5-2017

Understanding three-body interactions in hexagonal close packed solid He-4

Ashleigh Locke Barnes
University of Tennessee, Knoxville, alocke5@vols.utk.edu

Understanding three-body interactions in hexagonal close packed solid He-4

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Ashleigh Locke Barnes

May 2017

This dissertation is dedicated to my best friend and loving husband, Austin Barnes,
for seeing me through the best and worst times

Acknowledgements

This dissertation would not have been possible without the guidance of my advisor, Dr. Robert Hinde. Thank you, Dr. Hinde, for encouraging me to take advantage of every opportunity to grow in this field through conferences, symposia, and summer schools. I will never forget the knowledge I've gained and the people I've met from those experiences, which would not have been possible without your support. I would like to thank the members of my committee for giving of their time and effort to see me through my graduate studies, and I acknowledge the University of Tennessee and the National Science Foundation for funding this research.

To the current and former members of the Hinde group: you have made grad school a fun and memorable experience. You are simultaneously the ones who will laugh first when I make a mistake, and encourage me when I find out I have to redo the last three months of work. Matt Dutra, Kiran Dhah, Dan D'Andrea, Randi Beil, Paul Mott, and James Smith, you have become my wonderfully sarcastic, entertaining family and I don't think I could have made it through the toughest semesters without all of you.

I would like to thank my incredibly supportive husband, Austin, who always encouraged me to put school first while he managed his career and our home. Thank you for taking care of pretty much everything during the semesters when I was too stressed and busy to get the laundry done. You help me to take everything one day at a time, and you have motivated me to keep working when I felt like I was burned out. Coming home to you, Ron, and Scout always makes the worst days better.

To my friends and family, thank you for listening to me ramble on about research and providing reassurance when I felt overwhelmed. But more than that, thank you for reminding me to enjoy this time of my life. Samantha, thank you for being the one person who always understands my anxiety and who can help me put things into perspective (and for giving me the cutest niece and nephew on the planet). Dad and Leslie, thank you for encouraging me and understanding when I'm too busy to come visit, and for always being willing to provide a much needed escape to the mountains. Mom and James, thank you for always letting me know you are proud of me (and Mom, thanks for sharing my love of the X-Files). Jamie and Robert, Knoxville became home when you decided to move here. From roommates in Waco to neighbors in Knoxville, you will always be a part of our family, no matter where life takes us next.

I could fill a novel listing all the many friends, family members, and teachers who have helped me get here, and I wouldn't be who I am without each of you: David, Meg, Adam, Jenn, Nicole, Hana, my grandparents, aunts, uncles, cousins, my wonderful in-laws who made me feel like part of the family since day one, Brian Dalak, Kevin Pinney, Darrin Bellert, Bob Compton, and so many others. "Thank you" doesn't begin to express how much I appreciate having you in my life.

It is evident that an acquaintance with natural laws

means no less than an acquaintance with

the mind of God therein expressed.

- James Joule

Abstract

The ground state properties of hexagonal close packed (hcp) solid ^{4}He [He-4] are dominated by large atomic zero point motions which make the primary contribution to the solid's low-temperature Debye-Waller (DW) factors. Preliminary investigations have also suggested that three-body interactions can play an important role in this system, particularly at higher densities. However, due to their computational cost, these interactions are not generally incorporated into theoretical models of solid ^{4}He [He-4]. In order to accurately treat both zero point motion and three-body interactions, we have developed a perturbative treatment in which the three-body energy is added as a correction to the two-body energy obtained from variational quantum Monte Carlo (VMC) and variational path integral Monte Carlo (VPI) simulations. The accuracy of this approach is verified via comparison to simulations in which a three-body potential energy function is fully incorporated into the potential energy calculations throughout the simulations. These methods are used to calculate the ground state energy and DW factors of hcp ^{4}He [He-4] over a range of molar volumes from 2.5 cm^3/mol [cubic centimeters/mol] to 21.3 cm^3/mol [cubic centimeters/mol] at $T = 0$ K. DW factors from two-body simulations are found to be in good agreement with existing two-body models; however, neither two- nor three-body simulations can account for the 20% anisotropy in the DW factors recently reported by Blackburn, *et al.* Pressure-volume equations of state (EOSs) are derived from the energies obtained from all simulations. Incorporating three-body interactions brings the calculated pressures into much closer agreement with experimental values, and EOSs derived from both the perturbative and full-incorporation treatments of three-body interactions are nearly indistinguishable. This indicates that over this molar volume range, the computationally efficient perturbative method is sufficient to account for three-body interactions. Finally, the nonzero elastic constants are calculated via the bulk modulus, K, and the three pure shear constants C_0 [C_0], C_{66} [C_66], and C_{44} [C_44] which are obtained from simulations of distorted hcp ^{4}He [He-4] lattices. The results show that while three-body interactions affect the pure shear constants at higher densities, their influence on K is non-negligible even at low densities.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

1.1 The hcp ^{4}He System

Solid ^{4}He has been a topic of experimental and theoretical interest due to its highly quantum nature and the simplicity of the ^{4}He atoms which make accurate quantum calculations feasible. In addition, recent controvery over the possibility of a supersolid state has increased interest in studying solid ^{4}He both experimentally and theoretically[1]. Rare gas solids including solid ^{4}He have also been utilized as inert matrices for stabilizing reactive species for spectroscopic analysis[2, 3]. The data from these experiments can only be accurately interpreted if the influence of the matrix on the dopant is well understood, which requires well characterized properties of the pure solid. Beyond this application, interest in solid ^{4}He stems from the fact that large zero point motions, which arise due to the lightness of the ^{4}He atoms, result in expanded yet highly compressible lattices and have earned this material the classification of a "quantum solid"[4]. This unique class of materials includes solid H_2, D_2, and LiH, among others, which are characterized by reduced vibrational amplitudes (defined to be the square root of the mean squared displacement divided by the nearest neighbor distance) which exceed the Lindemann melting criterion[5]. This lends interesting properties to quantum solids such as high diffusion and exchange rates not commonly observed in classical solids at low temperatures. In fact, it is these large zero point motions which are responsible for keeping ^{4}He in the liquid phase at absolute zero and atmospheric pressure. It is only at pressures above approximately 25.2 bar that ^{4}He exists as a solid in the hexagonal close packed (hcp) phase[8].

As the simplest member of the quantum solids, solid ^{4}He is a sensible starting point for the development of efficient quantum mechanical models for the larger class of quantum solids. Reliable models of solid ^{4}He require accurate treatment of the zero point motions which dominate the low-temperature properties and are responsible for the nonzero kinetic contribution to the ground state energy. In addition, correlation of the atomic motions must be taken into consideration. This has generally been accomplished using quantum Monte Carlo (QMC) simulation methods which utilize a trial wavefunction with a two-body correlation term to calculate the system's properties. Commonly used trial wavefunctions for solid ^{4}He in QMC simulations generally follow the Jastrow-Nosanow form[6] shown in Eq. 1.1,

$$\Psi = \prod_i g(i) \prod_{i<j} f(i,j), \tag{1.1}$$

where $g(i)$ is a (typically Gaussian) function that depends only on the displacement of atom i. The two-body function $f(i,j)$ is a function of the instantaneous interatomic distance between atoms i and j (R_{ij}) and accounts for correlation in the atomic zero point motions. As atoms get farther apart, their motions are less correlated and therefore this function goes to 1 at long interatomic distances. A well-established form of this function is the McMillan form[7] shown in Eq. 1.2,

$$f(i,j) = e^{-\frac{1}{2}(b/R_{ij})^5} \tag{1.2}$$

where b is a variational parameter which can be tuned along with the Gaussian parameters in $g(i)$ to optimize the trial wavefunction. Using approximate ground state methods such as variational quantum Monte Carlo (VMC)[8] or exact methods such as Green's Function Monte Carlo (GFMC)[9], diffusion Monte Carlo (DMC)[10], or variational path integral Monte Carlo (VPI)[8], this wavefunction can be used to calculate approximate or exact ground state properties, respectively. In these simulations, the ground state potential energy is often calculated using a pairwise-additive model[11, 12, 13] described by Eq. 1.3,

$$V = \sum_i \sum_{j<i} V_2(R_{ij}), \tag{1.3}$$

where V_2 is a two-body potential energy function which depends only on the interatomic distance between atoms i and j. The delocalization of the ^{4}He atoms, however, also results in atoms coming into closer contact than their average lattice spacing suggests, which increases the probability that three ^{4}He atoms will be in close contact simultaneously. It is therefore possible that even at low densities, three-body interactions may play a significant role in this and other quantum solids. In the next section, we present a brief survey of the current theoretical models of three-body interactions in solid ^{4}He and review a recently reported three-body potential.

1.2 Three-Body Interactions

It is first useful to define what is meant by "three-body interactions". This term refers to the additional contribution to the potential energy associated with three particles interacting simultaneously that cannot be accounted for in the sum of pairwise contributions to the potential energy. Mathematically, this is described by Eq. 1.4.

$$V_3 = V_{\text{tot}} - \sum_{i}^{3} \sum_{j<i}^{3} V_2(R_{ij}) \tag{1.4}$$

where V_2 is the the pair potential and V_3 is the nonadditive three-body contribution. In any condensed phase system, three-body and higher many-body interactions have the potential to make a significant contribution to the potential energy[14]. However, many-body contributions to the potential energy are often computationally challenging and expensive to incorporate into simulations of large, dense systems. Therefore, in systems such as solid ^{4}He where three-body interactions are relatively weak compared to the pairwise-additive potential contribution[15], these interactions are often omitted for the sake of computational efficiency. However, a number of theoretical investigations have been performed that suggest that three-body interactions are neccessary in order to calculate properties of solid ^{4}He that are in agreement with experimental data. Here we provide a brief survey of those investigations.

1.2.1 Survey of Earlier Studies of Three-Body Interactions in Condensed Phase ^{4}He

Initial calculations of the three-body contribution to the properties of solid ^{4}He utilized the Axilrod-Teller (AT) three-body dispersion potential[16], ignoring three-body exchange contributions. VMC and GFMC studies of liquid as well as face centered cubic (fcc) and hcp solid ^{4}He by Whitlock and coworkers[17, 18] in 1979 and 1980 utilized a Lennard-Jones 12-6 ^{4}He-^{4}He interatomic potential[19] with perturbative three-body corrections using the AT three-body potential. They concluded in these investigations that the three-body interactions resulted in no significant change in the calculated melting and freezing densities, although better agreement in the energies from VMC simulations with those from GFMC could be obtained by including an explicit three-body correlation term in the wavefunction. Soon after, a nonadditive three-body potential was developed by Bruch and McGee[20] which included a three-body exchange term along with the AT contribution (referred to hereafter as the BM potential). Analysis of this three-body potential by Loubeyre in 1987 using Self Consistent Phonon as well as Monte Carlo calculations showed that the incorporation of the exchange term brought the calculated pressure-volume equations of state in both the liquid and solid phases at 300 K into much better agreement with experimental data[21]. This was later contradicted by Boronat and Casulleras who utilized quadratic DMC simulations to evaluate two different pair potentials as well as the AT and BM three-body potentials in liquid ^{4}He. Utilizing the same perturbative method as in Ref. [17], they found that the energies calculated with the AT and BM potentials often made the agreement with experimental data worse. However, when the exchange contribution to the BM three-body potential was reduced by a factor of three the calculated energies agreed very well with experiment[22]. A similar conclusion was reached by Boninsegni, *et al.*[23], who found that a prefactor of $\frac{2}{3}$ in the exchange term of the BM potential was necessary to accurately reproduce the isotopic shift in the melting pressure.

These early investigations shed light on the need for more careful parameterization of the exchange contribution to the nonadditive three-body energy. In 1996, Cohen and Murrell[24] published a new nonadditive three-body potential (CM) which was parameterized using high level *ab initio* energy calculations of 53 $^4He_3$ configurations, all with C_{2V} symmetry. The resulting potential was the sum of a exchange term and a product of a damping function and the AT potential. A comparison of both the CM and BM potentials using path integral Monte Carlo (PIMC) simulations at 300 K by Chang and Boninsegni, however, once again concluded that in the low density region, accurate agreement with the experimental pressure-volume equation of state could only be obtained by using a prefactor of $\frac{2}{3}$ before the exchange term in both potentials[25]. At higher densities, this correction no longer resulted in perfect agreement with experiment. Instead, the adjusted CM potential was found to underestimate the experimental pressure, while the BM potential overestimated the pressure. Additional DMC investigations by Ujevic and Vitiello called into question the validity of the CM potential in solid phase ^{4}He when no combination of two-body potentials with this three-body potential could accurately reproduce experimental binding energies[26]. This lead to a follow-up study in which the CM potential was adjusted by introducing two phenomenological parameters to scale up and dampen the exchange and dispersion terms, respectively[27]. While this method resulted in significantly improved calculations of energetic and elastic data, it also shed light on the problems in the CM potential.

More recently, due to the uncertainty in the available three-body potentials, DMC simulations have been carried out by Cazorla and Boronat in which contributions from all many-body interactions were accounted for perturbatively using density functional theory (DFT) calculations in the generalized and local gradient approximations[28]. This computationally efficient correction resulted in much better agreement in the high density pressure-volume equation of state in hcp solid ^{4}He with experimental pressure-volume data compared to simulations without the many-body correction. However, individual contributions from three-body interactions apart from four-

and higher many-body interactions could not be determined, and therefore it was unclear at what densities three-body interactions are important and when they become insufficient for an accurate description of the system. Later calculations by the same group utilized effective three-body potentials parameterized in the Slater-Kirkwood form (related to the form of the BM potential) from DFT calculations[29]. Forms of the potential parameterized from trimer energies, atomic forces, or a combination of the two quantities were added to the two-body potential throughout the DMC simulations and were used to calculate the high pressure equation of state, bulk compressibility, classical shear modulus (in the absence of zero point motion), and kinetic contribution to the total energy. These effective three-body potentials typically resulted in calculated quantities which were in better agreement with experiment than the two-body values, however the relative accuracy of each potential varied depending on the quantity being calculated. Therefore, the conclusion of all of these investigations was that three-body interactions do have a real impact on the calculated properites of condensed phase ^{4}He, though the reliability of existing three-body potentials is still up for debate.

1.2.2 The Cencek Three-Body Potential

In the midst of these reparameterizations of the CM potential and development of DFT-based effective three-body potentials, a new nonadditive three-body potential was published by Cencek, *et al.*, seemingly without being noticed by the condensed phase ^{4}He community[15]. This new potential was developed with the intention of accounting for correlation effects in the nonadditive three-body energy beyond the CCSD(T) level of theory by expanding to the full-configuration-interaction (FCI) level of theory. The total trimer energy was evaluated for 253 different trimer configurations with both $C_{2\mathrm{V}}$ and C_{s} symmetries according to Eq. 1.5,

$$E_{int}^{FCI}[3] = E_{int}^{HF}[3] + \delta E_{int}^{CCSD(T)}[3] + \delta E_{int}^{FCI}[3], \tag{1.5}$$

where $E_{int}^{FCI}[3]$ is the total nonadditive three-body potential energy at the FCI level of theory, $E_{int}^{HF}[3]$ is the Hartree-Fock nonadditive three-body energy, $\delta E_{int}^{CCSD(T)}[3]$ is the correlation contribution to the CCSD(T) three-body energy not accounted for in the Hartree-Fock energy, and $\delta E_{int}^{FCI}[3]$ is the correlation contribution to the FCI three-body energy not accounted for at the CCSD(T) level of theory[15]. Using this relationship, Cencek and coworkers were able to calculate the various terms of Eq. 1.5 independently, allowing for the use of smaller basis sets as the level of theory increased in order to maintain computational feasibility of the calculations.

The resulting potential was able to calculate the nonadditive three-body potential energy in an equilateral trimer with side lengths $R = 5.6\ a_o$ (approximately the minimum of the two-body potential well) to within 1.7 mK, a significant improvement over the 10 mK error calculated from an earlier three-body potential parameterized by the same group at the CCSD(T) level of theory[30]. Moreover, the new Cencek potential was found to have an overall uncertainty equal to one-fifth of that of their earlier potential.

To our knowledge, the Cencek nonadditive three-body potential represents the highest level of theory yet implemented in the derivation of the ^{4}He three-body energy. However, it has so far only been used in a handfull of gas-phase simulations where three-body interactions do not make a significant contribution[31, 32, 33, 34, 35]. It therefore remains for this potential to be implemented in condensed phase simulations in order to verify its reliability.

1.3 Overview

The following chapters detail the development and evaluation of a theoretical model of hcp solid ^{4}He which incorporates three-body interactions as a perturbative correction to the two-body energy of the system. Two-body energies are calculated from VMC and VPI simulations and the atomic positions recorded throughout are used to calculate a three-body correction using the recently developed three-body potential of Cencek, *et al.*[15] discussed above. This study constitutes the first implementation of the Cencek three-body potential in simulations of condensed phase ^{4}He. The perturbative treatment utilized here is evaluated against a full-incorporation treatment where the Cencek nonadditive three-body potential is added to the two-body potential energy function throughout the VMC and VPI simulations. This is done to confirm that the perturbative treatment does not result in any significant loss of accuracy.

The reliability of these models is assessed against existing experimental and theoretical data. To date, hcp solid ^{4}He has been investigated using a number of experimental techniques such as neutron diffraction[36], neutron inelastic scattering[37], x-ray diffraction[38], torsional oscillator experiments[39], and isochoric pressure measurements[40], among others methods. From these experiments the Debye-Waller (DW) factors, equations of state, and (to some extent) the elastic constants have been calculated. The theoretical models of hcp solid ^{4}He established here are therefore used to calculate these properties to verify that they produce results in good agreement with experiment. The same properties are calculated from two-body simulations in order to quantify the effect of three-body interactions on the system. We note that the initial motivation for the calculation of the DW factors stems from a discrepancy in the experimental data. In 2007, a low temperature neutron scattering study of hcp solid ^{4}He reported DW factors parallel and perpendicular to the basal plane of the crystal which differed by nearly 20%[36]. This is in contrast to a number of neutron scattering and x-ray diffraction studies performed at higher temperatures which found no evidence of anisotropy[37, 38, 41, 42, 43]. This anisotropy has not been explained by existing two-body models, and therefore we utilize both our two-body and three-body simulations in order to investigate this difference.

In the next chapter, the computational methods employed in the development of the three-body theoretical models of hcp solid ^{4}He are discussed. In addition, we describe the calculation of the DW factors, equations of state, and elastic constants. Chapter 3 presents the evaluation of the two-body model to which the three-body perturbative correction will be added. Specifically, this chapter investigates trends in the VMC trial wavefunction parameters, the two-body DW factors, and a preliminary equation of state. Within the appendix of Chapter 3 we also report DW factors calculated from full-incorporation VMC simulations. Chapter 4 provides an in-depth comparison of the pressure-volume equations of state derived from both VMC and VPI simulations with and without three-body interactions to existing experimental and theoretical data. Finally, in Chapter 5 we evaluate the elastic constants calculated from VPI simulations with and without a perturbative three-body correction in order to determine the significance of three-body interactions in the elastic properties of hcp solid ^{4}He. Chapter 6 provides a brief summary of the conclusions drawn from these studies as well as suggestions for future investigations.

1.4 References

[1] M. H. W. Chan, R. B. Hallock, and L. Reatto, *Journal of Low Temperature Physics* **172**, 317 (2013). 2

[2] E. B. Gordon, G. Frossati, A. Usenko, Y. Aratono, and T. Kumada, *Physica B: Condensed Matter* **329-333**, 404 (2003). 2

[3] M. E. Jacox, *International Journal of Mass Spectrometry* **267**, 268 (2007). 2

[4] E. Polturak and N. Gov, *Contemporary Physics* **44**, 145 (2003). 2

[5] F. Lindemann, *Physikalische Zeitschrift* **11**, 609 (1910). 2

[6] L. H. Nosanow, *Phys. Rev. Lett.* **13**, 270 (1964). 3

[7] W. L. McMillan, *Phys. Rev.* **138**, A442 (1965). 3

[8] D. M. Ceperley, *Rev. Mod. Phys.* **67**, 279 (1995). 2, 3

[9] M. H. Kalos, D. Levesque, and L. Verlet, *Phys. Rev. A* **9**, 2178 (1974). 3

[10] P. Reynolds, D. M. Ceperley, B. J. Alder, and W. A. Lester, *Journal of Chemical Physics* **77**, 5593 (1982). 3

[11] E. W. Draeger and D. M. Ceperley, *Phys. Rev. B* **61**, 12094 (2000). 3

[12] C. Cazorla, Y. Lutsyshyn, and J. Boronat, *Phys. Rev. B* **85**, 024101 (2012). 3

[13] R. Pessoa, M. De Koning, and S. A. Vitiello, *Journal of Low Temperature Physics* **173**, 143 (2013). 3

[14] K. Szalewicz, R. Bukowski, and B. Jeziorski, Chapter 33 - On the importance of many-body forces in clusters and condensed phase, in *Theory and Applications of Computational Chemistry*, edited by C. E. Dykstra, G. Frenking, K. S. Kim, and G. E. Scuseria, pp. 919 – 962, Elsevier, Amsterdam, 2005. 4

[15] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 4, 7, 8, 9

[16] B. M. Axilrod and E. Teller, *The Journal of Chemical Physics* **11**, 299 (1943). 5

[17] P. A. Whitlock, D. M. Ceperley, G. V. Chester, and M. H. Kalos, *Phys. Rev. B* **19**, 5598 (1979). 5

[18] P. A. Whitlock, M. H. Kalos, G. V. Chester, and D. M. Ceperley, *Phys. Rev. B* **21**, 999 (1980). 5

[19] J. De Boer and A. Michels, *Physica V* (1938). 5

[20] L. W. Bruch and I. J. McGee, *The Journal of Chemical Physics* **59**, 409 (1973). 5

[21] P. Loubeyre, *Physical Review Letters* **58**, 1857 (1987). 5

[22] J. Boronat and J. Casulleras, *Physical Review B* **49**, 8920 (1994). 5

[23] M. Boninsegni, C. Pierleoni, and D. M. Ceperley, *Physical Review Letters* **72**, 1854 (1994). 5

[24] M. J. Cohen and J. N. Murrell, *Chemical Physics Letters* **260**, 371 (1996). 6

[25] S.-Y. Chang and M. Boninsegni, *The Journal of Chemical Physics* **115**, 2629 (2001). 6

[26] S. Ujevic and S. A. Vitiello, *Physical Review B - Condensed Matter and Materials Physics* **71**, 1 (2005). 6

[27] S. Ujevic and S. A. Vitiello, *Journal of Physics: Condensed Matter* **19**, 116212 (2007). 6

[28] C. Cazorla and J. Boronat, *Journal of Physics: Condensed Matter* **20**, 015223 (2008). 6

[29] C. Cazorla and J. Boronat, *Phys. Rev. B* **92**, 224113 (2015). 7

[30] W. Cencek, M. Jeziorska, O. Akin-Ojo, and K. Szalewicz, *The Journal of Physical Chemistry A* **111**, 11311 (2007), PMID: 17595067. 8

[31] G. Garberoglio and A. H. Harvey, *Journal of Research of the National Institute of Standards and Technology* **114**, 249 (2009). 8

[32] G. Garberoglio, M. R. Moldover, and A. H. Harvey, *Journal of research of the National Institute of Standards and Technology* **116**, 729 (2011). 8

[33] G. Garberoglio and A. H. Harvey, *The Journal of chemical physics* **134**, 134106 (2011). 8

[34] K. R. S. Shaul, A. J. Schultz, and D. A. Kofke, *The Journal of Chemical Physics* **137**, 184101 (2012). 8

[35] K. R. Shaul, A. J. Schultz, D. A. Kofke, and M. R. Moldover, *Chemical Physics Letters* **531**, 11 (2012). 8

[36] E. Blackburn, J. M. Goodkind, S. K. Sinha, J. Hudis, C. Broholm, J. van Duijn, C. D. Frost, O. Kirichek, and R. B. E. Down, *Phys. Rev. B* **76**, 024523 (2007). 9

[37] J. Eckert, W. Thomlinson, and G. Shirane, *Phys. Rev. B* **18**, 3074 (1978). 9

[38] D. A. Arms, R. S. Shah, and R. O. Simmons, *Phys. Rev. B* **67**, 094303 (2003). 9

[39] E. Kim and M. H. W. Chan, *Nature* **427**, 225 (2004). 9

[40] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 9

[41] C. T. Venkataraman and R. O. Simmons, *Phys. Rev. B* **68**, 224303 (2003). 9

[42] C. A. Burns and E. D. Isaacs, *Phys. Rev. B* **55**, 5767 (1997). 9

[43] C. Stassis, D. Khatamlan, and G. Kline, *Solid State Communications* **25**, 531 (1978). 9

Chapter 2

Computational Methods

2.1 Introduction

This chapter provides an overview of the computational methods employed in this work. First, the two methods used to calculate the energy and coordinate-space observables of the hcp solid ^{4}He system – variational quantum Monte Carlo (VMC) and variational path integral Monte Carlo (VPI) – are discussed. This includes the reweighting procedure implemented to facilitate wavefunction optimization in the VMC simulations, as well as the calculation of the long-range correction to the potential energy in both VMC and VPI. Additionally, two different methods of incorporating three-body interactions into the VMC and VPI simulations are described. This is followed by a description of the energy-volume and pressure-volume equations of state. Finally, the implementation of shear distortions in the hcp lattice and the subsequent calculation of the nonzero elastic constants is detailed.

In general, it can be assumed that the calculations performed on ideal hcp systems ($c/a = 1.633$) utilize a nearly cubic hcp ^{4}He simulation cell with a fixed number of atoms in which periodic boundary conditions have been applied in all directions. Distortions are applied to this lattice while maintaining constant volume and number of atoms. All simulations are performed at $T = 0$ K. Details specific to each part of this work can also be found in the Computational Methods sections of the corresponding chapters.

2.2 VMC

We have previously reported the VMC method discussed in this section in Ref. [1] (Chapter 3). VMC is a quantum Monte Carlo method in which configurations of atomic positions are generated from a fixed trial wavefunction. The sequence of configurations generated throughout the simulations is used to calculate average values of the energy along with any other coordinate-space observables such as the Debye-Waller (DW) factors (see Sec. 2.2.2). For sufficiently long simulations, these averages converge to the expectation values for the given trial wavefunction, within statistical uncertainty.

For these simulations, we employ a modified Jastrow-McMillan[2] style trial wavefunction of the form shown in Eq. 2.1,

$$\Psi = A \prod_{i} e^{-a_{xy}(s_{i,x}^2+s_{i,y}^2)} e^{-a_z s_{i,z}^2} \prod_{(i,j)\in \mathrm{IPs}} e^{-\frac{1}{2}(b/R_{ij})^5}, \tag{2.1}$$

where A is a normalization factor, $\vec{s_i} = (s_{i,x}, s_{i,y}, s_{i,z})$ is the displacement vector of atom i from its average lattice site, R_{ij} is the instantaneous distance between atoms i and j, and a_{xy}, a_z, and b are variational parameters. The original Jastrow-McMillan wavefunction utilized a single a parameter[2], however due to the possibility of anisotropy in an hcp lattice, we have introduced two parameters which independently govern zero point motion in the x, y-plane and along the z-azis, corresponding to motion in and perpendicular to the basal plane of the crystal, respectively. The two a_i parameters affect the delocalization of the ^{4}He atoms, which are assumed to be loosely bound to their average lattice sites. This "tethering" has been previously implemented in the study of solid ^{4}He[3] and prevents the need to consider Bose-Einstein exchange statistics, instead allowing us to treat each atom as a Boltzmann particle[4]. The b parameter, however, accounts for correlation in the atomic displacements by preventing neighboring atoms from coming into close contact with one another. The term IPs in the product governed by the b parameter denotes the set of atoms which are considered to be interacting pairs. For these investigations, this includes all pairs of atoms whose average lattice sites are separated by $< 2.05R_{nn}$, where R_{nn} is the nearest neighbor distance. Unless otherwise specified, R_{nn} is determined from the ideal lattice at a given density before distortions are applied. Beyond this cutoff distance, correlation in the atomic motions is negligible, and therefore only pairs of atoms in the set of IPs contribute to the the two-body part of the wavefunction.

