figure 3.3a we can see that the predictive error in Si, with the inclusion of up to third-order IFCs, is less than 5%, while for NaCl the error is much higher at $\approx 10\%$. To get the same level of predictive capability in NaCl, it is necessary to include up to fifth-order IFCs. BTE, however, only includes second and third-order IFCs. For Si, this is adequate, but in the case of NaCl, BTE is missing the inclusion of higher-order anharmonicity. The result is that NaCl appears as a more harmonic system than is really is, resulting in an *underestimation* of anharmonicitiy, which in turn leads to an *overestimation* of κ_L across the entire temperature range. This behavor is observed in Figure 3.4b.

The takeaway from these results is that the BTE method is well suited for lightly-anharmonic systems in which a majority of anharmonic effects are captured by third-order IFCs. For strongly anharmonic systems, in which fourth-order or higher IFCs are important, a different method is needed in order to obtain more accurate values of κ_L.

A final note must also be made regarding systems exhibiting unstable phonon modes, such as tetrahedrite. The dispersion curves of tetrahedrite are plotted in Figure 3.5 in which the unstable modes are represented as negative frequencies. In actuality, the unstable modes are complex values which makes their use in the BTE method impossible. As a result, an alternative to BTE is also necessary for systems exhibiting unstable modes.

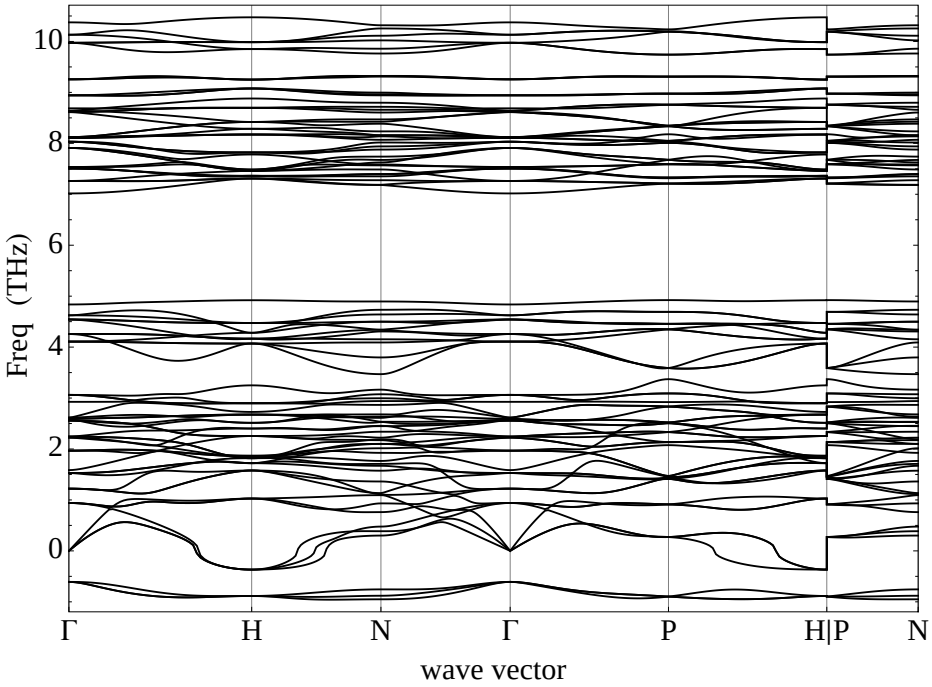


Figure 3.5: Phonon dispersion curves for tetrahedrite which exhibits unstable phonon modes (represented as negative frequencies).

2.2 Lattice Molecular Dynamics

Molecular dynamics (MD) is a useful tool for calculating a range of material properties and observing various physical phenomena, including lattice thermal conductivity. As mentioned previously, most MD packages model atomic interactions in one of two ways; (i) through the use of emprical potentials or (ii) via first-principles. Empirical potentials are computationally favorable, but tend to result in poor estimations of κ_L due to their relatively low-accuracy interatomic force predictions. First-principles approaches, on the other hand, offer very high-quality interatomic forces, but are prohibitively expensive to use in long-time MD simulations.

Lattice molecular dynamics (LMD) approaches the problem in a different way. In LMD, atomic interactions are modeled using the Taylor expansion of lattice energy from Equation 2.29. Taking the derivative, with respect to displacement, allows the direct calculation

of forces within the system

$$F_{\mathbf{a}} = -\frac{\partial V}{\partial u_{\mathbf{a}}} = -\left(\mathbf{\Phi}_a + \sum_{\mathbf{b}} \mathbf{\Phi}_{ab} u_{\mathbf{b}} + \frac{1}{2} \sum_{\mathbf{b},\mathbf{c}} \mathbf{\Phi}_{ab} u_{\mathbf{b}} u_{\mathbf{c}} + \cdots \right). \tag{3.16}$$

The only input necessary are the calculated IFCs for a given lattice. This removes the dependency on empirical potentials, while providing predictive capabilities that are on-par with DFT, but at a fraction of the computational cost. The IFCs calculated via CSLD can be used directly to run long-time MD simulations using LMD.

To get an idea of where LMD fits in performance-wise with respect to empirical potential based LAMMPS and the first-principles DFT package VASP, Table 3.1 shows the average time taken to perform a single MD time-step for a 512-atom NaCl system. Each value was obtained by running the various packages in parallel on 8 CPU cores. LAMMPS is on the order of microseconds, LMD milliseconds and VASP thousands of seconds.

Package	**Iteration Time (seconds)**
LAMMPS	7.12e-4
LMD	1.80e-2
VASP	15798.21

Table 3.1: Average MD iteration time for various software packages. Tests were performed using 512-atom NaCl system running on 8 CPU cores.

A variety of MD-based methods have been developed for calculating lattice thermal conductivity. The main advantage of using MD is that the anharmonicity of the system is implicitly included though the interactions between the constituent atoms of the system. The following sections discuss the theory behind and application of a range of techniques for calculating κ_L via MD, including both equilibrium and non-equilibrium methods.

2.2.1 Equilibrium MD

A material in thermal equilibrium will have an average temperature that is constant throughout the entire volume. Fourier's law of heat conduction, Equation 3.17, also shows that in the absence of a temperature gradient the heat flux, $\mathbf{q}$, will also be zero.

$$\mathbf{q} = -\kappa \nabla T \tag{3.17}$$

This makes sense intuitively because in the absence of a temperature gradient there is no driving force for thermal diffusion. However, instantaneous atomic vibrations will lead to a finite heat flux given by

$$\mathbf{q}(t) = \frac{d}{dt}\frac{1}{V}\sum_{i}^{N} \mathbf{r}_i(t) e_i(t), \tag{3.18}$$

where r_i and e_i are the instantaneous position and energy of particle i, respectively, at time t and V is the volume of the material. According to Equation 3.18 it is possible to have a non-zero heat flux even when a material is in thermal equilibrium, leading Equation 3.18 and Equation 3.22 to appear slightly contradictory. This discrepancy can be resolved by considering that it is the *average* heat flux that is zero for a material in thermal equilibrium, namely

$$\big\langle \mathbf{q}(t) \big\rangle = 0, \tag{3.19}$$

and the instantaneous heat flux oscillates about zero.

Green [12] and Kubo [17] developed the formalism for determining transport coefficients from

long-time correlations of such instantaneous fluctuations. In the case of thermal conductivity, the relationship is given by

$$\kappa_{\alpha\beta} = \frac{1}{Vk_BT^2} \int_0^\infty \left\langle J_\alpha(0) J_\beta(t) \right\rangle dt, \tag{3.20}$$

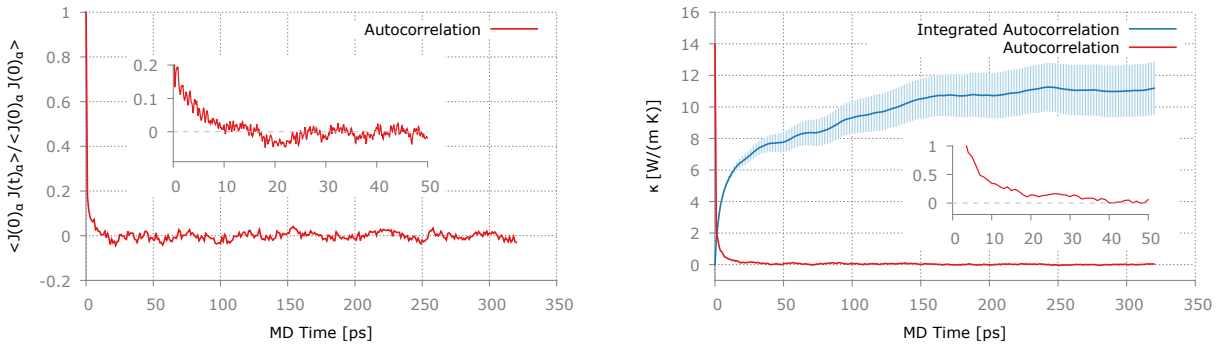
where α and β are coordinate directions and J is the heat current which is related to the heat flux via $\mathbf{J} = V\mathbf{q}$. This equation allows one to calculate the lattice thermal conductivity, κ_L, from nothing more than the fluctuation of the heat flux of Equation 3.18.

The term $\left\langle J_\alpha(0) J_\beta(t) \right\rangle$ is referred to as the heat current autocorrelation function. In many crystals, the current autocorrelation *should* decay quickly to zero, however in practice this is rarely observed in a single simulation. What is typically seen instead is the behavior illustrated in Figure 3.6a in which the current correlation drops off quickly but then continues to oscillate about zero. As discussed by Esfarjani[9] the time average in Equation 3.20 assumes that the system in question is ergodic, when in fact such finite simulations rarely are. The solution to this problem is to average not only over time but also over ensembles. This leads to a modified form of Equation 3.20

$$\kappa_{\alpha\beta} = \frac{1}{Vk_BT^2} \frac{1}{N} \sum_n^N \left(\frac{\Delta t_n}{\tau_n} \sum_{t=0}^{\tau_n} \sum_{u=1}^{\tau_n - t - 1} J_{\alpha,n}(u) J_{\beta,n}(u+t) \right) \tag{3.21}$$

where N is the number of ensembles, τ_n is the number of timesteps for a given ensemble and Δt_n is the timestep. Note that this summation scales as $O\left(\frac{\tau_n(\tau_n - 1)}{2}\right)$, which for large τ_n becomes extremely computationally expensive. Appendix B discusses this issue in more detail and presents a solution for reducing the cost of evaluating this expression.

Figure 3.6a shows the typical behavior for the heat current autocorrelation function resulting



(a) (b)

Figure 3.6: Heat current autocorrelation functions for NaCl simulations at $300K$ for $N = 26$ and a supercell of $4 \times 4 \times 4$ (512 atoms). (a) Current correlation for a single MD run. (b) Ensemble averaged integrated current correlation (blue) and associated error (light blue), in which the current correlation has been scaled by a constant to fit in the plot (red). Inset shows behavior of the current autocorrelation over the first 50 ps.

from a single simulation. This particular example is of NaCl at 200 K. The current correlation quickly drops to zero and then continues to oscillate around zero. Figure 3.6b shows Equation 3.21 plotted with the ensemble averaged current correlations. The current correlation (red line) behaves differently than that in Figure 3.6a due to the ensemble averaging, dropping off to zero after 40ps (see inset).

The question then becomes, how does one calculate κ_L from such plots? The integrated autocorrelation function in Figure 3.6b (blue line) tells us what κ_L is, however, the value depends on where one reads the curve. Li[14] suggests two methods for interpreting such plots. The first is to integrate out to the point at which the current correlation first drops to zero. The second is to fit to an exponential to the current correlation over some time range. Using the first drop method, we can see that the current correlation first reaches zero at 40ps in Figure 3.6b giving a value of 7.68 ± 0.5W m^{-1} K^{-1}. According to Li[14], little difference is typically observed in the exponential fitting method.

However, these methods are unsuitable for some materials which do not exhibit a monotonic exponential decay, such as various silica, germanium and carbon systems[6, 7, 21], which instead exhibit large initial fluctuations. We have observed similar behavior in $Cu_{12}Sb_4S_{13}$,

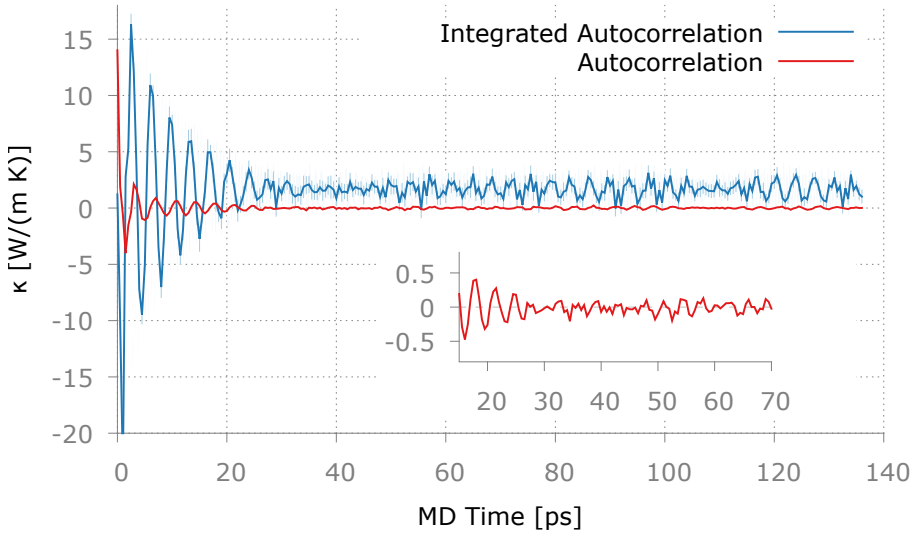
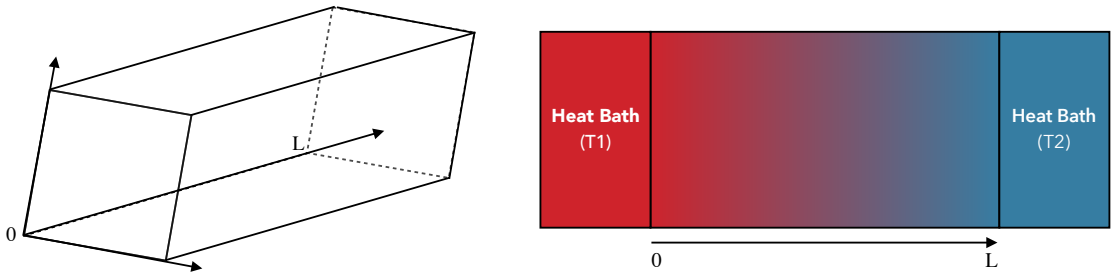


Figure 3.7: Heat current autocorrelation function for tetrahedrite simulations at 150 K for $N = 7$ and a supercell of $2 \times 2 \times 2$ (464 atoms). The current correlation (red line) has been scaled by a constant to fit in the plot.

an earth-abundant natural mineral tetrahedrite, shown in Figure 3.7. It is clear that neither proposed methods will work for such behavior.

A more general approach to this problem was suggested by McGaughey[22] in which an average is taken over a time range from the location where oscillations reach a minimum. In Figure 3.6b this would be between 50 and 100 ps, giving a value for NaCl of $\kappa = 8.51 \pm 0.82$W m^{-1} K^{-1}, which is within roughly 35% of the experimental value of 5.9W m^{-1} K^{-1}. For tetrahedrite this would be between 35 and 55 ps, giving a value of $\kappa = 1.71 \pm 0.685$W m^{-1} K^{-1}, which is within about 60% of the experimental value.

Numerous studies have also been performed to understand the effects that system size has on results obtained via the GK method. Esfarjani[9] studied Si via the GK method and, while they did not draw a conclusion on size dependence, they did notice a slight increase of κ_L when using larger system size. However, in a similar study of Si by Sellan[27] no discernible size dependence was observed. Our studies also show a lack of system size-dependence, which will be discussed in Section 3.2.1.



(a) (b)

Figure 3.8: NEMD simulation setup. (a) The simulation cell is elongated along a particular axis. (b) Heat baths lead to the development of a temperature gradient across the material.

2.2.2 Non-equilibrium MD

Non-equilibrium MD (NEMD) takes a conceptually simpler approach, compared to EMD, by drawing on the relationship between heat conduction and temperature gradient as defined by Fourier's law

$$\mathbf{J} = -\kappa \nabla T, \tag{3.22}$$

where $\mathbf{J}$ is the heat flux. If a system in equilibrium at time $t = 0$ is then placed in contact with two heat baths, at temperatures T_1 and T_2, where $T_1 > T_2$, then heat will flow from one side of the material to the other. Once this rate of heat flow reaches steady state, a linear temperature profile can be observed between two the heat baths along the length of the material. By measuring the amount of heat that flows between the two heat baths, and the resulting temperature gradient, one can use Equation 3.22 to calculate the thermal conductivity.

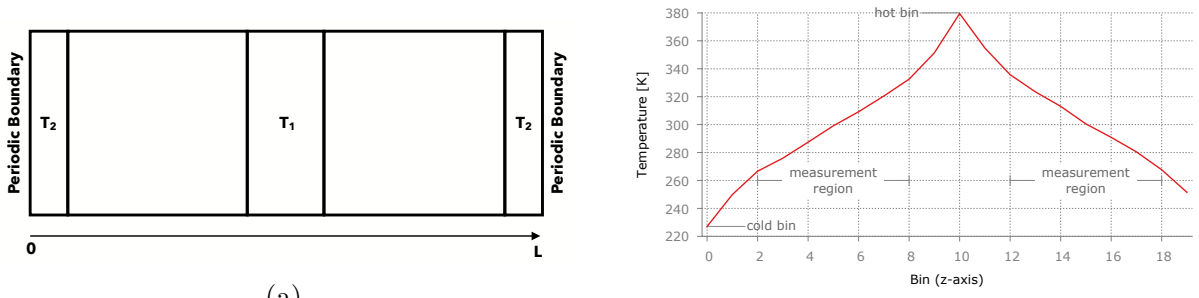
In practice, this is often done by elongating the system along a particular axis, for example the z-axis, and then applying the heat baths at positions $z = 0$ and $z = L$ as shown in Figure 3.8a. There are a number of issues that arise from this setup however. The first is that periodic boundary conditions can no longer be employed in the elongated axis. This

can give rise to finite size effects and, as a result, one must determine through a series of simulations at increasing cell lengths what simulation cell size gives converged results. Another consequence of this is that, typically, very large cell sizes are required, resulting in increased computational cost. Perhaps the most serious issue with this method, however, is that the resulting temperature difference between the two ends of the simulation cell often tends to be quite large. This issue is more significant in low TC materials due to the fact that the material is unable to efficiently transport heat away from the hot side, resulting in rather large thermal gradients across the material. As κ is a function of temperature, this also results in fundamentally different values of thermal conductivity in either end of the material, which in turn can lead to large uncertainty in the calculated value of κ.

2.2.3 Reverse Non-equilibrium MD

Reverse non-equilibrium MD (RNEMD) was developed to solve some of the issues involved with NEMD. In particular, RNEMD does away with the need for heat baths while also allowing periodic boundary conditions to be applied to all axes. The dependency on heat baths is removed by artificially inducing a heat flux in the material. The setup of the system is similar to NEMD in which one axis is elongated. The system is then logically divided into N equal volume 'slabs'. Slab N is then designated as the 'hot slab' and slab $N/2$ as the 'cold slab'. A simple microcanonical (NVE) MD simulation is then carried out. At a fixed frequency, every t_s steps, the velocity of the hottest atom in the cold slab is swapped with the velocity of the coldest atom in the hot slab. It is this swapping of velocities that generates the heat flux according to the equation

$$J = \frac{\sum_{i}^{N_{\text{swaps}}} \frac{1}{2} m_i (v_{i_h}^2 - v_{i_c}^2)}{2tA}, \tag{3.23}$$



(a) (b)

Figure 3.9: (a) Typical RNEMD topology where $T_1 > T_2$. (b) Example temperature profile resulting from an RNEMD simulation. Temperature gradient should be taken away from the hot and cold bins to ensure better sampling.

where v_{i_h} and v_{i_c} are the velocities of the hot and cold atoms, respectively, per swap event, $\mathbb{A}$ is the cross sectional area perpendicular to the elongated axis, t is the total simulation time and N_{swaps} is the total number of velocity swap events. The temperature difference between the hot and cold slabs, ΔT, can then also be measured and combined with the calculated heat flux to give the thermal conductivity as

$$\kappa = -J \left(\frac{dT}{dz} \right)^{-1} \tag{3.24}$$

where dz is the distance between the hot and cold slabs.