Throughout the VMC simulations, new atomic positions are selected from the atomic wavefunction which can be written by collecting all terms involving a particular atom i in Eq. 2.1 above. This equation then becomes

$$\psi_i = Ae^{-a_{xy}(s_{i,x}^2+s_{i,y}^2)}e^{-a_z s_{i,z}^2}\prod_{j\in \mathrm{IP}_i} e^{-\frac{1}{2}(b/R_{ij})^5}, \tag{2.2}$$

which can be further grouped into a product of one-body and two-body terms:

$$\psi_i = \Psi_1\Psi_2. \tag{2.3}$$

where Ψ_1 contains the a_i terms and Ψ_2 contains the product of b terms. Metropolis-style Monte Carlo moves[5] are used to generate new atomic configurations for each atom in the system sequentially by randomly sampling the one-body probability density $|\Psi_1|^2$ using the random number generator detailed in Ref. [6]. For each atom, the new configuration is accepted or rejected according to the two-body probability density, $|\Psi_2|^2$. If the two-body probability density of the new configuration is higher than that of the old configuration, the move is accepted and the atomic position is updated. Otherwise, the move is conditionally accepted with the probability $|\Psi_2(\mathbf{Q}')|^2/|\Psi_2(\mathbf{Q})|^2$ where $\mathbf{Q}$ and $\mathbf{Q}'$ correspond to the old and new configurations, respectively. After the move is accepted or rejected, the process is repeated for the next atom in the system. A single Monte Carlo cycle (MCC) in these simulations corresponds to attempting to move each atom in the ensemble once.

The atomic configurations can be used to calculate the instantaneous kinetic and potential energies of the system. However, because not every atomic move is accepted in a given MCC (in practice, approximately 45% of the total moves are accepted[1]), there is strong correlation in the ensemble configurations from one MCC to the next which must be taken into account when evaluating the statistical uncertainty of the average energies obtained from the instantaneous values. In order to eliminate this correlation, snapshots of the atomic positions are only recorded every 50 MCCs. The use of this interval is justified in Chapter 3, Sec. 3.2.1[1] where it is demonstrated that there is negligible correlation between the potential energies calculated from sequential snapshots.

The instantaneous kinetic and potential energies of the system are therefore evaluated only when snapshots of the atomic positions are recorded. The kinetic energy is calculated according to Eq. 2.4,

$$T = T_1 + T_2 = N\frac{\hbar^2(2a_{xy} + a_z)}{2m_{He}} + \sum_{(i,j)\in \mathrm{IP}} \frac{5\hbar^2 b^5}{2\mu_{i,j} R_{ij}^7}, \tag{2.4}$$

where T_1 and T_2 are the one- and two-body contributions to the total kinetic energy T, N is the total number of atoms in the ensemble, m_{He} is the mass of a single He atom, and $\mu_{i,j} = m_{He}/2$. Because T_1 does not depend on the atomic positions, its value can be determined exactly. In our initial simulations, the potential energy is assumed to be pairwise-additive, following the relationship

$$V = \sum_{(i,j)\in \mathrm{IP}} V_{A2}(R_{ij}), \tag{2.5}$$

where V_{A2} refers to the Aziz HFD-B(He) pair potential[7]. This formula assumes no contribution to the potential energy from pairs of atoms outside of the set of IPs. However, their contribution is accounted for via a long-range correction procedure detailed below in Sec. 2.2.3. Three-body interactions are also eventually incorporated into these calculations throughout the simulations, however this will be discussed in Sec. 2.4.

The instantaneous total energy of the system is then calculated from the sum of the kinetic and potential energies. When averaged over the total number of snapshots, p, this average total energy (in the absence of the long-range correction) is given by Eq. 2.6:

$$\langle E \rangle = T_1 + \frac{1}{p}\sum_{n=1}^{p}(T_2(\mathbf{Q}_n) + V(\mathbf{Q}_n)). \tag{2.6}$$

The expectation values in Eq. 2.6 are functions of the three variational parameters which are optimized to find the values which minimize the average total energy per atom.

2.2.1 Reweighting

The VMC wavefunction optimization procedure involves varying the a_{xy} and a_z parameters simultaneously while the b parameter is held fixed until a minimum energy per atom is found, after which point the b parameter is varied while the a_{xy} and a_z parameters remain fixed. This process is repeated until an additional round of optimization shows no significant change in the variational parameters. However, manual scanning of the variational parameters can be computationally expensive and requires longer VMC simulations in order to see statistically significant differences in the energy for smaller changes in the variational parameters. Therefore, in order to more precisely determine the optimized parameters without significantly increasing the computational cost, we implement a reweighting method during the optimization procedure. This method takes advantage of the fact that for small changes in the variational parameters, the distributions of the atomic configurations do not change significantly. Snapshots generated from one set of wavefunction parameters can therefore be used to estimate the average energy associated with a new set of parameters by reweighting the observables according to Eq. 2.7,

$$\langle\Psi'|\hat{H}|\Psi'\rangle = \int |\Psi'(\mathbf{Q})|^2 E'(\mathbf{Q})d\mathbf{Q} = \int |\Psi(\mathbf{Q})|^2 w(\mathbf{Q})E'(\mathbf{Q})d\mathbf{Q}, \tag{2.7}$$

where $w(\mathbf{Q}) = \left|\frac{\Psi'(\mathbf{Q})}{\Psi(\mathbf{Q})}\right|^2$, Ψ is the original wavefunction, Ψ' is the wavefunction with the new set of parameters, and E' is the local energy of the wavefunction with the new parameters. Using this relationship, the average energy associated with the new parameters is given by Eq. 2.8,

$$\langle E\rangle \approx \frac{\left(\sum\limits_{Q} E'(\mathbf{Q})w(\mathbf{Q})\right)}{\sum\limits_{Q} w(\mathbf{Q})}, \tag{2.8}$$

which is essentially a weighted average over all of the configurations sampled based on the overlap between the old and the new wavefunctions.

The VMC optimization procedure begins with a manual scanning of the a_{xy} and a_z parameters until an approximate minimum energy is determined, at which point the reweighting method is used to estimate the energies of wavefunctions with parameters $a_{xy} = a'_{xy} \pm d_{xy}$ and $a_z = a'_z \pm d_z$ where a'_{xy} and a'_z are the approximate optimal parameters and d_{xy} and d_z are small changes applied to these parameters. Three different values of d_{xy} and d_z are used to evaluate the reweighted energies. When a'_{xy} and a'_z are close to the exact optimized parameters, the energy contours fitting the reweighted energies are found to be described by ellipses according to Eq. 2.9.

$$\begin{aligned} E(a_{xy}, a_z) = C_1(a_{xy} - a^o_{xy})^2 + 2C_2(a_{xy} - a^o_{xy})(a_z - a^o_z) \\ + C_3(a_z - a^o_z)^2 + E(a^o_{xy}, a^o_z), \end{aligned} \tag{2.9}$$

where a^o_{xy} and a^o_z are the optimized parameters and $E(a^o_{xy}, a^o_z)$ is the minimum energy. Fitting the reweighted energies to this equation therefore provides an improved estimate of the optimal parameters. VMC snapshots are then generated using these updated parameters and the reweighting procedure is repeated once more. The motivation behind multiple applications of the reweighting method is explained in detail in Chapter 3, Sec. 3.2.4.

After two rounds of reweighting, the final set of optimized a_{xy} and a_z parameters are fixed and the b parameter is manually adjusted in the VMC simulations until an approximate minimum energy per atom is determined. The reweighting procedure is then applied to the b parameter and the reweighted energies are fit to a quadratic equation in order to determine the optimal b parameter. Again, two rounds of this reweighting procedure are performed in order to determine the next set of optimized parameters.

Optimization of the VMC trial wavefunctions always begins and ends with the a_i parameters, and therefore a_i optimization is repeated after the b parameter is optimized. After these three optimization steps, if a reweighting calculation of the b parameter shows no significant change, the wavefunction is considered to be optimized. Otherwise the b parameter is reoptimized, followed by the a_i parameters. Once the wavefunction has been fully optimized, another VMC simulation is performed using the optimal parameters in order to calculate the approximate ground state energy per atom.

2.2.2 The Debye-Waller Factors and Atomic Probability Density

Using snapshots of atomic configurations from the optimized wavefunction, the DW factors corresponding to motion in and perpendicular to the basal plane of the crystal can be calculated. The many different forms of the DW factor can all be reduced to the mean squared displacement, $\langle u_j^2 \rangle$ and therefore this quantity will be used in place of the DW factor to allow for a more direct comparison to previously published results. The formula for $\langle u_j^2 \rangle$ is given by Eq. 2.10,

$$\langle u_j^2 \rangle = \frac{1}{N} \cdot \frac{1}{M} \sum_{n=1}^{N} \sum_{m=1}^{M} (s_{n,j}^m)^2, \tag{2.10}$$

where $s_{n,j}^m$ is a Cartesian component ($j = x$, y, or z) of the displacement vector of atom n in configuration m. $\langle u_j^2 \rangle$ is therefore equal to the square of the displacement vector along direction j over N atoms and M configurations sampled in the VMC simulations.

The VMC atomic snapshots can also be used to determine the form of the atomic probability density function for our simulations. In Chapter 3, Fig. 3.2 we present a histogram of the atomic displacements of atom 1 in the x-direction which appear to follow a Gaussian distribution. Similar histograms are observed for displacements in the y and z-directions as well. Analysis of the atomic displacements in Chapter 3, Sec. 3.2.2 confirms that the distributions of the atomic displacements in the x, y, and z directions are well represented by Gaussian functions, and that these distributions are independent of one another[1].

Assuming a Gaussian form of the probability density given in Eq. 2.11,

$$P_j = \frac{\sqrt{\alpha_j}}{\sqrt{\pi}} e^{-2\alpha_j (s_j)^2}, \tag{2.11}$$

the Gaussian parameter α_j for a given Cartesian direction j can be calculated from $\langle u_j^2 \rangle$ according to Eq. 2.12,

$$\alpha_j = \frac{1}{4\langle u_j^2 \rangle^2}. \tag{2.12}$$

Further justification for the use of a Gaussian probability density function can be found in Chapter 3, Sec. 3.2.2.

In the ideal hcp lattice and those which are distorted only by compression or expansion along the z-axis, symmetry in the x,y-plane results in $\alpha_x = \alpha_y$ and therefore this quantity will be called α_{xy}. Using this notation and the independent nature of the one-dimensional probability density functions, the three-dimensional probability density function can be written as the product of P_x, P_y, and P_z, as shown in Eq. 2.13.

$$P = \frac{\alpha_{xy}\sqrt{\alpha_z}}{(\sqrt{\pi})^3} e^{-2\alpha_{xy}(s_x^2+s_y^2)-2\alpha_z(s_z^2)} \tag{2.13}$$

2.2.3 Long-Range Corrections

The average energy calculated from the optimized wavefunction includes only contributions from atomic pairs that are considered to be interacting. Atomic pairs separated by a distance larger than the cutoff distance of $2.05R_{nn}$, however, make a non-negligible contribution to the potential energy and therefore must be accounted for. Because all atoms in the ^{4}He system are identical, we can arbitrarily choose a central atom from which to calculate the long-range correction (LRC) to the potential energy without loss of accuracy. For our purposes, we consider all atoms outside of the interacting pair cutoff distance from atom 1 for these calculations.

The LRC procedure considers two different regions of atoms: those atoms within the N atom simulation cell which fall outside of the $2.05R_{nn}$ cutoff distance from atom 1 (region 1), and the infinite number of atoms beyond the finite simulation cell represented by the periodic boundary conditions (region 2).

For large interatomic distances R, the Aziz HFD-B(He) potential energy used in the VMC simulations[7] becomes

$$V_{lrc}(R) = -\frac{C_6}{R^6} - \frac{C_8}{R^8} - \frac{C_{10}}{R^{10}}. \tag{2.14}$$

Contributions to the potential energy from atoms in region 1 can be calculated by evaluating the following integral for each of these atoms paired with atom 1:

$$\langle V_{lrc} \rangle = \int\int P_1 V_{lrc}(R) P_i d\vec{s_1} d\vec{s_i}, \tag{2.15}$$

where $\vec{s_1}$ and $\vec{s_i}$ are the instantaneous displacement vectors of atoms 1 and i from their average lattice positions, P_1 and P_i are the three-dimensional probability densities of atoms 1 and i (Eq. 2.13), and $R = |\vec{R}|$ is the interatomic distance corresponding to the instantaneous interatomic vector

$$\vec{R} = \vec{R}_{latt} + \vec{s_i} - \vec{s_1}, \tag{2.16}$$

where the component in the j-direction is $R_j = R_{latt,j} + s_{i,j} - s_{1,j}$. $\vec{R}_{latt}$ is the vector from the average lattice position of atom 1 to atom i and is constant. R_j is therefore a function of only the displacement vector components $s_{1,j}$ and $s_{i,j}$.

The atomic pairs considered in the LRC are not defined to be interacting, and therefore correlation in the atomic motion is not considered. Taking advantage of the separable form of the probability densities, the integral above can be written as a product of six Gaussian terms and the potential energy function $V_{lrc}(R)$. Rewriting $s_{i,j}$ as $x_{i,j}/\sqrt{2\alpha_j}$ in Eq. 2.13 allows us to evaluate the integral in Eq. 2.15 using Gaussian quadrature according to the relationship shown in Eq. 2.17,

$$\int_{-\infty}^{\infty} \exp(-x^2) f(x) dx \approx \sum_{k=1}^{N_G} w_k f(x_k), \tag{2.17}$$

where the weights, w_k, and abscissas, x_k, are determined by the number of nodes used, N_G. For our calculations, 8 nodes are sufficient to reach the converged value of $\langle V_{lrc} \rangle$ for region 1.

While this treatment is computationally efficient for the atoms in region 1, a different approach must be used to treat the infinite number of atoms in region 2. For this region, we make use of the fact that as the interatomic separation between two atoms increases, the zero point motions of the atoms change the interatomic distance negligibly, and therefore region 2 atoms can be treated by considering only their average lattice positions (see Chapter 3, Sec. 3.2.3). In addition, at longer interatomic distances, the $1/R^6$ term makes the primary contribution to the potential energy, and therefore only this term is calculated for the atoms in region 2.

The sum of the $1/R^6$ contributions for an infinite number of atoms in an ideal hcp lattice has previously been reported by Hirchfelder, Curtiss, and Bird[8] as a lattice sum, $S_6^* = \sum_{i=2}^{\infty} \frac{R_{nn}^6}{R_{1i}^6}$, times $1/R_{nn}^6$. Therefore in an ideal lattice, the contribution of region 2 atoms to the LRC is calculated by subtracting contributions of interacting pairs of atom 1 and those atoms in region 1 from the sum $S_6^* C_6/R_{nn}^6$ so that these atoms are not accounted for twice.

In simulations which consider distorted lattices, however, the S_6^* sum has not been previously calculated, and therefore we have developed a method to derive this value for any distortion of the hcp lattice. Fig. 2.1 shows how a finite sum over

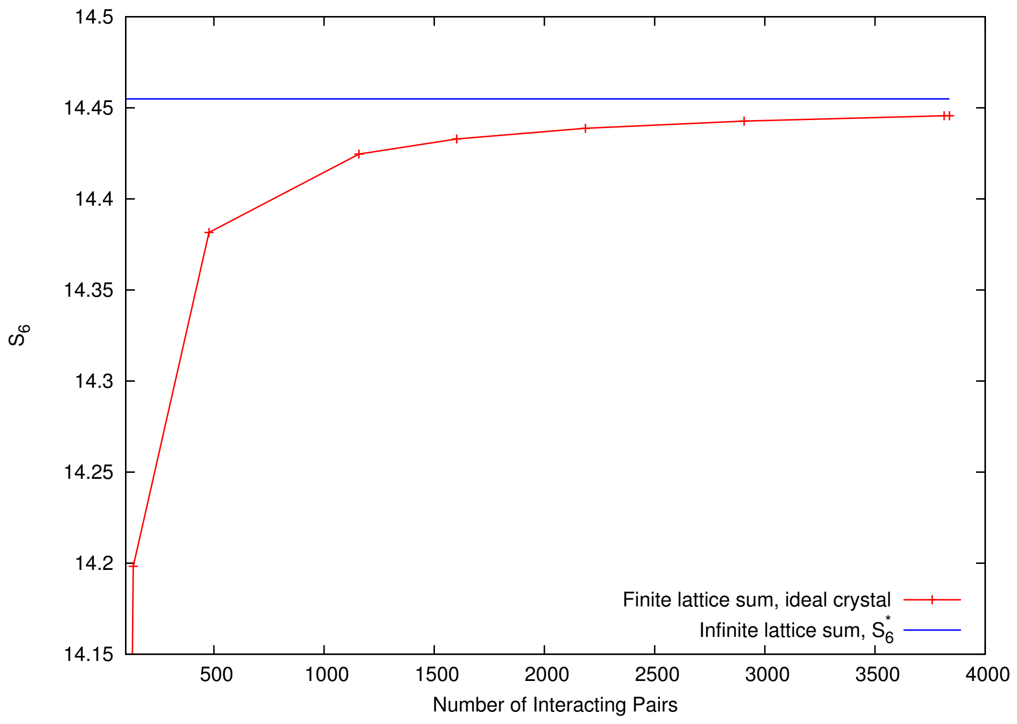


Figure 2.1: Truncated lattice sum S_6 vs. the number of interacting pairs considered in an ideal hcp lattice (red line). The infinite sum from Hirschfelder, Curtiss, and Bird[8] is provided for comparison (blue line).

$1/R^6$ contributions, S_6, approaches that of the inifinite lattice sum S_6^* in an ideal lattice. For each point at which S_6 was evaluated, a correction factor of S_6^*/S_6 can be calculated to quantify the difference between the truncated and infinite sums. Similar truncated sums in lattices with varying c/a ratios are shown in Fig. 2.2 where care has been taken to ensure that the same atoms are included in the distorted lattice calculations as were used for ideal lattice. We found that correcting these truncated sums at each point using the same correction factor from the ideal hcp lattice results in corrected sums which agree to within 5.0x10^{-4}. This corrected sum is taken to be S_6^* for these distorted lattices. We have reported these values for a number of distorted lattices with varying c/a ratios[1]. However this method is not limited to distortions of the c/a ratio and is in fact used to calculate the LRC in all simulations with distorted lattices. Using this procedure, the total LRC is given by the sum of the region 1 Gaussian quadrature calculation and the $-C_6/R^6$ contribution of the region 2 atoms.

2.3 VPI

The VMC method described above allows for the calculation of the average energy of a selected trial wavefunction which can be optimized in order to approximate the ground state wavefunction. However, this method is limited by the variational principle, and as such the minimum energies from the VMC simulations will always be higher than the true ground state energies. In order to eliminate error due to the variational principle, we utilize VPI[9], also known as the path integral ground state method (PIGS)[10], which is an exact method from which exact ground state energies and other coordinate-space observables can be obtained, within statistical uncertainty. We have previously reported the VPI method described below in Ref. [11] (Chapter 4, Sec. 4.2.2).

In the VPI method, the hcp ^{4}He system is modeled as a p-bead polymer chain where each bead is a replica of the full N atom system. Progression down the chain

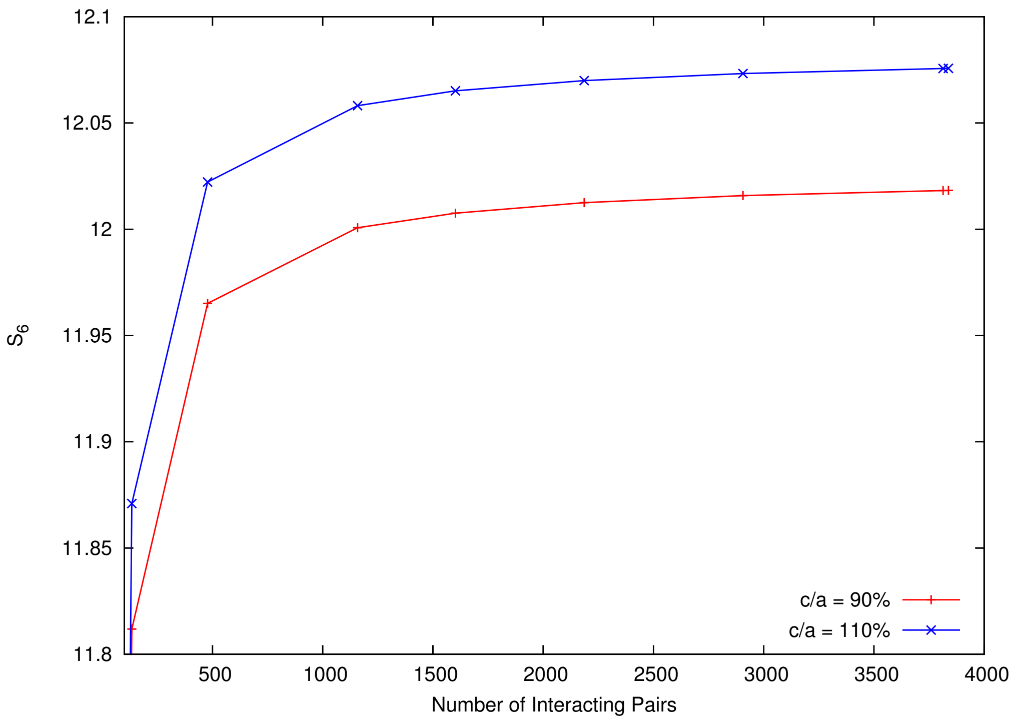


Figure 2.2: Truncated lattice sum S_6 vs. the number of interacting pairs considered in distorted hcp lattices with c/a ratios of 90% (red) and 110% (blue) of the ideal c/a ratio.

in either direction corresponds to the evolution of the trial wavefunction in imaginary time according to the imaginary time propagator $\exp[-\hat{H}\Delta\tau/\hbar]$[4]. The links between adjacent beads therefore represent the evolution of the wavefunction through $\Delta\tau$ units of imaginary time.

This method takes advantage of the fact that a trial wavefunction Ψ_{tr} can be written as a linear combination of the eigenfunctions (Ψ_i) of the exact Hamiltonian as shown in Eq. 2.18, where the eigenfunctions are ordered according to their corresponding eigenvalues from lowest energy to highest.

$$\Psi_{tr} = \sum_{i=0}^{\infty} c_i \Psi_i \tag{2.18}$$

Application of the imaginary time propagator to this trial wavefunction will project out the lowest energy eigenfunction of the Hamiltonian, Ψ_0, which corresponds to the ground state wavefunction of the system, provided that c_0 is not too small. We will refer to this wavefunction as Ψ_{gs}. As the coefficient c_0 increases, indicating a greater overlap between the trial and ground state wavefunctions, fewer applications of the imaginary time propagator are required for Ψ_{tr} to converge to Ψ_{gs}. We therefore use the optimized wavefunctions from VMC as our starting trial wavefunctions as these closely approximate the ground state wavefunction.

Our VPI simulations are performed using the QSATS code[4] which we have altered to accept independent a_{xy} and a_z parameters for the trial wavefunction as well as lattice distortion parameters, where necessary. This implementation samples configurations of the beads in the polymer chain using Metropolis Monte Carlo moves[5] according to the probability density in Eq. 2.19,

$$P(C) = A\Psi_{tr}(\mathbf{Q}_1)\Psi_{tr}(\mathbf{Q}_p)exp\Big(-\frac{\Delta\tau}{\hbar}\sum_{j=1}^{p-1} F(\mathbf{Q}_j, \mathbf{Q}_{j+1})\Big), \tag{2.19}$$

where $C = \mathbf{Q}_1, \mathbf{Q}_2, ..., \mathbf{Q}_p$ is the configuration of the entire polymer chain and $\mathbf{Q}_j$ represents the configuration in the jth replica[4]. The function $F(\mathbf{Q}_j, \mathbf{Q}_{j+1})$ is given by Eq. 2.20[4],

$$F(\mathbf{Q}_j, \mathbf{Q}_{j+1}) = \frac{V(\mathbf{Q}_j) + V(\mathbf{Q}_{j+1})}{2} + \frac{1}{2(\Delta\tau)^2} \sum_{n=1}^{N} m_{He}(\mathbf{r}_{n;j+1} - \mathbf{r}_{n;j})^2 \tag{2.20}$$

where $V(\mathbf{Q}_j)$ is the full potential energy of the of the jth replica, and $r_{n;j}$ is the instantaneous position of atom n in replica j. The exponential term in Eq. 2.19 is related to the Trotter factorization[12] of the imaginary time propagator shown in Eq. 2.21.

$$exp(-\Delta\tau\hat{H}/\hbar) \approx exp(-\Delta\tau\hat{V}/2\hbar) \times exp(-\Delta\tau\hat{T}/\hbar) \times exp(-\Delta\tau\hat{V}/2\hbar) \tag{2.21}$$

The probability density in Eq. 2.19 therefore represents the evolution of two trial wavefunctions at either end of the polymer chain (i.e., the $j = 1$ and $j = p$ beads) in either direction along the chain. For sufficiently large p and short $\Delta\tau$, the distribution of the interior beads approaches $|\Psi_{gs}|^2$ while the terminal beads sample the probability density $|\Psi_{tr}\Psi_{gs}|$. Atomic snapshots generated from the interior beads can therefore be used to calculate the average ground state properties of the system.

Similar to the VMC simulations, a single MCC in these simulations corresponds to an attempt to move each atom in each replica once, sequentially. The QSATS algorithm automatically calculates all contributions to the probability density in Eq. 2.19 except for the potential energy contribution to the F function. This quantity, referred to as V, is evaluated in the old and new configuration for each attempted move. If ΔV is negative, the move is accepted and the atomic position is updated. Otherwise, the move is conditionally accepted with the probability $e^{-\Delta V \Delta\tau/\hbar}$.

These simulations utilize the same cutoff criterion when determining interacting pairs as the VMC method above, and as before only atoms defined to be interacting pairs contribute to the potential energy calculated throughout the simulations. In order to calculate the true ground state energy, then, the same LRC procedure reported above is used to calculate a correction to the potential energy for every bead in the ensemble. This requires calculating the α_{xy} and α_z Gaussian parameters separately for each bead.

2.4 Three-Body Interactions

Investigations into the role of three-body interactions in hcp solid ^{4}He presented here make use of the nonadditive three-body potential reported by Cencek, *et al.* in 2009[13]. As mentioned in Chapter 1, 253 individual ^{4}He$_3$ configurations with side lengths as small as $R = 1.75a_o$ were used to parameterize the Cencek potential which approaches full-configuration-interaction accuracy. This nonadditive three-body contribution to the potential energy is incorporated into our VMC and VPI simulations using two different methods: a computationally efficient perturbative approach where snapshots of atomic configurations from the two-body QMC simulations are used to calculate a three-body correction, and a full-incorporation method where the nonadditive three-body potential is added to the Aziz two-body potential throughout the simulations. In both cases, only contributions from interacting trimers (ITs) are considered, where an interacting trimer is defined to consist of a central atom and two of its twelve nearest neighbors. The details of both methods, discussed below, have been previously reported in Ref. [11] (Chapter 4, Sec 4.2.3).