While this method does solve some of the issues present in NEMD, it still suffers from many of the same drawbacks and in some cases the issues become even more severe. For example, consider in NEMD two heat baths at temperatures $T_1 = 325K$ and $T_2 = 275K$. After enough time has passed the system will be in steady state and the difference between the hot and cold sides will be $\Delta T = T_1 - T_2 = 50K$. In the case of RNEMD, however, we have no direct control over the temperature of the hot and cold bins. Instead, we can only control the heat flux by adjusting the velocity swapping rate. For a material with unknown thermal conductivity, it is impossible to choose a swapping rate that results in a specific

ΔT. This can be problematic, especially in low thermal conductivity materials where heat cannot be transported away from the hot slab very quickly. In this case, if the swapping rate is too high, the hot slab will continue to get hotter and hotter, which can ultimately lead to melting–an obviously undesirable side effect. Even if melting doesn't occur, it is still possible to end up with a ΔT that is on the order of hundreds of Kelvin, which can make determining κ_L rather tricky. Finally, RNEMD also suffers from the need to use relatively large simulation cells.

2.2.4 Homogenous Non-equlibrium MD

A major drawback of the (R)NEMD approach is the spatial inhomogeneity resulting from the temperature gradient. This is undesirable as it leads to a variation in thermal conductivity throughout the sample. For small temperature gradients this issue is mitigated somewhat, but still results in some ambiguity as to what the true value is for κ_L. Homogeneous non-equilibrium MD (HNEMD) is an attempt to solve this issue by removing the temperature gradient all together.

HNEMD starts by modifying the equations of motion, such that the force on atom a is given by

$$\mathbf{F}_a = F_a - \sum_b \mathbf{F}_{ab} \left(\mathbf{r}_{ab} \cdot \mathbf{F}_e \right) + \frac{1}{N} \sum_{b,c} \mathbf{F}_{bc} \left(\mathbf{r}_{bc} \cdot \mathbf{F}_e \right), \tag{3.25}$$

where F_a is the unmodified force calculated from Eq. (3.16) and $\mathbf{F}_{ab}$ is the force on atom a due to b [1]. The external field $\mathbf{F}_e$ has the effect of driving higher energy (hotter) particles with the field and lower energy (colder) particles against the field. A Gaussian thermostat is also employed to remove the heat generated by $\mathbf{F}_e$. Using linear response, the average heat

[1] Contributions from third- and higher-order interactions to $\mathbf{F}_{ab}$ were obtained by partitioning the energy evenly among all interacting atoms, including repeated sites.

flux is given by

$$\left\langle \mathbf{J}(t) \right\rangle = -\beta V \int_0^t ds \left\langle \mathbf{J}(t-s) \otimes \mathbf{J}(0) \right\rangle \cdot \mathbf{F}_e. \tag{3.26}$$

As $\mathbf{F}_e \to 0$ and $t \to \infty$, one recovers the Green-Kubo formula Equation 3.20. For cubic systems the external field can be set to $\mathbf{F}_e = (0, 0, F_z)$, and we get

$$\kappa_L = \frac{V}{k_B T^2} \int_0^\infty dt \left\langle J_z(t) J_z(0) \right\rangle = \lim_{F_z \to 0} \frac{-\left\langle J_z(\infty) \right\rangle}{T F_z}. \tag{3.27}$$

As this method is rooted in linear response theory, the above equations are only valid within the linear regime.

The process then involves a series of simulations at varying external fields $\mathbf{F}_e$ and constant T, with a simple linear extrapolation to zero field resulting in the true κ_L. Figure 3.10a shows the heat flux for NaCl plotted at different external fields. Each point represents the average heat flux over a minimum of five simulations. For each point, the associated *reduced* κ_L, κ_L^*, is then calculated as

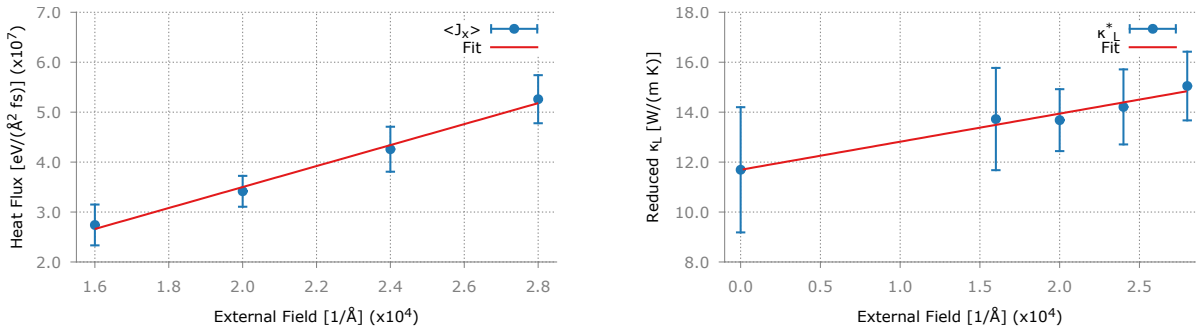
$$\kappa_L^* = \frac{\left\langle J_z \right\rangle}{T F_z} \tag{3.28}$$

As is the case for GK, multiple ensembles must also be used to sufficiently sample the phase space. Therefore, Equation 3.28 can be rewritten as

$$\kappa_L^* = \frac{1}{N} \sum_n^N \frac{\left\langle J_z \right\rangle_n}{T F_z} \tag{3.29}$$

Figure 3.10b shows the extrapolated κ_L for a sample system of NaCl at 200 K. The fitted line (red) gives the true κ_L where $F_z = 0$, yielding a value of 11.69 W m^{-1} K^{-1}.

The most difficult part of this method is choosing an appropriate external field F_z, since



(a) (b)

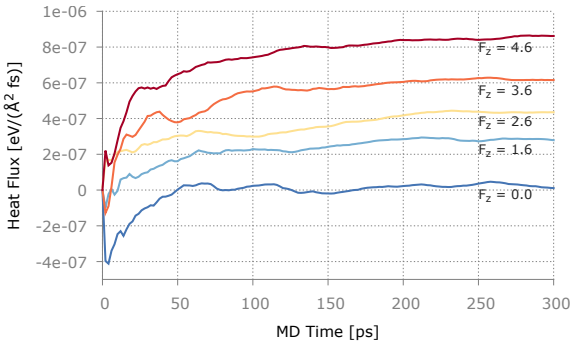
Figure 3.10: HNEMD results for NaCl at 200 K and a supercell of $4 \times 4 \times 4$ (512 atoms). (a) Average heat flux and (b) reduced kappa κ_L^* over a range of external fields. Extrapolation to zero field plotted as red line.

that is the only free variable in Equation 3.28 (not including temperature). The matter is further complicated by the rather odd units of the field (Å^{-1}), which makes it difficult to apply any physical significance to the chosen value. To gain a better idea of how to choose F_z, it is instructive to understand how the heat flux behaves with respect to different fields.

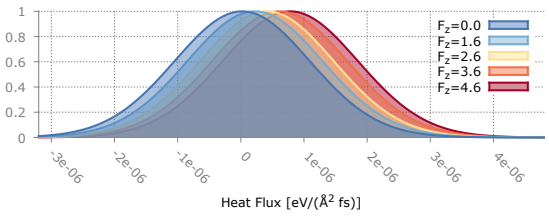
When there is no external field, the heat flux will oscillate about zero, giving an average that is zero. When an external field is applied, the average heat flux will begin to deviate, as predicted by Equation 3.28. Figure 3.11a shows the running average heat flux, at varying external fields, for NaCl. As expected, the average increases with increasing field.

What Equation 3.28 doesn't tell us, however, is how the heat flux distribution will change with respect to the external field. Figure 3.11b shows the heat flux distribution for a range of external fields. At modest external fields, the average $\langle J_z \rangle$ increases, but the distribution is unaffected. In Figure 3.11b the standard deviations of all curves are within 1% of the equilibrium value σ_e.

At larger external fields the story is different. As shown in Figure 3.12a, increasing large external fields give way to a divergent average heat flux. Additionally, the distribution of the heat flux begins to increase from σ_e. This behavior is illustrated in Figure 3.12b. At a field strength of 4.6e-4 (light blue curve), σ is still equal to σ_e. However, at the slightly larger

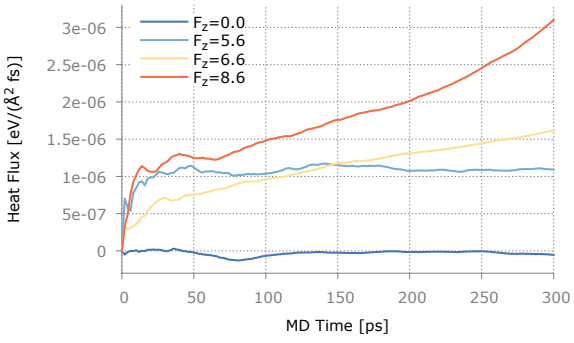


(a)

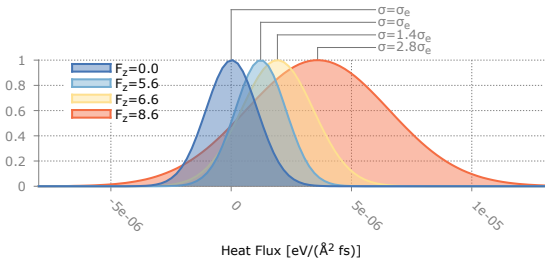


(b)

Figure 3.11: HNEMD results for NaCl at 200 K and a supercell of $4 \times 4 \times 4$ (512 atoms). (a) Running average of $\langle J_z \rangle$ at varying external field strengths. (b) Heat flux distribution for a range of external fields, fitted to a Gaussian distribution. At modest external fields, the average $\langle J_z \rangle$ increases (seen as a shift in the peaks), but the shape of the distribution does not change (standard deviation is constant).



(a)



(b)

Figure 3.12: Behavior of HNEMD simulations when subjected to large external fields. (a) The heat flux diverges when the external field is too large. (b) The distribution of the heat flux is no longer equal to the equilibrium value at large fields.

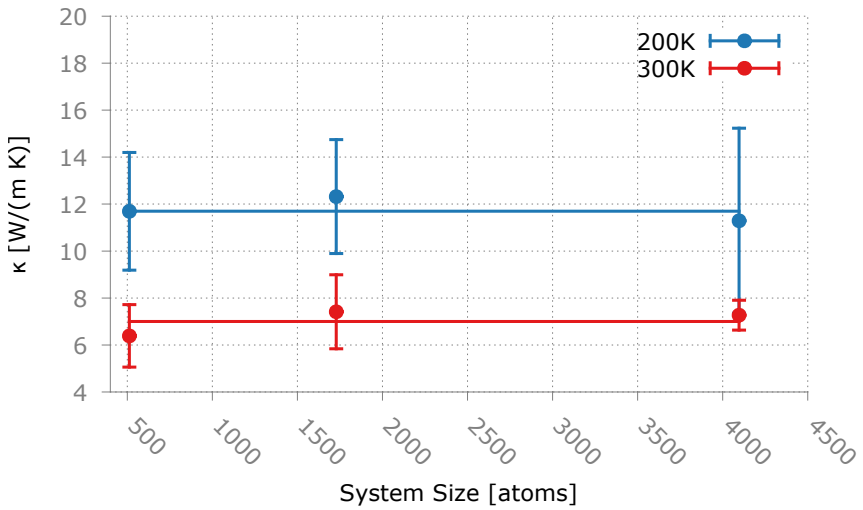


Figure 3.13: System size versus calculated κ_L for NaCl. No discernible system size dependence is observed.

fields of 6.6e-4 and 8.6e-4, the standard deviation increases to $1.4\sigma_e$ and $2.8\sigma_e$, respectively. It is important to note that when the system exhibits this behavior it is no longer within the linear regime, rendering Equation 3.27 invalid. Therefore, caution must be employed when choosing a value for the external field. As a solution to this problem, Section 1 in Chapter 5 discusses a possible algorithm for automating the choice of $\mathbf{F}_e$.

One of the main advantages of the HNEMD method is that it doesn't suffer from undesirable system-size effects. Figure 3.13 shows how the calculated κ_L for NaCl changes with respect to system-size. At both temperatures there are no discernible system-size effects. This is a huge improvement over (R)NEMD methods. The other benefit to this approach is that HNEMD requires significantly less simulation time than the Green-Kubo EMD method, resulting in reduced computational cost. Specifics are proivided in Section 3.2.2.

3 Example Systems

In order to evaluate the efficacy and broad applicability of a new method, it is instructive to study a selection of systems that exhibit a wide range of behavior in the properties that one

wishes to calculate. Thermal conductivity is a rather tricky property due to the fact that values differ heavily between systems and across temperatures within systems. Therefore, one must test systems that exhibit very different κ_L while also testing each system across a broad range of temperatures. To this end, we have selected three systems which we believe highlight the strengths and weaknesses of the this approach to calculating lattice thermal conductivity. These systems include silicon (Si), sodium chloride (NaCl) and tetrahedrite ($Cu_{12}Sb_4S_{13}$). Each of these systems posses increasing levels of anharmonicity, resulting in orders-of-magnitude differences in κ_L. Additionally, the complexity of the units cell range from the simple, single species, diamond-cubic structure of silicon, to the 29 atom primitive cell of tetrahedrite.

3.1 Si

Silicon is a relatively low anharmonicity system with only two atoms in the primitive cell. It is a well studied system, both experimentally and computationally, with previous studies showing excellent agreement between calculated and experimental values for κ_L. Therefore, silicon can be seen as a sort of 'litmus test'.

Force constants for Si were calculated using CSLD. We included pair and up to 6-th order IFC tensors, giving a total of 712 symmetry-distinct elements and 258 non-zero elements from compressive sensing. Lattice thermal conductivity was calculated using the ShengBTE package, including pair and third order IFCs. The results are plotted in Figure 3.14 and show great agreement across the entire temperature range tested.

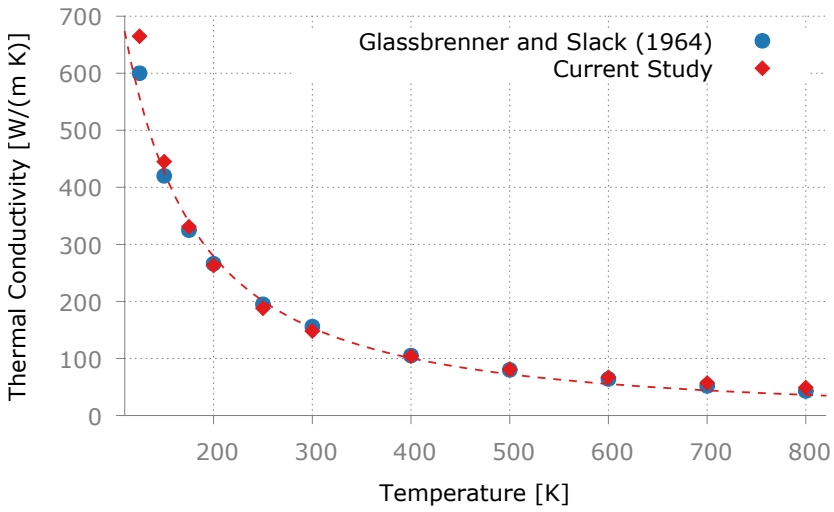
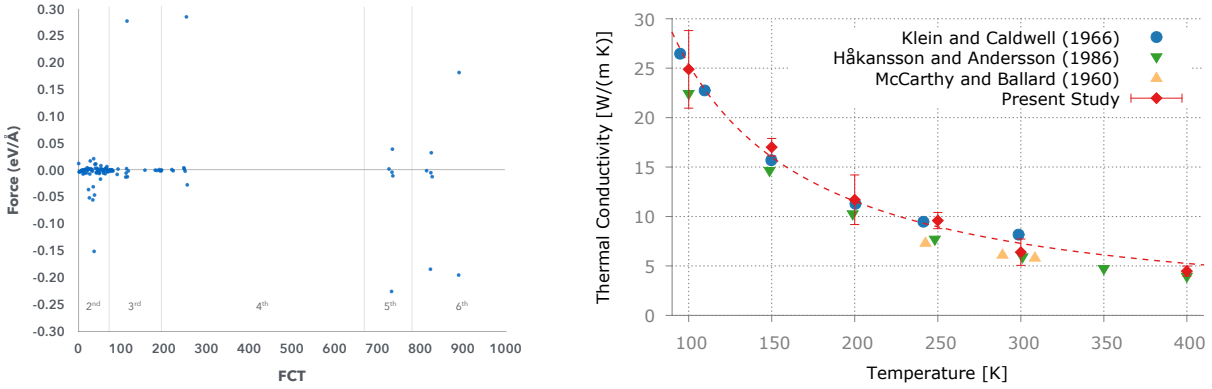


Figure 3.14: Lattice thermal conductivity versus temperature for silicon. Calculated values were obtained using CSLD and BTE.

3.2 NaCl

Sodium chloride is a relatively high anharmonicity system and a good test system because there is a plethora of experimental data on it's thermal conductivity. It is a simple enough system that a 'robust' method should be able to calculate κ_L relatively easily, yet this is something that, as mentioned in Section 1.1.1 of Chapter 2, even specialized empirical potentials fail to accomplish. Additionally, Figure 3.4b highlights how BTE overestimates κ_L across the entire temperature range tested. Since the BTE approach only includes up to third-order IFCs, it is likely that the omission of higher-order interactions leads to an underestimation of the anharmonicity in NaCl. The results shown in Figure 3.3a further reinforce this, as predictive error is quite large ($\sim 20\%$) when only pair and thee-body terms are included. Therefore, we used CSLD to include higher-order ICFs and LMD to calculate κ_L for NaCl.

Force constants for NaCl were calculated using CSLD. We included pair and up to 6-th order IFC tensors, giving a total of 1375 symmetry-distinct elements and 199 non-zero elements from compressive sensing. Figure 3.15a shows the distribution of IFCs calculated via CSLD.



(a) (b)

Figure 3.15: Results for NaCl. (a) Distribution of IFCs calculated via CSLD. (b) Thermal conductivity versus temperature calculated using CSLD, LMD and the HNEMD method.

A majority of IFCs belong to 2- and 3-body clusters, with a few higher-order terms. To calculate lattice thermal conductivity, LMD simulations were performed for NaCl between 100 and 300 K.

3.2.1 Equilibrium MD

LMD was used to run a series of equilibrium (NVE) simulations, which were subsequently analyzed using the Green-Kubo method (Equation 3.21) to determine κ_L. For the $4 \times 4 \times 4$ system, 50 ensembles were used, while for other system sizes a minimum of 25 ensembles were used. All simulations were run for 500 ps with 1 fs timesteps. The results are shown in Table 3.2. While there is moderate variation between κ_L and system size, there is no discernible pattern, and in general all values of κ_L are in good agreement with experimental values.

One of the major drawbacks to this approach, however, is that a large amount of simulation time is needed to get the converged results shown in Table 3.2. In the case of the $4 \times 4 \times 4$ system, a total of 25 nanoseconds worth of simulation time was required. Even for the highly-optimized LMD package, this represents a significant amount of computation time. Additionally, as discussed in Section 2.2.1, there are a number of different ways to interpret

Temperature	Simulation Size	κ_L^{FD}	κ_L^{Avg}	κ_L^{Exp}
150 K	$4 \times 4 \times 4$	17.09 ± 1.21	17.54 ± 1.68	15.67
	$6 \times 6 \times 6$	14.71 ± 1.22	14.47 ± 2.13	
	$8 \times 8 \times 8$	13.43 ± 0.62	15.16 ± 1.53	
300 K	$4 \times 4 \times 4$	7.68 ± 0.50	6.86 ± 0.73	5.85
	$6 \times 6 \times 6$	6.06 ± 0.32	5.47 ± 0.79	
	$8 \times 8 \times 8$	6.80 ± 0.30	6.46 ± 0.68	

Table 3.2: Thermal conductivity for NaCl from GK simulations and experimental values. All κ_L values are in units of W/(m K). κ_L^{FD} values were calculated using the first-drop method while the κ_L^{Avg} values were calculated by averaging over the stable section of the integrated correlation function (see Section 2.2.1 for description of different methods).

the results of Equation 3.21. The results in Table 3.2 were determined using two methods; (i) first-drop (FD) method and (ii) the averaging method. For the most part, the values calculated using the averaging method are better than those calculated using the FD method. The fact that one must choose how to interpret the results, however, leaves a little to be desired.

3.2.2 HNEMD

To calculate κ_L via the HNEMD method, at least four different values for F_z (Equation 3.29) were used at a given T with a minimum of five ensembles, with system sizes ranging from 512 to 4096 atoms. The lengths of the simulations ranged from 100 ps to 1 ns and all used a timestep of 1 fs. The results obtained are shown for NaCl in Figure 3.15b. Very good agreement is seen between the calculated and experimental values across the entire temperature range tested. Figure 3.13 and Table 3.3 compare κ_L calculated across a range of system sizes and temperatures. No discernible system-size dependence is seen at the temperatures tested.

The HNEMD method also addresses the two main issues present in the Green-Kubo method.

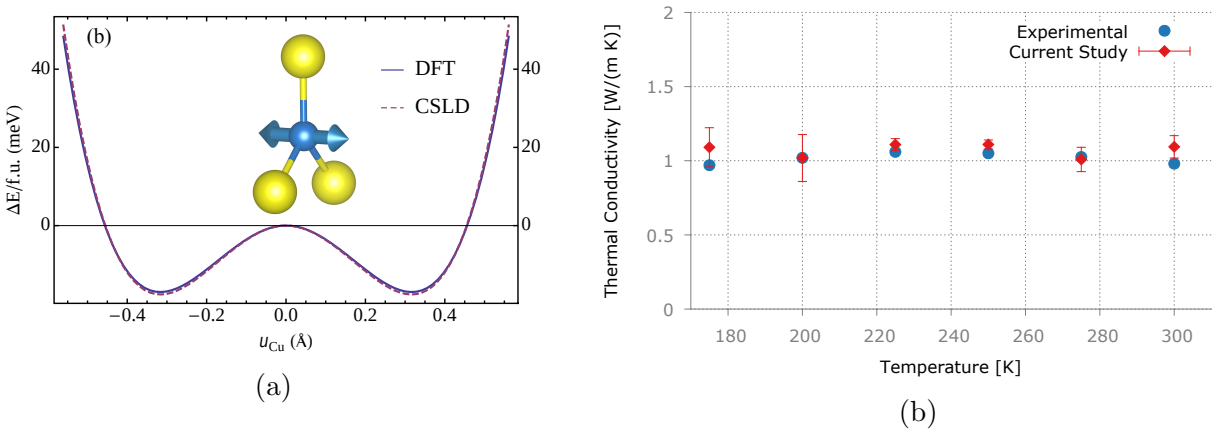
Temperature	Simulation Size	κ_L^{HNEMD}	κ_L^{Exp}
200 K	$4 \times 4 \times 4$	11.69 ± 2.51	11.29
	$6 \times 6 \times 6$	12.32 ± 2.43	
	$8 \times 8 \times 8$	11.29 ± 3.94	
300 K	$4 \times 4 \times 4$	6.39 ± 1.33	5.85
	$6 \times 6 \times 6$	7.41 ± 1.58	
	$8 \times 8 \times 8$	7.27 ± 0.64	

Table 3.3: Thermal conductivity for NaCl from HNEMD simulations and experimental values. All κ_L values are in units of W/(m K).

The first is that less total simulation time is needed. The $4 \times 4 \times 4$ results here were calculated in roughly half the simulation time needed for the Green-Kubo method. Additionally, Equation 3.29 can be used directly to calculate κ_L. This results in the HNEMD method requiring absolutely no emotional involvement. This is a huge advantage because (i) it lends itself to use in a fully-automated manner and (ii) results in a consistent, repeatable process that does not require any subjective interpretation of the data.

3.3 $Cu_{12}Sb_4S_{13}$

The final material is $Cu_{12}Sb_4S_{13}$, a parent compound for the earth-abundant natural mineral tetrahedrite, which was recently shown to be a high-performance thermoelectric[20]. An interesting property of this material is that it exhibits an extremely low κ_L, experimentally found to be $\lesssim 1$ W/(m K) in phase and compositionally pure samples[20]. Additionally, previous studies found several harmonically unstable phonon modes, pointing to very strong anharmonicity. $Cu_{12}Sb_4S_{13}$ has body-centered cubic (space group $I\bar{4}3m$) structure with 29 atoms in the primitive cell, a large number that complicates the computation of FCTs using existing methods. For example, there are 188 distinct atomic pairs within a radius of $a = 10.4$ Å, 116 triplets within $a/2$, etc. Taking into account the 3^n elements of each tensor, the number of unknown coefficients is very large (55584 in our setting including up to 6th-



(a) (b)

Figure 3.16: CSLD and LMD results for tetrahedrite. (a) Relative energy per formula unit of an unstable optical mode involving out-of-plane displacements of trigonally coordinated copper atoms (blue) bonded to sulfur (yellow sphere). DFT and CSLD are shown as solid and dashed lines, respectively. (b) Calculated κ_L for tetrahedrite across a range of temperatures. Experimental results from Lu et al. [20].

order terms). After symmetrization, this is reduced to $N = 3188$, which still represents a formidable numerical challenge.

A large number of non-zero FCT elements (2101) were obtained by CSLD. Figure 3.3b shows the overall accuracy of the model over a prediction set from *ab initio* MD snapshots at 300 K. The root-mean-square error of the predicted force components is 0.02 eV/Å, or 4%. Figure 3.16a shows the DFT potential energy surface (solid line) along an unstable Γ point mode involving displacement of trigonally coordinated Cu atoms (inset). The double-well behavior points to strong 4th-order anharmonicity. Our CSLD model (dashed line) is able to reproduce the potential energy to an absolute accuracy of 2 meV per atom.

The HNEMD method was used to calculate κ_L of $Cu_{12}Sb_4S_{13}$, employing the same approach as described for NaCl above. All simulations were done with a supercell of 464 atoms and a minimum of 4 separate external fields at each temperature. The HNEMD results are compared with the experimental κ_L from Lu et al. [20] with electronic contributions subtracted in Figure 3.16b. Once again, very good agreement is seen across the entire temperature range tested. This example shows that CSLD, coupled with LMD, extends the

accuracy of DFT to treat lattice dynamics of compounds with large, complex unit cells and strong anharmonic effects previously beyond the reach of non-empirical studies.

Bibliography

[1] S. Baroni, S. de Gironcoli, A. Dal Corso, and P. Giannozzi. Phonons and related crystal properties from density-functional perturbation theory. *Rev. Mod. Phys.*, 73(2):515–562, 2001.

[2] Stefano Baroni, Stefano de Gironcoli, Andrea Dal Corso, and Paolo Giannozzi. Phonons and related crystal properties from density-functional perturbation theory. *Rev. Mod. Phys.*, 73:515–562, Jul 2001. doi: 10.1103/RevModPhys.73.515. URL `http://link.aps.org/doi/10.1103/RevModPhys.73.515`.

[3] P. E. Blöchl. Projector augmented-wave method. *Phys. Rev. B*, 50:17953–17979, Dec 1994. doi: 10.1103/PhysRevB.50.17953. URL `http://link.aps.org/doi/10.1103/PhysRevB.50.17953`.

[4] D. A. Broido, M. Malorny, G. Birner, Natalio Mingo, and D. A. Stewart. Intrinsic lattice thermal conductivity of semiconductors from first principles. *Applied Physics Letters*, 91(23):231922, 2007. doi: http://dx.doi.org/10.1063/1.2822891. URL `http://scitation.aip.org/content/aip/journal/apl/91/23/10.1063/1.2822891`.

[5] E.J. Candes and M.B. Wakin. An introduction to compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):21–30, March 2008. ISSN 1053-5888. doi: 10.1109/MSP.2007.914731.

[6] Jianwei Che, Tahir Çağın, Weiqiao Deng, and William A. Goddard. Thermal conductivity of diamond and related materials from molecular dynamics simulations. *The Journal of Chemical Physics*, 113(16):6888–6900, 2000. doi: http://dx.doi.org/10.1063/1.1310223. URL `http://scitation.aip.org/content/aip/journal/jcp/113/16/10.1063/1.1310223`.

[7] Jianjun Dong, Otto F. Sankey, and Charles W. Myles. Theoretical study of the lattice thermal conductivity in ge framework semiconductors. *Phys. Rev. Lett.*, 86:2361–2364, Mar 2001. doi: 10.1103/PhysRevLett.86.2361. URL http://link.aps.org/doi/10.1103/PhysRevLett.86.2361.

[8] D.L. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *Information Theory, IEEE Transactions on*, 47(7):2845–2862, Nov 2001. ISSN 0018-9448. doi: 10.1109/18.959265.

[9] Keivan Esfarjani, Gang Chen, and Harold T. Stokes. Heat transport in silicon from first-principles calculations. *Phys. Rev. B*, 84:085204, Aug 2011. doi: 10.1103/PhysRevB.84.085204. URL http://link.aps.org/doi/10.1103/PhysRevB.84.085204.

[10] Paolo Giannozzi, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, Guido L Chiarotti, Matteo Cococcioni, Ismaila Dabo, Andrea Dal Corso, Stefano de Gironcoli, Stefano Fabris, Guido Fratesi, Ralph Gebauer, Uwe Gerstmann, Christos Gougoussis, Anton Kokalj, Michele Lazzeri, Layla Martin-Samos, Nicola Marzari, Francesco Mauri, Riccardo Mazzarello, Stefano Paolini, Alfredo Pasquarello, Lorenzo Paulatto, Carlo Sbraccia, Sandro Scandolo, Gabriele Sclauzero, Ari P Seitsonen, Alexander Smogunov, Paolo Umari, and Renata M Wentzcovitch. Quantum espresso: a modular and open-source software project for quantum simulations of materials. *Journal of Physics: Condensed Matter*, 21(39):395502, 2009. URL http://stacks.iop.org/0953-8984/21/i=39/a=395502.

[11] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, Ph. Ghosez, J.-Y. Raty, and D.C. Allan. First-principles computation of material properties: the {ABINIT} software project. *Computational Materials Science*, 25(3):478 – 492, 2002. ISSN 0927-0256. doi: http://dx.doi.org/10.1016/S0927-0256(02)00325-7. URL http://www.sciencedirect.com/science/article/pii/S0927025602003257.

[12] Melville S. Green. Markoff random processes and the statistical mechanics of time-dependent phenomena. ii. irreversible processes in fluids. *The Journal of Chemical Physics*, 22(3):398–413, 1954. doi: http://dx.doi.org/10.1063/1.1740082. URL `http://scitation.aip.org/content/aip/journal/jcp/22/3/10.1063/1.1740082`.

[13] K M. Ho, C L. Fu, and B N. Harmon. Vibrational Frequencies via Total-Energy Calculations - Applications to Transition-Metals. *Phys. Rev. B*, 29(4):1575–1587, 1984.

[14] S. Yip J. Li, L. Porter. Atomistic modeling of finite-temperature properties of crystalline b-sic ii. thermal conductivity and effects of point defects. *Journal of Nuclear Materials*, 255, 1998.

[15] FIZ Karlsruhe. Icsd web. URL `http://icsd.fiz-karlsruhe.de`.

[16] G. Kresse and J. Hafne. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47(558), 1993.

[17] Ryogo Kubo. Statistical-mechanical theory of irreversible processes. i. general theory and simple applications to magnetic and conduction problems. *Journal of the Physical Society of Japan*, 12(6):570–586, 1957. doi: 10.1143/JPSJ.12.570. URL `http://dx.doi.org/10.1143/JPSJ.12.570`.

[18] K Kunc and R M Martin. Ab Initio Force Constants of GaAs: A New Approach to Calculation of Phonons and Dielectric Properties. *Phys. Rev. Lett.*, 48:406–409, February 1982. doi: 10.1103/PhysRevLett.48.406.

[19] Wu Li, Jesús Carrete, Nebil A. Katcho, and Natalio Mingo. ShengBTE: a solver of the Boltzmann transport equation for phonons. *Comp. Phys. Commun.*, 185:1747–1758, 2014. doi: 10.1016/j.cpc.2014.02.015.

[20] Xu Lu, Donald T. Morelli, Yi Xia, Fei Zhou, Vidvuds Ozolins, Hang Chi, Xiaoyuan Zhou, and Ctirad Uher. High performance thermoelectricity in earth-abundant compounds based on natural mineral tetrahedrites. *Advanced Energy Materials*, 3(3):342–348, 2013. ISSN 1614-6840. doi: 10.1002/aenm.201200650. URL http://dx.doi.org/10.1002/aenm.201200650.

[21] A.J.H. McGaughey and M. Kaviany. Thermal conductivity decomposition and analysis using molecular dynamics simulations: Part ii. complex silica structures. *International Journal of Heat and Mass Transfer*, 47(8–9):1799 – 1816, 2004. ISSN 0017-9310. doi: http://dx.doi.org/10.1016/j.ijheatmasstransfer.2003.11.009. URL http://www.sciencedirect.com/science/article/pii/S0017931003006203.

[22] A.J.H. McGaughey and M. Kaviany. Phonon transport in molecular dynamics simulations: Formulation and thermal conductivity prediction. In Avram Bar-Cohen George A. Greene, James P. Hartnett† and Young I. Cho, editors, *Advances in Heat Transfer*, volume 39 of *Advances in Heat Transfer*, pages 169 – 255. Elsevier, 2006. doi: http://dx.doi.org/10.1016/S0065-2717(06)39002-8. URL http://www.sciencedirect.com/science/article/pii/S0065271706390028.

[23] M Omini and A Sparavigna. An iterative approach to the phonon boltzmann equation in the theory of thermal conductivity. *Physica B: Condensed Matter*, 212(2):101 – 112, 1995. ISSN 0921-4526. doi: http://dx.doi.org/10.1016/0921-4526(95)00016-3. URL http://www.sciencedirect.com/science/article/pii/0921452695000163.

[24] K. Parlinski, Z-Q Li, and Y. Kawazoe. First-Principles Determination of the Soft Mode in Cubic ZrO_2. *Phys. Rev. Lett.*, 78(21):4063–4066, May 1997.

[25] R. Peierls. Kinetic theory of heat conduction in crystals. *Ann. Physik*, 3:1055–1101, 1929.

[26] John P. Perdew, Kieron Burke, and Matthias Ernzerhof. Generalized gradient approximation made simple. *Phys. Rev. Lett.*, 77:3865–3868, Oct 1996. doi: 10.1103/PhysRevLett.77.3865. URL `http://link.aps.org/doi/10.1103/PhysRevLett.77.3865`.

[27] D. P. Sellan, E. S. Landry, J. E. Turney, A. J. H. McGaughey, and C. H. Amon. Size effects in molecular dynamics thermal conductivity predictions. *Phys. Rev. B*, 81:214305, Jun 2010. doi: 10.1103/PhysRevB.81.214305. URL `http://link.aps.org/doi/10.1103/PhysRevB.81.214305`.

[28] M.J. Verstraete and Z. Zanolli. Density functional perturbation theory. Lecture Notes of the 45th IFF Spring School "Computing Solids - Models, ab initio methods and supercomputing", 2014.

[29] H Wendel and Richard M Martin. Charge Density and Structural Properties of Covalent Semiconductors. *Phys. Rev. Lett.*, 40(14):950–953, April 1978.

CHAPTER 4

Performance

The most computationally expensive portion of any MD code is the calculation of interatomic forces. Just how expensive this is depends on how complicated Equation 2.3 actually is. As discussed in Section 2.2 of Chapter 3, this can range rather wildly. In this chapter, we focus on the particular case of optimizing Equation 2.28 and Equation 2.30 within the context of LMD.

Ultimately, the goal of this work is to devise an efficient method for solving the crystal potential so that it can be used as the Hamiltonian in a classical MD program. The solution should make it possible to simulate thousands of atoms on nanosecond timescales with runtimes on the order of hours, not days. Additionally, we look to exploit the massively parallel nature of GPU devices in an effort to get more performance per dollar, while also increasing performance per Watt.

1 Strategies & Hardware

When it comes to increasing the performance of a computationally expensive problem, there are two primary techniques that can be employed; (i) do work more *efficiently* and (ii) do more work *simultaneously*. The first technique is rather nebulous as it involves tuning an algorithm to be more efficient, and there are many ways to accomplish this. One way to make an algorithm more efficient is by reducing the number of floating point (mathematical) operations required to calculate the desired result. A trite example of this is the distributive property, which can be used to reduce the number of operations in the following example

$$(5 \times 3) + (5 \times 2) + (5 \times 4) = 45 \tag{4.1}$$

$$5 \times (3 + 2 + 4) = 45 \tag{4.2}$$

in which Equation 4.1 requires five operations while Equation 4.2 only requires three, but both produce identical results. Real problems will, of course, typically be far more complicated, but identifying ways like this to reduce the number of floating point operations can often yield significant performance improvements.

Another approach to increasing efficiency is through redesigning the algorithm such that the hardware is used more effectively. This can also be difficult to achieve, especially in a cross-platform manner, as hardware differences can range tremendously. Nonetheless, one of the most effective methods for increasing the performance of an algorithm, on current hardware, is by designing for memory locality. This is best understood by taking a brief detour to discuss the basic hardware layout and memory hierarchy of modern computers.

A modern computer can be boiled down to essentially three main components; (i) the processor (CPU), (ii) memory (RAM) and (iii) the hard drive (HDD). The CPU does all of the work that we request of the computer, the RAM holds temporary information needed by programs that are running and the HDD stores long term data that persists indefinitely. When a program is launched, the CPU loads the program from the HDD into RAM and begins executing it. When the program requests a piece of data it has to flow through the following stages of memory (these numbers are rough estimates based on current hardware)

1. HDD: > 1000 cycles
2. RAM: 100 cycles
3. Cache
    (a) Level 3 (L3): 50 cycles

(b) Level 2 (L2): 10 cycles

(c) Level 1 (L1): 5 cycles

4. Processor registers - 1 cycle

The number of 'cycles' refers to how many CPU cycles the CPU must wait for that memory operation to be complete, and the number of cycles a CPU can perform per second depends on the clock speed of a given CPU. Each level of caching also involves a reduction in storage capacity, with RAM on the order of gigabytes while caches (L1, L2, etc) are usually only a few megabytes or less. Without getting too caught up in the details, the closer the memory is to the processor registers, the faster it can be loaded, because the CPU doesn't have to wait as long for the data to be loaded. When memory is requested that is not in the cache–called a *cache miss*–the CPU must wait until the data is loaded. Forcing the CPU to wait while data is loaded is a large bottleneck that should be avoided for high-performance algorithms.

Caching mechanisms are typically quite simple. When data is requested, rather than just loading the desired value the CPU will load the desired value in addition to a *block* of data that is in the same region of memory. This whole block of data is then cached. Therefore, optimizing the algorithm to operate on data that is stored close together will result in a decreased number of cache misses, thereby increasing the performance of the algorithm.