2.4.1 Perturbative Treatment

The perturbative treatment of three-body interactions in the hcp ^{4}He system relies on the assumption that three-body interactions do not significantly impact the ground state wavefunction of the system, and therefore snapshots generated from a two-body wavefunction can be used to calculate the three-body energy. This is essentially a first order perturbation of the two-body energy (Eq. 2.22).

$$E = E_2 + \langle V_3 \rangle \tag{2.22}$$

In our VMC simulations, the calculation of $\langle V_3 \rangle$ can be further simplified by considering the trimer geometries formed by a central atom and its twelve nearest neighbors in their equilibrium lattice positions. The resulting 66 trimers can be classified based on their central angles as follows: 60°(24), 90°(12), 109.47°(3), 120°(18), 146.44°(6), and 180°(3)[14], where the number in parentheses denotes the number of occurrences of that trimer geometry. The average three-body energy per atom can therefore be calculated by adding contributions from one representative of each trimer geometry, weighted by the number of times they occur in the 66 ITs. The three-body contribution from equilateral trimers is divided by three because these trimers would appear in the set of ITs for each of the three atoms involved. Because every ^{4}He atom in the hcp lattice is identical, this calculation is only performed for one central atom (atom 1). We show in Chapter 4, Sec. 4.2.3.1 that both the choice of this central atom and the use of the representative trimers result in no loss of accuracy compared to treating each IT in the system individually[11]. Although the uncertainty in the three-body energy is increased when this method is employed, the added uncertainty is still significantly lower than the error due to the variational principle (calculated as the difference between the VMC and VPI total energies). VPI, however, is an exact method which does not suffer from variational error, and therefore all of the ITs for each central atom are accounted for in the calculation of $\langle V_3 \rangle$ in order to reduce the uncertainty, again being careful to avoid triple counting. Using this perturbative treatment, the computational cost (quantified in CPU hours) of the VMC and VPI simulations increases by approximately 0.2% and 5.0%, respectively.

2.4.2 Full Incorporation Method

In order to evaluate the assumption that three-body interactions do not have a strong effect on the ground state wavefunction, we fully incorporate the Cencek three-body potential into the VMC and VPI simulations. This requires substituting the pairwise-additive model of the potential energy given by Eq. 2.5 with Eq. 2.23:

$$V = \sum_{(i,j)\in\text{IPs}} V_{A2}(R_{ij}) + \sum_{(i,j,k)\in\text{ITs}} V_3(R_{ij}, R_{jk}, R_{ik}) \tag{2.23}$$

where V_3 is the Cencek nonadditive three-body potential[13]. The three-body energy of all ITs is therefore evaluated any time the two-body potential energy is calculated. In the VMC simulations, this equates to every 50 MCCs when the atomic snapshots are recorded. This additional three-body calculation increases the computational cost eight-fold. VPI simulations, however, require the calculation of the potential energy in order to accept or reject every attempted Monte Carlo move. Full incorporation of the three-body potential into these simulations increases the computational cost approximately 256-fold. For this reason, full-incorporation VPI simulations are only performed at four higher densities where three-body interactions are more significant and are therefore expected to make a greater contribution to the ground state wavefunction.

2.5 Equations of State

The ground state energies calculated from the simulations decribed above are used to derive the zero-temperature energy-volume and pressure-volume equations of state (EOSs). These calculations are detailed in Chapter 4 and were previously reported in Ref. [11]. The form of the EOS employed was taken from the experimental EOS reported by Driessen, *et al.*[15],

$$E(V_m) = E_o - P_o V_m + aV_m^{-\frac{8}{3}} + bV_m^{-2} + cV_m^{-\frac{4}{3}} + dV_m^{-\frac{2}{3}} \tag{2.24}$$

where Eq. 2.24 is a modified Birch EOS that has been rearranged so that the equation is linear with respect to the fitting parameters E_o, P_o, a, b, c, and d. This is done simply to allow for faster convergence of the Levenberg-Marquardt[16] fitting algorithm employed in Gnuplot version 4.2[17]. As in the experimental EOS[15], no single set of parameters is able to accurately fit the data across the full density range studied here, and therefore we divide the data into high and low density regions. For our calculations, 11.02 cm^3/mol is used as the transition point between the two regions. We find that uncertainties in the fitting parameters are significantly reduced by constraining $P_o = 0$ in the high density region, and $a = 0$ in the low density region. These constraints have a negligible impact on the residuals for each fit.

Using Eq. 2.24 above, the pressure-volume EOS is described by Eq. 2.25,

$$P(V_m) = -\frac{\delta E}{\delta V_m} = P_o + \frac{8}{3}aV_m^{-\frac{11}{3}} + 2bV_m^{-3} + \frac{4}{3}cV_m^{-\frac{7}{3}} + \frac{2}{3}dV_m^{-\frac{5}{3}}. \tag{2.25}$$

For each data set, the resulting $P(V_m)$ equation is compared to the experimental pressure-volume data from Driessen, *et al.*[15] in Chapter 4, Sec. 4.3.3.

2.6 Elastic Constants

The impact of three-body interactions on the elastic properties of hcp solid ^{4}He is also investigated by calculating the elastic constants at $T = 0$ K with and without three-body interactions. An hcp lattice has five nonzero elastic constants: C_{11}, C_{12}, C_{13}, C_{33}, and C_{44}, also known as the shear modulus. Following the procedure reported by Cazorla and Boronat[18], these quantities depend on the the derivative of the c/a ratio with respect to volume, as well as the second derivative of energy with respect to volume (i.e., the bulk modulus K) and the three heterogeneous strain variables, η, γ, and ϵ which correspond to the pure shear constants C_0, C_{66}, and C_{44}, respectively. Changing η, γ, and ϵ corresponds to changing the c/a ratio, the angle between the x- and y-axes in the basal plane of the crystal, and the angle between the basal plane of the crystal and the z-axis, respectively, at a constant volume.

Following Ref. [18], the c/a ratio in the equilibrium geometry is assumed to be constant and therefore $\left(\frac{\delta \ln c/a}{\delta V}\right)_{V=V_0} = \frac{C_{33}-C_{11}-C_{12}+C_{13}}{C_0} = 0$, where the subscript $V = V_0$ refers to the equilibrium geometry of the system at a given molar volume. The bulk modulus and pure shear constants can then be defined according to Eqs. 2.26-2.29,

$$K = -V_0\left(\frac{\delta P}{\delta V}\right)_{V=V_0} \tag{2.26}$$

$$C_0 = \frac{2}{V_0}\left(\frac{\delta^2 E}{\delta \eta^2}\right)_{V=V_0} \tag{2.27}$$

$$C_{66} = \frac{1}{V_0}\left(\frac{\delta^2 E}{\delta \gamma^2}\right)_{V=V_0} \tag{2.28}$$

$$C_{44} = \frac{1}{V_0}\left(\frac{\delta^2 E}{\delta \epsilon^2}\right)_{V=V_0} \tag{2.29}$$

where the primitive lattice vectors of the hcp ^{4}He unit cell are defined in Eq. 2.30,

$$\begin{aligned} \mathbf{a}_1 &= a\phi^{-1}\gamma^{1/2}(\gamma^{-1}\tfrac{\sqrt{3}}{2}\mathbf{i} + \tfrac{1}{2}\mathbf{j} + \tfrac{\epsilon}{2}\mathbf{k}) \\ \mathbf{a}_2 &= a\phi^{-1}\gamma^{1/2}(\gamma^{-1}\tfrac{\sqrt{3}}{2}\mathbf{i} - \tfrac{1}{2}\mathbf{j} + \tfrac{\epsilon}{2}\mathbf{k}) \\ \mathbf{a}_3 &= c\phi^2\mathbf{k}, \end{aligned} \tag{2.30}$$

where a and c are the standard hcp lattice parameters in and perpendicular to the basal plane of the crystal and $\phi = \sqrt{1+\eta}$. When $\eta = \epsilon = 0$ and $\gamma = 1$, these primitive lattice vectors correspond to the equilibrium hcp geometry. For practical purposes in our simulations, changing η, γ, and ϵ is more easily thought of in terms of changes in the atomic x, y, and z coordinates, shown in Eq. 2.31.

$$\begin{aligned} x &\rightarrow x\gamma^{-1/2}\phi^{-1} \\ y &\rightarrow y\gamma^{1/2}\phi^{-1} \\ z &\rightarrow z\phi^2 + y\epsilon \end{aligned} \tag{2.31}$$

By changing one of the three heterogeneous strain variables at a time and fitting the resulting responses in the ground state energy to appropriate functions, we can calculate C_0, C_{66}, and C_{44} above. Taking advantage of previously determined relationships between the elastic constants of an hcp system[18], the remaining four nonzero elastic constants can be calculated according to Eqs. 2.32-2.35.

$$C_{11} = K + C_{66} + \frac{1}{18}C_0 \tag{2.32}$$

$$C_{12} = K - C_{66} + \frac{1}{18}C_0 \tag{2.33}$$

$$C_{13} = K - \frac{1}{9}C_0 \tag{2.34}$$

$$C_{33} = K + \frac{2}{9}C_0 \tag{2.35}$$

We calculate the elastic constants as well as the bulk modulus and pure strain constants at a range of molar volumes from 7.88 cm^3/mol to 20.78 cm^3/mol using VPI simulations with and without perturbatively corrected three-body interactions. For each of the pure strains, simulations are performed using eight different values of the corresponding heterogeneous strain variable in addition to the equilibrium value. Additional details for these calculations as well as their results are discussed in Chapter 5.

2.7 References

[1] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **144**, 084505 (2016). 15, 17, 22, 25

[2] J.-P. Hansen and D. Levesque, *Physical Review* **165**, 293 (1968). 16

[3] E. W. Draeger and D. M. Ceperley, *Phys. Rev. B* **61**, 12094 (2000). 16

[4] R. J. Hinde, *Computer Physics Communications* **182**, 2339 (2011). 16, 28, 29

[5] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *The Journal of Chemical Physics* **21**, 1087 (1953). 17, 28

[6] P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton, *Operations Research* **50**, pp. 1073 (2002). 17

[7] R. A. Aziz, F. R. McCourt, and C. C. Wong, *Molecular Physics* **61**, 1487 (1987). 18, 23

[8] J. Hirschfelder, C. Curtiss, and R. Bird, *Molecular theory of gases and liquids*, Structure of matter series, Wiley, 1954. 24, 26

[9] D. M. Ceperley, *Rev. Mod. Phys.* **67**, 279 (1995). 25

[10] A. Sarsa, K. E. Schmidt, and W. R. Magro, *The Journal of Chemical Physics* **113**, 1366 (2000). 25

[11] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **146**, 094510 (2017). 25, 30, 31, 32

[12] H. F. Trotter, *Proceedings of the American Mathematical Society* **10**, 545 (1959). 29

[13] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 30, 32

[14] R. J. Hinde, *Chemical Physics Letters* **460**, 141 (2008). 31

[15] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 32, 33

[16] J. J. Moré, The Levenberg-Marquardt algorithm: Implementation and theory, in *Numerical Analysis*, edited by G. Watson, volume 630 of *Lecture Notes in Mathematics*, pp. 105–116, Springer Berlin Heidelberg, 1978. 33

[17] T. Williams, C. Kelley, and many others, Gnuplot 4.2: an interactive plotting program, http://gnuplot.sourceforge.net/, 2009. 33

[18] C. Cazorla, Y. Lutsyshyn, and J. Boronat, *Phys. Rev. B* **85**, 024101 (2012). 33, 34, 35

Chapter 3

Search for Anisotropy in the Debye-Waller Factors of hcp ^{4}He

A version of this chapter was originally published in The Journal of Chemical Physics in 2016 by Ashleigh Barnes and Robert Hinde. Only minor revisions have been made to allow for consistency of formatting. The original citation is given below.

Ashleigh L. Barnes and Robert J. Hinde. *The Journal of Chemical Physics*, **144**(8), 084505 (2016).

The article was written by Ashleigh Barnes with revision suggestions from Robert Hinde.

The Journal of Chemical Physics is an AIP publication. The AIP policy on article reproduction is as follows:

Abstract

The properties of hexagonal close packed (hcp) solid ^{4}He are dominated by large atomic zero point motions. An accurate description of these motions is therefore necessary in order to accurately calculate the properties of the system, such as the Debye-Waller (DW) factors. A recent neutron scattering experiment reported significant anisotropy in the in-plane and out-of-plane DW factors for hcp solid ^{4}He at low temperatures, where thermal effects are negligible and only zero point motions are expected to contribute. By contrast, no such anisotropy was observed either in earlier experiments or in path integral Monte Carlo (PIMC) simulations of solid hcp ^{4}He. However, the earlier experiments and the PIMC simulations were both carried out at higher temperatures where thermal effects could be substantial. We seek to understand the cause of this discrepancy through variational quantum Monte Carlo (VMC) simulations utilizing an accurate pair potential and a modified trial wavefunction which allows for anisotropy. Near the melting density, we find no anisotropy in an ideal hcp ^{4}He crystal. A theoretical equation of state is derived from the calculated energies of the ideal crystal over a range of molar volumes from 7.88 to 21.3 cm^3, and is found to be in good qualitative agreement with experimental data.

3.1 Introduction

Solid ^{4}He is the simplest quantum solid, and therefore provides a reasonable starting point for the development of theoretical models to better describe this unique class of solids. At absolute zero and pressures above approximately 25 bar, solid ^{4}He is found in the hexagonal close packed (hcp) phase; however at higher temperatures and pressures body centered cubic and face centered cubic phases are also accessible[1]. For the purpose of this investigation, only the hcp phase is of interest. Because hcp ^{4}He is a quantum solid, its properties are dominated by large zero point motions of the atoms about their average lattice positions[2]. An accurate description of these zero point motions, therefore, is essential to better understanding and predicting the properties of this material. Previous neutron scattering[3, 4] and x-ray diffraction[5, 6, 7] studies have reported no significant difference between atomic zero point motions parallel or perpendicular to the crystal's c axis. Finite temperature path integral Monte Carlo (PIMC) simulations[8], as well as Green's-function Monte Carlo (GFMC) investigations[9, 10] conducted at $T = 0$ K, have suggested the same. This is surprising because of the inherent anisotropy in the hcp lattice, and goes against predictions of the properties of quantum solids reported by Chui[11]. More recently, however, a low temperature (0.14 K $< T <$ 1 K) neutron diffraction study was published by Blackburn *et al.*[12] in which a difference of approximately 20% in the Debye-Waller (DW) factors for in-plane and out-of-plane zero point motions was observed. The DW factors were temperature independent over the temperature range in the study, indicating that the authors were observing quantum contributions to the DW factors only. In order to determine the reason for the discrepancy between these experimental findings and the earlier simulations, we begin by utilizing variational quantum Monte Carlo (VMC) simulations with a realistic pair potential in order to determine the DW factors for in- and out-of-plane zero point motions in hcp ^{4}He. VMC simulations will be performed at $T = 0$ K, in contrast to the previous finite temperature PIMC calculations, utilizing a more reliable two-body potential energy function than was employed in earlier GFMC studies at absolute zero. We will demonstrate that when only two-body interactions are considered, our results agree with previous theoretical and experimental findings in that no anisotropy is observed.

3.2 Computational Methods

3.2.1 VMC

VMC is a quantum Monte Carlo simulation technique used to generate a sequence of configurations of a many-particle system. The probability that a given configuration appears in the sequence is given by the probability density function associated with a trial wavefunction. Expectation values of the energy and other coordinate-space observables are calculated by averaging the appropriate functions of the particles' coordinates over the sequence of configurations. In the limit of infinitely long simulations these expectation values converge to the exact values for the trial wavefunction. For finite-length simulations, statistical uncertainties in the expectation values can be estimated using standard statistical methods.

For our system, we employ a Jastrow-Mcmillan[13] style trial wavefunction of the form:

$$\Psi = A \prod_i e^{-a_{xy}(s_{i,x}^2 + s_{i,y}^2)} e^{-a_z s_{i,z}^2} \prod_{i<j} e^{-\frac{1}{2}(b/R_{ij})^5}, \tag{3.1}$$

where A is a normalization factor, $\vec{s_i} = (s_{i,x}, s_{i,y}, s_{i,z})$ is the displacement vector of atom i from its average lattice site, and R_{ij} is the instantaneous distance between atoms i and j. The one-body terms keep atoms localized around their lattice sites according to the variational a_{xy} and a_z parameters, while the two-body terms prevent neighboring atoms from coming into close contact, as determined by the b parameter. In an hcp crystal, a_{xy} and a_z can have different values, and therefore we make a distinction between the one-body terms for motions parallel (x,y) or perpendicular (z) to the basal plane.

When atoms are far apart from one another, there is negligible correlation in their motions, and therefore their contribution to the two-body part of the wavefunction can be ignored. We therefore include in the two-body term only pairs of atoms defined to be interacting pairs (IPs) based on a distance-dependent criterion that is described in detail below. The wavefunction then becomes

$$\Psi = A \prod_i e^{-a_{xy}(s_{i,x}^2 + s_{i,y}^2)} e^{-a_z s_{i,z}^2} \prod_{(i,j)\in \text{IPs}} e^{-\frac{1}{2}(b/R_{ij})^5}. \tag{3.2}$$

If we collect all terms in the wavefunction involving some atom i, we obtain the atomic wavefunction in Eq. 3.3:

$$\psi_i = Ae^{-a_{xy}(s_{i,x}^2+s_{i,y}^2)}e^{-a_z s_{i,z}^2}\prod_{j\in \mathrm{IP}_i} e^{-\frac{1}{2}(b/R_{ij})^5}, \tag{3.3}$$

which we can think of as a product of a one-body term and a set of two-body terms,

$$\psi_i = \Psi_1\Psi_2. \tag{3.4}$$

In this study, Metropolis-style[14] Monte Carlo moves are used to generate new configurations. The fundamental event is a single-atom update in which a provisional new position for atom i is chosen by sampling directly from the one-body probability density $|\Psi_1|^2$. The random number generator utilized in this step is detailed in Ref. [15]. Each movement is then accepted or rejected according to the two-body contribution Ψ_2 in Eq. 3.4. If $\frac{|\Psi_2(\mathbf{Q}')|^2}{|\Psi_2(\mathbf{Q})|^2} > 1$, where $\mathbf{Q}'$ and $\mathbf{Q}$ are the new and old configurations, respectively, the move is accepted, otherwise it is conditionally accepted with the probability $|\Psi_2(\mathbf{Q}')|^2/|\Psi_2(\mathbf{Q})|^2$. A single Monte Carlo cycle (MCC) consists of attempting to move each atom in the ensemble once in this manner.

For each configuration, the instantaneous energy is calculated as the sum of the potential and kinetic energies. In this study, the potential energy is assumed to follow a pairwise-additive relationship:

$$V = \sum_{(i,j)\in \mathrm{IP}} V_{A2}(R_{ij}), \tag{3.5}$$

where V_{A2} is the Aziz HFD-B(He) potential energy[16] between atoms i and j. Potential energy contributions from those atoms not considered to be interacting pairs are accounted for using a long-range correction procedure detailed below. The instantaneous kinetic energy is calculated according to

$$T = T_1 + T_2 = N\frac{\hbar^2(2a_{xy} + a_z)}{2m_{He}} + \sum_{(i,j)\in\mathrm{IP}} \frac{5\hbar^2 b^5}{2\mu_{i,j}R_{i,j}^7}, \tag{3.6}$$

where $\mu_{i,j} = m_{He}/2$[17]. The one-body contribution to the kinetic energy, T_1, is independent of the atomic positions, and therefore its value is exactly known. The expectation value of the total energy, assuming no contribution from atoms outside of the interacting-pair cutoff, is then found by averaging the instantaneous energy over all configurations (Eq. 3.7).

$$\langle E\rangle = T_1 + \frac{1}{p}\sum_{n=1}^{p}(T_2(\mathbf{Q}_n) + V(\mathbf{Q}_n)). \tag{3.7}$$

These expectation values are functions of the three variational parameters a_{xy}, a_z, and b. The optimal parameter values are those for which the average energy per atom is minimized.

Our simulations make use of an ensemble of N=448 atoms arranged in an hcp lattice with an ideal c/a ratio of 1.633. This corresponds to a cell consisting of 8 layers of atoms parallel to the basal plane, each containing 56 atoms, where the basal plane is taken to be perpendicular to the z-axis. Periodic boundary conditions are applied in all three directions. The nearest neighbor distance, R_{nn}, is determined from the average lattice positions scaled to the desired density, and atoms whose lattice sites are separated by the distance $2.05R_{nn}$ or less are considered to be interacting pairs. In the ideal crystal each atom belongs to 56 interacting pairs. Densities studied include the experimental density from the Blackburn study (4.1896x10$^{-3}a_o^{-3}$), corresponding to an experimental external pressure of approximately 25.2 bar[18], as well as higher densities corresponding to pressures of up to 8 kbar. Cell dimensions at the Blackburn density are 48.23934a_o x 48.73935a_o x 45.48050a_o with a nearest neighbor distance of $R_{nn} = 6.96a_o$.

Crystals with distorted lattices ($c/a \neq 1.633$) are also investigated at the Blackburn density. These simulations are initialized using the same cell parameters as above, and various c/a ratios are achieved by scaling the z coordinates and proportionally adjusting the x and y coordinates to obtain the desired c/a ratio while maintaining constant cell volume. R_{nn} is then calculated from the distorted lattice positions. In lattices with $c/a < 1.633$, R_{nn} is the distance between nearest neighbors in adjacent planes, otherwise R_{nn} is the distance between nearest neighbors in the same plane. The same distance criterion used to determine interacting pairs in the ideal crystal is used with distorted lattices. As the crystal is distorted fewer pairs are included within this cutoff region, however the configuration of interacting pairs retains hexagonal symmetry, and contributions to the potential energy from atoms excluded from this region are accounted for in the long-range correction procedure.

In order to minimize sequential correlation, new atomic positions are chosen randomly from the one-body probability density rather than as a shift from the previous position. Additionally, configurations are only recorded every 50 MCCs. In a typical simulation, approximately 45% of all moves are accepted, indicating that on average an atom will move 22 times between these snapshot recordings. Fig. 3.1 investigates sequential correlation in the Monte Carlo simulation by comparing the potential energy felt by atom 1 for consecutive snapshots. The distribution has a Pearson correlation coefficient of -0.0017, indicating no significant relationship between consecutive values. This allows for easy determination of the statistical uncertainties in our observables without the need to account for sequential correlation.

3.2.2 $\langle u^2 \rangle$ Calculation

Once the wavefunction is optimized, the DW factor can be calculated. Since there are many formulations of the DW factor, which can all be related to the mean squared displacement, $\langle u^2 \rangle$, we calculate $\langle u^2 \rangle$ for in-plane and out-of-plane zero point motions for easier comparison to previously published results.

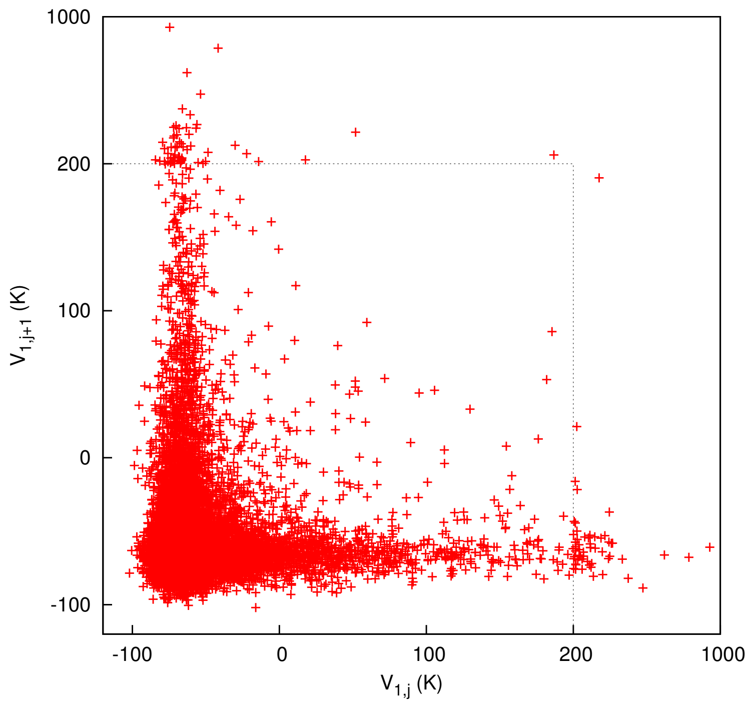


Figure 3.1: Correlation in the potential energy of atom 1 during VMC simulations. $V_{1,j}$ corresponds to the potential energy (K) felt by atom 1 in the jth configuration sampled. Energies beyond 200 K have been scaled for improved visualization.

Snapshots are generated from the optimized wavefunction, and $\langle u^2 \rangle$ is calculated in each direction by averaging the square of the atomic displacements over all N atoms and all M configurations (Eq. 3.8),

$$\langle u_j^2 \rangle = \frac{1}{N} \cdot \frac{1}{M} \sum_{n=1}^{N} \sum_{m=1}^{M} (s_{n,j}^m)^2, \tag{3.8}$$

where $s_{n,j}^m$ is a Cartesian component (j = x, y, or z) of the displacement vector of atom n in configuration m. For these calculations, 2.56x10^6 snapshots were used.

These snapshots are also used to calculate the probability density function for the atomic displacements in each direction. Fig. 3.2a shows a histogram of the x-displacements of an atom in the ideal lattice, which appear to follow a Gaussian distribution. Similar histograms were generated in which only atomic snapshots having a z-displacement between 0.0 and 1.0, 1.0 and 1.5, 1.5 and 2.0, and 2.0 and 3.0a_o were considered. The means of these distributions agreed with that of the full distribution within 0.018a_o, and statistical variances differed from that of the full distribution by less than 10%. This suggests that s_x and s_z are not strongly correlated, which was further confirmed by calculating Pearson correlation coefficients. The Pearson correlation coefficient for s_x and s_z was found to be on the order of 10^{-3}. Similar values were observed for correlation between s_x and s_y, as well as s_y and s_z. This allows us to treat the distributions of displacements in each direction as independent from one another.