The other way to increase performance is by doing more than one thing *simultaneously*, or in *parallel*. Most CPUs today contain at least two *cores*–each core can perform its own work independent of the others–while high-end CPUs can contain ten or more cores. Additionally, mutiple computers can be networked together to create a pool of thousands of cores. There are a variety of software libraries available to make developing such parallel applications easier. In general, these parallelization techniques can be broken down into two categories; (i) shared-memory and (ii) distributed-memory.

The shared-memory approach involves the use of so-called *threads*, each of which is capable of running independent operations. All threads belonging to a single *process*, however, are able

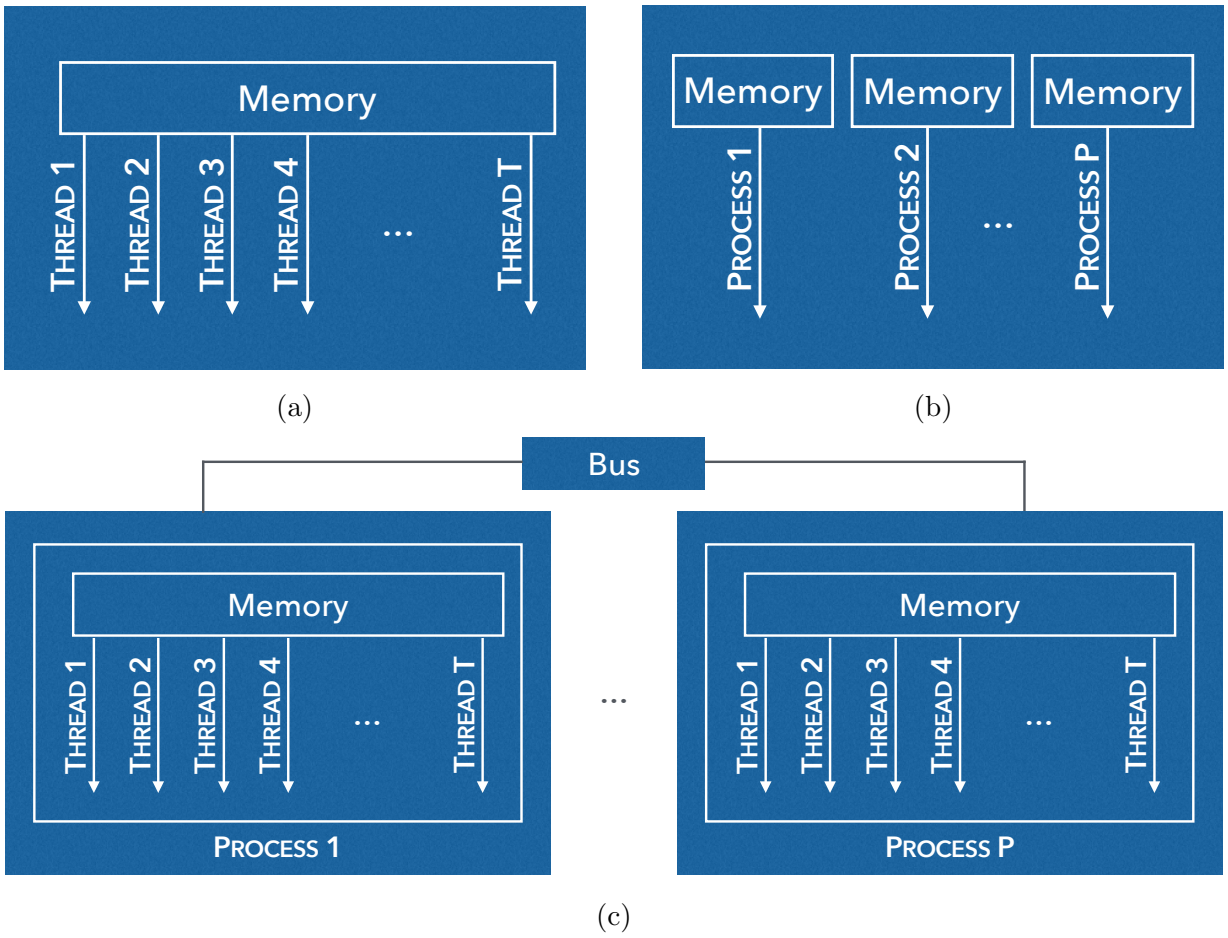


Figure 4.1: Basic parallelization techniques (a) shared-memory and (b) distributed-memory. Combining the two techniques (c) is also possible.

to access the same memory space–if one thread changes a variable in memory, all threads will able to see the modified variable instantly. This is convenient because memory doesn't need to be kept in sync, but it must be accessed carefully to avoid two or more threads writing to the same memory location at the same time–an occurrence known as a 'race condition'. This sort of behavior is undesirable and caution must be used to avoid such conflicts. 'OpenMP' is one example of a very popular shared-memory parallelization library. Figure 4.1a illustrates the basic layout of a shared-memory application, in which T independent threads belong to a single process and share the same memory space.

Distributed-memory applications differ quite substantially from shared-memory ones. In-

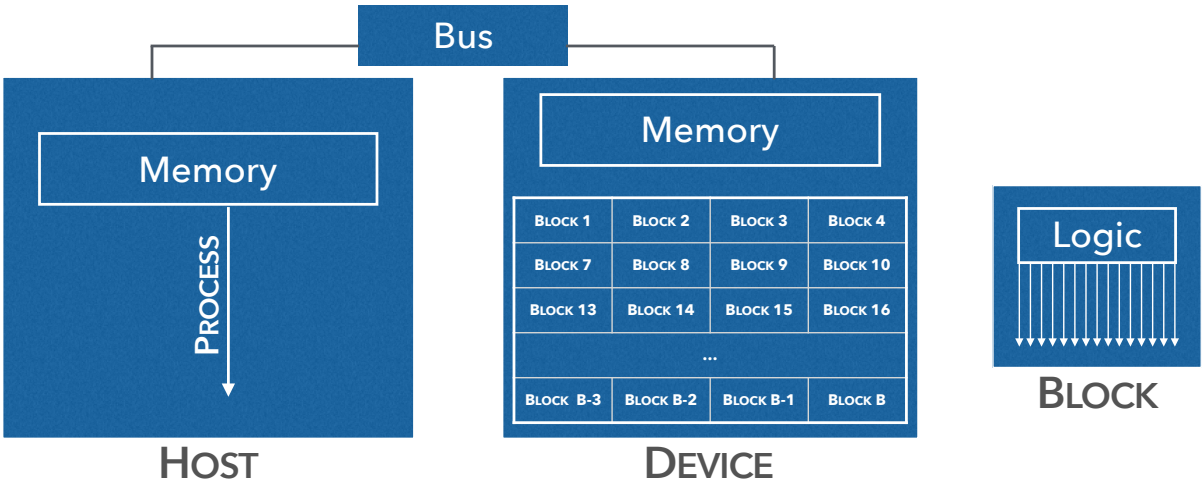


Figure 4.2: Simplified layout of a system involving a GPU (device). The GPU contains a large number of floating-point processors, but only a few logic processors. The number of processing block varies widely between devices. In order to maximize GPU utilization, SIMD should be employed.

stead of a single process containing multiple threads, a distributed-memory application will contain a group of multiple processes, each with their own (private) memory space. Like threads, each process can do its own work independent of the others, but in order to share memory from one process to another the memory must be explicitly copied between processes. 'MPI' is one of the most widely used methods for distributed-memory applications. Figure 4.1b illustrates the basic layout of a distributed-memory application, where P independent processes contain their own private memory space.

Distributed-memory applications have the added benefit of being able to run across multiple, physically distinct computers whereas shared-memory applications are limited to the number of CPU cores on a single computer. However, the two techniques can also be combined as illustrated in Figure 4.1c. In this mode of operation, processes running on physically distinct machines must communicate to each other through the use of a common connection, or bus. This bus can be a common network connection, or a specialized high-speed link, such as InfiniBand. The faster the interconnect, the quicker data can be shared between processes.

Recently, there has been growing interest in utilizing the massive computing capability

present in graphics processing units (GPU). Optimizing an algorithm for a GPU is quite different than the approaches just listed for the CPU-based parallelization techniques. To understand why, it is important to understand the basics of how a GPU works. Figure 4.2 depicts a typical GPU-based setup. In this setup, the *host* is where the program runs and where the main application memory resides–this includes the CPU and RAM. The *device* is the GPU and internally it is very different from a CPU. As mentioned, a CPU typically contains any where from two up to about 10 cores while a GPU can contain thousands of cores, which we'll refer to as threads. However, the threads on a GPU are very different from the cores on a CPU. The CPU cores run at much higher frequencies (meaning they can do more work per unit time) and each core contains all the hardware necessary for processing numerical and logic operations. The GPU threads, however, are grouped into *blocks*, where each block only has a single logic unit. The logic unit can only process one thread at a time which means that if two threads in a block try to do a different operation, one thread will have to wait for the other to finish. Therefore, in order to obtain maximum utilization on a GPU, it is necessary to have each thread doing the same operation (or 'instruction') but on different data. This technique is referred to as same-instruction multiple-data (SIMD) and is the key to unlocking the full performance of GPUs.

2 Notation

Before diving in, it is necessary first to define the problem and the associated notation that will be used to describe the various solutions. Much of the notation used throughout the remainder of this section relies on familiarity with sets and lists. A brief primer is provided in Appendix C.

Ultimately, the goal is to devise an efficient method for solving Equation 2.29 and Equation 2.30 for a system of N atoms. In an MD simulation we are concerned primarily with the interactions between atoms, and we do so by keeping track of various properties such as

displacements, forces and energies. For a system with N atoms, we will need to keep track of N energies, but for displacements and forces we need to keep track of three values per atom (for the x-, y-, and z-components). Therefore, atomic displacements are stored as an array of $3N$ elements. To make accessing values in a $3N$ array easier, we define an indexing function, Ω, that maps the displacement of atom a in direction α to a single value as

$$\Omega(a, \alpha) = 3a + \alpha = \omega_{3a+\alpha} \tag{4.3}$$

where α is one of the three Cartesian axes ($x = 0$, $y = 1$, $z = 2$) and $0 \leq \omega_i < 3N$. Under this notation, a given displacement-index, ω_i, corresponds to atom $a = \text{floor}(\omega_i/3)$ and direction $\alpha = \omega \mod (3)$, with the value of a certain displacement given by $\boldsymbol{u}[\omega_i]$, where $\boldsymbol{u}$ is a list of displacements of length $3N$. For convenience, ω_i will sometimes be referred to as an 'atom-direction'.

We then also define a function that returns an atom-number given an atom-direction, $\Gamma(\omega_i)$, as

$$\Gamma(\omega_i) = \left\lfloor \frac{\omega_i}{3} \right\rfloor \tag{4.4}$$

Additionally, given a set (or list) of atom-directions as an input, this function returns a set (or list) of atom-numbers. More specifically

$$\Gamma\left(\left\{\omega_i, \omega_j\right\}\right) = \left\{\Gamma(\omega_i), \Gamma\left(\omega_j\right)\right\} \tag{4.5}$$

$$\Gamma\left(\left[\omega_i, \omega_i\right]\right) = \left[\Gamma(\omega_i), \Gamma(\omega_i)\right] \tag{4.6}$$

The next step is to expand the first few terms of Equation 2.29, which gives

$$V = \Phi_{0_x 0_x} u_{0_x} u_{0_x} + \Phi_{0_x 0_y} u_{0_x} u_{0_y} + \Phi_{0_x 0_z} u_{0_x} u_{0_z} + ... \tag{4.7}$$

which can more compactly be written as

$$V = v_0 + v_1 + v_2 + ... + v_{T-1} \quad \text{where,} \tag{4.8}$$

$$v_0 = \Phi_{0_x 0_x} u_{0_x} u_{0_x},\ v_1 = \Phi_{0_x 0_y} u_{0_x} u_{0_y},\ ... \tag{4.9}$$

We then define the list $\boldsymbol{v}$ as the collection of all v_i

$$\boldsymbol{v} = \left[v_0, v_1, ..., v_{T-1}\right] \tag{4.10}$$

where the number of items in $\boldsymbol{v}$ is $|\boldsymbol{v}| = T$. Note that we define this as a list and not a set because the values v_i are not guaranteed to be unique.

Next, we define the set of distinct, non-zero force constants, $\boldsymbol{\Phi}$, for a given system as

$$\boldsymbol{\Phi} = \left(\Phi_0, \Phi_1, ..., \Phi_{P-1}\right) \tag{4.11}$$

where the total number of distinct, non-zero force constants is given by $|\boldsymbol{\Phi}| = P$, and in general $0 < P \leq T$. We also define the function $\boldsymbol{\phi}\,(i)$ that maps from an index i in the range $0 \leq i < T$ to the unique force constant in $\boldsymbol{\Phi}$. Equation 4.9 can now be rewritten as

$$v_0 = \boldsymbol{\phi}(0)\,\boldsymbol{u}[\omega_0]\boldsymbol{u}[\omega_0],\ v_1 = \boldsymbol{\phi}(1)\,\boldsymbol{u}[\omega_0]\boldsymbol{u}[\omega_1],\ ... \tag{4.12}$$

With this notation it is now possible to convert the terms in Equation 4.10 and Equation 4.12 into a series of lists consisting of unique indexing values needed to calculate a specific product Equation 4.10

$$\mathbf{v}_0 = [\omega_0, \omega_0],\ \mathbf{v}_1 = [\omega_0, \omega_1],\ ... \tag{4.13}$$

where $0 \leq \omega_i < 3N$. Note that $\mathbf{v}_i$ is defined as list and not a set because improper clusters will contain duplicate ω_i. Also, the number of sites in cluster i is given by $|\mathbf{v}_i|$. For a given i in the range $0 \leq i < T$, $\mathbf{v}_i$ can be used to directly calculate the corresponding value of v_i as

$$v_i = \boldsymbol{\phi}(i) \prod_{j=0}^{|\mathbf{v}_i|-1} \boldsymbol{u}\left[\mathbf{v}_i[j]\right] \tag{4.14}$$

and the distinct set of all $\mathbf{v}_i$ is

$$\mathbb{V} = \left(\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{T-1}\right) \tag{4.15}$$

Sometimes it will be necessary to know how many times a specific atom, a, or atom-direction, a_α, appears in a list of atom-directions $\mathbf{s}$. The two forms are defined as

$$\mathbf{\Lambda}\left(\mathbf{s}, a\right) = \text{count } a \text{ in } \Gamma\left(\mathbf{s}\right) \tag{4.16}$$

$$\mathbf{\Lambda}\left(\mathbf{s}, a, \alpha\right) = \text{count } \Omega\left(a, \alpha\right) \text{ in } \mathbf{s} \tag{4.17}$$

$$\tag{4.18}$$

The following examples illustrate the behavior in more detail

$$\begin{aligned}
\mathbf{s} &= \left[0, 3\right] \\
\mathbf{\Lambda}\left(\mathbf{s}, 0\right) &\rightarrow 1 \\
\mathbf{\Lambda}\left(\mathbf{s}, 2\right) &\rightarrow 0 \\
\mathbf{\Lambda}\left(\mathbf{s}, 0, 0\right) &\rightarrow 1 \\
\mathbf{\Lambda}\left(\mathbf{s}, 0, 1\right) &\rightarrow 0
\end{aligned}$$

Finally, we will represent the total number of atoms in the system as N, where a given atom a is in the range $0 \leq a < N$. We can then define all of the atoms as a set of integers, $\mathbb{N}$, as

$$\mathbb{N} = \left\{a : a \in \mathbf{Z} \ \wedge \ 0 \leq a < N\right\} \tag{4.19}$$

where $\mathbf{Z}$ is the set of all integers.

3 Implementations

Previously, a variety of different techniques and hardware that can be used to increase the performance of a demanding computational problem was discussed. The following sections cover different approaches taken in optimizing the force-energy problem in LMD.

3.1 Atom Decomposition

Perhaps the simplest form of parallelization in classical MD codes is the so-called 'atom decomposition' (AD) method [1]. This method is a type of "divide and conquer" approach, as it involves dividing the system into smaller groups, each of which can be processed independently. Since each group is independent, all groups can be worked on simultaneously. For a system of N atoms divided into G groups, each group g, in the range $0 \leq g < G$, is tasked with calculating the forces and energies of N/G atoms. Parallelization is achieved when $G > 1$ and the maximum amount of parallelization possible is reached when $G = N$.

The first step requires partitioning the atoms in the system into groups. There are actually many ways to go about doing this. In a non-heterogeneous environment, for example, it may be beneficial for certain groups to contain more atoms than others due to differences in computing power. Additionally, when N is not perfectly divisible by G, some groups will contain fewer atoms than others. For simplicity, however, we will consider the case where N is perfectly divisible by G, such that every group has exactly N/G atoms for which it is responsible. We can then define the groups as subsets of all atoms, $a \in \mathbb{N}$, where each group g is defined as a set of atoms $\mathbb{N}^g$

$$\mathbb{N}^g = \left\{ P\left(a, g\right) : a \in \mathbb{N} \right\} \text{ where,} \tag{4.20}$$

$$P\left(a, g\right) = a \bmod \left(\frac{N}{G}\right) + g\left(\frac{N}{G}\right) \tag{4.21}$$

and

$$\mathbb{N}^0 \cup \mathbb{N}^1 \cup ... \cup \mathbb{N}^{G-1} = \mathbb{N} \tag{4.22}$$

The terms in Equation 4.15 can then be partitioned into these groups as well, denoted as $\mathbb{V}^g$, where

$$\mathbb{V}^g = \left\{ \mathbf{v}_i : 0 \leq i < T \ \wedge \ \Gamma\left(\mathbf{v}_i\right) \cap \mathbb{N}^g \neq \varnothing \right\} \tag{4.23}$$

Each set $\mathbb{V}^g$ will contain at most T elements, but in practice $|\mathbb{V}^g| < T$.

The potential energy of a given atom a in group g, can then be calculated as

$$V_a^g = \sum_{i=0}^{|\mathbb{V}^g|-1} \frac{\boldsymbol{\Lambda}\left(\mathbb{V}_i^g, a\right)}{|\mathbb{V}_i^g|} \, \boldsymbol{\phi}_g\left(i\right) \prod_{j=0}^{|\mathbb{V}_i^g|-1} \boldsymbol{u}\left[\mathbb{V}_i^g[j]\right] \tag{4.24}$$

where $\mathbb{V}_i^g = \mathbb{V}^g[i]$ and $\boldsymbol{\phi}_g$ maps to the correct force constant in $\boldsymbol{\Phi}$. If atom a does not appear in the list $\mathbb{V}_i^g$ then $\boldsymbol{\Lambda} = 0$. Otherwise, for proper clusters, $\boldsymbol{\Lambda} = 1$, which will result in the potential energy being partitioned evenly to all sites. In the case of an improper cluster, $\boldsymbol{\Lambda} > 1$ and the fraction $\boldsymbol{\Lambda}/|\mathbb{V}_i^g|$ will ensure that the energy is partitioned correctly.

Finally, we must derive an expression for the force. To do so, we start with Equation 2.30. The result is actually quite similar to Equation 4.24, with the exception that we must divide the cluster energy by the atom's displacement *and* we do not want to partition the energy. Additionally, to calculate the force on atom a in direction α, we need to filter the terms in $\mathbb{V}_g$ such that only those that contain atom-direction $\omega_i = \Omega\left(a, \alpha\right)$ are included.

$$F^g_{a_\alpha} = -\frac{1}{\boldsymbol{u}\left[\Omega\left(a,\alpha\right)\right]} \sum_{i=0}^{|\mathbb{V}^g|-1} \boldsymbol{\Lambda}\left(\mathbb{V}^g_i, a, \alpha\right)\ \boldsymbol{\phi}_g(i) \prod_{j=0}^{|\mathbb{V}^g_i|-1} \boldsymbol{u}\left[\mathbb{V}^g_i[j]\right] \tag{4.25}$$

This method is ideally suited for use with MPI because each process can be assigned a single group g. Additionally, the sums in Equation 4.24 and Equation 4.25 can all, with a little work, be done in parallel as well, lending themselves to parallelization with OpenMP. This MPI-OpenMP approach has been implemented in LMD and the results are discussed later in Section 4.

3.1.1 Decomposed Forces

For thermal conductivity calculations utilizing the Green-Kubo or HNEMD method, it is necessary to calculate the so-called *decomposed force*, or $\mathbf{F}_{ab}$, which is the force on atom a due to atom b. Mathematically, it is given by

$$\mathbf{F}_{ab} = -\frac{\partial e_b}{\partial u_a} \tag{4.26}$$

In simulations, this is stored as a list with $3N^2$ elements[1], because it is really the force on atom a in direction α due to atom b, $F_{a_\alpha,b}$. In order to calculate this value, we must modify Equation 4.25.

$$F^g_{a_\alpha,b} = -\frac{1}{\boldsymbol{u}\left[\Omega\left(a,\alpha\right)\right]} \sum_{i=0}^{|\mathbb{V}^g|-1} \frac{\boldsymbol{\Lambda}\left(\mathbb{V}^g_i, b\right)}{|\mathbb{V}^g_i|}\ \boldsymbol{\phi}_g(i) \prod_{j=0}^{|\mathbb{V}^g_i|-1} \boldsymbol{u}\left[\mathbb{V}^g_i[j]\right] \tag{4.27}$$

Additionally, $|\mathbb{V}_g\left[i\right]|$ has reappeared in the denominator because the energy needs to be partitioned evenly among sites.

Finally, the total force on atom a in direction α can be calculated directly from the decom-

[1]There are ways around this, as will be discussed later

posed forces as

$$F_{a_\alpha} = \sum_{i=0}^{N-1} F_{a_\alpha,i} \tag{4.28}$$

3.2 Potential Decomposition

The task-parallel approach of the atom-decomposition method is well suited for the MPI/OpenMP models of parallelization, however, this approach does not translate well to the GPU. The reason for this is due primarily to the hardware design of current GPUs. As discussed previously in Section 1, GPUs are designed for crunching numbers. Excessive branching and constant memory requests can lead to tremendous performance penalties, which is why many CPU-based algorithms do not translate directly to GPUs. Another reason that typical algorithms do not directly translate to a GPU is because they are not parallel enough. In the case of the AD method, the maximum level of parallelization is equal to the number of atoms. A system with hundreds of atoms would be unable to meet the level of parallelization possible on current GPU hardware[2].

A common technique used to exploit the massively-parallel capabilities of a GPU is to convert the problem from one that is task-parallel to one that is data-parallel. In a data-parallel environment, parallelism in the *data* must be exploited, which can sometimes be difficult to identify. In the case of the force-energy problem in LMD, this is best done by breaking the problem down into smaller chunks. In each 'chunk' the goal is to *decrease* the complexity (same instruction) while simultaneously *increasing* the parallelism (multiple data)–this is how to achieve SIMD. The result is a two-stage process for solving the force-energy problem in LMD, where each step is completed sequentially but each step is as SIMD as possible. Since this data-parallel approach involves decomposing the potential into smaller, bite-size chunks, this method is referred to as potential decomposition (PD).

[2]Current high-end GPUs contain thousands of threads.

In the first step, the individual elements of Equation 4.8 are calculated. This involves each thread performing a single product as shown in Equation 4.14, with threads ranging from $0 \leq i < T$. The total number of energy terms is T, therefore this stage is parallel up to T. SIMD is achieved in this stage because, in general $T \gg N$, which increases parallelism, and the operations (instructions) are all the same. Ignoring individual forces and energies, the total potential energy can simply be calculated as as sum over Equation 4.10

$$V = \sum_{i=0}^{|\boldsymbol{v}|-1} \boldsymbol{v}[i] \tag{4.29}$$

However, per-atom forces are necessary for MD, therefore we need a second step. To make this second step as simple as possible, we need to construct a list that holds the terms Equation 4.10 for a given atom-direction, like so

$$\boldsymbol{v}_{a_\alpha} = \left\{ v_i : v_i \in \boldsymbol{v} \ \wedge \ \boldsymbol{\Lambda}\left(\mathbf{v}_i, a, \alpha\right) \geq 1 \right\} \tag{4.30}$$

In words we can say that $\boldsymbol{v}_{a_\alpha}$ contains all of the products in Equation 4.10 for which the displacement of atom a in direction α contributes at least once. The equation for calculating the force on atom a in direction α then becomes

$$F_{a_\alpha} = -\frac{1}{\boldsymbol{u}\left[\Omega(a,\alpha)\right]} \sum_{i=0}^{|\boldsymbol{v}_{a_\alpha}|-1} \boldsymbol{\Lambda}\left(\boldsymbol{v}_{a_\alpha}, a\right) \boldsymbol{v}_{a_\alpha}\left[i\right] \tag{4.31}$$

which is simply a sum of the terms in $\boldsymbol{v}_{a_\alpha}$, with $\boldsymbol{\Lambda}$ adding adding a multiplicity. Since this step requires the terms in $\boldsymbol{v}_{a_\alpha}$, the two two steps must be done sequentially.

Without doing anything fancy, this step is $3N$ parallel. However, in practice it is quite easy to parallelize over the summation in Equation 4.31 and then do a quick parallel reduction to get the final value of the force. This increases the parallelization to $3NW$, where W

is a device-dependent parameter that typically yields optimal performance when chosen somewhere between 32 and 256.

The whole process is illustrated in Figure 4.3 and can be summarized thusly. Stage 1 involves calculating all of the products in Equation 4.7 by evaluating Equation 4.14 and storing the results into the list $\boldsymbol{v}$ (Equation 4.10). All T products can be worked on by independent *work items*, giving this stage a huge level of parallelization. Once the products have been calculated, we move to Stage 2. In Stage 2, the problem is divided into N groups, which are then subdivided into 3 groups of W work items. Each group of W work items computes the sum of Equation 4.31, for a given atom a and direction α, operating on only a subset of the total items in $\boldsymbol{v}_{a_\alpha}$. The partial sums are then reduced to a final value for F_{a_α}. This final reduction takes a total of $\log_2(W)$ operations to complete. As the parallelization relating to W is essentially an "unrolling" of the summation in Equation 4.31, it follows that $W < |\boldsymbol{v}_{a_\alpha}|$.

3.2.1 Decomposed Forces

To calculate $\mathbf{F}_{ab}$ we must define a few new subsets. First, we must define the subset of $\boldsymbol{v}$ (Equation 4.10) that contains terms contributing to a given $F_{a_\alpha,b}$. This involves including terms that (i) contain at least one contribution from the displacement of atom a in direction α and (ii) have a contribution from the displacement of atom b in *any* direction. To satisfy (i), we need to identify all $\mathbf{v}_i$ in $\mathbb{V}$ containing at least one occurrence of index $\omega_{a_\alpha} = \Omega(a, \alpha)$. The following set contains all indicies i corresponding to elements, $\mathbf{v}_i$, in $\mathbb{V}$ that contain ω_{a_α}

$$\mathbb{A} = \left(i \ : \ 0 \leq i < T \ \wedge \ \mathbf{\Lambda}\left(\mathbf{v}_i, a, \alpha\right) \geq 1 \right) \tag{4.32}$$

To satisfy (ii), we identify all $\mathbf{v}_i$ in $\mathbb{V}$ containing at least one contribution from atom b. The following set contains the indicies containing clusters with atom b

$$\mathbb{B} = \left(i \ : \ 0 \leq i < T \ \wedge \ \mathbf{\Lambda}\left(\mathbf{v}_i, b\right) \geq 1 \right) \tag{4.33}$$

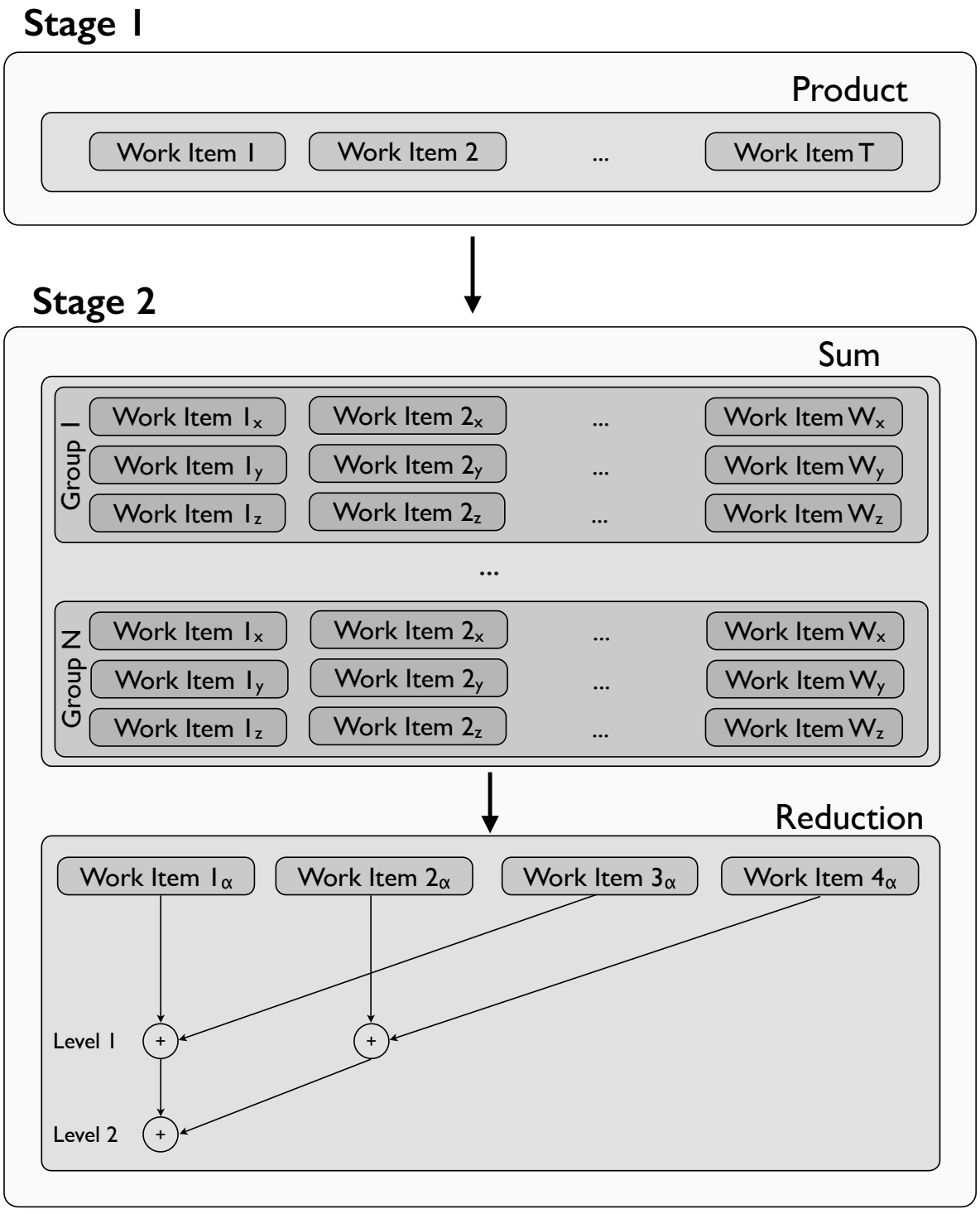


Figure 4.3: Overview of the potential decomposition method. Stage 1 involves calculating every product in the potential. Once complete, Stage 2 involves calculating the per-atom forces and energies. To increase parallelization, each work group can be increased to include W work items. When $W > 1$, a final reduction step is needed to calculate the final values.

The intersection of these two sets, $\mathbb{I}_{a_\alpha,b}$, gives the indicies needed to calculate $F_{a_\alpha,b}$

$$\mathbb{I}_{a_\alpha,b} = \mathbb{A} \cap \mathbb{B} \tag{4.34}$$

Now we have everything necessary to calculate $F_{a_\alpha,b}$

$$F_{a_\alpha,b} = -\frac{1}{\boldsymbol{u}\left[\Omega\left(a,\alpha\right)\right]} \sum_{i}^{\mathbb{I}_{a_\alpha,b}} \frac{\boldsymbol{\Lambda}\left(\mathbb{V}\left[i\right],b\right)}{\left|\mathbb{V}\left[i\right]\right|} \boldsymbol{v}\left[i\right] \tag{4.35}$$

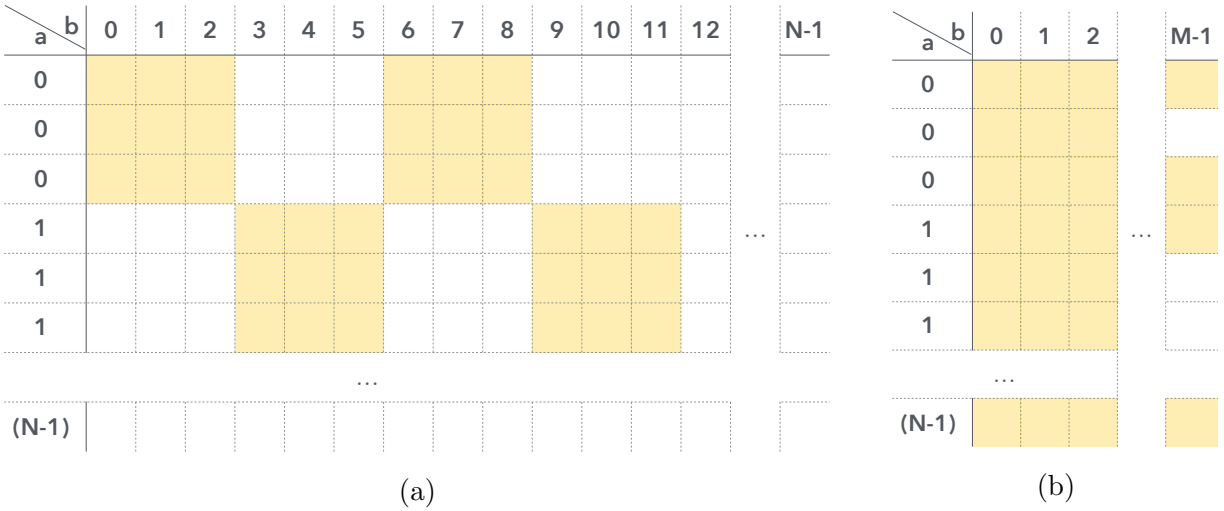
The potential energy can also be calculated easily

$$V_{a_\alpha,b} = \sum_{i}^{\mathbb{I}_{a_\alpha,b}} \frac{\boldsymbol{\Lambda}\left(\mathbb{V}\left[i\right],b\right)}{\left|\mathbb{V}\left[i\right]\right|} \boldsymbol{v}\left[i\right] \tag{4.36}$$

To implement this, each work group calculates a single force component as illustrated in Figure 4.3. Work items within a single group compute the sum shown in Equation 4.35. Once all values of $F_{a_\alpha,b}$, for a fixed a_α, have been computed in a single work group, the final force, F_{a_α}, is computed using Equation 4.28. The total potential energy of atom a is then calculated as

$$V_a = \sum_{\alpha=0}^{2} \sum_{b=0}^{N-1} V_{a_\alpha,b} \tag{4.37}$$

A final comment must be made regarding the memory requirements of $\mathbf{F}_{ab}$. The straightforward way to store these values is by allocating $3N^2$ memory locations. With this approach, however, systems with more than a few thousand atoms begin to require vast amounts of memory–pushing what is available on all but the highest-end GPUs. Fortunately, we can take advantage of the fact that $\mathbf{F}_{ab}$ is generally quite sparse, even for systems with complex unit cells. This property makes it possible to use a compressed representation of $\mathbf{F}_{ab}$. To make accessing accessing elements of $\mathbf{F}_{ab}$ relatively easy, we have employed a fixed-width, compressed $\mathbf{F}_{ab}$ that only requires $3NM$ memory locations, where $M < N$.



(a) (b)

Figure 4.4: Visual representation of $\mathbf{F}_{ab}$, where (a) is the full (uncompressed) matrix with N columns and (b) the compressed form with M columns, where $M \leq N$.

Figure 4.4a demonstrates the matrix-like layout of the uncompressed $\mathbf{F}_{ab}$, for a fictitious system where the colored squares indicate non-zero values. M is computed as the number of non-zero columns in the row with the *maximum* number of non-zero columns. This is not the most memory efficient storage method due to the fact that rows shorter than M will be zero-padded to length M. The advantage to this design, however, is that rows can easily be accessed in a random manner.

4 Results

The AD methods was implemented using MPI as the method of parallelization. All performance data for the AD method was collected by running across multiple threads (CPU cores) as well as physical nodes (multiple CPUs). Two versions of the PD algorithm were implemented; one as a CPU-only version that was parallelized using OpenMP, the other as a GPU-only implementation parallelized with OpenCL.

Figure 4.5 shows how well the two different CPU-based implementations scale with the level of parallelization. The AD method sees near perfect, one-to-one scaling, while the PD

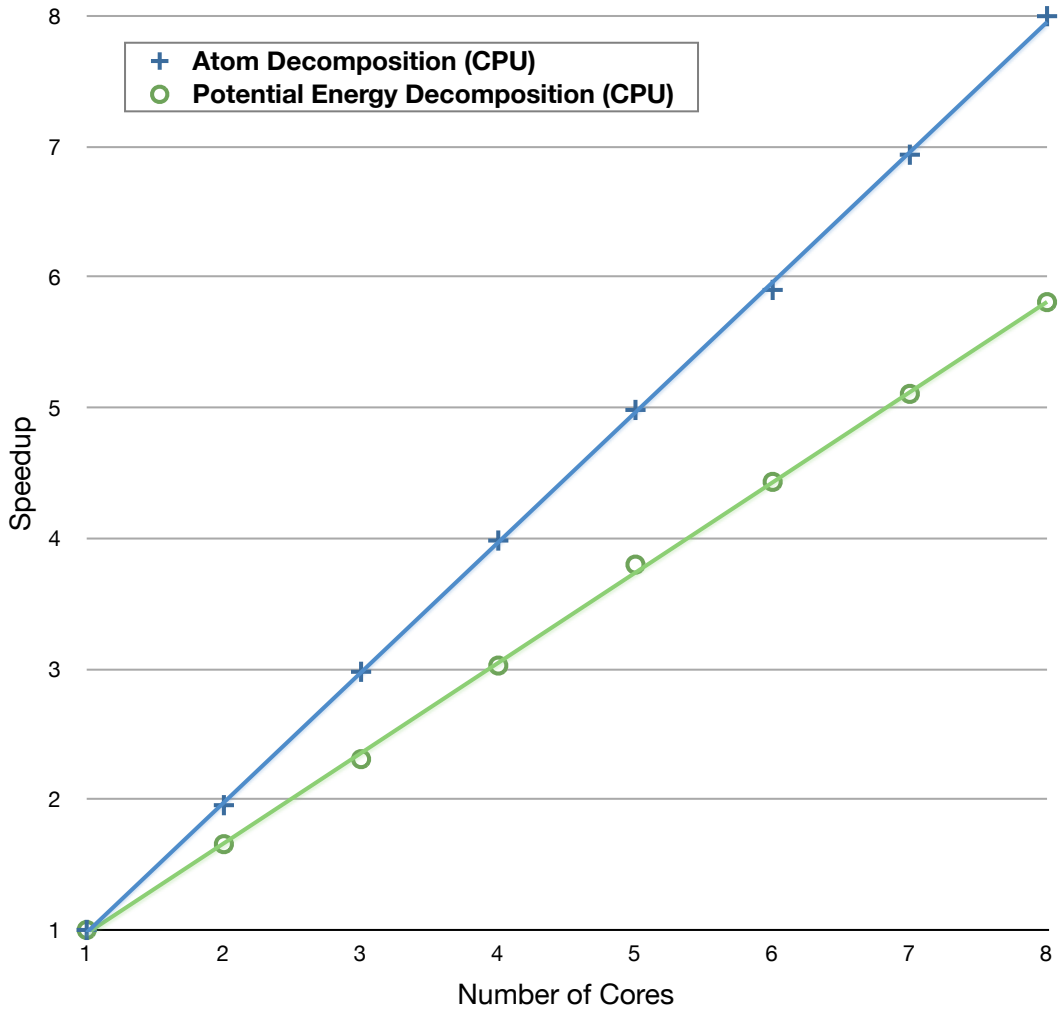


Figure 4.5: Speedup of the two CPU-based implementations. The AD method achieves near perfect, one-to-one speedup, while the PD method does not.