To determine the analytical form of the probability density function, the kurtosis is first calculated according to Eq. 3.9:

$$\kappa = \frac{\langle u_j^4 \rangle}{\langle u_j^2 \rangle^2}. \tag{3.9}$$

At each density, the distributions in the x, y, and z directions are found to have a kurtosis of approximately 3, which is consistent with a one-dimensional Gaussian distribution. From $\langle u_j^2 \rangle$ we can calculate the corresponding α_j for the one-dimensional

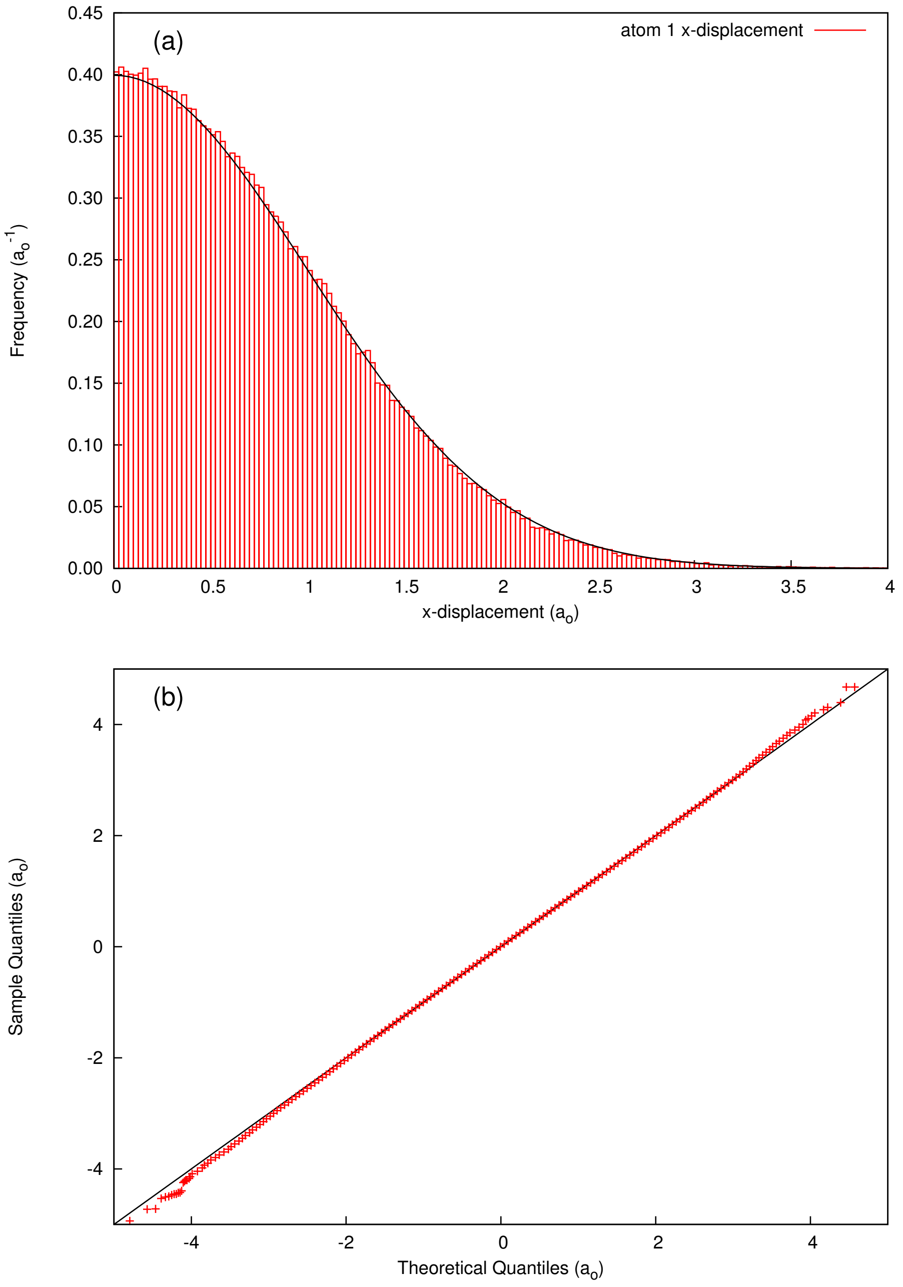


Figure 3.2: (a) Histogram of atom 1 x-displacements over 6.4x10^5 MCCs. For ease of visualization, only the positive half of the probability density is shown, however the negative half demonstrates the same behavior. (b) Quantile-quantile plot of atom 1 x-displacements. In both figures, the best-fit Gaussian, P_x, is shown in black for comparison. Atomic displacements are taken from VMC simulations using the optimized wavefunction at the Blackburn density.

Gaussian probability density P_j in Eq. 3.10:

$$P_j = \frac{\sqrt{\alpha_j}}{\sqrt{\pi}} e^{-2\alpha_j (s_j)^2}. \tag{3.10}$$

P_x determined from the optimized wavefunction at the Blackburn density is shown in black in Fig. 3.2a, along with a quantile-quantile plot of the x-displacements compared to P_x in Fig. 3.2b. The calculated best-fit Gaussian agrees very well with the observed frequencies, differing by less than 0.012 a_o^{-1}across the distribution. The quantile-quantile plot also shows that only slight deviations from Gaussian behavior occur in the wings of the distribution where these values make a relatively small contribution to the overall probability density. Similar agreement is found for the y- and z-distributions at the Blackburn density, as well as higher densities. The effects of the slight deviation from Gaussian behavior are investigated in Section 3.2.3.

The independence of the distributions in the x-, y-, and z-directions has already been demonstrated. Therefore, the three-dimensional probability density is just the product of P_x, P_y, and P_z (Eq. 3.11). Because of symmetry in the x,y-plane, α_x and α_y are the same and will be denoted α_{xy}.

$$P = \frac{\alpha_{xy}\sqrt{\alpha_z}}{(\sqrt{\pi})^3} e^{-2\alpha_{xy}(s_x^2+s_y^2)-2\alpha_z(s_z^2)}. \tag{3.11}$$

We note that the three-dimensional probability density looks similar to the square of the one-body part of the atomic wavefunction in Eq. 3.3. The α_j values are in fact related to the a_j variational parameters, however the α_j values also depend on the density and b variational parameter.

3.2.3 Long-Range Corrections

Atoms outside of the interacting pair region still make a small contribution to the overall energy of the crystal that must be accounted for. Corrections to the potential energy are made by considering all of those atoms outside of the interacting pair cutoff from atom 1, though, due to symmetry, selection of the center atom is arbitrary and does not affect the results.

The long-range correction calculations focus on two regions: those atoms in the 448-atom cell outside of the interacting-pair cutoff (region 1), and the infinite number of atoms beyond these 448 represented in the periodic boundary conditions applied to our model (region 2).

In the limit of long-range interactions, the potential energy function becomes

$$V_{lrc}(R) = -\frac{C_6}{R^6} - \frac{C_8}{R^8} - \frac{C_{10}}{R^{10}}. \tag{3.12}$$

In order to treat the atoms in region 1, we must evaluate the following integral for each of these atoms paired with atom 1:

$$\langle V_{lrc} \rangle = \int \int P_1 V_{lrc}(R) P_i d\vec{s_1} d\vec{s_i}, \tag{3.13}$$

where $\vec{s_1}$ and $\vec{s_i}$ are the instantaneous displacement vectors of atoms 1 and i from their average lattice positions, P_1 and P_i are the three-dimensional probability densities of atoms 1 and i defined in Eq. 3.11, and $R = |\vec{R}|$ is the interatomic distance corresponding to the instantaneous interatomic vector

$$\vec{R} = \vec{R}_{latt} + \vec{s_i} - \vec{s_1}, \tag{3.14}$$

where the component in the j-direction is $R_j = R_{latt,j} + s_{i,j} - s_{1,j}$. $\vec{R}_{latt}$ is the vector from the average lattice position of atom 1 to atom i and is constant. The interatomic distance is therefore a function of the six displacement vector components $s_{1,x}$, $s_{1,y}$, $s_{1,z}$, $s_{i,x}$, $s_{i,y}$, and $s_{i,z}$.

Because the atomic pairs considered are separated by a distance greater than the interacting pair cutoff, correlation in the atomic motions does not need to be considered. This along with the separable form of the probability densities allows the integral to be written as a product of six Gaussian terms and the potential energy function $V_{lrc}(R)$. By rewriting $s_{i,j}$ as $x_{i,j}/\sqrt{2\alpha_j}$ in Eq. 3.11 we can make use of Gaussian quadrature to evaluate the integral using the relationship shown in Eq. 3.15,

$$\int_{-\infty}^{\infty} \exp(-x^2) f(x) dx \approx \sum_{k=1}^{N} w_k f(x_k), \tag{3.15}$$

where the weights, w_k, and abscissas, x_k, are determined by the number of nodes used, N. For our calculations, convergence within the statistical uncertainty of the energy expectation value is reached using 8 nodes.

We investigated the effects of using the best-fit Gaussian approximation for our atomic distributions in these calculations by comparing the average pair energies calculated using Gaussian quadrature and VMC snapshots for six representative interatomic distances in region 1 ranging from $2.24R_{nn}$ to $4.87R_{nn}$. For each distance, 10 atomic pairs were selected and the pair potential energy was computed from the Gaussian approximation and compared with that obtained directly from VMC. The difference between these values was found to decrease as the interatomic distance increased, and at the largest distance the error is below 10^{-6} K/pair. Using this method, the maximum error estimated for V_{lrc} at the Blackburn density was only 0.005 K/atom, approximately 0.4% of the long-range correction energy.

As atomic separations increase, we expect that zero point motions change the interatomic distance negligibly, and that the C_6/R^6 term constitutes the dominant contribution to the potential energy. Both of these trends are demonstrated in Fig. 3.3 where the potential energy of a pair of atoms calculated using Gaussian quadrature is compared to the C_6/R^6 term calculated from average lattice positions (not considering zero point motions). As the distance between atoms increases, the difference between the two values becomes negligible (Fig. 3.3b). This allows for a simplified treatment of region 2 atoms in which only the C_6/R^6 contributions from each pair in the absence of zero point motions are added.

The sum of $1/R^6$ contributions for an infinite number of atoms in an hcp lattice has been previously reported[19] in terms of a lattice sum, S_6^*, times $1/R_{nn}^6$. The

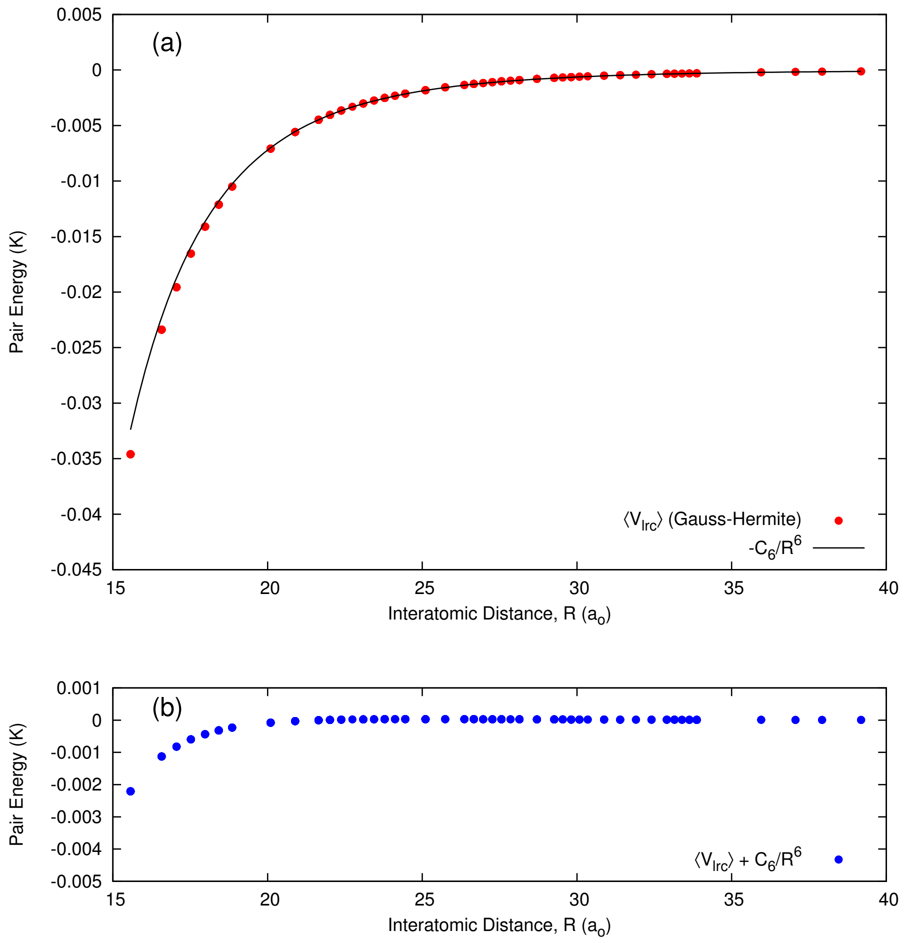


Figure 3.3: (a) Comparison of the Gauss-Hermite calculation of $\langle V_{lrc} \rangle$ including ZPM (red dots) to $-C_6/R^6$ calculated without ZPM (black line). The difference between these two calculations is shown in (b). Gaussian parameters used in the Gauss-Hermite integration are obtained from VMC simulations using the optimized wavefunction at the Blackburn density

value for S_6^* includes contributions from interacting pairs and region 1 atoms, which we have treated more carefully, and therefore these atoms' contribution to the rigid lattice $1/R^6$ sum is subtracted to get the total long-range correction from region 2.

However, the value of S_6^* has not yet been reported for non-ideal hcp lattices. For these distorted lattices, the equivalent value was found as follows, using a lattice containing 7920 atoms (20 atoms x 18 atoms x 22 atoms). We first identify the central atom in the lattice with an ideal c/a ratio and construct a series of spheres of increasing radius centered on that atom. For each sphere we calculate the truncated $1/R^6$ sum as a product of $1/R_{nn}^6$ times a constant S_6, which should converge to the exact value (S_6^*) as the sphere gets infinitely large. We then calculate the ratio between the truncated sum and the exact value for each of the finite spheres. The lattice is then distorted as described in Section 3.2.1 to achieve non-ideal c/a ratios. We distort the spheres in the same way so that we end up with ellipsoids containing the same lattice sites as in the case of the ideal c/a ratio. The truncated sum for each ellipsoid is calculated and the S_6/S_6^* ratio from the corresponding sphere in the ideal lattice is used to extrapolate the estimated S_6^* for the distorted lattice. We observe that this approach leads to estimated S_6^* values for the distorted lattices that agree to three decimal places, or about 1 part in 10^5. Results are shown in Table 3.1.

The final long-range correction to the potential energy is calculated from the sum of the Gaussian quadrature calculations from region 1 atoms and the rigid lattice R^{-6} contributions from region 2 atoms.

3.2.4 Reweighting

A reweighting method is used to determine the optimized wavefunction parameters to a greater precision without significantly impacting computing time. This method takes advantage of the fact that, although the energy depends on both the variational parameters and the atomic positions, for small changes in parameter values, the distribution of the snapshots changes only slightly. The effect of this small change can

Table 3.1: R_{nn} and S_6^* values determined for lattices with various c/a ratios. The value for $c/a = 100\%$ is taken from Ref. 19.

c/a (% of ideal value)	R_{nn} (a_o)	S_6^*
90	6.739	12.0259
93.33	6.812	12.7388
96.67	6.886	13.5470
100	6.963	14.45485
103.33	6.887	13.5553
106.67	6.814	12.7687
110	6.745	12.0834

be approximated by reweighting observables calculated from the original snapshots. Therefore, the energy of a wavefunction with a new set of parameter values can be calculated from snapshots taken from a different wavefunction according to the following equation:

$$\langle \Psi' | \hat{H} | \Psi' \rangle = \int |\Psi'(\mathbf{Q})|^2 E'(\mathbf{Q}) d\mathbf{Q} = \int |\Psi(\mathbf{Q})|^2 w(\mathbf{Q}) E'(\mathbf{Q}) d\mathbf{Q}, \tag{3.16}$$

where $w(\mathbf{Q}) = \left| \frac{\Psi'(\mathbf{Q})}{\Psi(\mathbf{Q})} \right|^2$, Ψ is the original wavefunction, Ψ' is the wavefunction with the new set of parameters, and E' is the local energy of the wavefunction with the new parameters.

This relationship shows that the local energy of a new wavefunction can be evaluated at a given configuration $\mathbf{Q}$ from the old wavefunction via the weighting factor w. Therefore the expectation value of the total energy essentially becomes a weighted average over all configurations sampled:

$$\langle E \rangle \approx \frac{\left(\sum\limits_{Q} E'(\mathbf{Q}) w(\mathbf{Q}) \right)}{\sum\limits_{Q} w(\mathbf{Q})}. \tag{3.17}$$

Statistical errors are present in both the numerator and denominator in Eq. 3.17, and correlations in the two errors due to the common $w(\mathbf{Q})$ terms must be taken into account when determining the uncertainty in $\langle E \rangle$. This is done according to the procedure reported in Ref. [20].

3.3 Results and Discussion

3.3.1 Ideal Lattice, Blackburn Density

Initial simulations focused on the optimization of the a_i parameters while the b parameter was held constant at $5.6029 a_o$. Using 8x10^4 MCCs, approximate optimized a_{xy} and a_z values were determined to the nearest $0.02 a_o^{-2}$ by scanning through values of the a_i parameters and calculating their VMC energies. At these near-optimal parameters, referred to below as centering points, 2.56x10^6 snapshots were generated to use in reweighting in order to obtain more precise optimized parameters.

Energies obtained from Eq. 3.17 were determined by scanning over a fine grid of (a_{xy}, a_z) values in the vicinity of the centering point; increments of $0.005a_o^{-2}$ were used in the scans. A representative set of results is shown in Fig. 3.4. Near the optimal parameter values, the contours can be approximated by ellipses:

$$E(a_{xy}, a_z) = C_1(a_{xy} - a_{xy}^o)^2 + 2C_2(a_{xy} - a_{xy}^o)(a_z - a_z^o) + C_3(a_z - a_z^o)^2 + E(a_{xy}^o, a_z^o), \quad (3.18)$$

where a_{xy}^o and a_z^o are the optimized parameters and $E(a_{xy}^o, a_z^o)$ is the minimum energy. At the minimum energy, the first derivatives of E must be zero, and therefore we can assume that for a_{xy}, a_z near the optimal parameters, $E(a_{xy}, a_z)$ behaves quadratically. This simplifies calculations of the first and second partial derivatives of E with respect to the parameters a_{xy} and a_z, from which we can easily calculate the six constants $C_1, C_2, C_3, a_{xy}^o, a_z^o$, and $E(a_{xy}^o, a_z^o)$. Overall, this analysis requires only nine reweighted energies to complete. We use this method to calculate estimates of the six constants. These estimates are then used to initiate a Levenberg-Marquardt fit[21] (as implemented in gnuplot version 4.2[22]) in order to obtain more accurate values which also take into account the uncertainties in the energies.

Fig. 3.5 shows a series of slices through contour plots of the form in Eq. 3.18 in which the a_z parameter is held constant while a_{xy} is altered. The slices were generated using three different sets of centering points. These figures show the 95% confidence interval of the energies obtained from reweighting, as well as VMC energies where available. In each of the slices in Fig. 3.5, the values determined from reweighting and VMC agree within their mutual 95% confidence intervals. However, while the results agree with one another, we see that as the difference between the reweighting parameters and the centering point values increases, so does the uncertainty in the

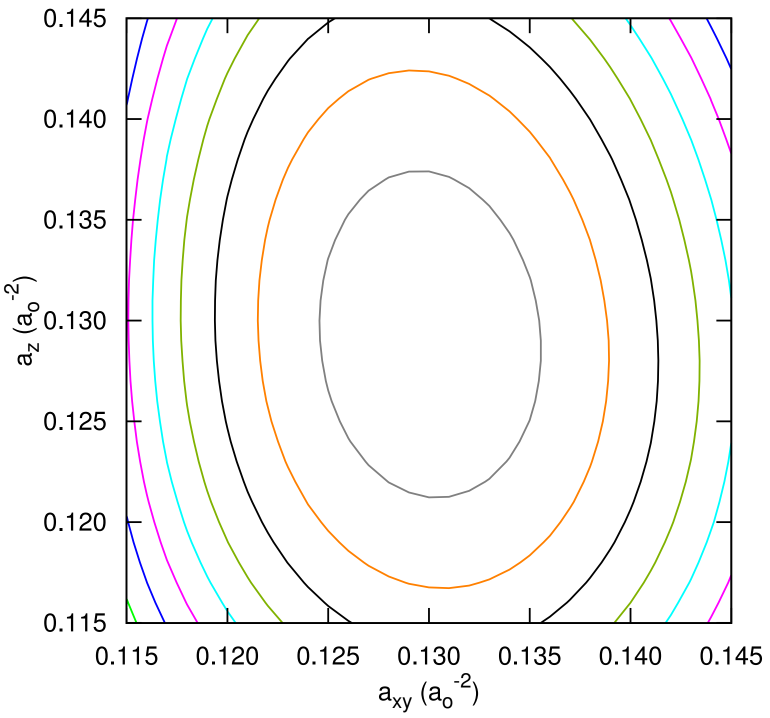


Figure 3.4: Contour plot of reweighted energies at a density of $0.0041896a_o^{-3}$ from the centering point $a_{xy} = a_z = 0.13a_o^{-2}$. The innermost contour corresponds to an energy of -3.76 K/atom, and each successive contour is 0.01 K/atom higher in energy.

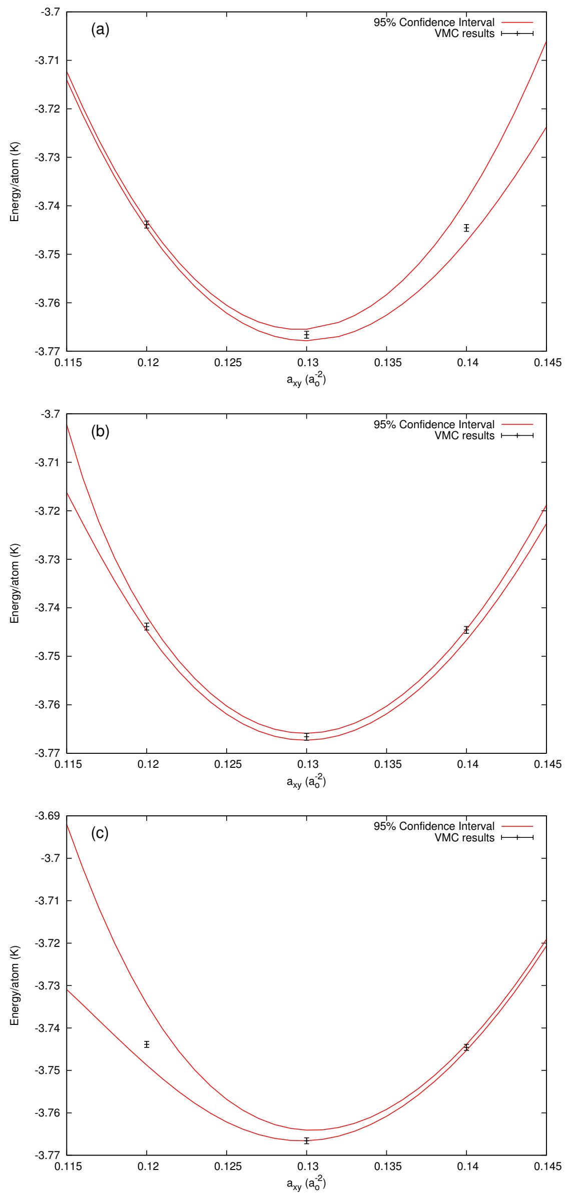


Figure 3.5: Reweighting results for various centering points at the Blackburn density. (a) $a_{xy} = 0.12a_o^{-2}, a_z = 0.13a_o^{-2}$; (b) $a_{xy} = 0.13a_o^{-2}, a_z = 0.13a_o^{-2}$; (c) $a_{xy} = 0.14a_o^{-2}, a_z = 0.13a_o^{-2}$.

energy. This indicates that reweighting from points far from the optimal parameter values can bring us closer to the true values, however the uncertainty will be much greater. Therefore performing two rounds of reweighting, in which the snapshots for the second round are generated from the improved parameters determined in the first round, allows for more accurate and precise determination of the optimal parameter values. For the second round of reweighting, we decreased the step size to $0.001a_o^{-2}$ in order to improve the precision of our reweighting calculations and optimized parameters. The initial scanning of the a_i parameters along with the two rounds of reweighting constitute a single optimization step.

The second optimization step focused on changing the b parameter while the a_i parameters were held fixed at the values determined in the first step. A similar scanning procedure was used to determine the approximate optimized b parameter to the nearest $0.02a_o$, after which 2.56x10^6 VMC snapshots were generated from this wavefunction. Using the near-optimal b parameter as the centering point, the first round of reweighting calculations were performed using a step size of $0.002a_o$. Energies from 10 reweighting calculations were fit to a parabola in order to estimate the minimum energy b value. Snapshots generated using this b value were then used for the second round of reweighting. For this round, the step size was decreased to $0.001a_o$. This whole process constitutes the second optimization step.

Following the second optimization step, a third step was performed in which the a_i parameters were reoptimized in exactly the same manner as in the first optimization step. 2.56x10^6 snapshots were generated from the optimized wavefunction after the three optimization steps. In order to determine if the wavefunction was fully optimized, an additional b-parameter reweighting calculation was performed using these snapshots and the optimized parameters as the centering point. The estimated optimal b parameter from this calculation was about $0.05a_o$ lower than the centering point b, corresponding to a 0.6% change in the minimum energy. This prompted reoptimization of the b and a_i parameters by repeating optimization steps two and three (see Appendix, Fig. 3.12). The optimized wavefunction parameters are given in Table 3.2.

Fig. 3.6 shows a contour plot of the energy as a function of the b and a parameters (where $a = a_{xy} = a_z$) determined by reweighting from snapshots from the new optimized wavefunction. We see in this figure that the optimal parameters determined from our five optimization steps are just outside of the minimum energy contour, differing in energy from the predicted minimum by less than 0.15%, or about 0.005 K/atom. As this is the greatest improvement that could be expected from additional steps, and because VMC is an approximate method that carries inherent error due to the variational principle, no further optimization was performed. A fit of the data in Fig. 3.6 to an equation analogous to Eq. 3.18 estimates the uncertainty in the a_i and b parameters at the end of the optimization procedure to be approximately $\pm\ 0.004a_o^{-2}$ and $\pm\ 0.034a_o$, respectively. Over the range of parameters included in Fig. 3.6, the LRC energy (determined via reweighting) changed by less than 0.005 K/atom, confirming that omission of this correction during reweighting does not shift the minimum energy considerably. We also note that the orientation of the contours (nearly aligned with the axes) indicates that there is very little correlation in the a and b variational parameters at this density. This simplifies the optimization procedure by allowing a relatively small number of optimization steps to achieve the optimal parameters.

2.56x10^6 snapshots were generated from the fully optimized wavefunction and used to calculate the mean squared displacements for in-plane and out-of-plane motions as described above. The results are shown in Table 3.2. Only a slight difference of 0.14% between $\langle u_{xy}^2 \rangle$ and $\langle u_z^2 \rangle$ is observed at this density, in contrast to the 20% difference reported by Blackburn, *et al.* Additionally, the difference between the a_{xy} and a_z parameters is very small, as is expected in the absence of anisotropy. Therefore with our current model, anisotropy is not detected at the Blackburn density in an ideal hcp crystal.

Table 3.2: Summary of optimized wavefunctions for various densities and c/a ratios. Statistical uncertainties in the average energy per atom and mean squared displacement are less than $\pm$ 5x10^{-4} K/atom and $\pm$ 3x10$^{-5}\mathring{A}^2$, respectively, for densities below 8.0x10$^{-3}a_o^{-3}$, and less than $\pm$ 1x10^{-3} K/atom and $\pm$ 6x10$^{-6}\mathring{A}^2$ for all higher densities.

Density ($10^{-3}a_o^{-3}$)	c/a (% of ideal value)	a_{xy} (a_o^{-2})	a_z (a_o^{-2})	b (a_o)	E (K/atom)	V_{lrc} (K/atom)	E_{tot} (K/atom)	$\langle u_{xy}^2 \rangle$ ($\mathring{A}^2$)	$\langle u_z^2 \rangle$ ($\mathring{A}^2$)
	90	0.1374	0.1649	5.435	-3.404	-1.393	-4.797	0.3052	0.2461
4.1896	100	0.1481	0.1476	5.422	-4.026	-1.229	-5.255	0.2793	0.2789
	110	0.1601	0.1309	5.425	-3.107	-1.656	-4.763	0.2572	0.3321
4.2943	100	0.1555	0.1549	5.418	-3.811	-1.291	-5.102	0.2650	0.2647
4.3991	100	0.1630	0.1624	5.414	-3.559	-1.354	-4.913	0.2519	0.2516
4.5038	100	0.1706	0.1701	5.410	-3.266	-1.418	-4.684	0.2399	0.2395
4.6086	100	0.1785	0.1779	5.405	-2.932	-1.485	-4.417	0.2286	0.2283
4.7133	100	0.1864	0.1857	5.402	-2.554	-1.552	-4.106	0.2182	0.2180
4.8180	100	0.1945	0.1937	5.399	-2.128	-1.622	-3.750	0.2085	0.2083
4.9228	100	0.2026	0.2021	5.395	-1.656	-1.692	-3.348	0.1996	0.1992
5.0275	100	0.2114	0.2109	5.392	-1.132	-1.765	-2.897	0.1909	0.1905
5.5000	100	0.2595	0.2589	5.334	1.895	-2.109	-0.214	0.1571	0.1568
6.0000	100	0.3090	0.3082	5.310	6.543	-2.508	4.035	0.1314	0.1312
6.5000	100	0.3620	0.3617	5.289	12.922	-2.942	9.980	0.1118	0.1115
7.0000	100	0.4186	0.4180	5.269	21.317	-3.411	17.906	0.09636	0.09613
7.5000	100	0.4791	0.4786	5.250	32.014	-3.914	28.100	0.08398	0.08376
8.0985	100	0.5556	0.5548	5.226	48.250	-4.563	43.687	0.07223	0.07206
9.2112	100	0.7046	0.7040	5.192	90.006	-5.901	84.105	0.05645	0.05630
10.467	100	0.8824	0.8819	5.157	158.507	-7.621	150.886	0.04452	0.04438
11.320	100	1.010	1.009	5.133	219.937	-8.913	211.024	0.03860	0.03850

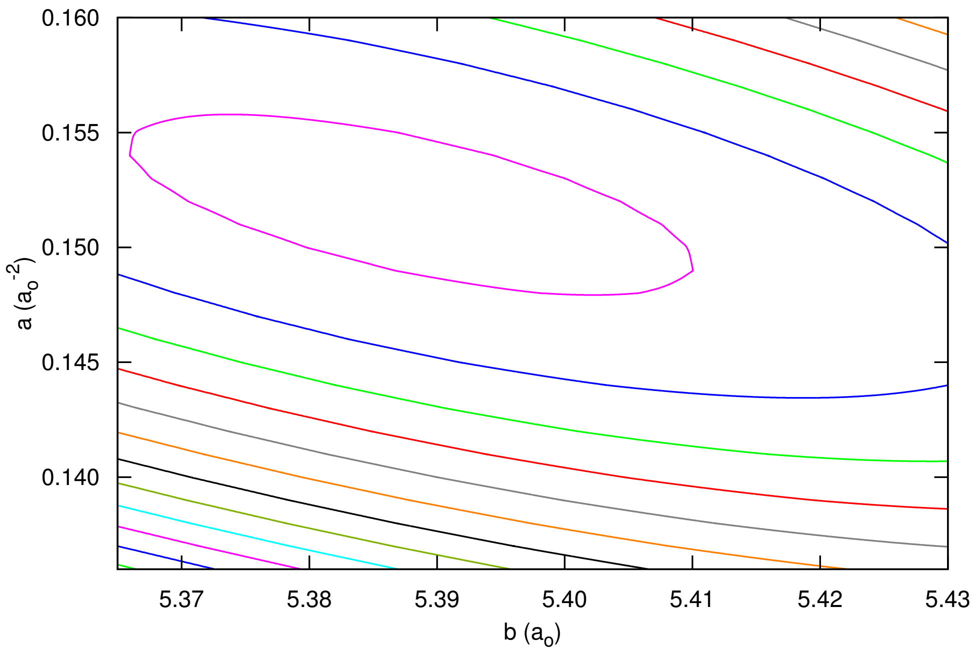


Figure 3.6: Contour plot of energy vs. b and $a = a_{xy} = a_z$ parameters where energies are determined through reweighting calculations at the Blackburn density using the optimized wavefunction parameters $b = 5.422a_o$ and $a = 0.148a_o^{-2}$ as the centering point. The innermost contour corresponds to an energy of -4.03 K/atom, and each successive contour is 0.01 K/atom higher in energy.

3.3.2 Distorted Lattices, Blackburn Density

In order to determine the effects of forced anisotropy on our model's predictions, the same optimization procedure was used for a series of distorted lattices. Initially crystals with six different c/a ratios equal to 90%, 93.33%, 96.67%, 103.33%, 106.67%, and 110% of the ideal c/a ratio were considered. Again, the b parameter was held constant at $5.6029a_o$ while the a parameters were optimized. After the optimal a parameters were obtained, the $\langle u^2 \rangle$ values were calculated; they are shown in Fig. 3.7. We see that at the ideal c/a ratio, there is no significant difference between $\langle u_{xy}^2 \rangle$ and $\langle u_z^2 \rangle$; however, even the smallest of the lattice distortions leads to a considerable difference between the two mean squared displacements, indicating that our model is able to detect anisotropy when we explicitly force it on the system.

Once we determined that our model was responding to distortions of the lattice, we continued to fully optimize the wavefunctions of those lattices with c/a ratios of 90% and 110% of the ideal value. As with the ideal lattice, this required five optimization steps overall (see Appendix, Fig. 3.12). The fully optimized wavefunction parameters and the in- and out-of-plane $\langle u^2 \rangle$ values are included in Table 3.2. We note that in our current model, a 10% change in the c/a ratio results in a difference between $\langle u_{xy}^2 \rangle$ and $\langle u_z^2 \rangle$ comparable to that reported by Blackburn *et al.*

3.3.3 Ideal Lattice, Higher Densities

The system was also studied at higher densities to determine what effect density has on the wavefunction parameters, as well as to observe whether the degree of anisotropy in the zero point motions is density-dependent. For the low density region, the original Blackburn density was increased by up to 20% in increments of 2.5% and the wavefunction was fully optimized as before. Wavefunctions were also optimized in the middle ($0.0055a_o^{-3}$ to $0.0075a_o^{-3}$) and high ($0.0080985a_o^{-3}$ to $0.0113198a_o^{-3}$) density regions following a similar procedure. The full range of densities studied corresponds to experimental pressures of approximately 25.2 to 7885 bar.

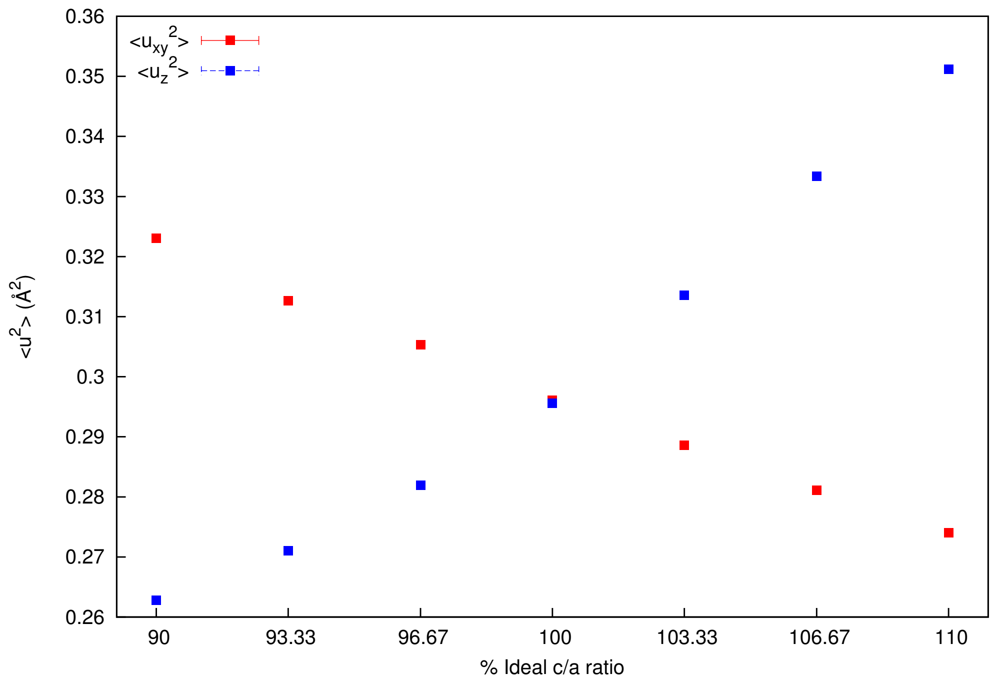


Figure 3.7: Mean squared displacements in the x,y-plane (red) and along the z-axis (blue) for lattices with various c/a ratios after first a-parameter optimization at the Blackburn density. The b-parameters have all been held constant at $5.6029a_o$. Error bars are included but are smaller than the data points.

In order to bring the initial parameters closer to the true optimal values, initial scanning of the a_i parameters at the new densities was performed with the b parameter fixed at the optimal value from the second optimization step at the Blackburn density ($5.4709a_o$), rather than at the initial value of $5.6029a_o$. At the lower densities, this b parameter was close enough to the optimal value that only three optimization steps were necessary. In the high density region another round of b and a_i optimization were required. Additionally, in the high density region scanning could only determine the optimal a_i parameters to the nearest $0.05a_o^{-2}$ due to greater uncertainties in the VMC energies, and therefore step sizes in the first and second rounds of reweighting of the a_i parameters were increased to $0.01a_o^{-2}$ and $0.005a_o^{-2}$, respectively. Step sizes for the b parameter reweighting were kept at $0.002a_o$ for both rounds of reweighting.

The middle density region was the last region investigated, and we determined that the VMC scanning at the beginning of each optimization step could be replaced with an additional round of reweighting (see Appendix, Fig. 3.13), reducing the computational cost. This initial round of reweighting used step sizes of $0.01a_o^{-2}$ and $0.005a_o$ for a_i and b parameter reweighting, respectively. With this method only three optimization steps were necessary in this density region. Optimization results are included in Table 3.2. We observe that at all densities studied, the $\langle u_{xy}^2 \rangle$ and $\langle u_z^2 \rangle$ differ by less than 0.32%, indicating no significant anisotropy in the atoms' zero point motions.

The need for additional rounds of optimization in the high density region is partially due to the fact that the initial scanning parameters were farther away from the true optimal values. However another reason for this can be deduced from Fig. 3.8, which shows the estimated minimum energy at the highest density as a function of the $a = a_{xy} = a_z$ and b parameters. From this figure, it is apparent that the contours are more angled with respect to the axes than was observed at the Blackburn density (Fig. 3.6). Therefore, the degree of correlation between a_i and b is slightly greater in this high density range, resulting in more optimization steps. We also note that the energy at our optimal parameters shown in Table 3.2 is again not located

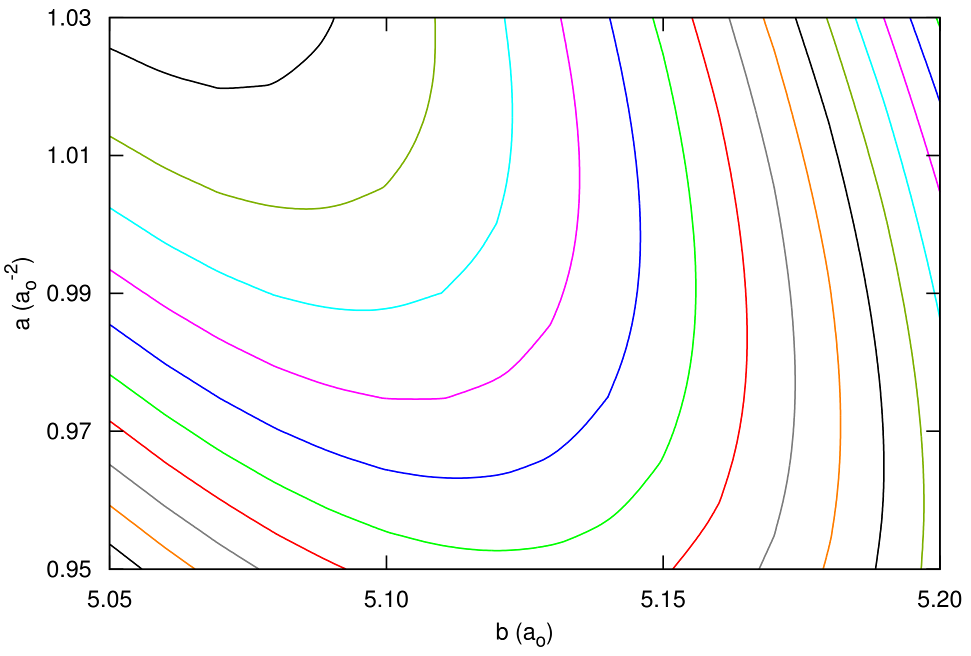


Figure 3.8: Contour plot of energy vs. b and $a = a_{xy} = a_z$ parameters at density $= 0.0113198 a_o^{-3}$. Energies are determined through reweighting calculations using $b = 5.132 a_o$ and $a = 1.01 a_o^{-2}$ as the centering point. The innermost contour corresponds to an energy of 219.8 K/atom, and each successive contour is 0.05 K/atom higher in energy.

within the minimum energy contour, but the predicted change in energy with an additional reweighting step is less than 0.1%, and therefore further optimization was not performed. Analysis of the data in Fig. 3.8 estimates the uncertainties in a_i and b at this highest density to be approximately $\pm\ 0.08a_o^{-2}$ and $\pm\ 0.10a_o$, respectively.

Fig. 3.9 shows the dependence of the variational parameters on density. There is a slight jump in the b parameter between the low and middle density regions which could be smoothed out if more optimization steps were used, however this changes the calculated observables only slightly. The difference between the b values on either side of the jump in Fig. 3.9b (corresponding to the densities 5.0275x10^{-3} and 5.5x$10^{-3}a_o^{-3}$) is less than $0.06a_o$. If we consider the contour plot in Fig. 3.6, a similar change in the b value corresponds to only a 0.01K/atom change in the energy. The difference in the a parameter between the same two densities considered above is about $0.05a_o^{-2}$, which is twice as large as the range of a values studied in Fig. 3.6. This difference, therefore, corresponds to a much more significant change in energy. This indicates that there is a wider range of acceptable b values that would give approximately the same energy, resulting in greater uncertainty in the b parameter, whereas the a_i parameters can be determined more precisely. This is in accord with the uncertainties obtained by analysis of the data in Figs. 3.6 and 3.8.

3.3.4 Equation of State

The energy dependence on molar volume is shown in Fig. 3.10. Where available, experimental energies per atom [23] have been provided. We see good agreement with experiment in the shape of our energy–volume curve, though there is a constant difference of about 0.77 K/atom between our calculated energies and the experimental values in the low density range.

In order to more accurately compare our results to experimental data, we used the computed energy–volume data to derive a pressure–volume equation of state (EOS) following the procedure outlined in Ref. [24]. This was done by first fitting a

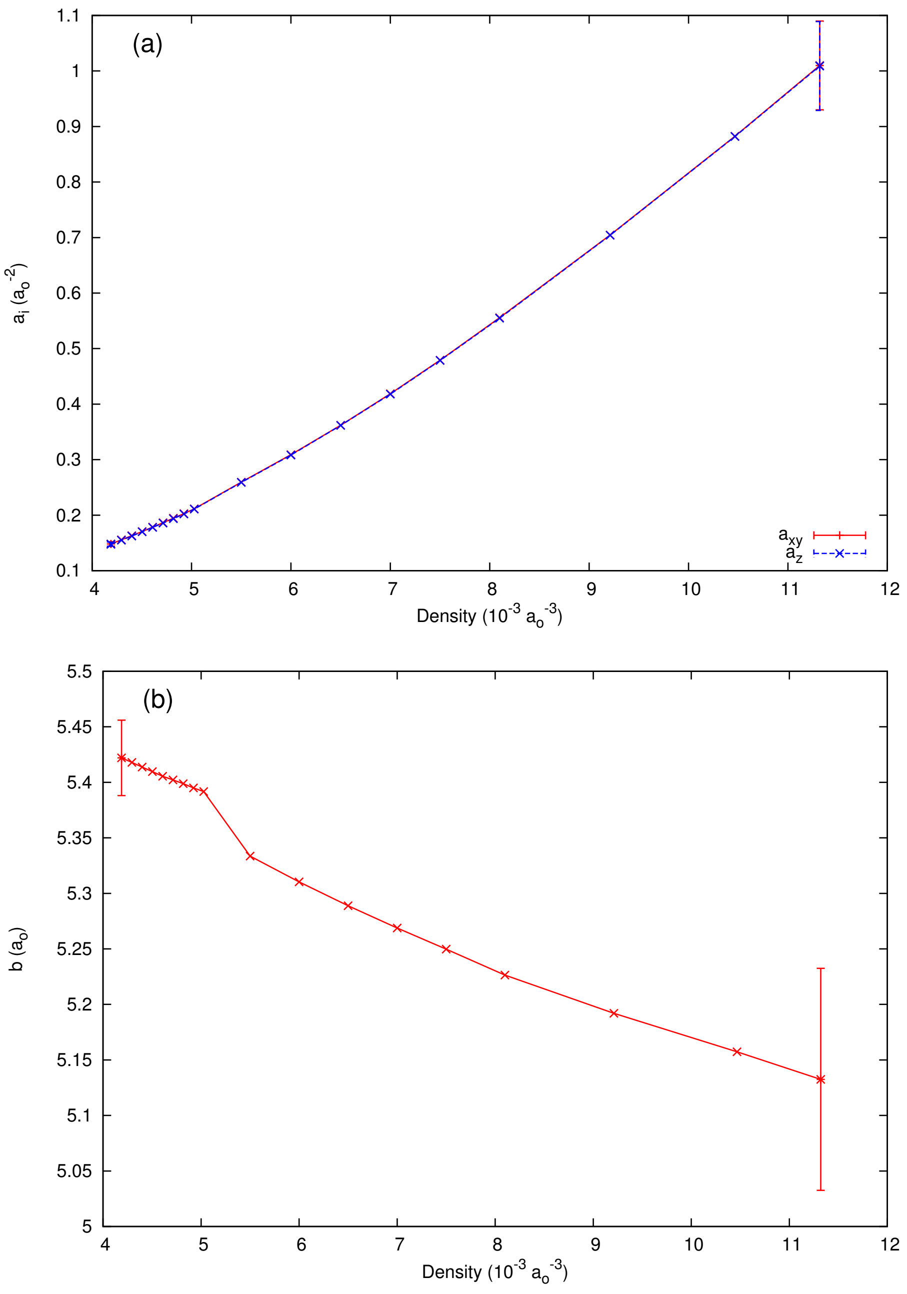


Figure 3.9: Variational parameter dependence on density: (a) a_{xy}, a_z vs. Density, (b) b vs. Density. Estimated error bars are shown for the highest and lowest densities.

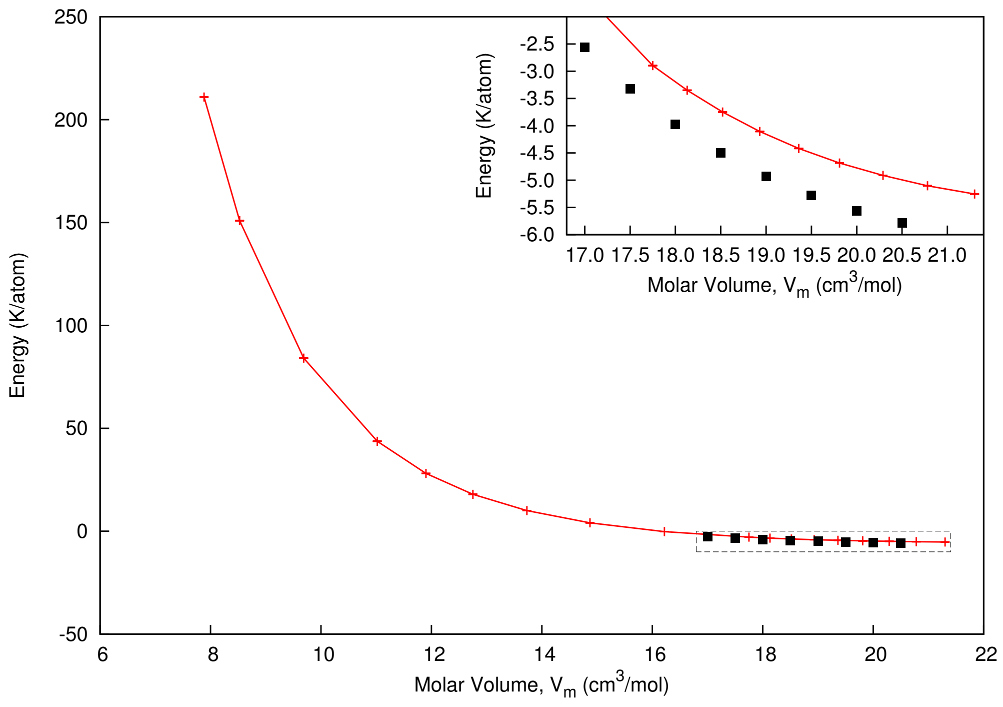


Figure 3.10: Energy vs. molar volume (V_m) for an ideal lattice (red). Error bars are smaller than the data points. Experimental data from [23] (black squares) is provided where available.

fourth order polynomial of $(\frac{1}{V_m})$ to the energy–volume data to obtain $E(V_m)$. This polynomial fit the data with less than 0.6% error at all points where $|E(V_m)| \geq 0.5$ K/atom. The EOS ($P(V_m)$) was then found by taking the negative derivative of $E(V_m)$ with respect to V_m (Eq. 3.19). The parameters obtained from this fitting procedure are given in Table 3.3. Fig. 3.11 compares the derived EOS to experimental data reported by Driessen *et al.*[18].

$$P(V_m) = \frac{1}{V_m^2}\left(\frac{4a}{V_m^3} + \frac{3b}{V_m^2} + \frac{2c}{V_m^1} + d\right) \tag{3.19}$$

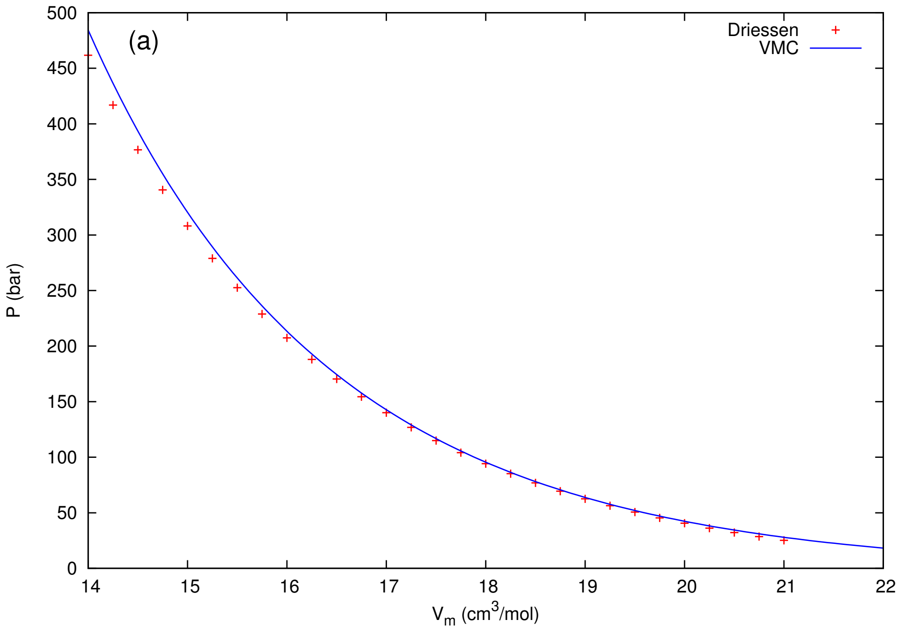
Overall the EOS agrees qualitatively with the experimental data, however as the molar volume decreases, we see that the theoretical EOS begins to diverge from experiment. At the lowest molar volume considered, our predicted pressure differs from the experimental pressure by approximately 1.1 kbar or about 12%. This suggests that improvements to our current model are necessary in order to better predict experimental observables, particularly at higher densities.

3.4 Summary and Conclusions

From the various simulations performed in this study, we have shown that our model behaves as expected when the density is changed. Increasing density was shown to cause an increase in the a_i parameters and a decrease in the b parameter. Both of these trends are expected when atoms are forced into closer contact with one another, reducing the amount of available space in which an atom can move without exchange occurring. An increase in the a_i parameters leads to greater localization of the atoms, and a decrease in b causes the two-body term to take effect at smaller distances. Additionally, we have shown that correlation between the variational parameters is small but increases with increasing density, and that the atomic wavefunction utilized in these simulations is more sensitive to changes in the b parameter than the a_i

Table 3.3: Fitting parameters for the P(V_m) equation of state shown in Eq. 3.19.

Parameter	Value ($\times 10^6$)
a	-17.339 ± 6.609 bar(cm^3/mol)^5
b	27.639 ± 2.180 bar(cm^3/mol)^4
c	-2.827 ± 0.261 bar(cm^3/mol)^3
d	0.1010 ± 0.0134bar(cm^3/mol)^2



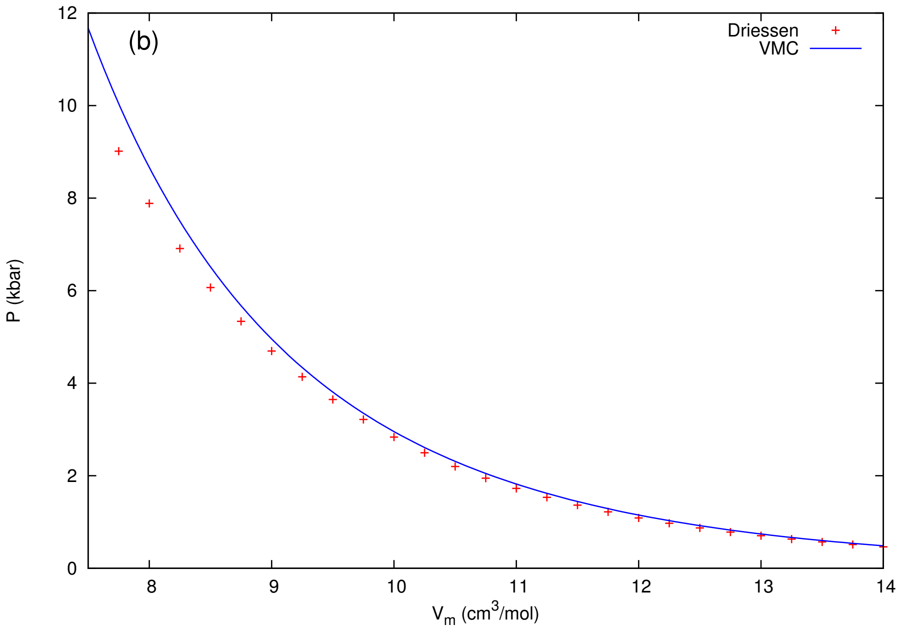


Figure 3.11: Pressure-Volume equation of state derived from VMC energies (blue) in the (a) low density and (b) high density regions. Experimental data from Driessen *et al.* [18] is shown in red for comparison.

parameters, i.e. a 1% change in the b parameter has a greater impact on the calculated energy than a 1% change in the a_i parameters.