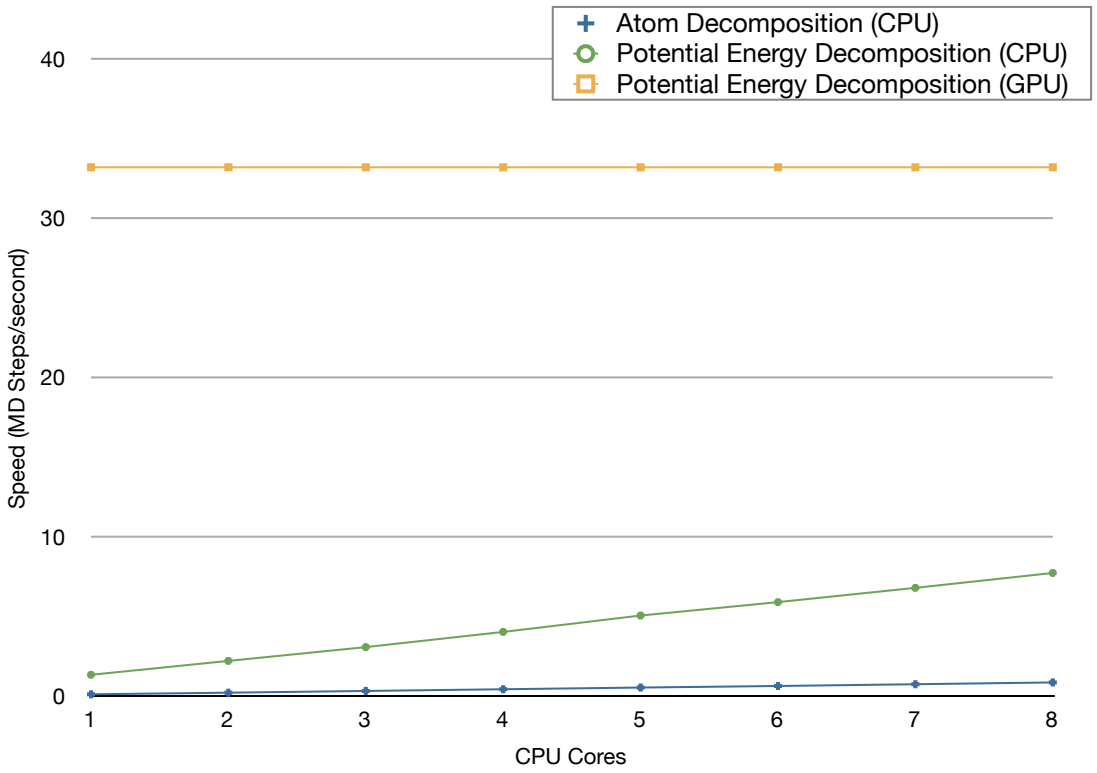


Figure 4.6: Relative speed of the three implementations on a 1440 atom NaCl system. The GPU-based PD on a single GPU outperforms both CPU-based methods. The GPU used was an NVIDIA Tesla C2050 and the CPU was a 4-core Intel Xeon E5520 (2.27GHz).

method does not. The deviation of the PD from the ideal behavior is likely due to the fact that the two stages scale differently. Figure 4.6, however, illustrates just how much more efficient the PD method is compared to the AD method. On a single CPU (4 cores), the CPU-based PD implementation is nearly 10 times faster than the CPU-based AD implementation, while the GPU-based PD implementation is just over 8 times faster than the CPU-based PD. Assuming linear scaling, extrapolating the CPU-based implementation in Figure 4.6, 36 cores (9 CPUs) for the CPU-based PD implementation and 313 cores (79 CPUs) for the CPU-base AD method would be required to match the performance of the single GPU-based PD implementation.

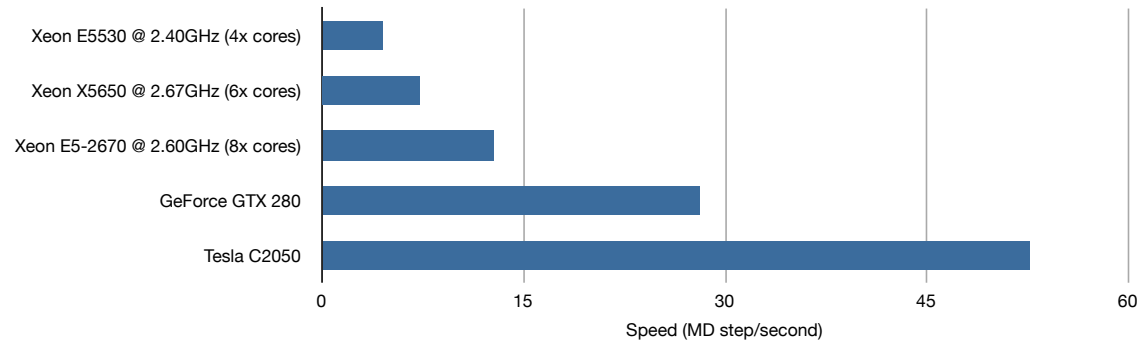


Figure 4.7: Performance of the PD on a variety of CPUs and GPUs for a 1440 atom NaCl system. All GPUs tested outperformed the CPUs by a large margin.

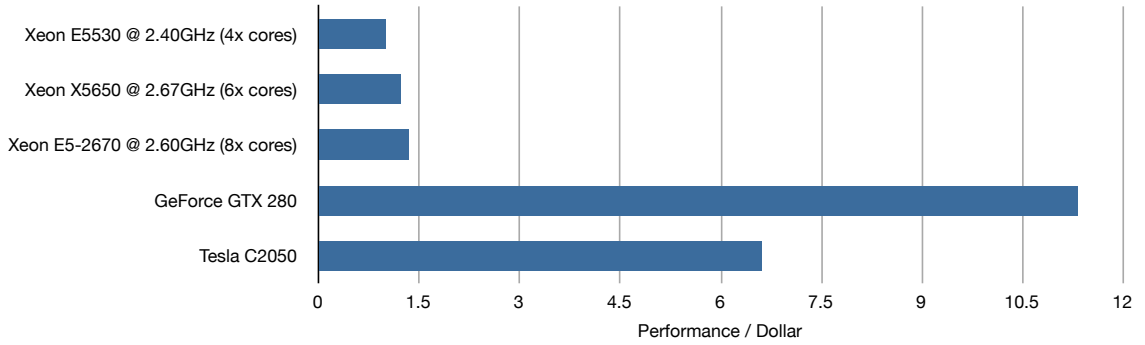


Figure 4.8: Performance per dollar for a 1440 atom NaCl system. Pricing data for each device based on current retail price retrieved from the internet.

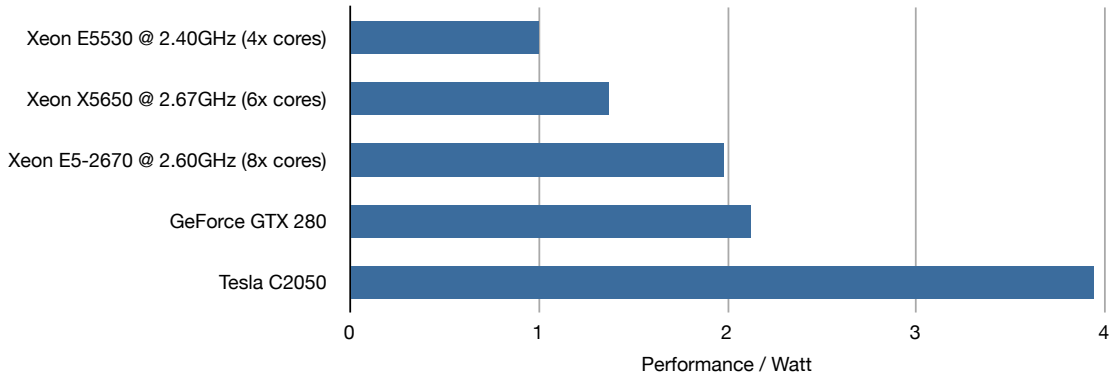


Figure 4.9: Performance per Watt for a 1440 atom NaCl system. All power ratings are based off manufacturers maximum listed TDP.

We tested the performance of the PD method on a variety of other CPUs and GPUs as well. The results are shown in Figure 4.7. Once again both GPUs outperformed all CPUs tested. The Tesla C2050 outperformed the best CPU by a factor of more than 4. Additionally, the performance per dollar is shown in Figure 4.8. The consumer-level, gaming GPU is the big winner here, beating out the best CPU by a factor of more than 8. It is also important to note that we only use the price of the CPU, which does not include the cost of the memory, whereas the cost of the GPU does include memory. Factoring in the cost of current generation DDR3 memory, for example, would further increase the separation between the CPU and GPU results here. Finally, Figure 4.9 highlights the performance per Watt. The GPU gains here are less significant over the CPU, with the best GPU coming in just under 2 times that of the best CPU.

Bibliography

[1] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1 – 19, 1995. ISSN 0021-9991. doi: http://dx.doi.org/10.1006/jcph.1995.1039. URL `http://www.sciencedirect.com/science/article/pii/S002199918571039X`.

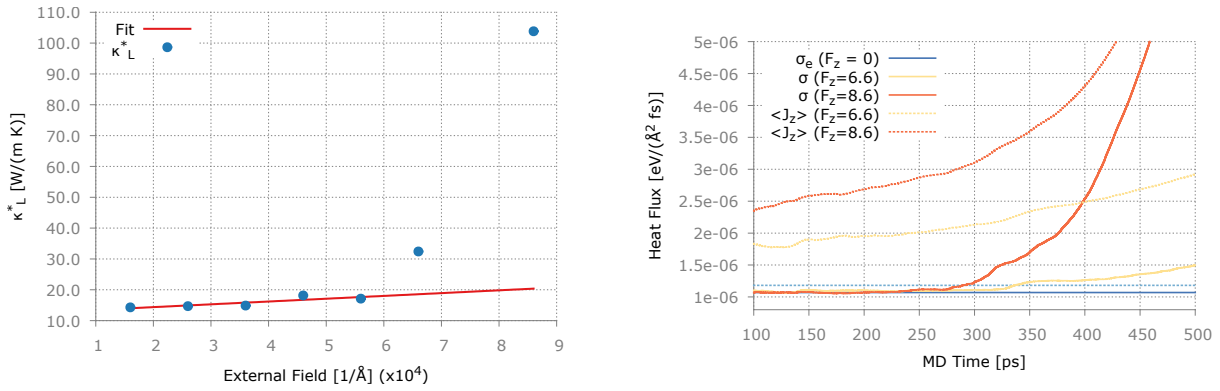
CHAPTER 5

Future Work

1 Automatic HNEMD

The key parameter in the HNEMD method is the external field, $\mathbf{F}_e$ that is applied to the system. As mentioned in Section 2.2.4, this field has the effect of driving higher energy (hotter) particles with the field and lower energy (colder) particles against it. This in turn results in a non-zero average heat flux. For small external fields, the system will remain within the linear response regime. However, large fields will drive the system into the non-linear regime in which Equation 3.26 is no longer valid. Therefore, one must choose $\mathbf{F}_e$ judiciously such that the system remains in the linear regime. Determining an appropriate field, however, can be a somewhat tedious task, involving lots of trial and error.

The non-linear behavior can be identified in a number of ways. Figure 5.1 illustrates the behavior observed when plotting κ_L^* against applied external field. In this case, the rightmost points falls well outside the linear fit (red line) of the other, smaller fields, indicating a departure from the linear regime. Therefore, the values of κ_L^* corresponding to these high-field points must be excluded from the fitting procedure. While it is easy to identify the non-linear behavior using this technique, it has a number of draw backs. First, it doesn't lend itself very well to automation, because it takes manual intervention to identify the outliers. Second, and more importantly, the non-linear points correspond to wasted computation. Therefore, an ideal solution would be one in which the system could *automatically* determine appropriate $\mathbf{F}_e$ values.



(a) (b)

Figure 5.1: (a) Reduced kappa κ_L^* versus external field F_z. The two right-most points lay outside the linear regime. (b) Behavior of the average heat flux and heat flux standard deviation for large fields. The blue dashed line is $1.1\sigma_e$.

In order to automate the process though, there needs to be some value that can be used to identify when the system has entered into the non-linear region. The most obvious value, the heat flux (more specifically the average heat flux), is not the best choice because it is difficult to know when the average begins to deviate from its expected value. Consider Figure 5.1b in which the heat flux running average is plotted over the course of the simulation for various values of external field F_z (dashed yellow and orange lines). Both of these values exhibit a slow upward slope, when in the linear regime they would be straight flat lines, as seen in Figure 3.10a. In practice one could implement a scheme to identify the case when the running average does not converge, but it turns out that there is a better way to approach the problem which includes using the standard deviation of the heat flux.

In order to determine when an HNEMD simulation is in the non-linear regime, we first need to know some details about how the system behaves during equilibrium. In an equilibrium (NVE) simulation, the average heat flux will be zero and the heat flux distribution will be roughly Gaussian in shape (see Figure 3.11b), with a standard deviation σ_{eq}. When subjected to an external field F_z, the average heat flux will deviate away from zero, but in the linear-regime the standard deviation will remain (approximately) constant. When

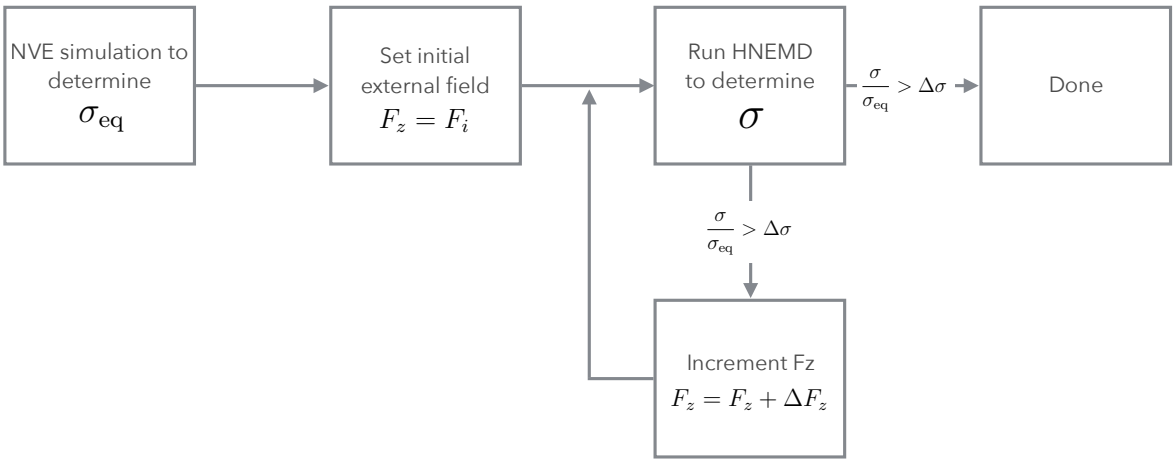


Figure 5.2: Proposed flow for automatic HNEMD. Only three parameters must be defined; initial external field F_i, external field increment ΔF_z and maximum deviation $\Delta\sigma$.

F_z becomes too large, and the system leaves the linear-regime, then the standard deviation increases. Figure 5.1b illustrates this behavior in NaCl. The solid blue line in the standard deviation of the heat flux for the equilibrium case, σ_{eq}. The solid yellow and orange lines are the standard deviation for systems with large external fields F_z. The dashed blue line is simply a visual guide drawn arbitrarily at $1.1\sigma_{\text{eq}}$ (a 10% increase in σ_{eq}).

The proposed algorithm for automatic HNEMD proceeds like so

1. Perform NVE simulation to determine σ_{eq}.
2. Set the initial external field F_z to F_i.
3. Run HNEMD and calculate σ.
4. If $\frac{\sigma}{\sigma_{\text{eq}}} < \Delta\sigma$, increment $F_z = F_z + \Delta F_z$ and go to 3, otherwise.
5. Done.

There are then three user-defined variables that control the process. The first, F_i defines the starting value for the external field and the second, ΔF_z controls how much to increment F_z by from one HNEMD simulation to the next. The last variable, $\Delta\sigma$, defines the stopping

value for $\sigma/\sigma_{\text{eq}}$. More specifically, $\Delta\sigma$ defines when the calculated σ at a given F_z deviates enough from σ_{eq} to be considered outside of the linear-regime. A value of $\Delta\sigma = 1.1$ appears to be a relatively good default value. Choosing an appropriate F_i and ΔF_z, however, is more complicated. To begin to understand how to choose F_i, we start be rearranging Equation 3.28 to get

$$F_i = \frac{\langle J_z \rangle}{\kappa_L^* T} \tag{5.1}$$

Here we have two unknown quantities, κ_L^* and $\langle J_z \rangle$. A good way to choose $\langle J_z \rangle$ is by taking some fraction, say 10%, of σ_{eq}. This gives

$$F_i = \frac{0.1\sigma_{\text{eq}}}{\kappa_L^* T} \tag{5.2}$$

The justification for taking a small percent of σ_{eq} is that we want to 'gently' perturb the system. In practice we've found that pushing F_z to greater than $\sim 0.7\sigma_{\text{eq}}$ typically results in a departure from the linear regime, so choosing a target $\langle J_z \rangle$ that is relatively small (compared to σ_{eq}) is a good place to start.

Finally, the quantity that we ultimately desire from the HNEMD simulations is κ_L^*, but in order to choose an appropriate F_i according to Equation 5.2 we need to know κ_L^*, which is somewhat of a 'chicken or the egg' dilemma. This apparent impasse can be circumvented by replacing κ_L^* with an order-of-magnitude estimate κ_L^M. Now our starting value for F_i becomes

$$F_i = \frac{0.1\sigma_{\text{eq}}}{\kappa_L^M T} \tag{5.3}$$

The suggested approach to determine κ_L^M is to use the initial NVE simulation (used to calculate σ_{eq}) and the Green-Kubo formalism (see Section 2.2.1).

CHAPTER 6

Conclusion

The main goal of this project was to devise a robust computational approach, capable of obtaining high quality lattice thermal conductivity, κ_L, values for crystalline materials. In order to accomplish this, a number of entireley new computational tools were developed. First, a novel compressive sensing approach, CSLD, was developed for solving the force constant problem of lattice dynamics. CSLD was show to be a powerful tool for highly anharmonic lattice dynamics in complex materials based on the robust and mathematically rigorous framework of compressive sensing and compressive sampling. The main advantage of CSLD over the current methods is that it is widely applicable, computationally efficient, systematically improvable, and straightforward to implement. Importantly, it works with general-purpose DFT codes and can be used in an automated manner, with minimal human intervention. Additionally, with the solid, mathematical foundation of lattice dynamics, we were able to do away with the lower quality emperical potentials and develop an entirely new MD package, LMD. LMD is capable of achiving near-DFT quality forces at a fraction of the computational cost, paving the way for exceptionally efficient simulation of crystalline materials. Not only is LMD capable of performing very high quality calcuations of κ_L, but it has also been highly optimized to take advantage of a wide range of high-performance computing hardware. We expect the CSLD and LMD approach to be extrememly useful for large scale, systematic studies of κ_L, with an eye towards finding new, promising thermoelectric materials.