A comparison of the $\langle u^2 \rangle$ values calculated in this study to previously reported values is given in Table 3.4. We see that our values are in fairly good agreement with earlier PIMC [8] and GFMC [10] results, however we report values approximately twice as large as those found in the Blackburn study [12]. Anisotropy was also not detected in the ideal hcp ^{4}He crystal at the Blackburn density of $0.0041896 a_o^{-3}$, nor at the higher densities studied. The $\langle u^2 \rangle$ values for in-plane and out-of-plane zero point motions in ideal crystals given in Table 3.2 differ by less than 0.32%, which does not agree with the approximate 20% difference reported by Blackburn *et al.* However, using our model it was possible to induce anisotropy of that magnitude in the zero point motions by uniaxially compressing or expanding the crystal in order to change the c/a ratio by $\pm$ 10%. Further investigation is needed in order to understand the discrepancies between our results and the experimental findings of Blackburn *et al*.

Where available, we have compared our calculated energies with experimental results [23] in Fig. 3.10. In this region, our values agree qualitatively with experiment, though the calculated energies are consistently about 0.77 K/atom higher than the experimental values. We were also able to derive an EOS relating the molar volume of our simulation cells to experimental pressures. Although qualitatively similar, comparison to experimental data from Driessen *et al.* shows better agreement in the low density range than at higher densities. We acknowledge two sources of error which may contribute to these differences, namely that VMC energies are bound by the variational principle and will therefore always be higher in energy than the exact ground state, and also that three-body interactions have not been taken into account. An exact ground state method such as variational path integral Monte Carlo (VPI) is necessary in order to remove error associated with the variational principle. Using the optimized wavefunctions from this VMC study as the trial wavefunctions, the uncertainty and projection time for these VPI simulations could be significantly

Table 3.4: Comparison of previously reported $\langle u^2 \rangle$ values to those found in this study. An asterisk represents data obtained via interpolation of directly calculated results.

Ref	Method	V_m (cm^3/mol)	T (K)	c/a	$\langle u_{xy}^2 \rangle$ ($\mathring{A}^2$)	$\langle u_z^2 \rangle$ ($\mathring{A}^2$)
[12]	Neutron scattering	21.3	<1	1.638(5)	0.122(1)	0.150(1)
This work	VMC	21.3	0	1.633	0.2793(1)	0.2789(1)
This work	VMC	21.3	0	1.470	0.3052(1)	0.2461(1)
This work	VMC	21.3	0	1.796	0.2572(1)	0.3321(1)
[10]	GFMC	19.12	0	1.633	0.261(9)	
This work	VMC	19.36	0	1.633	0.2286(1)	0.2283(1)
This work*	VMC	19.12	0	1.633	0.2228	0.2225
[8]	PIMC	10.98	5	1.633	0.0778(3)	
This work	VMC	11.02	0	1.633	0.07223(1)	0.07206(1)
This work*	VMC	10.98	0	1.633	0.07173	0.07157
[8]	PIMC	12.12	5	1.633	0.0952(5)	
This work	VMC	12.75	0	1.633	0.09636(1)	0.09613(1)
This work*	VMC	12.12	0	1.633	0.08709	0.08687

reduced, making elimination of variational error fairly simple. Additionally, we expect that, particularly at the higher densities, three-body interactions make a significant contribution to the overall energy of solid ^{4}He and must be incorporated in some way by considering one of the available three-body potential energy surfaces for ^{4}He [25, 26]. Previous theoretical results which have reported no evidence of anisotropy have relied on a pairwise additive potential energy function, and therefore it is not certain what impact the incorporation of non-additive three-body interactions might have on the DW factors. Future efforts will focus on addressing both of these sources of error in order to develop a more reliable model for the zero point motions in hcp ^{4}He, in addition to calculating the theoretical elastic constants for solid ^{4}He in order to shed light on the importance of three-body interactions in the system. Some of the solid's elastic constants are associated with anisotropic distortions of the crystal lattice and therefore reliable trial wavefunctions for these distorted lattices will be needed to carry out the VPI energy calculations from which elastic constants can be derived. The reweighting method presented above provides a computationally efficient means of optimizing these wavefunctions while allowing a_{xy} and a_z to be different.

3.5 References

[1] D. M. Ceperley, *Rev. Mod. Phys.* **67**, 279 (1995). 40

[2] E. Polturak and N. Gov, *Contemporary Physics* **44**, 145 (2003). 40

[3] J. Eckert, W. Thomlinson, and G. Shirane, *Phys. Rev. B* **18**, 3074 (1978). 41

[4] C. Stassis, D. Khatamlan, and G. Kline, *Solid State Communications* **25**, 531 (1978). 41

[5] D. A. Arms, R. S. Shah, and R. O. Simmons, *Phys. Rev. B* **67**, 094303 (2003). 41

[6] C. T. Venkataraman and R. O. Simmons, *Phys. Rev. B* **68**, 224303 (2003). 41

[7] C. A. Burns and E. D. Isaacs, *Phys. Rev. B* **55**, 5767 (1997). 41

[8] E. W. Draeger and D. M. Ceperley, *Phys. Rev. B* **61**, 12094 (2000). 41, 73, 74

[9] P. A. Whitlock, D. M. Ceperley, G. V. Chester, and M. H. Kalos, *Phys. Rev. B* **19**, 5598 (1979). 41

[10] P. A. Whitlock, M. H. Kalos, G. V. Chester, and D. M. Ceperley, *Phys. Rev. B* **21**, 999 (1980). 41, 73, 74

[11] S. T. Chui, *Phys. Rev. B* **41**, 796 (1990). 41

[12] E. Blackburn, J. M. Goodkind, S. K. Sinha, J. Hudis, C. Broholm, J. van Duijn, C. D. Frost, O. Kirichek, and R. B. E. Down, *Phys. Rev. B* **76**, 024523 (2007). 41, 73, 74, 80

[13] J.-P. Hansen and D. Levesque, *Physical Review* **165**, 293 (1968). 42

[14] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *The Journal of Chemical Physics* **21**, 1087 (1953). 43

[15] P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton, *Operations Research* **50**, pp. 1073 (2002). 43

[16] R. A. Aziz, F. R. McCourt, and C. C. Wong, *Molecular Physics* **61**, 1487 (1987). 43, 80

[17] J.-P. Hansen, *Phys. Rev.* **172**, 919 (1968). 44

[18] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 44, 70, 72

[19] J. Hirschfelder, C. Curtiss, and R. Bird, *Molecular theory of gases and liquids*, Structure of matter series, Wiley, 1954. 51

[20] A. M. Ferrenberg, D. P. Landau, and R. H. Swendsen, *Physical Review E* **51**, 5092 (1995). 55

[21] J. J. Moré, The Levenberg-Marquardt algorithm: Implementation and theory, in *Numerical Analysis*, edited by G. Watson, volume 630 of *Lecture Notes in Mathematics*, pp. 105–116, Springer Berlin Heidelberg, 1978. 56

[22] T. Williams, C. Kelley, and many others, Gnuplot 4.2: an interactive plotting program, http://gnuplot.sourceforge.net/, 2009. 56

[23] D. O. Edwards and R. C. Pandorf, *Phys. Rev.* **140**, A816 (1965). 68, 69, 73

[24] C. Cazorla and J. Boronat, *Journal of Physics: Condensed Matter* **20**, 015223 (2008). 68

[25] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 75, 80

[26] M. J. Cohen and J. N. Murrell, *Chemical Physics Letters* **260**, 371 (1996). 75

3.6 Appendix

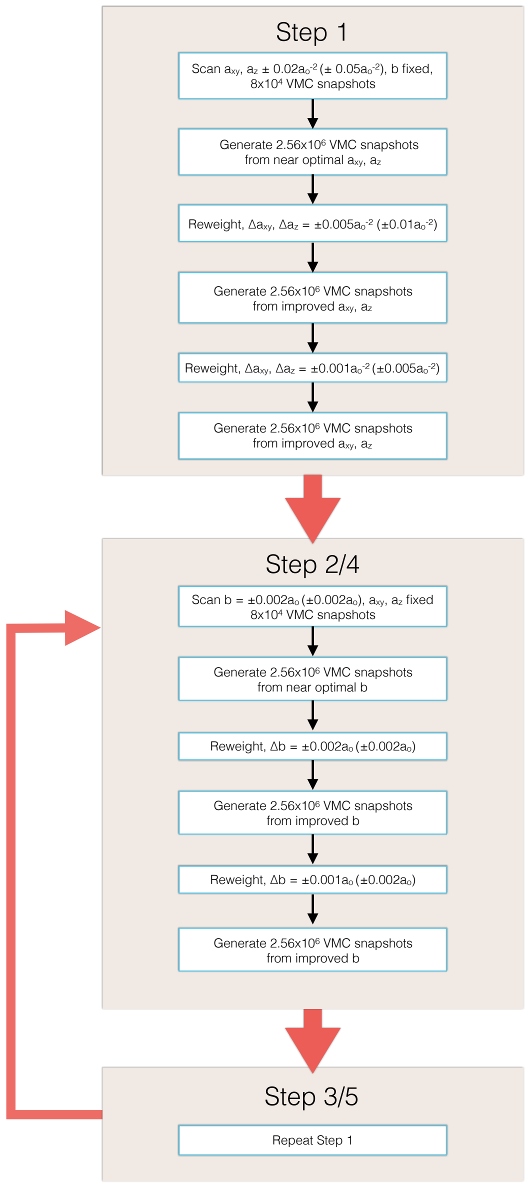

Step 1
Scan a_{xy}, $a_z \pm 0.02a_o^{-2}$ ($\pm 0.05a_o^{-2}$), b fixed, $8x10^4$ VMC snapshots
Generate $2.56x10^6$ VMC snapshots from near optimal a_{xy}, a_z
Reweight, Δa_{xy}, $\Delta a_z = \pm 0.005a_o^{-2}$ ($\pm 0.01a_o^{-2}$)
Generate $2.56x10^6$ VMC snapshots from improved a_{xy}, a_z
Reweight, Δa_{xy}, $\Delta a_z = \pm 0.001a_o^{-2}$ ($\pm 0.005a_o^{-2}$)
Generate $2.56x10^6$ VMC snapshots from improved a_{xy}, a_z
Step 2/4
Scan $b = \pm 0.002a_o$ ($\pm 0.002a_o$), a_{xy}, a_z fixed $8x10^4$ VMC snapshots
Generate $2.56x10^6$ VMC snapshots from near optimal b
Reweight, $\Delta b = \pm 0.002a_o$ ($\pm 0.002a_o$)
Generate $2.56x10^6$ VMC snapshots from improved b
Reweight, $\Delta b = \pm 0.001a_o$ ($\pm 0.002a_o$)
Generate $2.56x10^6$ VMC snapshots from improved b
Step 3/5
Repeat Step 1

Figure 3.12: Optimization scheme for low density (high density) region.

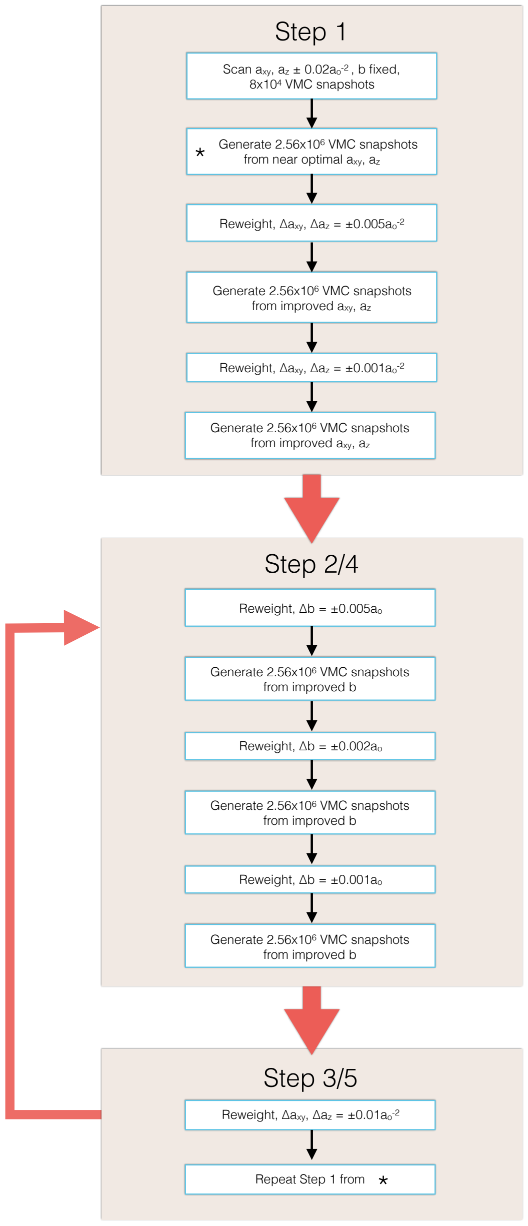

Step 1
Scan a_{xy}, a_z ± $0.02a_o^{-2}$, b fixed, $8x10^4$ VMC snapshots
* Generate $2.56x10^6$ VMC snapshots from near optimal a_{xy}, a_z
Reweight, Δa_{xy}, Δa_z = $\pm 0.005a_o^{-2}$
Generate $2.56x10^6$ VMC snapshots from improved a_{xy}, a_z
Reweight, Δa_{xy}, Δa_z = $\pm 0.001a_o^{-2}$
Generate $2.56x10^6$ VMC snapshots from improved a_{xy}, a_z
Step 2/4
Reweight, Δb = $\pm 0.005a_o$
Generate $2.56x10^6$ VMC snapshots from improved b
Reweight, Δb = $\pm 0.002a_o$
Generate $2.56x10^6$ VMC snapshots from improved b
Reweight, Δb = $\pm 0.001a_o$
Generate $2.56x10^6$ VMC snapshots from improved b
Step 3/5
Reweight, Δa_{xy}, Δa_z = $\pm 0.01a_o^{-2}$
Repeat Step 1 from *

Figure 3.13: Optimization scheme for middle density region.

3.6.1 Additional Calculation of the DW Factors Using a Three-Body Potential

In order to determine if three-body interactions are responsible for the anisotropy in the low-temperature DW factors reported by Blackburn, *et al.*[12], trial wavefunctions have been reoptimized for an ideal hcp lattice at each of the densities reported above using the same VMC optimization procedure as before with a three-body potential in place of the pairwise additive potential (Eq. 3.20). These calculations are referred to as VMC+3B simulations. The three-body potential used in these optimizations is the sum of the Aziz HFD-B(He) pair potential[16] and the Cencek nonadditive three-body potential[25],

$$V = \sum_{(i,j)\in \mathrm{IPs}} V_{A2}(R_{ij}) + \sum_{(i,j,k)\in \mathrm{ITs}} V_3(R_{ij}, R_{jk}, R_{ik}). \tag{3.20}$$

As in the two-body simulations, only those atoms which are defined to be interacting pairs are accounted for in the evaluation of the pair potential. Similarly, the nonadditive three-body potential is only evaluated for the set of interacting trimers (ITs), defined to be all trimers formed from a central atom and two of its nearest neighbors. In order to reduce the number of optimization steps, optimized a_{xy}, a_z, and b variational parameters from the two-body optimizations are used as the starting trial wavefunction parameters.

The in-plane and out-of-plane mean squared displacements are calculated from 2.56x10^6 VMC snapshots as before, and are tabulated in Table 3.5 along with their percent differences. From this data, we can conclude that the incorporation of three-body interactions does not result in any significant increase in the difference between $\langle u_{xy}^2\rangle$ and $\langle u_z^2\rangle$. In fact, the percent difference decreases at the experimental density from the Blackburn study compared to the two-body simulation results. In addition, a comparison with Table 3.2 indicates that the mean squared displacements calculated from VMC optimization using a three-body potential do not differ significantly from those obtained from the two-body simulations. This suggests that the incorporation of three-body interactions does not strongly affect the optimized wavefunction, and therefore a perturbative treatment of three-body interactions may be a viable option for modeling this system.

Table 3.5: Mean squared displacements calculated from VMC+3B simulations. Statistical uncertainties in the mean squared displacements are less than $\pm$ 3x$10^{-5}\mathring{A}^2$, for densities below 8.0x$10^{-3}a_o^{-3}$, and less than $\pm$ 6x$10^{-6}\mathring{A}^2$ for all higher densities.

Density ($10^{-3}a_o^{-3}$)	$\langle u_{xy}^2 \rangle$ ($\mathring{A}^2$)	$\langle u_z^2 \rangle$ ($\mathring{A}^2$)	% Difference
4.1896	0.2776	0.2776	0.00
4.2943	0.2635	0.2632	0.11
4.3991	0.2506	0.2502	0.16
4.5038	0.2384	0.2381	0.13
4.6086	0.2271	0.2269	0.09
4.7133	0.2168	0.2166	0.09
4.8180	0.2070	0.2067	0.15
4.9228	0.1979	0.1974	0.25
5.0275	0.1895	0.1893	0.11
5.5000	0.1568	0.1564	0.26
6.0000	0.1312	0.1309	0.23
6.5000	0.1117	0.1114	0.27
7.0000	0.09636	0.09614	0.23
7.5000	0.08409	0.08392	0.20
8.0985	0.07286	0.07269	0.23
9.2112	0.05709	0.05695	0.25
10.467	0.04519	0.04506	0.29
11.320	0.03929	0.03919	0.25

Chapter 4

Effect of Three-Body Interactions on the Zero-Temperature Equation of State of hcp Solid ^{4}He

A version of this chapter was originally published in The Journal of Chemical Physics in 2017 by Ashleigh Barnes and Robert Hinde. Only minor revisions have been made to allow for consistency of formatting. The original citation is given below.

Ashleigh L. Barnes and Robert J. Hinde. *The Journal of Chemical Physics* **146**(9), 094510 (2017).

The article was written by Ashleigh Barnes with revision suggestions from Robert Hinde.

The Journal of Chemical Physics is an AIP publication. The AIP policy on article reproduction is as follows:

Abstract

Previous studies have pointed to the importance of three-body interactions in high density ^{4}He solids. However the computational cost often makes it unfeasible to incorporate these interactions into the simulation of large systems. We report the implementation and evaluation of a computationally efficient perturbative treatment of three-body interactions in hexagonal close packed (hcp) solid ^{4}He utilizing the recently developed nonadditive three-body potential of Cencek, *et al.* This study represents the first application of the Cencek three-body potential to condensed phase ^{4}He systems. Ground state energies from quantum Monte Carlo simulations, with either fully incorporated or perturbatively treated three-body interactions, are calculated in systems with molar volumes ranging from 21.3 cm^3/mol down to 2.5 cm^3/mol. These energies are used to derive the zero-temperature equation of state for comparison against existing experimental and theoretical data. The equations of state derived from both perturbative and fully incorporated three-body interactions are found to be in very good agreement with one another, and reproduce the experimental pressure-volume data with significantly better accuracy than is obtained when only two-body interactions are considered. At molar volumes below approximately 4.0 cm^3/mol, neither two-body nor three-body equations of state are able to accurately reproduce the experimental pressure-volume data, suggesting that below this molar volume four-body and higher many-body interactions are becoming important.

4.1 Introduction

The highly quantum nature of condensed phase ^{4}He makes it an interesting system for the study and development of quantum mechanical models. Existing at absolute zero and pressures above 25 bar[1], hcp solid ^{4}He is a quantum solid whose properties are dominated by large atomic zero point motions. These zero point motions lead to atoms coming into much closer contact than the average lattice spacing at a given density would ordinarily suggest, thereby increasing the probability of three atoms simultaneously coming into close contact with one another. Typical simulations of this system employ a pairwise-additive model of the potential energy, as described by Eq. 4.1,

$$V = \frac{1}{2}\sum_{i}^{N}\sum_{j\neq i}^{N} V_2(R_{ij}), \tag{4.1}$$

where V_2 is any of the reliable two-body potential energy functions known for ^{4}He. This approach reduces computational cost by ignoring contributions to the total energy from three-body and higher many-body interactions. However, previous work by Ujevic and Vitiello[2] utilizing the three-body potential energy function developed by Cohen and Murrell[3] suggests that three-body interactions can have a significant impact on the calculated properties of both solid and liquid ^{4}He including the melting and freezing densities.

The high computational cost of evaluating these interactions can make full incorporation of a three-body potential intractable even for moderately small system sizes. However, there have been some attempts to account for these contributions perturbatively[4]. Shortly after the findings of Ujevic and Vitiello[2] were published, another study was reported by Cazorla and Boronat[5] in which electronic density functional theory (DFT) calculations were used to perturbatively correct two-body energies from diffusion Monte Carlo (DMC) simulations for all many-body interactions, resulting in a pressure-volume equation of state (EOS) which was able to predict experimental pressures at molar volumes down to 2.5 cm^3/mol with much greater accuracy than previously obtained from two-body simulations. These calculations, however, did not differentiate between three-body contributions and four-body or higher many-body contributions. Following this study, another investigation from the same group utilized DFT with Grimme-D2 van der Waals corrections (DFT-D2)[6] to parameterize effective three-body potentials based on an original model publised by Bruch and McGee[7] which were then incorporated into

DMC simulations[8]. These effective potentials were shown to improve agreement with experimental equations of state and bulk moduli, among other properties, over traditional two-body models. However, the effective three-body models were parameterized from calculations of a relatively small number of configurations (16) of a system of 96 ^{4}He atoms, and relied on the assumption that any discrepancy between the DFT-D2 potential energy and the pairwise additive potential energy calculated from the Aziz HFD-B(He) potential[9] can be attributed to three-body interactions alone.

In order to more carefully examine the contributions of three-body interactions and to understand at which densities these contributions become significant, we perform quantum Monte Carlo (QMC) simulations which incorporate a three-body potential either through a perturbative treatment or by fully incorporating it into the potential energy calculations throughout the simulation. This study utilizies a newer three-body potential developed by Cencek, *et al.*[10] using the full configuration interaction method. A number of recent investigations have applied this non-additive three-body function to the study of the gas phase properties of ^{4}He[11, 12, 13, 14, 15], however it has not yet been implemented in the studies of condensed phases.

In Sec. 4.2 and 4.3 below we will briefly discuss the computational methods employed, including the proposed perturbative treatment of three-body interactions, and assess the accuracy of our treatment and the Cencek potential by comparing the calculated energies and pressure-volume EOS to existing theoretical and experimental data.

4.2 Computational Methods

The following sections detail the computational methods employed in this investigation. In the first section, we review the previously reported variational Monte Carlo (VMC) simulation technique utilized to optimize trial wavefunctions[16] from which approximate ground state energies can be calculated. Next, we introduce the variational path integral Monte Carlo (VPI) method that was implemented in order to obtain exact ground state properties, within statistical uncertainty, from the VMC-optimized trial wavefunctions. In the VMC and VPI simulations, three-body interactions are treated both perturbatively and by fully incorporating the Cencek non-additive three-body potential[10] into the potential energy function throughout the simulation. These two approaches to the treatment of three-body interactions are discussed next. Finally, we present the procedure used to derive energy-volume and pressure-volume equations of state from the VMC and VPI energies with and without three-body interactions.

4.2.1 VMC

Trial wavefunctions are optimized using VMC[1] in order to obtain approximate ground state wavefunctions. The VMC simulations reported here make use of a Jastrow-McMillan style trial wavefunction[17] of the following form:

$$\Psi = A \prod_{i} e^{-a_{xy}(s_{i,x}^2 + s_{i,y}^2)} e^{-a_z s_{i,z}^2} \prod_{(i,j)\in \mathrm{IPs}} e^{-\frac{1}{2}(b/R_{ij})^5}, \tag{4.2}$$

where A is a normalization factor, $\vec{s}_i = (s_{i,x}, s_{i,y}, s_{i,z})$ is the displacement vector of atom i from its average lattice site, R_{ij} is the instantaneous distance between atoms i and j, and a_{xy}, a_z, and b are variational parameters which are optimized to give the minimum average energy per atom. Eq. 4.2 adopts the convention that the basal plane of the crystal lies parallel to the (x, y) plane and perpendicular to the z azis. In the second term, IPs denote the set of interacting pairs of atoms defined as those whose average lattice sites are separated by a distance less than $2.05R_{nn}$, where R_{nn} is the nearest neighbor distance. Only those atoms in the set of IPs contribute to the two-body potential energy calculated throughout the VMC simulations. Two-body contributions from all other atomic pairs are accounted for using a long-range correction precedure[16]. VMC provides a means of statistically sampling atomic configurations from the probability density of the fixed trial wavefunction, and therefore the wavefunction does not evolve throughout the simulation. Instead, the simulation progresses through a specified number of Monte Carlo cycles (MCC), where one MCC is defined as an attempt to displace each of the atoms once sequentially. Snapshots of the atomic positions and the average observables are recorded every 50 MCCs. We have previously verified that this interval is adequate to eliminate correlation between sequential snapshots[16]. The a_{xy}, a_z, and b variational parameters are optimized for a range of molar volumes from 2.5 cm^3/mol to 21.3 cm^3/mol following the previously reported method[16] using a simulation cell of $N_{\mathrm{VMC}} = 448$ ^{4}He atoms with periodic boundary conditions applied in all directions. Snapshots from 2.56x10^6 MCCs are generated from each optimized wavefunction and used to calculate the average total energy utilizing the Aziz HFD-B(He) pair potential[9].

4.2.2 VPI

VMC is an approximate method and therefore suffers from systematic error due to the variational principle. In order to account for this error and observe its effect on the calculated equations of state, simulations are also performed at each density using VPI as implemented in QSATS[18], where the QSATS code has been modified to accept independent a_{xy} and a_z wavefunction parameters.

Also known as the path integral ground state method (PIGS)[19], VPI allows for the determination, within statistical uncertainty, of the exact ground state properties of a quantum system at $T = 0$ K. The system is modeled as a p-bead polymer chain where each bead is a replica of the hcp ^{4}He system consisting of N_{VPI} atoms. For these simulations, $N_{\mathrm{VPI}} = 180$. Progression down the chain in either direction corresponds to the evolution of the wavefunction in imaginary time as described by the imaginary time propagator $\exp[-\hat{H}\Delta\tau/\hbar]$, where the links between adjacent beads represent the evolution of the system for $\Delta\tau$ units of imaginary time. A trial wavefunction is initialized at each end of the polymer chain. Given that the trial wavefunction can be written as a linear combination of the eigenfunctions of $\hat{H}$, applying the imaginary time propagator with each step along the chain projects out the lowest energy state of the system (Ψ_{gs}) from the trial wavefunction while contributions from all higher energy states decay to zero.

The configurations of the p-bead polymer chain are generated using conventional Metropolis Monte Carlo moves[20] and can be described by the probability density in the following equation:

$$P(C) = A\Psi_{tr}(\mathbf{Q}_1)\Psi_{tr}(\mathbf{Q}_p)exp\Big(-\frac{\Delta\tau}{\hbar}\sum_{j=1}^{p-1} F(\mathbf{Q}_j, \mathbf{Q}_{j+1})\Big) \tag{4.3}$$

where $C = \mathbf{Q}_1,\ \mathbf{Q}_2,\ ...,\ \mathbf{Q}_p$ is the configuration of the entire polymer chain and $\mathbf{Q}_j$ represents the configuration in the jth replica[18]. In the above equation, $\Psi_{\mathrm{tr}}(\mathbf{Q})$ represents the trial wavefunction which approximates the ground state wavefunction, and $F(\mathbf{Q}_j, \mathbf{Q}_{j+1})$ is a function which depends on the potential energy of the total N-particle system, particle mass, and the displacement of each particle from bead j to bead $j+1$[18]. When multiplied by the factor $-\frac{\Delta\tau}{\hbar}$ this term can be related to the Trotter factorization[21] of the imaginary time propagator.