APPENDIX A

LMD Package

1 Overview

Lattice Molecular Dynamics (LMD) is the name of the molecular dynamics code developed, from scratch, and used throughout this project. It is written in C++ and has no required dependencies other than the C++98 standard library[1] and a POSIX-compliant operating system[2]. Throughout development this code has been heavily tested on Mac OS X and a range of Linux-based operating systems, and on hardware ranging from laptops to huge national super computers.

The behavior of LMD is controlled largely by two main concepts– *commands* and *plugins*. Commands control what *mode* the program will be in when it is launched. This can range from running an MD simulation (see Section 3.1.1) to processing output data to calculate thermal conductivity (see Section 3.1.2). Plugins can be used to modify the behavior of a specific command.

2 Installation

To ease the process of compiling across a range of different systems, LMD comes with Autotools support. Any system with Autotools installed can simply follow the usual process of `./configure` followed by `make lmd`. By default, the build system will attempt to find

[1] There are, however, optional dependencies for higher-performance features.

[2] Technically, the only feature that is explicitly operating-system specific is the dependency on `dlopen`.

and include the optional libraries listed in the following section. Should that not be desired, there are a number of flags accepted by `configure` to manually control the inclusion of such libraries. Specifics are discussed in Section 2.1.

Additionally, there are a few flags that can be used during configuration to inform Autotools where to look for certain things such as compilers, libraries and headers. These include

1. `CC` - Used to set the desired compiler.
2. `CXXFLAGS` - Used to set flags to be passed to the compiler
3. `CPPFLAGS` - Use to set flags to the pre-processor

Here is an example that uses a specific version of `g++` with certain compiler flags and alternate paths for libraries and headers

```
./configure CC=/usr/local/bin/g++ \
CXXFLAGS="-O3" \
CPPFLAGS="-I/home/myuser/include -L/home/myuser/lib"
```

2.1 Optional Libraries

There are a variety of optional libraries that LMD can take advantage of to increase performance. They are discussed in more detail below.

2.1.1 FFTW3

FFTW3 (http://www.fftw.org/) is a library for performing fast Fourier transforms. It is required in order to perform thermal conductivity analysis using the `tc` command (see Section 3.1.2). There are a number of other libraries that provide a compatible interface (such as Intel's Math Kernel Library) and these can be used provided that you supply the correct paths during configuration.

FFTW3 support can explicitly be disabled by passing `--disable-fftw3` to `configure`.

2.1.2 OpenMP

OpenMP is a shared-memory parallelization library. It is implemented as compiler directives, which means that in order to use OpenMP you must use an OpenMP-compatible compiler. If you with to use OpenMP, LMD requires version 2.0 or better. Compilers such as GCC and Intel have had support for OpenMP 2.0 for a number of years, so any relatively recent version should work. One major caveat, however, is OS X. Newer versions of OS X (since 10.9), have moved away from GCC as the default compiler to Clang/LLVM. As of LLVM version 6.0 (the default in OS X 10.9 and 10.10) OpenMP support is unavailable. As a result, if one wishes to use OpenMP on OS X, it is necessary to use a difference compiler. GCC 4.9 obtained from http://hpc.sourceforge.net/ has been used successfully on OS X with OpenMP.

OpenMP support can explicitly be disabled by passing `--disable-openmp` to `configure`.

2.1.3 MPI

The Message Passing Interface, MPI, specification defines a common interface for distributed-memory parallelization. This is an extremely efficient method for parallelizing a single simulation over many different networked computers. There are a number of different MPI implementations including OpenMP, Intel and MPICH, to name a few. Like OpenMP, MPI is a compiler-dependent library. There can be rather large difference between the different MPI implementations and how they compile code, therefore configuring and compiling LMD with a specific MPI implementation is beyond the scope of this thesis.

MPI support can explicitly be disabled by passing `--disable-mpi` to `configure`.

2.1.4 OpenCL

The final type of parallelization available in LMD is via OpenCL. It is targeted at using the massively parallel, floating point processing capabilities available in graphics processing

units (GPUs). Not all GPUs are compatible with OpenCL, so you must verify that OpenCL is even supported by the GPU you intend to use. Additionally, OpenCL support is typically provided by vendor-specific GPU drivers. For example, AMD, Nvidia and Intel each provide their own implementation of OpenCL for their specific hardware, so in order to compile LMD with OpenCL support you should consult the various vendors' documentation on how to compile OpenCL for their devices. The one major exception to this rule is Mac OS X. As of OS X 10.7, Apple has provided built-in support for OpenCL on it's hardware. As a result, it is extremely easy to compile LMD with OpenCL support on OS X and nothing additional needs to be installed–`configure` will automatically find and include OpenCL on newer versions of OS X.

OpenCL support can explicitly be disabled by passing `--disable-opencl` to `configure`.

3 Usage

LMD has different modes of operations depending on how the application is launched, the format of which is

```
lmd [command] [arguments...]
```

Without any command or arguments (simply running `lmd`), the program will output a detailed overview of compilation settings, available commands and discovered plugins.

Under the 'Enabled acceleration extensions' section, one can see which parallelization libraries LMD has been built with. Possible options include `openmp`, `mpi`, and `opencl`. If a desired parallelization library is missing, refer to Section 2.1 and try recompiling. The 'Commands' section displays all available commands that can be passed to the program at run time (see Section 3.1), while the 'Plugins' section shows all discovered plugins (see Section 3.2).

3.1 Commands

LMD ships with a number of built-in commands. Some commands will only be available when certain optional libraries are included during configuration and building of LMD. Detailed information about each command is given below. Additionally, users can easily create their own custom commands to further extend the functionality of LMD. To learn more about extending LMD with custom commands see Section 4.1.

3.1.1 `run` - Run MD

The `run` command is used for running an MD simulation and requires three separate input files–`in.sim`, `in.lat` and `in.pot`. These files should be located within the same directory that the `lmd` binary is executed. If any of these files cannot be found, or any of them are invalid, the program will issue and error and exit.

Following the INI-file syntax, there are two main sections that should be defined in `in.sim`.

Simulation

The 'Simulation' section defines the basic settings for the MD simulation. This includes parameters such as starting temperature, simulation length, size of the timestep, etc. Table A.1 describes the available settings.

Potential

The 'Potential' section is used to control how to evaluate the potential energy and calculate forces. This is typically the most time-consuming part of any MD code and as a result it has been highly optimized. There is quite a bit of flexibility on how to perform such operations in LMD, depending on what hardware is available. The simple takeaway here is that if LMD has not been compiled with OpenCL support, the `BACKEND` option should be set to 'default'. If OpenCL support is available, then Table A.2 can be used to determine how to use the available OpenCL hardware. The IDs in this table can be determined using the `dev`

TBEG	Beginning temperature, in Kelvin.
NSTEPS	Number of time steps to run the simulation for. Total simulation duration will be equal to NSTEPS × TSTEP.
TSTEP	Length of timestep, in femtoseconds.
TSTAT	Enable a thermostat. Currently the only available option is 'Gaussian'. Default if no thermostat.
WALLTIME	Maximum CPU seconds to use. This is useful in environments will hard limits on how long a program can run before being forcibly stopped. If WALLTIME is reached before the desired NSTEPS have been completed, the program will gracefully exit and can be resumed again (see Section 3.1.1). If no value if given the program will run until NSTEPS is reached.
PLUGINDIR	Defines a specific location to look for plugins (see Section 4).

Table A.1: Available LMD simulation settings. Required settings are in bold.

BACKEND	Which backend to use. Available options are 'default' or 'opencl'.
PLATFORM-ID	Platform selection. Default: 0.
DEVICE-TYPE	Device type to use during selection. Available options are 'all' (default), 'cpu', 'gpu', 'accelerator', 'custom'.
DEVICE-ID	Which device to use. Default 0.
WG-SIZE	Size of the workgroup. **Must be a multiple of 2**. Default is 64, but depending on the hardware 32 or 128 might be better choices.
PROFILE	Enable OpenCL debugging, set to 'true'. Default is 'false'.

Table A.2: Available LMD potential settings. Required settings are in bold.

command (see Section 3.1.4).

Resuming a Run

Resuming a run is simply a matter of copying the file `out.res` to `in.res`. This can be useful in resuming a simulation that was ended abruptly or simply to run for a longer period of time. Another way to use this is equilibrate the system before running a canonical (NVT) simulation. First, perform a short, equilibrium (NVE) simulation by leaving the thermostat disabled. Then, copy `out.res` to `in.res`, enable the thermostat and start the simulation.

Parallelization

As mentioned earlier in Section 3.1.1, OpenCL parameters can be tuned using the input file. In the event that OpenCL is not available, LMD can still take advantage of OpenMP and/or MPI for parallelization. If LMD has been compiled with OpenMP, nothing specific must be done at run time to take advantage of these parallelization options. However, the number of threads used by OpenMP can be controlled via the `OMP_NUM_THREADS` environmental variable. On POSIX systems under a Bash prompt, this can be set using the command `export OMP_NUM_THREADS=4`, in which case a maximum of 4 threads will be used. If OpenMP is an option, it's use should seriously be considered. Many parts of the code have been tuned to take advantage of the parallelization provided by OpenMP.

3.1.2 `tc` - Thermal Conductivity

The thermal conductivity command is available only if LMD has been built with FFTW3 support. This command calculates the thermal conductivity using the Green-Kubo formalism. For more details on the theory behind this method, see Section 2.2.1. The real-space integration of the heat flux correlation function scales with simulation length T as $O(T^2)$. Therefore, Fourier transforms are used to reduce this to $O(T \log T)$, hence the requirement for the FFTW3 library. For details on how the FFT is used in evaluating correlation functions, see Appendix B.

3.1.3 `help` - Help

The `help` command is useful for understanding how a command or plugin can be used. It's usage is as simple as

```
lmd help [command or plugin]
```

Some plugins have names with spaces in them, such as the heat flux (see Section 3.2.4). In that case, the plugin name must be surrounded by quotation marks. For example, the

following will output the help information for the heat flux plugin.

```
lmd help "Heat Flux"
```

3.1.4 dev - OpenCL Device List

The `dev` command is only available when LMD has been compiled with OpenCL support. It is useful for determining what devices and platforms are available and also their IDs for use in defining the potential settings (Section 3.1.1). As an example of how to use the output to select an OpenCL device, consider the following output of running `lmd dev`

```
Found 1 OpenCL Platform
  Platform 0 [1/1]: Apple [OpenCL 1.2 (Jul 29 2014 21:24:39)]
    Device 0 [1/2]: Intel(R) Core(TM)2 Duo CPU     L9600  @ 2.13GHz
    Device 1 [2/2]: GeForce 320M
```

In this example one platform, with ID 0, has been found–OpenCL 1.2 provided by Apple. Additionally, two devices have been found. The first device, with ID 0, is the Intel CPU and the second, with ID 1, is the NVidia GeForce GPU.

3.2 Plugins

The default behavior of LMD, when runinng MD, is to perform a simple microcanonical (NVE) simulation. The only output file is `out.res`, which can be used for resuming a simulation (see Section 3.1.1), while temperatures and energies are printed to the standard output each timestep. This behavior, however, can be modified by using plugins. LMD ships with a number of built-in plugins for performing different types of MD and for writing out various system properties to file, but it is also possible to develop custom plugins as well (see Section 4.2). The built-in plugins are discussed in more detail below.

NSLAB	Number of slabs (or bins) to partition the system into. Default: 0.
NSWAP	Number of atoms to swap during a swap event. Default: 1.
FREQ	Number of time steps in between swapping events. Default: 100.
DIR	Axis to swap velocities. Available options are 0, 1 or 2 (for x, y and x, respectively). Default 0.
TFREQ	Number of time steps between writing slab temperatures to file. Default: FREQ
NEQ	Number of time steps to run before the first swap event.

Table A.3: Available LMD settings for the RNEMD plugin.

3.2.1 RNEMD - Reverse Non-equilibrium MD

The Reverse Non-equilibrium MD technique, discussed in Section 2.2.3, can be enabled in LMD by defining the RNEMD section in in.sim. There are a number of settings that can be defined to modify how this method is used which are listed in **??**.

In order to obtain the best results with this method, one should ensure that the system is elongated along a particular axis and the define the appropriate value for the DIR setting. If NSLAB is zero or not evenly divisible by two, then this plugin will be disabled. It is also important to choose NSLAB such that there are enough atoms in each slab for sufficient temperature averaging—a few hundred atoms is usually sufficient.

In addition to the settings listed in Table A.3, there are a few experimental options. The first, TDIFF, can be used to define the target temperature difference between the hot and cold slabs, in Kelvin. When this setting is defined, it will override whatever value is set for FREQ, as the system will attempt to swap at the appropriate rate to obtain the desired temperature profile. The other, slightly less useful option HFLUX, can be used to fix the desired heat flux. When this value is set, whatever value is defined for FREQ is ignored and will be replaced with a value calculated automatically by the system. Additionally, NSWAP will be reset to NSWAP = 1.

An example configuration to enable RNEMD is shown in Listing 1.

```
[RNEMD]
NSLAB = 8      # Divide system into 8 slabs...
DIR   = 2      # ...along z-axis
NSWAP = 1      # Swap 1 atom...
FREQ  = 50     # ...every 50 time steps
NEQ   = 1000   # Do 1000 steps before first swap event
```

Listing 1: Example settings to enable RNEMD.

Output Files

When enabled, this plugin will output a number of files. The first, `out.tbin`, will contain a list of slab temperatures. The format of this file is quite simple, but is in binary format, with triplets of numbers representing the MD step, slab ID and it's associated temperature. The MD step and slab ID are both 4-byte unsigned integers and the temperature is an 8-byte double. A simple Python program to print out this file is shown in Listing 2.

```
import functools
import struct

with open("out.tbin", "rb") as fh:
  for chunk in iter(functools.partial(fh.read, 16), ""):
    step, slab, temp = struct.unpack("<IId", chunk)
    print step, slab, temp
```

Listing 2: Example Python script to read binary formatted `out.tbin`.

The second file generated by this plugin is named `out.vs`. It is also a binary encoded file that is written in groups of four values, totaling 28 bytes. The first 4-byte integer is the time step, the next 8-byte double is the total heat flux, followed by 8-byte double representing the calculate thermal conductivity (in W/mK) and the last 8-bytes represents the fit quality. The thermal conductivity and fit quality shouldn't really be used directly, as they are really only meant to be used internally. To calculate the true thermal conductivity, one should determine the average temperatures of each slab in `out.tbin`. Then, calculate the temperature difference dT between slabs 1 and $(N/2) - 1$. Using Equation 3.24 and the final value of the heat flux J from `out.vs`, one can then get the calculated thermal

conductivity. However, the units of heat flux are

$$\frac{\text{electron volts}}{\text{Angstrom} \times \text{femtosecond} \times \text{Kelvin}} \tag{A.1}$$

Thus, the conversion factor of 1602176.46 should be used to convert to W/mK.

3.2.2 `HNEMD` - Homogeneous Non-equilibrium MD

Homogeneous Non-equilibrium MD is a very efficient method for calculating thermal conductivity. This method is best used to study isotropic systems as it is only able to determine the thermal conductivity along a given direction. This is in contrast to the Green-Kubo method, which is technically able to calculate the entire thermal conductivity tensor simultaneously. Nevertheless, HNEMD is an extremely efficient technique, capable of yielding very high thermal conductivity results. For specifics on the theory behind HNEMD, see Section 2.2.4.

Enabling this plugin is very easy as there is only one setting needed under the `HNEMD` section in `in.sim`. To turn on HNEMD, simply define a setting named `FIELD` as a comma-separated list of three numbers. Each number represents the strength of the external field, in inverse Ångtroms, along the x, y and z axes. For example the following will enable a field along the z-axis

```
[HNEMD]
FIELD = 0.0,0.0,1e-4
```

Additionally, the Gaussian thermostat should be enabled. See Section 3.2.3 for details on the Gaussian thermostat.

Output Files

The HNEMD plugin creates a single output file, named `dat.jq`, that contains four values

per line. The first value is the integer time step, followed by the three values of the heat flux (J_x, J_y, J_z).

Calculating Thermal Conductivity

In order to calculate the thermal conductivity using HNEMD, a number of steps must be taken. First, at least three different values for `FIELD` should be used. For each value of `FIELD` a few simulations should be done with different initial conditions. For each simulation, the average heat flux should be calculated. The simple Python script shown in Listing 3 demonstrates how to calculate the average heat flux for a given run.

```
with open("dat.jq", "r") as fh:
  count, J = 0, [0,0,0]
  for line in fh:
    for i, v in enumerate([float(v) for v in line.split()[1:]]):
      J[i] = J[i] + v
    count = count + 1
  print "<J> = %g,%g,%g" % (J[0]/count, J[1]/count, J[2]/count)
```

Listing 3: Python script for calculating the average heat flux for an HNEMD run.

Once the average heat flux has been calculated for every run, the reduced thermal conductivity, $\kappa^*(F_{\text{ext}})$, can be determined using Equation 3.28. There should be a single κ^* for each external field value used. Simple linear regression can then be applied to get the true thermal conductivity at zero field.

3.2.3 `Gaussian` - Gaussian Thermostat

Currently the only thermostat included in LMD is the Gaussian thermostat. For details on how this thermostat works, refer to Section 2.3.1. By enabling the thermostat, one can perform a canonical (NVT) ensemble. To enable the thermostat, the `TSTAT = gaussian` setting should be defined in the input file under the `Simulation` section. Internally, the Gaussian thermostat is implemented as a plugin, so it is entirely possible to implement other types of thermostats using the plugin system as well. For more details on how to write

plugins, see Section 4.2.

3.2.4 `Heat Flux` - EMD Green-Kubo

To enable LMD to calculate the heat flux in an a simulation, the `HeatFlux` section should be defined with a single setting `ENABLE` = true. When enabled, heat fluxes will be written, every time step, into a file named `dat.jq`. Each line in this file will contain the step number followed by the three heat fluxes divided by the system volume ($J_x/V, J_y/V, J_z/V$). This file can be used by the `tc` command (Section 3.1.2) to calculate thermal conductivity using the Green-Kubo formalism (Section 2.2.1).

Calculating Thermal Conductivity

To calculate thermal conductivity using this method, it is typically necessary to do ensemble averaging over quite a large number of initial conditions. The required number of simulations will vary from system to system, so one must analyze the results to ensure the results are converged.

3.2.5 `Trajectories`

To enable LMD to write out atomic displacements, velocities and forces, one can define a setting under the `Simulation` section of the input file named `LOGTRAJ`. The value should be an integer representing how frequently (in time steps) the trajectories should be written to disk. Keep in mind that for every write there will be $N * 10$ values written. For a large system and/or a long simulation, setting `LOGTRAJ` to a small number can result in a very large file.

4 Extending

To make LMD more robust and flexible, it has been designed to be easily extended and modified *without* having to modify the main source code and, in many cases, without having to recompile the program. In order to accomplish this, a light-weight plugin system was designed. Additionally, LMD employs a flexible event-based system that can easily be used to modify the default behavior.

There are two distinct concepts used throughout this section–'command' and 'plugin'. From a technical standpoint, a plugin is used to describe something that add additional functionality using the event-based system, while a command adds an entirely new user-facing command to the LMD program. As an example, the Gaussian plugin implements a Gaussian thermostat to the MD simulation, but it does not provide a user-facing command–this is a *pure plugin*. The Thermal Conductivity plugin does not add any additional functionality to the MD simulation and instead simply provides the `tc` command to the LMD program–this is a *pure command*. It is entirely possible to create a plugin that is both a plugin and a command.