The form of the probability density in Eq. 4.3 indicates that the two trial wavefunctions at beads $j = 1$ and $j = p$ are being propagated through imaginary time along the chain in either direction. As the number of replicas p increases and the imaginary time step $\Delta\tau$ decreases, the distributions of the interior beads approach that of the exact ground state probability density, $|\Psi_{gs}|^2$, while terminal beads sample the probability density $|\Psi_{tr}\Psi_{gs}|$. By sampling the interior beads' distributions we are able to calculate the average ground state properties of the system. This is true as long as the overlap between the trial and exact ground state wavefunctions is not too small; however, convergence can be significantly improved with the use of a more accurate trial wavefunction. For this reason, optimized wavefunctions from VMC are used as the starting trial wavefunctions for the VPI simulations. For a more in-depth description of this simulation method, the reader is referred to Refs. [1] and [19].

A single MCC in these simulations corresponds to an attempt to move each atom in each replica once, in sequential fashion. The moves are carried out using an algorithm that automatically incorporates all contributions to the F function of Eq. 4.3 except for those related to the many-body potential energy V. If V in the new configuration is lower than the previous configuration, the move is accepted, otherwise it is conditionally rejected with the probability $e^{-\Delta V \Delta\tau/\hbar}$. Each simulation begins with a 1x10^5 MCC warmup run, followed by a 1x10^6 MCC production run, from which snapshots of atomic positions are recorded every 1000 MCCs. In this study, 430 replicas and a time step $\Delta\tau = 200$ au are used. The time step is consistent with previous simulations of hcp solid ^{4}He using the QSATS code which found the calculated observables to be independent of the time step parameter for 200 au $\leq \Delta\tau \leq$ 500 au[18].

Fig. 4.1 shows how the average two-body potential energy (V_2) changes with progression along the replica chain in the VPI simulations when $V_m = 2.50$ cm^3/mol. V_2 is expected to converge to the exact ground state potential energy in the center replicas when p is sufficiently large. In Fig. 4.1 we observe that in all but the end replicas shown, the average two-body potential energies agree within their 95% confidence intervals. Using the VMC optimized wavefunction as the trial wavefunction at this molar volume, V_2 is considered to be converged after 23 replicas, corresponding to an imaginary time of 4600 au. At the remaining molar volumes studied here, similar behavior is observed, though the VMC trial wavefunction is occassionally close enough to the ground state wavefunction that even the end replicas' V_2 values agree with the interior replicas within their 95% confidence intervals. This indicates that the optimized wavefunctions from VMC are very close to the exact ground state wavefunctions. However in order to ensure that the potential energies are representative of the true ground state wavefunctions, an interior subset of these converged replicas is used to calculate the average potenial energy. In most cases,

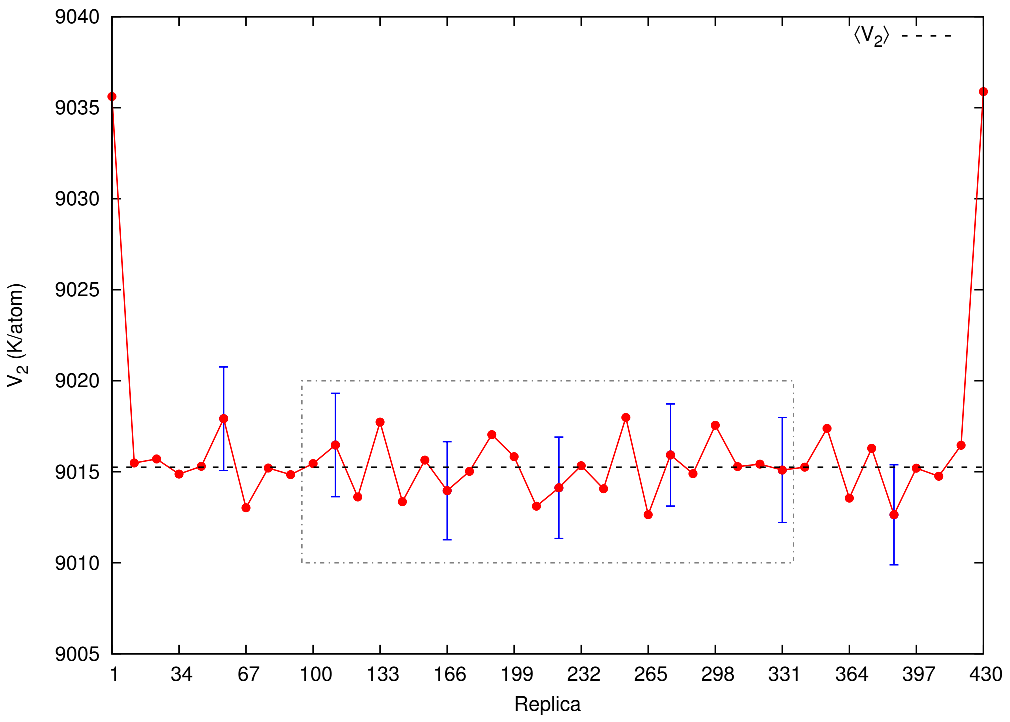


Figure 4.1: Average two-body potential energy, V_2, vs. replica number from VPI simulations with $V_m = 2.50$ cm^3/mol. Error bars represent the 95% confidence intervals. The grey box shows the replicas used to calculate the total average two-body potential energy, $\langle V_2 \rangle$, whose value is given by the black dashed line.

this subset encompasses replicas 100 to 331, however the interval is occasionally adjusted if the starting trial wavefunction is further from the exact ground state at a given density, based on the observed convergence of V_2. In order to reduce correlation between the replicas, every 11th replica within this subset is included in the calculation of the average potential energy and any other position-dependent observables.

4.2.3 Three-Body Interactions

This study utilizes the ^{4}He nonadditive three-body potential reported by Cencek, *et al.*[10], which was developed using electronic structure methods that included large atom-centered basis sets and approach full-configuration-interaction accuracy. Two methods are employed to incorporate three-body interactions in our quantum Monte Carlo simulations: a computationally efficient perturbative treatment, and a full-incorporation method in which the nonadditive three-body potential is added to the Aziz pair potential throughout the simulation. Both methods consider only those trimers formed by an atom and two of its nearest neighbors. The implementation of these two methods is explained below.

4.2.3.1 Perturbative Treatment

A perturbative treatment of three-body interactions makes the assumption that three-body interactions do not have a significant impact on the distribution of the ^{4}He atomic positions throughout the QMC simulations. The three-body correction to the potential energy can therefore be calculated by evaluating a three-body potential energy function using snapshots of atomic positions obtained from two-body simulations. For VMC simulations, the three-body correction makes use of the fact that a central atom can form 66 different trimers with its 12 nearest neighbors, which can be further categorized into 6 different geometries based on the central angle when the atoms are in their equilibrium positions: 60°(24), 90°(12), 109.47°(3), 120°(18), 146.44°(6), and 180°(3)[22]. The number in parentheses is the number of trimers with the given central angle. The average three-body energy can therefore be calculated from VMC snapshots from one representative trimer of each geometry, weighted by the number of times it occurs in the 66 trimers. Although the representative trimers are selected based on their equilibrium geometry, deviation from this configuration throughout the three-body energy calculation due to atomic delocalization is accounted for by applying the atomic displacements recorded in the VMC snapshots. The selection of the six trimers simply ensures that each of the possible trimer environments are accounted for in the three-body energy calculation while maintaining computational efficiency. This does not result in any significant loss of accuracy compared to treating each trimer individually (as shown in Fig. 4.2 for a system with $V_m = 16.22$ cm^3/mol). Additionally, the resulting increase in uncertainty is significantly less than the uncertainty in the two-body energies due to variational error, which is on the order of 1 K/atom at the density represented in Fig. 4.2 (see Appendix, Table 4.9). In order to prevent triple counting, contributions from equilateral trimers (which contain three nearest neighbor pairs and would therefore appear in the trimer list for three different central atoms) are divided by three. The final three-body correction is reported as the average three-body energy per atom. The added computational cost of this perturbative correction is negligible compared to the VMC snapshot generation steps, amounting to an increase in CPU time of approximately 0.2%. (This treatment is referred to as the perturbative treatment because the total energy of the system is evaluated using a first order perturbation of the two-body Hamiltonian to account for three-body interactions: $E = E_2 + \langle V_3 \rangle$.)

VPI is an exact method which does not suffer from variational error, and therefore a more precise approach is implemented in order to reduce the uncertainty in the perturbative three-body energy correction. All nearest neighbor trimers from all central atoms are evaluated individually at each VPI snapshot, again being sure to account for triple counting of equilateral trimers, and the average three-body energy per atom is reported. The consideration of each of the trimers individually increases

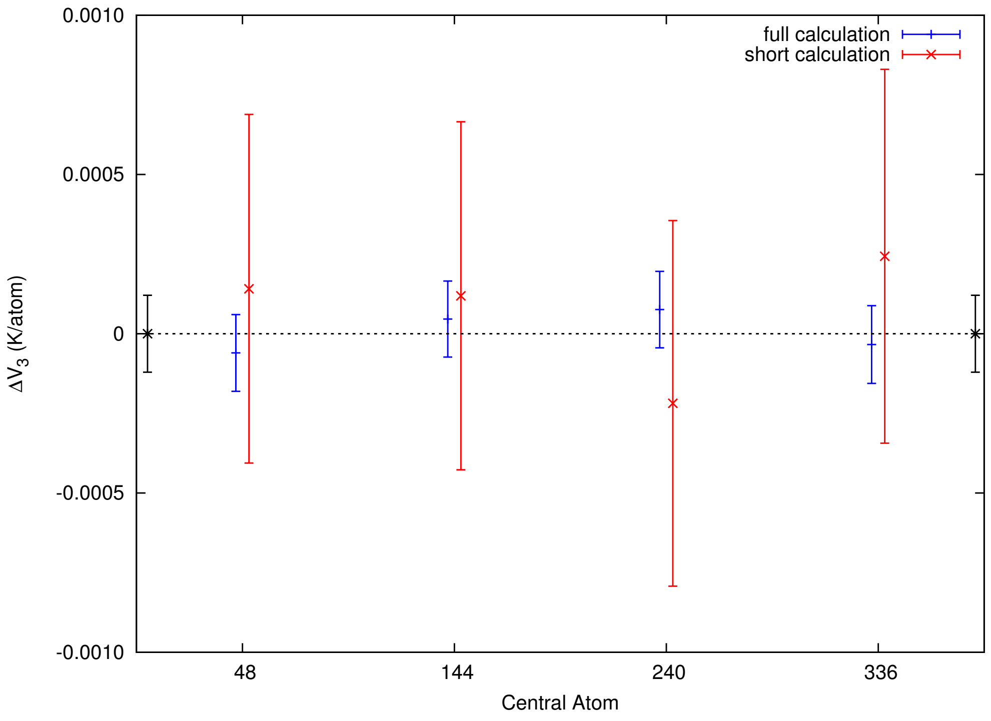


Figure 4.2: Difference in the three-body potential energy from the average calculated using various central atoms considering all 66 trimers individually (blue) or considering only one representative of each trimer geometry (red) when $V_m = 16.22$ cm^3/mol. Though only four data points are shown here, these calculations were repeated for 21 different central atoms. The average (black dashed line) ($\Delta V_3 = 0$) was determined from the average of the full calculation over all central atoms. Error bars represent 95% confidence intervals.

the computational cost of the perturbative treatment moreso than the implementation in the VMC simulations, however the increase in total CPU time is less than 5%. Convergence of V_3 with replica count is also considered in these simulations, and we find that V_3 converges slower than V_2 in most cases, requiring approximately 55 replicas, or 11000 au time to converge.

4.2.3.2 Full-Incorporation Method

In the perturbative treatment above, three-body interactions are not considered until the wavefunction has been fully optimized using only two-body interactions. This does not allow for the influence of three-body interactions on the wavefunction or the generated atomic configurations. In order to determine if this is a reliable approach, three-body interactions must be fully incorporated into the wavefunction optimizations. In the VMC simulations, the Cencek three-body potential must then be evaluated for all nearest neighbor trimers whenever the Aziz pair potential is calculated. This increases the computational cost of the VMC simulations approximately eight-fold. In order to reduce the number of optimization steps required for these full-incorporation simulations, the optimized wavefunction parameters from the VMC two-body simulations are used as starting parameter values. Full-incorporation VMC optimizations are performed at every molar volume for which two-body optimized parameters have been determined.

Because the accept/reject criterion for MCC moves in VPI depends on the potential energy, in full-incorporation simulations the full three-body energy must be calculated with each attempted move rather than after each snapshot collection. The increase in computational cost is therefore significantly higher than in VMC. For this reason, these simulations are only performed at the highest densities where three-body interactions are most important and thus where discrepancies between perturbative and full-incorporation treatments are most probable. Similar convergence of the potential energy is observed in these simulations as in the two-body simulations.

4.2.4 Equation of State Calculations

From each set of simulation data, we derive the zero-temperature EOS. This is done by first fitting an equation to the energy-volume relationship for each data set using the gnuplot fitting routine[23]. The form of the equation used is shown in Eq. 4.4 below and is a rearrangement of the $E(V_m)$ relationship obtained from the modified Birch EOS reported by Driessen, *et al.*[24] in their study of solid ^{4}He.

$$E(V_m) = E_o - P_o V_m + aV_m^{-\frac{8}{3}} + bV_m^{-2} + cV_m^{-\frac{4}{3}} + dV_m^{-\frac{2}{3}} \tag{4.4}$$

This representation simply removes dependencies between the fitting parameters, allowing for faster convergence of the fitting algorithm[25]. Following the procedure of Driessen, *et al.*, the data are divided into high and low density regions using $V_m = 11.02$ cm^3/mol as the dividing point. In the high density region, uncertainties in the fitting parameters are significantly reduced with negligible impact on the residuals when P_o is constrained at the value $P_o = 0$. Similarly, in the low density region uncertainties in the parameters are improved by setting $a = 0$ for each data set. The final pressure-volume EOS is then derived as shown in the following:

$$P(V_m) = -\frac{\delta E}{\delta V_m} = P_o + \frac{8}{3}aV_m^{-\frac{11}{3}} + 2bV_m^{-3} + \frac{4}{3}cV_m^{-\frac{7}{3}} + \frac{2}{3}dV_m^{-\frac{5}{3}} \tag{4.5}$$

4.3 Results and Discussion

4.3.1 Energy-Volume Equations of State

The energies calculated according to the above methods result in five sets of simulation data spanning the 2.5 cm^3/mol to 21.3 cm^3/mol molar volume range: VMC two-body simulations (VMC-2B), VMC with perturbatively treated three-body interactions (VMC(3B)), VMC with fully incorporated three-body interactions (VMC+3B), VPI two-body simulations (VPI-2B), and VPI with perturbatively treated three-body interactions (VPI(3B)). In addition, energies from VPI with fully incorporated three-body interactions (VPI+3B) are reported at four selected densities. For reference, the six different energy calculation methods and their associated labels are also summarized in Table 4.1. The VMC optimized wavefunction parameters, along with the calculated total and three-body energies for each set of simulations are tabulated in Appendix Tables 4.8, 4.9, and 4.10. From all but the VPI+3B simulations, there are enough data to derive reliable equations of state. Of these five data sets, the VPI(3B) data represents the highest level of theory, addressing error due to both the variational principle and many-body interactions. These calculated energies are shown below in Figs. 4.3a and b, separated into low and high density regions, respectively.

The energies from each of the simulations are used to fit energy-volume equations of state (EV-EOS) in the form of Eq. 4.4. Values of the fitting parameters determined for each data set are tabulated in Tables 4.2 and 4.3 below. The resulting EV-EOS fit the calculated energies within 0.35% in the low density region, with most data points fitting within 0.001-0.1%. In the high density region, the largest error in the fit is 1.14% with the majority of data points fitting the EV-EOS within 0.001-0.15%. Simulations which include three-body interactions have much lower residuals in the high density region with a maximum error of 0.47%. For all energy-volume equations of state, the statistical uncertainties in the fitting parameters are less than 10%. Fig. 4.3 shows the energy-volume relationship obtained from the VPI(3B) simulation data, with the difference between the calculated EV-EOS from both VPI(3B) and VPI-2B and previously reported experimental[26] and theoretical[5] energies shown in Fig. 4.3c. From this figure we see that at our highest level of theory we are able to reproduce experimental findings to better than 2.5 J/mol (energies in this region range from about -55 to -20 J/mol). The calculated EV-EOS from the VPI(3B) data is also in good agreement with the DMC energies reported by Cazorla and Boronat in Ref. [5]. However, the energies from Ref. [5] in this molar volume region do not include contributions from three-body interactions and are therefore expected to be

Table 4.1: Summary of the six different simulation methods implemented in this study and their associated labels.

Label	Method description
VMC-2B	VMC simulations utilizing the Aziz HFD-B(He) pair potential[9]
VMC(3B)	VMC simulations utilizing the Aziz HFD-B(He) pair potential with perturbative three-body corrections using the Cencek three-body potential[10]
VMC+3B	VMC simulations utilizing the Aziz HFD-B(He) pair potential with fully incorporated three-body interactions using the Cencek three-body potential
VPI-2B	VPI simulations utilizing the Aziz HFD-B(He) pair potential
VPI(3B)	VPI simulations utilizing the Aziz HFD-B(He) pair potential with perturbative three-body corrections using the Cencek three-body potential
VPI+3B	VPI simulations utilizing the Aziz HFD-B(He) pair potential with fully incorporated three-body interactions using the Cencek three-body potential

Figure 4.3: Energy vs. molar volume from VPI(3B) simulations in the (a) low and (b) high density regions along with the best fit Birch energy-volume EOS (blue line). Experimental data from [26] is provided for comparison in the low density region (black squares). (c) Difference between energies reported by [26] and [5] and the best fit Birch equation from VPI-2B and VPI(3B). In most cases, statistical uncertainties are smaller than the symbol size. *Continued on next page.*

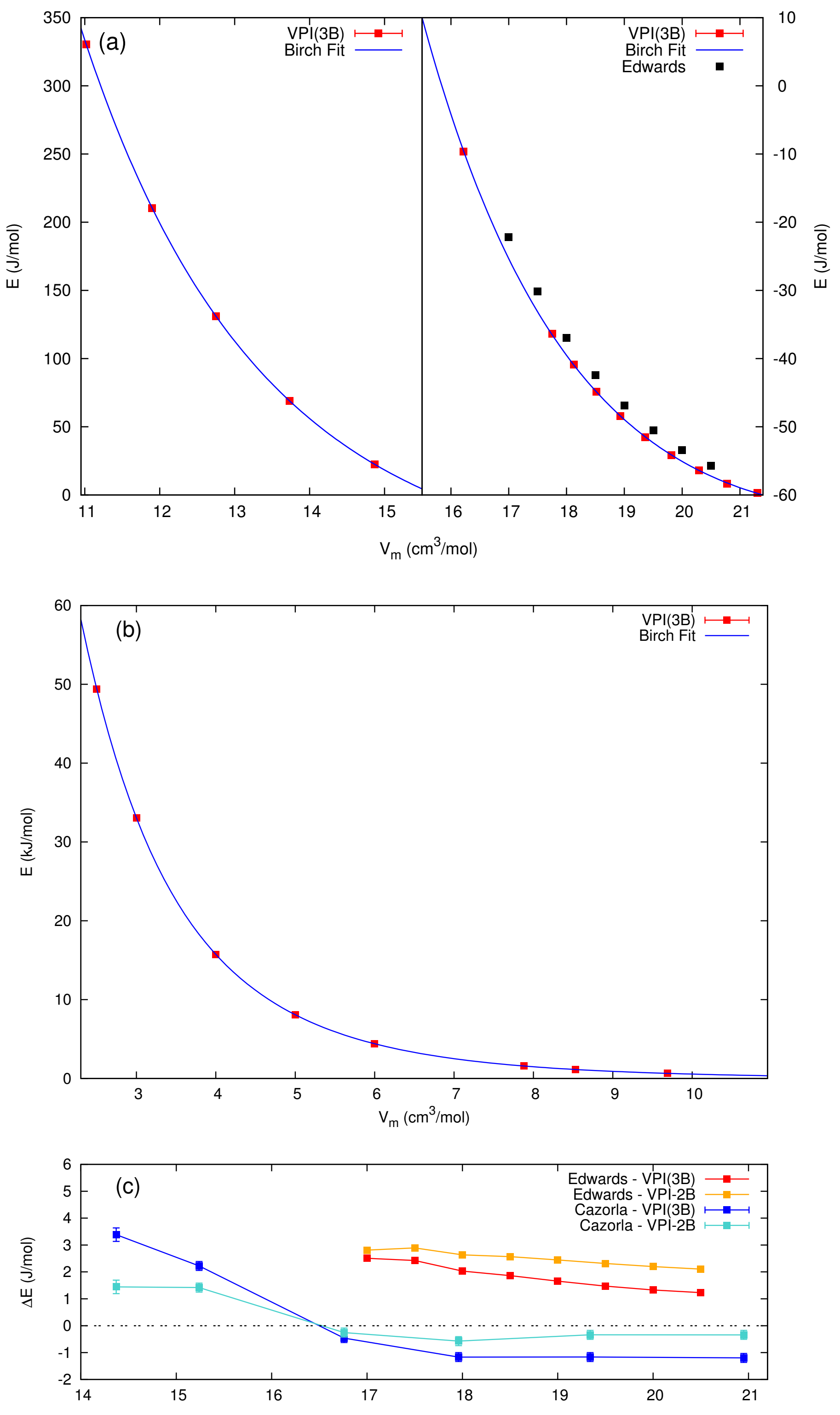


Figure 4.3: *Continued*

Table 4.2: Fitting parameter values for the low density ($V_m \geq 11.02$ cm^3/mol) EV-EOS for each simulation set. In this density range, $a = 0$. The reported equations fit the energies in this density range with RMS error of 0.05 J/mol and a maximum residual ($r_{\max}$) of 0.13 J/mol.

Simulation	E_o	P_o	b	c	d
	(J/mol)	(bar)	(bar(cm^3/mol)^3)	(bar(cm^3/mol)$^{7/3}$)	(bar(cm^3/mol)$^{5/3}$)
VMC-2B	-5589.6 ± 226.8	-398.8 ± 21.9	10550998.1 ± 179253.8	-4727328.6 ± 112551.3	797386.7 ± 25456.7
VMC(3B)	-4357.9 ± 304.5	-299.4 ± 28.5	8994196.1 ± 256993.2	-3906621.8 ± 157778.9	638657.9 ± 34912.7
VMC+3B	-4726.9 ± 193.7	-335.9 ± 18.7	9267951.2 ± 153323.1	-4082828.0 ± 96216.8	679339.9 ± 21751.1
VPI-2B	-5783.3 ± 198.2	-419.1 ± 19.1	10644347.1 ± 157538.9	-4801254.9 ± 98722.4	816351.9 ± 22288.8
VPI(3B)	-4441.2 ± 207.3	-309.8 ± 20.0	8981022.4 ± 164722.4	-3918818.9 ± 103222.5	644507.7 ± 23304.5

Table 4.3: Fitting parameter values for the high density ($V_m \leq 11.02$ cm^3/mol) EV-EOS for each simulation set. In this density range, $P_o = 0$. The reported equations fit the energies in this density range with RMS error of 17.1 J/mol and a maximum residual ($r_{\max}$) of 73.8 J/mol.

Simulation	E_o	a	b	c	d
	(J/mol)	(bar(cm^3/mol)$^{11/3}$)	(bar(cm^3/mol)^3)	(bar(cm^3/mol)$^{7/3}$)	(bar(cm^3/mol)$^{5/3}$)
VMC-2B	-12868.2 ± 911.9	-7652817.0 ± 804833.2	23652401.4 ± 1112120.0	-11063674.4 ± 553938.0	1987749.0 ± 118101.6
VMC(3B)	-10064.6 ± 733.8	-18041859.9 ± 931792.3	27527260.9 ± 1174835.5	-10763538.2 ± 533173.9	1715066.5 ± 103753.1
VMC+3B	-11033.1 ± 283.3	-19116327.0 ± 258296.3	28919163.1 ± 353839.5	-11420539.5 ± 174765.3	1847733.0 ± 36963.4
VPI-2B	-12934.8 ± 952.9	-7685758.7 ± 810681.7	23700628.5 ± 1128983.9	-11095794.5 ± 567531.1	1995188.6 ± 122217.9
VPI(3B)	-10951.6 ± 233.6	-18980798.8 ± 200662.4	28780753.6 ± 278766.9	-11365828.5 ± 139785.4	1836496.6 ± 30030.1

in better agreement with the VPI-2B data from this study, which is shown to be the case in Fig. 4.3c.

4.3.2 Evaluation of the Perturbative Treatment

The accuracy of the perturbative treatment of three-body interactions in the VMC and VPI simulations is assessed by comparing the two- and three-body potential energies to those obtained from VMC and VPI full-incorporation calculations. Due to the high computational cost, VPI+3B calculations are only performed at the first density in the high density region, as well as three of the highest densities, corresponding to molar volumes of 11.02 cm^3/mol, 7.88 cm^3/mol, 6.00 cm^3/mol, and 4.00 cm^3/mol. At these molar volumes, three-body interactions are more significant and are therefore more likely to impact the optimized wavefunction and ground state energies. The total energy with long-range corrections (E_{tot}), two-body (V_2), and three-body (V_3) potential energies at these molar volumes from the VMC(3B) and VMC+3B data sets are tabulated below in Table 4.4 along with the percent differences. Corresponding quantities from the VPI(3B) and VPI+3B simulations are given in Table 4.5.

The data in Tables 4.4 and 4.5 indicate that even at these low molar volumes, the total energies and three-body potential energies obtained from perturbative and full-incorporation simulations are in very good agreement with one another, differing by less than 0.26% and 2.5%, respectively. Interestingly, there is a more significant difference in the average two-body potential energies calculated from the perturbative and full-incorporation treatments. This suggests that the full incorporation of three-body interactions has a non-negligible effect on the ground state wavefunction. The percent differences in the VMC variational parameters are provided in the Appendix, Table 4.8. Though the optimized parameters are expected to change slightly from VMC-2B to VMC+3B simulations due to the additional round of optimization[16], from this data it is apparent that the percent difference in the b parameter arising from

Table 4.4: Comparison of E_{tot}, V_2, and V_3 energies from VMC(3B) and VMC+3B.

V_m (cm^3/mol)	Simulation	E_{tot} (J/mol)	V_2 (J/mol)	V_3 (J/mol)
11.02	VMC(3B)	344.68 ± 0.01	-283.98 ± 0.03	-18.54 ± 0.07
	VMC+3B	344.17 ± 0.01	-278.16 ± 0.03	-18.94 ± 0.01
	% difference	**0.148%**	**2.07%**	**2.13%**
7.88	VMC(3B)	1598.43 ± 0.28	570.65 ± 0.06	-156.02 ± 0.28
	VMC+3B	1595.81 ± 0.02	592.91 ± 0.05	-158.09 ± 0.01
	% difference	**0.164%**	**3.83%**	**1.32%**
6.00	VMC(3B)	4432.68 ± 0.82	3270.75 ± 0.09	-696.53 ± 0.83
	VMC+3B	4425.92 ± 0.02	3335.38 ± 0.09	-704.38 ± 0.01
	% difference	**0.153%**	**1.96%**	**1.12%**
4.00	VMC(3B)	15802.56 ± 3.55	17124.93 ± 0.16	-4795.29 ± 3.55
	VMC+3B	15765.70 ± 0.04	17327.61 ± 0.16	-4836.25 ± 0.03
	% difference	**0.234%**	**1.18%**	**0.851%**

Table 4.5: Comparison of E_{tot}, V_2, and V_3 energies from VPI(3B) and VPI+3B.