4.1 Commands

In order to create a custom command or plugin that can be loaded dynamically at runtime, a subclass of `Plugin` must be implemented. As an example, let's create a simple command, called `HelloWorld`, that will create a new command that will print out 'Hello World' when called by the user. To start, we define a basic header for the new plugin, shown in Listing 4.

The header file simply includes the `Plugin` header, and then defines a subclass called `HelloWorld` with a public constructor, a static function name `create_plugin` and a single member function named `command`. The names of these functions are important. `create_plugin` is called automatically by the plugin framework when `lmd` is executed. `command` is called automatically to execute the command features provided by our plu-

```
1  #include <lattice/Plugin.h>
2
3  class HelloWorld : public Plugin {
4  public:
5    HelloWorld();
6
7    int command(int argc, char* const argv[]);
8
9    inline static Plugin* create_plugin() {
10     return new HelloWorld;
11   };
12 };
```

Listing 4: `HelloWorld.h` - Header file for `HelloWorld` command.

gin. The real magic happens in the implementation of our `HelloWorld` plugin, shown in Listing 5.

```
1  #include "HelloWorld.h"
2
3  #include <lattice/Plugins.h>
4  #include <cstdio>
5
6  REGISTER_PLUGIN(HelloWorld, "HelloWorld");
7
8  HelloWorld::HelloWorld()
9  : Plugin("HelloWorld") {
10   command_name  = "hello";
11   command_desc  = "Prints 'Hello World' and exits";
12 };
13
14 int HelloWorld::command(int argc, char* const argv[]) {
15   printf("Hello World!\n");
16   return 0;
17 };
```

Listing 5: `HelloWorld.cpp` - Implementation of `HelloWorld` command.

Let's go through Listing 5 to see what's going on. Line 1 we simply include the header file for the `HelloWorld` class. Lines 3-4 include additional necessary headers–notice the include of `Plugins.h` as this is required. Line 6 we register our plugin with the plugin system. The first argument, `HelloWorld`, is the class we are registering and the second

argument, "HelloWorld", should be a unique string representing the new plugin. Lines 8 is the definition of the class constructor and line 9 calls the parent class, `Plugin`, constructor passing a single argument that is the human-readable name of the new plugin. Line 10 tells the plugin system that our plugin provides a command that is called `hello` (this will make it possible to call `lmd hello`). When a plugin defines the `command_name` variable, the plugin system expects that a `command` member function also be implemented (which we have on line 14). Line 11 describes the basic function of our command. Lines 15-16 are the implementation of our command. Notice that the `command` function returns an `int`–this is used as the return code when LMD is run using this command.

```
CXX=g++
FLAGS=-O3
LIBS=-lliblattice

PLATFORM := $(shell uname -s)

ifeq ($(PLATFORM),Darwin)
        FLAGS += -dynamiclib -flat_namespace
endif
ifeq ($(PLATFORM),Linux)
        FLAGS += -fPIC -shared
endif

all:
        $(CXX) $(LIBS) $(FLAGS) -o hello_world.plugin HelloWorld.cpp
```

Listing 6: `Makefile` - A simple makefile for an LMD plugin.

The last step is to compile our new plugin. This procedure will vary from platform to platform, but a simple makefile is shown in Listing 6. The main requirement is to compile the plugin as a shared library. On most Linux-based machines, this can be accomplished with `gcc`'s `-shared` option, while on Mac OS X this can be done with the `-dynamiclib` option.

Evented	Events/Actions
simulation	init beforeintegrate1 afterintegrate1 beforeforce afterforce beforeintegrate2 afterintegrate2 beforesample aftersample
integrator	**evolvevelocity1** **evolvevelocity2**
force	beforeinit init

Table A.4: Available LMD `Evented` objects and their associated events. Items in bold are actions.

4.2 Plugins

Creating a new plugin is very similar to creating a command. The main difference is that a pure plugin will not implement a `command` function, nor define the `command_name` and `command_desc` variables. Instead, a plugin will implement a `listen` function. This function will get called once by each `Evented` object and, if desired, the plugin can notify the `Evented` object that it wishes to listen to one or more events or actions provided by the `Evented`. Table A.4 lists the available `Evented` objects and their associated events and actions. The difference between and event and an action will be discussed later.

The best way to describe how to use events to modify the behavior of LMD is by example. The header for a simple plugin that will write the system temperature to a file every step is shown in Listing 7. The basic structure of the plugin is very similar to that of our custom command in Listing 4, except here we define the `listen` function, which is called

automatically by the plugin system. The name of the `simulationAfterSample` function is arbitrary, but it's signature is important. The first argument, `event`, will be one of the events listed in the second column of Table A.4. The second argument, `caller`, will be the instance of the calling `Evented` class. The third argument, `packet`, will vary depending on the specific event being called.

```
1  #include <lattice/Plugin.h>
2
3  class Temperature : public Plugin {
4  public:
5    Temperature();
6
7    bool listen(Evented* evented);
8
9    bool simulationAfterSample(const char*        event,
10                              Evented*           caller,
11                              Evented::Packet*   packet=NULL);
12
13   inline static Plugin* create_plugin() {
14     return new Temperature;
15   };
16 };
```

Listing 7: `Temperature.h` - Basic plugin header for LMD.

The implmentation of the plugin is shown in Listing 8. This is slightly more complicated than in the case of a `command`, so let's walk through line by line. The first nine lines simply include the necessary header files. Note that on line 7 we include `Simulation.h`, which we need because later we will need access to the `Simulation` class. Line 11 registers our new plugin with the plugin system, with the first argument as the plugin class and the second argument as a unique string name for the plugin. Lines 13-14 are the basic constructor for our class. Note that on line 14 we need to call the parent `Plugin` class constructor, passing in our plugin's name as the only argument. On line 16, we define the `listen` method which will get called automatically by the plugin system. This is where we can specify where our plugin should 'attach' itself to listen to events in the future. On line 17, we check to see if the calling `evented` instance is that of the 'simulation' class. If it is, then on lines

18-20 we bind our `simulationAfterSample` method to the 'aftersample' event of the `Simulation` class. Line 23 we simply return `true`–currently the return code is not used by the plugin system. On line 26 we begin the implmentation of `simulationAfterSample` that we want bound to the 'aftersample' event. On line 30 we cast the `caller` to an instance of `Simulation`. We known that `caller` will by of type `Simulation` because we checked on line 17. Lines 32-47 we calculate the temperature of the system and print it out. On line 42 we access the member function `getMass` of the `lattice` instance. This is available here because we included the `Lattice.h` header. Additionally, on line 47, the constant `KB_IN_EV_PER_K` is defined in `constants.h`.

To compile the plugin we can use a makefile similar to that in Listing 6.

```cpp
#include "Temperature.h"

#include <lattice/Plugins.h>
#include <lattice/Lattice.h>
#include <lattice/constants.h>

#include <lmd/simulation/Simulation.h>

#include <cstdio>

REGISTER_PLUGIN(Temperature, "Temperature");

Temperature::Temperature()
: Plugin("Temperature") {};

bool Temperature::listen(Evented* evented) {
  if (evented->isNamed("simulation")) {
    evented->addListener("aftersample",
                         this,
                         (Evented::Callback)&Temperature::simulationAfterSample);
  }

  return true;
};

bool Temperature::simulationAfterSample(const char*       event,
                                        Evented*          caller,
                                        Evented::Packet*  packet=NULL) {

  Simulation* sim = static_cast<Simulation*>(caller);

  double  mv2 = 0,
  mass,
  v_mag;

  float*  v = sim->data.v;
  int     idx;

  for (int i = 0; i < sim->data.nat; i++) {
    idx   = i*3;
    v_mag = sqrt(v[idx+0]*v[idx+0] + v[idx+1]*v[idx+1]+ v[idx+2]*v[idx+2]);
    mass  = lattice->getMass(i, true); // eV/(A/fs)^2

    mv2 += (mass * v_mag*v_mag);
  }

  return mv2/(KB_IN_EV_PER_K * (3*data.nat-3));
};
```

Listing 8: `Temperature.cpp` - Basic plugin implementation for LMD.

5 Input Files

5.1 `in.sim`

Most commands utilize a common input file named `in.sim`, which should be in INI format. An example INI file looks like

```
[Section 1]
key1 = value1

[Section 2]
key2 = value2
```

Each command will interpret the input file in a different way, so consult Section 3.1 for a list of commands and available input configuration options.

```
#
# Example LMD input file.
#
# Section names are case insensitive.
#

################################################
# Global Simulation Settings
################################################
[Simulation]

# Starting temperature, in Kelvin.
#
TBEG = 300

# Number of MD steps to run.
#
NSTEPS = 5000

# Length of each MD step, in femtoseconds.
#
TSTEP = 1

# Maximum length, in seconds, that this simulation
# should run for.  When the simulation has run
# for this long, it will stop gracefully.  This
# is useful when running on a system with a hard
# limit on CPU run time.
#
#WALLTIME = 15
```

```
# Enable a specific thermostat.
#
#TSTAT = gaussian

# Enable a specific debugging mode.  Any value
# greater than 0 enables debugging.
#
#DEBUG = 1

# Define a custom plugin directory.
#
#PLUGINDIR = /some/path/to/plugins


################################################
# Potential Settings
################################################
[Potential]

# Select the potential backend to use.  Possible
# options include:
#  default - Reference, CPU-based implementation
#  opencl  - OpenCL implmentation
#
# The default behavior is to use the best
# implementation available.
#
BACKEND = default


############################
# OpenCL Options
#

# Define the OpenCL platform to use, zero indexed.
# Defaults to 0.
#
#PLATFORM-ID = 0

# Define the type of OpenCL device to use.
# Possible options are:
#  all
#  cpu
#  gpu
#  accelerator
#  custom
#
#DEVICE-TYPE = all

# Select a specific OpenCL device, zero indexed.
```

```
# Defaults to 0.
#
#DEVICE-ID   = 0

# Define the work group size of the OpenCL
# kernels.  The default is 64. Note, this must be
# a factor of two.  For CPU devices, this usually
# needs to be set to 1.
#
#WG-SIZE     = 1

# Enable OpenCL performance profiling.  Profiling
# data will be written to a file named "profile.log"
#
#PROFILE     = true
```

5.2 in.lat

The `in.lat` file describes the lattice to be simulated. The first three lines should be the primitive lattice vectors, in Ångstroms. The next three lines define the supercell lattice vectors. The next line defines the number of different atomic species in the primitive cell. This should be followed by the same number of lines, with each line containing four values. The first three values describe the atom's position in the primitive cell and the last number is the atomic number of the element. The next line should be the total number of atoms in the supercell. This line should be followed a number of lines equal to the number of atoms in the supercell. Each line defines the position of an atom in the super cell as a series of 8 numbers. The first number is the unique ID of that atom, the next three numbers define the unit cell to which this atom belongs. The next number maps the atom to the atomic species defined on the sixth line. The last three numbers are the fractional coodinates of the atom within the supercell. As an example, the following in an input for a 4x4x4 NaCl lattice

```
2.82 2.82 0.0
-4 4 4
4 -4 4
4 4 -4
2
0 0 0 11
0.5 0.5 0.5 17
512
```

```
1 -3 3 3 1 0.75  0.0   0.0
2 -3 3 3 2 0.875 0.125 0.125
3 -3 3 4 1 0.875 0.125 0.0
4 -3 3 4 2 1.0   0.25  0.125
5 -3 4 3 1 0.875 0.0   0.125
6 -3 4 3 2 1.0   0.125 0.25
...
```

5.3 in.pot

The `in.pot` file describes the potential of the system. It describes, for the entire supercell, clusters and their associated interatomic force constants. An example entry for a pair-cluster looks like:

```
2
0 0 0
0 0 0
1
1 1
3
1 1 1.031973137
2 2 1.031973137
3 3 1.031973137
```

The first line defines the size of the cluster, which in this case is a pair cluster. The next two lines define the location of the sites involved in the cluster (these values are currently ignored). The fourth line states how many clusters there are of this type, which also defines how many lines will immediatley follow. The following lines, in this case just a single line, define the atomic indicies (1-indexed) of the atoms involved in this cluster. Line 6 defined the number of ICFs for this cluster, which also defined the number of lines that follow. Lines 7-9 define the directions of the atoms (integers) followed by the value of the force constant (float), where the x-, y- and z- direction are denoted as 1, 2, 3, respectively. Each entry should be separated by a blank new line.

If CSLD is used, the `in.pot` can be automatically generated for a given supercell.

APPENDIX B

Time Correlation Functions

A number of interesting physical properties can be calculated through the use of time correlation functions. In Section 2.2.1 (Equation 3.20 specifically), we showed how the Green-Kubo relations can be used to calculate thermal conductivity by using the time correlation function of the heat flux. However, this technique can be used to calculate other properties such as diffusion coefficients and viscosity. Therefore, a computationally efficient method for evaluating such equations can be extremely useful and is the focus of this chapter. Let's start by considering some property that is a function of time, $A(t)$. We can write it's autocorrelation function as

$$C(\tau) =< A(\tau)A(0) >= \frac{1}{\tau_n}\int_0^{\tau_n} A(\tau_0)A(\tau_0+\tau)d\tau_0 \tag{B.1}$$

where τ is a specific time origin and τ_n is the total number of time origins. In the discrete case, this becomes

$$C(\tau) =< A(\tau)A(0) >= \frac{\Delta t}{\tau_n}\sum_{\tau_0=0}^{\tau_n} A(\tau_0)A(\tau_0+\tau) \tag{B.2}$$

where Δt is the timestep used in the simulation. The Green-Kubo relation can then be used to relate the microscopic property, $A(t)$, to some macroscopic transport property, D, via

$$D = \beta\int_0^{\infty} \left\langle A(t)A(0)\right\rangle dt \tag{B.3}$$

Combining Equation B.1 and Equation B.3, leads to

$$D = \beta \int_0^{\infty} \left(\frac{1}{\tau_n} \int_0^{\tau_n} A(\tau_0) A(\tau_0 + t) d\tau_0 \right) dt \tag{B.4}$$

which in the discrete case becomes

$$D = \beta \sum_{t=0}^{\infty} \frac{\Delta t}{\tau_n} \sum_{\tau_0=0}^{\tau_n} A(\tau_0) A(\tau_0 + t) \tag{B.5}$$

Obviously in simulations we cannot evaluate such expressions to infinity, thus one must perform simulations that are long enough to give converged results. If we assume that the simulation length, T, is long enough to satisfy this criteria, then we can write this expression as

$$D = \beta \sum_{t=0}^{T} \frac{\Delta t}{\tau_n - t - 1} \sum_{\tau_0=1}^{\tau_n - t - 1} A(\tau_0) A(\tau_0 + t) \tag{B.6}$$

The main problem that arises here is that Equation B.6 scales as $O\left(\frac{T(T-1)}{2}\right)$. It is not uncommon for T to be on the order of 10^5 or 10^6, making this a very expensive operation. There are two routes we could take to decrease the amount to time needed to evaluate Equation B.6. The first is relativity straightforward and involves parallelizing over the first sum. The degree of parallelization is limited only by the number of physical processing cores present in a given machine.

However, there is a better method which involves a clever mathematical trick. This 'trick' comes from realizing that Equation B.6 is what is known as *convolution* and that convolution in the time domain is equivalent to multiplication in the frequency domain. All that is then needed is a fast way to convert between the time and frequency domains, which is a common task known as the Fast Fourier Transform (FFT). The technique looks something like

1. $\mathbf{A}(f) = \text{FFT}\{\mathbf{A}(t)\}$
2. $\mathbf{M}(f) = \mathbf{A}(f)\mathbf{A}^*(f)$
3. $\mathbf{N}(t) = \text{IFFT}\{\mathbf{M}(f)\}$
4. $D = \beta \sum_{t=0}^{T} \mathbf{N}(t)$

In the first step, we convert the time-domain value $\mathbf{A}(t)$ into the frequency domain using a forward FFT. Then in the next step we multiple the frequency-domain value, $\mathbf{A}(f)$, by its complex conjugate, which gives is the intermediary product $\mathbf{M}(f)$. An inverse FFT of $\mathbf{M}(f)$ gives back the time-domain equivalent $\mathbf{N}(t)$. $\mathbf{N}(t)$ represent the second sum in Equation B.6, thus all that is left in the final step is to sum up all the values of $\mathbf{N}(t)$.

At first, the FFT based method may seem more expensive than the straight forward approach. However, this isn't the case because even with the two FFT transforms (forward and reverse), we can reduce the whole process to one that scales as $O(T\ln T)$[1].

Bibliography

[1] D. J. Tildesley M. P. Allen. *Computer Simulation of Liquids*. Oxford University Press, 1991.

APPENDIX C

Sets & Lists

It is useful to be able to represent collections of items using consistent notation. Here we discuss the notation used in this paper to work with two similar types of collections; sets and lists. Both sets and lists can hold items that, most of the time, will be numbers.

1 Sets

To start, we define a set

$$\{1, 2, 3, 4\} \tag{C.1}$$

which is identical to

$$\{1, 1, 1, 2, 2, 3, 4\} \tag{C.2}$$

because sets contain unique items. A set that contains no items is called an empty set and is denoted as $\varnothing$ or $\{\}$.

We can define sets without explicitly enumerating the items like so

$$\{x \mid x \text{ is an even prime}\} \tag{C.3}$$

where the character |, pronounced "such that", denotes some criteria for creating the set. In this case, the criteria is that all items in this set are even primes, or more specifically that this set contains *all* even primes. Multiple conditions can be defined by separating them with $\wedge$, pronounced "and". We can also define functions that determine what items belong to the set.

$$\{P(x) \mid x \in \mathbf{Z}\} \text{ where,} \tag{C.4}$$

$$P(x) = 2x \tag{C.5}$$

where $\mathbf{Z}$ is the set of all integers and $\in$ means "in". This will result in a set containing all positive, even integers, because the function $P(x)$ multiplies every integer by 2.

The number of elements in, or cardinality of, a set A is denoted $|A|$. The following set contains three items

$$|\{5, 6, 7\}| = 3 \tag{C.6}$$

Two or more sets can be combined via a "union" ($\cup$)

$$\{1, 2, 3\} \cup \{3, 4, 5\} = \{1, 2, 3, 4, 5\} \tag{C.7}$$

Note that $|A| + |B|$ does not necessarily equal $|C|$. The difference between sets can also be

found via an "intersection" ($\cap$)

$$\{1, 2, 3\} \cap \{3, 4, 5\} = \{3\} \tag{C.8}$$

Sets are, by definition, unordered, however, we define the ordered set using parentheses. The following is an ordered set

$$\mathbf{A} = (a, b, c) \tag{C.9}$$

Ordered sets are useful because individual items can be accessed using the index operator. The first element (zero indexed) in ordered set $\mathbf{A}$ can be accessed like so

$$a \leftarrow \mathbf{A}\,[0] \tag{C.10}$$

2 Lists

Lists are very similar to sets, with the main exception being that lists hold an ordered collection of items that are not necessarily unique. To differentiate between lists and sets, we use the following notation

$\{1, 2, 3\}$ a set

$(1, 2, 3)$ an ordered set

$[1, 2, 3]$ a list

Index

amorphous solids, 21

compressive sensing, 43
covalent solids, 24
crystalline solids, 21
CSLD, 51

DFPT, 50

ionic solids, 24

lattice dynamics, 38
lattice thermal conductivity, 56
    BTE, 56
    Equilibrium MD, 62
    Green-Kubo, 62
    HNEMD, 69
    NEMD, 66
    RNEMD, 67
LMD, 60

molecular dynamics, 28
    ensembles, 35
    steps, 29

Peltier effect, 3
potentials
    empirical, 25
        EAM, embedded-atom method, 26
        EIM, embedded-ion method, 26
        Lennard Jones, 24

Seebeck
    coefficient, 10
    effect, 3

thermal conductivity, 9
thermocouple, 5
thermoelectrics
    applications, 5
    efficiency, 8
    figure of merit, 8
    history, 2
    optimization, 10
timestep, 33

unit cell, 22