V_m (cm^3/mol)	Simulation	E_{tot} (J/mol)	V_2 (J/mol)	V_3 (J/mol)
11.02	VPI(3B)	330.5 ± 0.7	-284.6 ± 2.6	-19.8 ± 0.1
	VPI+3B	330.1 ± 0.8	-277.1± 3.0	-20.3 ± 0.1
	% difference	**0.121%**	**2.67%**	**2.49%**
7.88	VPI(3B)	1574.1 ± 0.8	575.5 ± 1.1	-160.0 ± 0.1
	VPI+3B	1572.5 ± 1.0	603.5 ± 1.1	-162.8 ± 0.1
	% difference	**0.107%**	**4.75%**	**1.73%**
6.00	VPI(3B)	4401.4 ± 1.5	3255.9 ± 1.7	-701.0 ± 0.2
	VPI+3B	4394.9 ± 1.5	3323.7 ± 1.7	-711.3 ± 0.2
	% difference	**0.148%**	**2.06%**	**1.46%**
4.00	VPI(3B)	15736.4 ± 3.2	17058.9 ± 4.0	-4806.6 ± 0.8
	VPI+3B	15695.8 ± 2.5	17270.4 ± 4.4	-4862.6 ± 0.9
	% difference	**0.258%**	**1.23%**	**1.16%**

the inclusion of three-body interactions increases significantly at lower molar volumes. It has been demonstrated that changes in the b variational parameter have a greater impact on the wavefunction than changes in the a_i parameters[16], and therefore larger percent differences in the b parameter at lower molar volumes indicate an increasing influence of three-body interactions on the ground state wavefunction. One possible interpretation of the greater influence of the b parameter on the wavefunction is that changing the a_i parameters affects the delocalization of the ^{4}He atoms, which results in opposing changes in the kinetic and potential energies. Therefore the effects of the a_i parameters cancel out to an extent. Changes in the b parameter, however, influence which interatomic distances may be sampled. This significantly impacts the potential energy of the system without strong compensation from the kinetic energy.

The effect of three-body interactions in the VPI simulations can be analyzed by considering the sampled atomic configurations. The distribution of nearest neighbor distances in the VPI(3B) and VPI+3B simulations is visualized in Fig. 4.4 for simulations where $V_m = 4.0$ cm^3/mol. From this figure, we see that the VPI+3B simulations tend to sample shorter ^{4}He-^{4}He distances. This is due to the more attractive three-body energies at these interatomic distances which help to stabilize configurations that are energetically unfavorable when only two-body interactions are considered. This is consistent with the more repulsive values of V_2 and more attractive values of V_3 in the VPI+3B simulations reported in Table 4.5. The close agreement between the total energies for the two sets of simulations given the difference in V_2 values is therefore the result of a compensating change in the kinetic energy. It is worth noting that the kinetic energy in this molar volume region ranges from approximately 600 to 3500 J/mol, and therefore makes a non-negligible contribution to the total energy. This illuminates the need for methods such as QMC which can accurately treat zero point motion and its effects on the kinetic and potential energies. Overall, the results of this analysis show that properties depending on the total energy, or derivatives thereof, can be calculated using the faster perturbative approach without significant loss of accuracy. However, accurate analysis of the

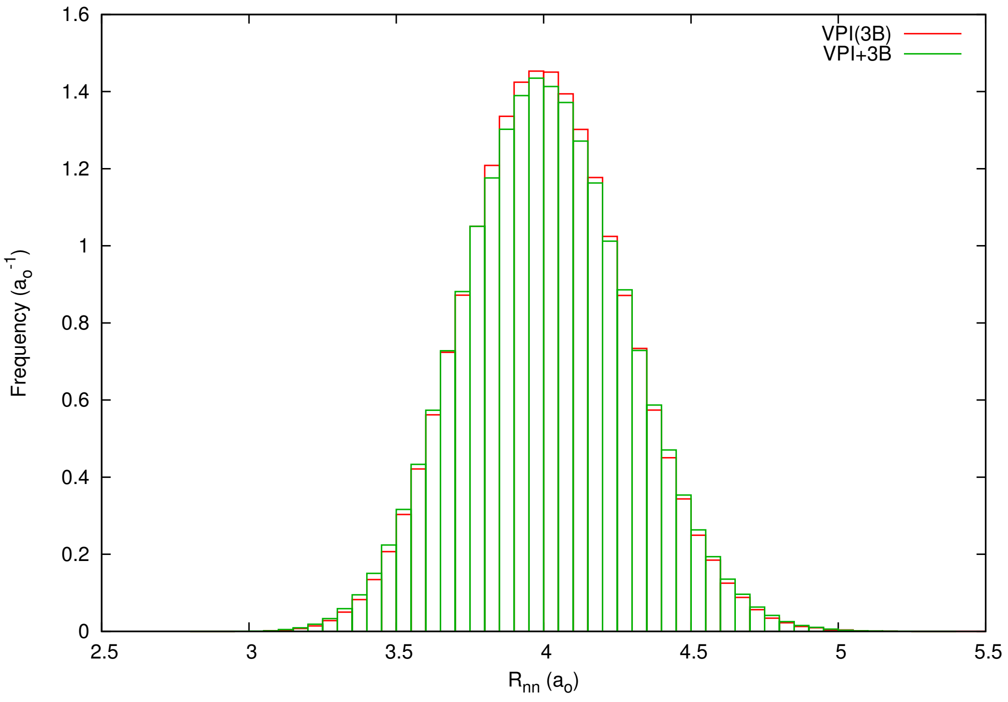


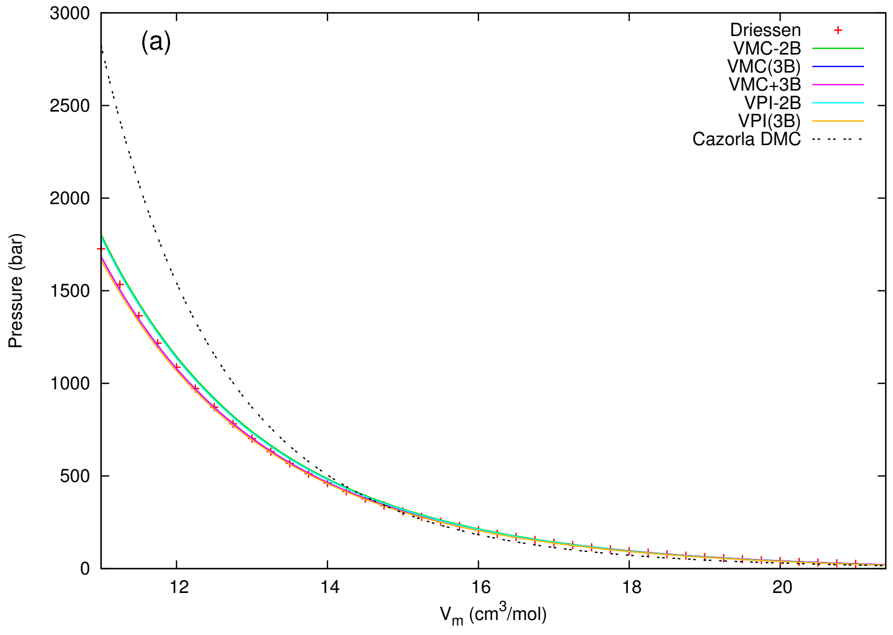
Figure 4.4: Comparison of average first nearest neighbor distances from VPI(3B) (red) and VPI+3B (green) simulations where $V_m = 4.00$ cm^3/mol.

potential and kinetic contributions to the total energy may require fully incorporating three-body interactions at higher densities.

4.3.3 Pressure-Volume Equations of State

The pressure-volume EOS for each of the simulation sets is constructed according to Eq. 4.5 from the fitting parameters reported above. The five resulting equations are shown in Fig. 4.5, along with experimental data from Driessen, *et al.*[24], and the theoretical EOS reported by Cazorla and Boronat derived from DMC energies[5]. The EOS reported in Ref. [5] included a perturbative correction for all many-body interactions from DFT calculations in the high density region. In Fig. 4.5, the equations of state derived in the present study appear to separate into two classes of overlaying equations: ones that include three-body interactions (VMC(3B), VMC+3B, and VPI(3B)), and ones that consider only two-body interactions (VMC-2B, VPI-2B). In addition to a visual comparison, the RMS error and maximum residual of each EOS compared to the Driessen data[24], provided in Table 4.6, are also quite similar within these two groups. Therefore, we will refer to three-body equations and two-body equations in general for simplicity.

At the lowest densities shown in Fig. 4.5a, where three-body interactions are not expected to be significant, each EOS agrees well with the experimental pressures reported in [24]. However, even at moderate densities when the molar volume is less than about 14.0 cm^3/mol, the two-body equations begin to diverge from the three-body equations, with the latter lying closer to the experimental data. Interestingly, the Cazorla and Boronat EOS diverges from the experimental data more rapidly than the VMC-2B or VPI-2B EOS from this study, resulting in an error of approximately 1 kbar near $V_m = 11$ cm^3/mol. In the high density region (Fig. 4.5b), the differences between simulations with and without three-body interactions become more obvious as two-body equations predict significantly higher pressures than the experimental values at molar volumes below about 7.0 cm^3/mol. The three-body simulations,



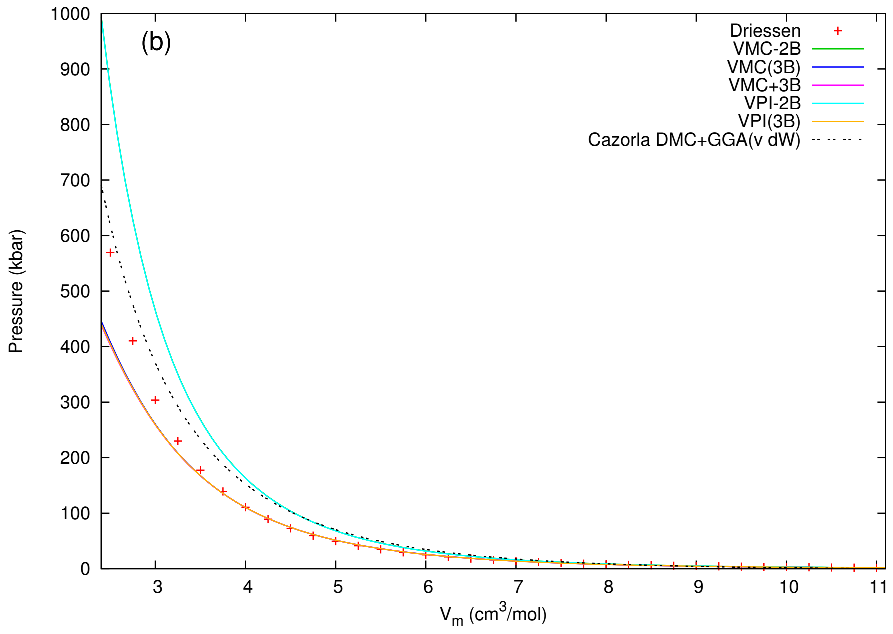


Figure 4.5: Comparison of all pressure-volume equations of state obtained for each simulation set in the (a) low and (b) high density regions. For comparison, experimental data from Ref. [24] is provided (red pluses) along with the theoretical EOS reported in Ref. [5] (dotted black line). In the high density region, this theoretical EOS is based on energies corrected for many-body interactions using DFT calculations[5] and was parameterized for molar volumes below 8.5 cm^3/mol. In both density regions, the VMC-2B and VPI-2B equations overlay one another, as do the VMC(3B), VMC+3B, and VPI(3B) equations.

Table 4.6: RMS error and maximum absolute error ($r_{\max}$) for each pressure-volume EOS compared to experimental values from Driessen *et al.*[24] In the high density region, the RMS error is calculated for $V_m \geq 4.0$ cm^3/mol. At lower molar volumes, four-body interactions are beginning to become significant and therefore both the two-body and three-body equations of state diverge from experimental values.

Simulation	Low Density EOS RMSE, $r_{\max}$ (bar)	High Density EOS RMSE, $r_{\max}$ (kbar)
VMC-2B	27.918, 73.51	15.113, 52.62
VMC(3B)	6.096, 28.58	0.826, 2.037
VMC+3B	6.228, 28.98	0.821, 2.043
VPI-2B	22.754, 60.69	15.031, 52.39
VPI(3B)	1.279, 45.82	0.811, 2.019

however, are in good agreement with experiment down to $V_m = 4.0$ cm^3/mol. This can be understood from the attractive nature of three-body interactions in this molar volume range (Fig. 4.6). The attractive three-body potential for configurations close to that of an equilateral triangle lowers the total energy, effectively making the system more compressible. The pressures calculated from three-body data are therefore lower than the corresponding pressures obtained from two-body simulations. At molar volumes near $V_m = 2.5$ cm^3/mol, the Cazorla and Boronat EOS, which has been corrected for all many-body interactions using DFT calculations, provides better agreement with experiment. This sheds light on the increasing importance of 4-body and higher many-body interactions as the density increases and suggests that below 4.0 cm^3/mol, three-body interactions alone may not be sufficient for an accurate model of this system.

The effect of three-body interactions on the EOS is considered in greater detail in Fig. 4.7, divided into low, middle, and high density regions. Here, the differences between the Driessen experimental pressure and both the VPI-2B and VPI(3B) EOS are shown. Experimental error, estimated by Driessen *et al.* to be $\pm$ 0.3% V_m, is shown on selected data points, as well as estimated uncertainties in the predicted pressures. These uncertainties are estimated by sequentially fixing each parameter from Eq. 4.4 at its fitted value $\pm 1\sigma$ or 2σ, where σ is the reported uncertainty in the parameter value given in Tables 4.2 and 4.3, and refitting the remaining parameters, resulting in 21 equations of state for each data set including the original EOS. At each point where vertical error bars are reported, the lower and upper bounds have been determined from the minimum and maximum values of these 21 equations at that density.

In general, these figures show that in every density region, the inclusion of three-body interactions using the Cencek, *et al.* potential[10] improves agreement with experiment. In addition, we see that two-body simulations consistently overestimate the pressure of the system. However, when the Cencek three-body potential energy is incorporated, the calculated pressures are typically slightly lower

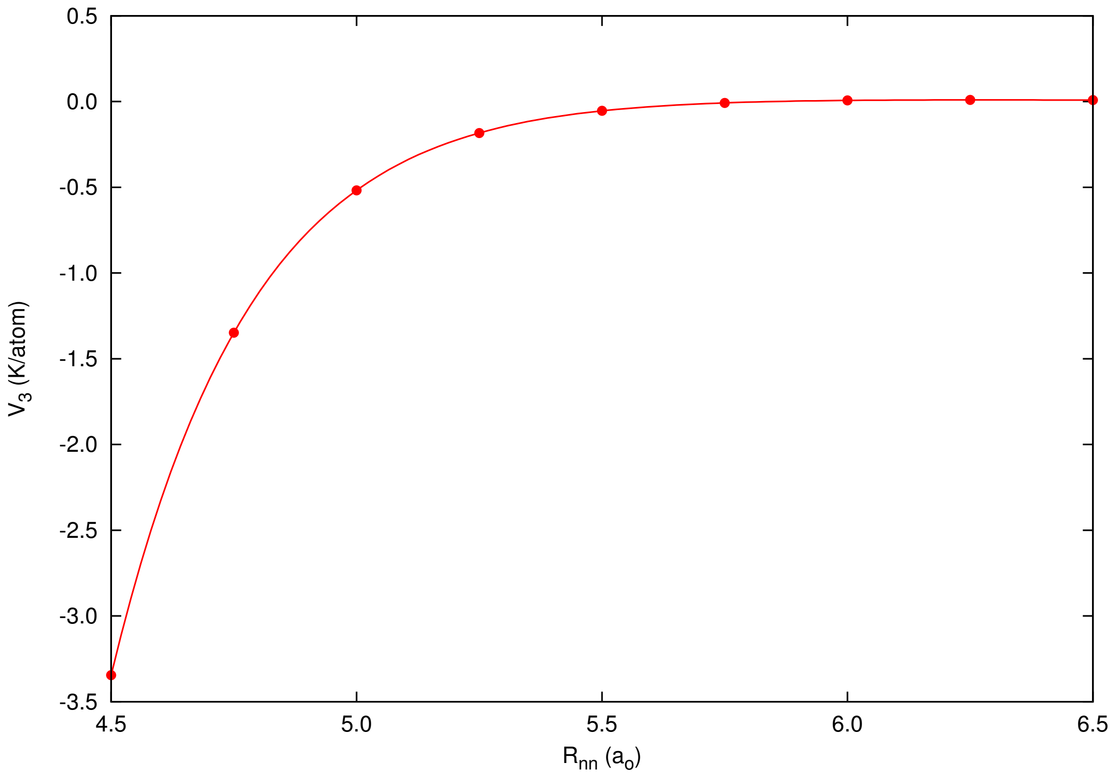
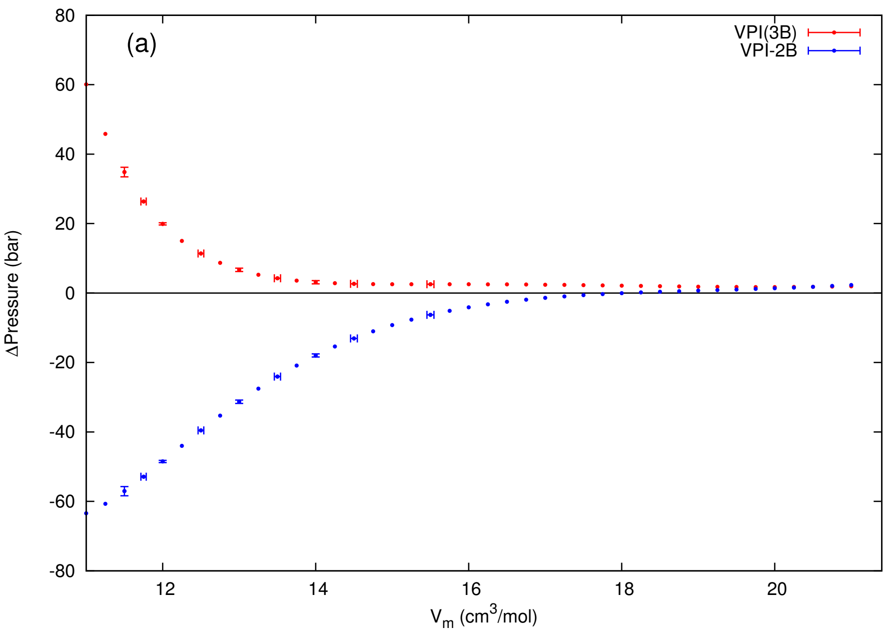


Figure 4.6: Nonadditive three-body potential energy for a ^{4}He equilateral trimer with side lengths R_{nn} calculated using the three-body potential from Cencek, *et al.*[10]. The energy is attractive where $R_{nn} < 6.0\ a_o$, corresponding to a molar volume of approximately 13.6 cm^3/mol.

Figure 4.7: Δ Pressure vs. V_m where Δ Pressure $\equiv P_{\text{Driessen}} - P_{\text{VPI-2B}}$ (blue) or $P_{\text{Driessen}} - P_{\text{VPI(3B)}}$ (red) calculated from the reported pressure-volume EOS. (a) Low density region. Vertical error bars are smaller than the symbol size for $V_m > 14.0$ cm^3/mol. (b) Middle density region. Vertical error bars are smaller than the symbol size at all molar volumes for the VPI(3B) EOS, and where $V_m > 7.0$ cm^3/mol for the VPI-2B EOS. (c) High density region. Vertical error bars are smaller than the symbol size at all molar volumes above 2.5 cm^3/mol. *Continued on next page.*



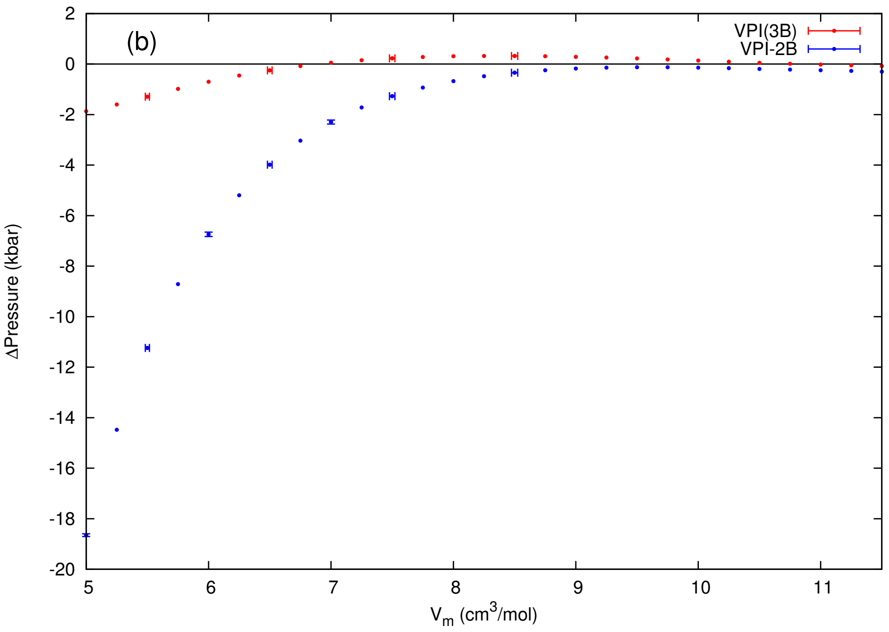


Figure 4.7: *Continued.*

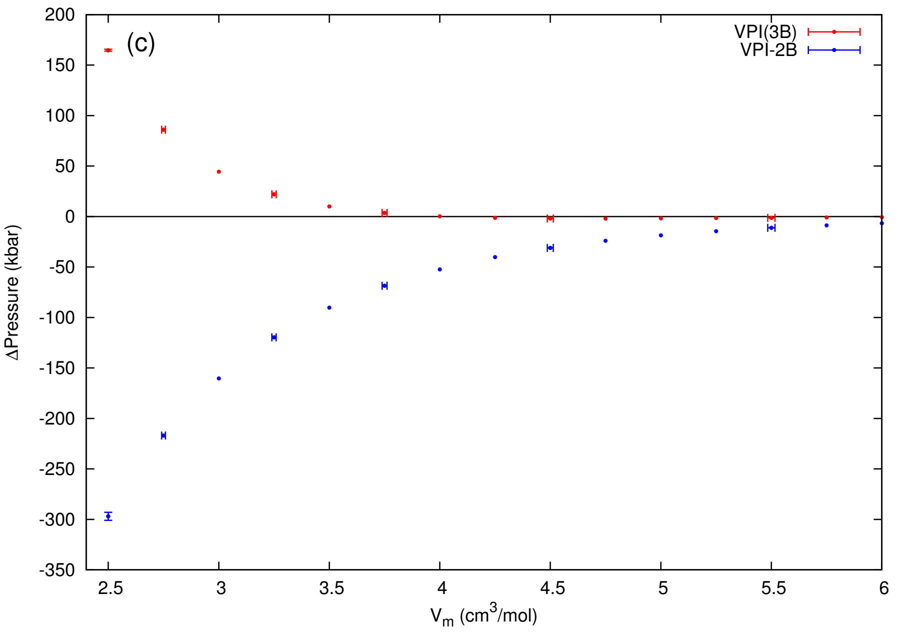


Figure 4.7: *Continued.*

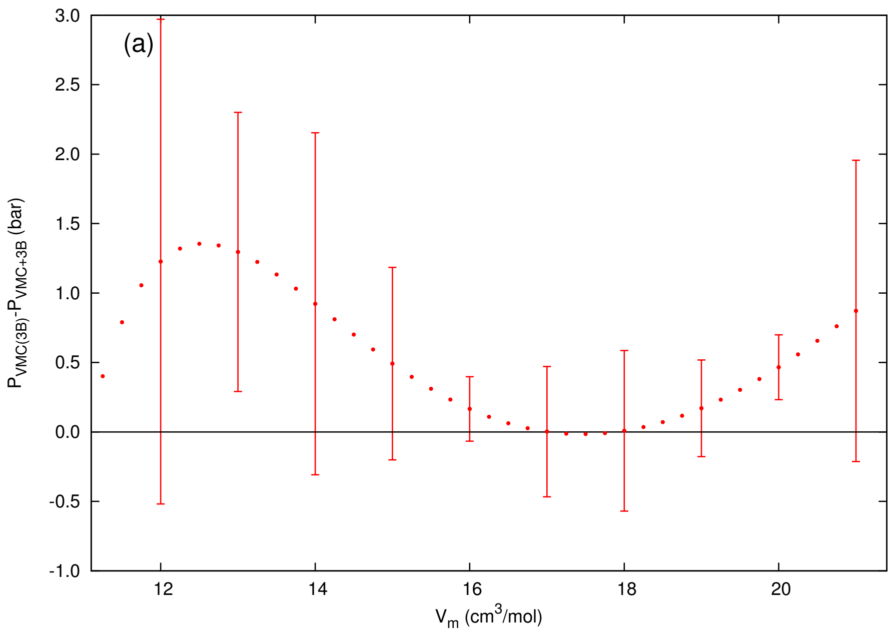
than the experimental values, though still in good agreement. Although these differences in behavior appear most pronounced in the middle and high density regions, the significant difference in curvature of the two trends in Fig. 4.7a suggests that properties such as the bulk compressibility, which depends on the derivative of pressure with respect to volume, will change significantly when three-body interactions are incorporated, even at relatively low densities. Therefore, when these properties are of interest, three-body interactions should be accounted for in some manner at all densities.

Fig. 4.8 assesses the relative accuracy of the perturbative treatment via comparison to the full-incorporation EOS. Here, both of the VMC three-body equations of state are used to allow for a more direct comparison of the different three-body treatments. Vertical error bars have been calculated in the same manner as before, and in the majority of cases the pressures predicted from the VMC perturbative and full-incorporation treatments agree within their statistical uncertainties. Overall, the difference between the two treatments is at least an order of magnitude less than the error compared to experimental data[24], indicating that the more computationally efficient perturbative treatment does not result in a significant loss of accuracy over the full-incorporation method in the molar volume range studied here.

4.4 Conclusion

In the above study we have demonstrated that the addition of the Cencek *et al.* three-body ^{4}He potential[10] into QMC simulations of high pressure hcp solid ^{4}He improves agreement with experimental energies, and results in more reliable equations of state from both VMC and VPI simulations than those obtained from corresponding two-body simulations at molar volumes from 21.3 cm^3/mol down to 4.0 cm^3/mol. Even at low densities where three-body interactions make a relatively small contribution the total energy, the effects of three-body interactions on the EOS are non-negligible.

Figure 4.8: Difference in predicted pressure from VMC(3B) and VMC+3B equations of state vs. V_m in the (a) low, (b) middle, and (c) high density regions. Uncertainties in the pressure are shown at selected molar volumes. *Continued on next page.*



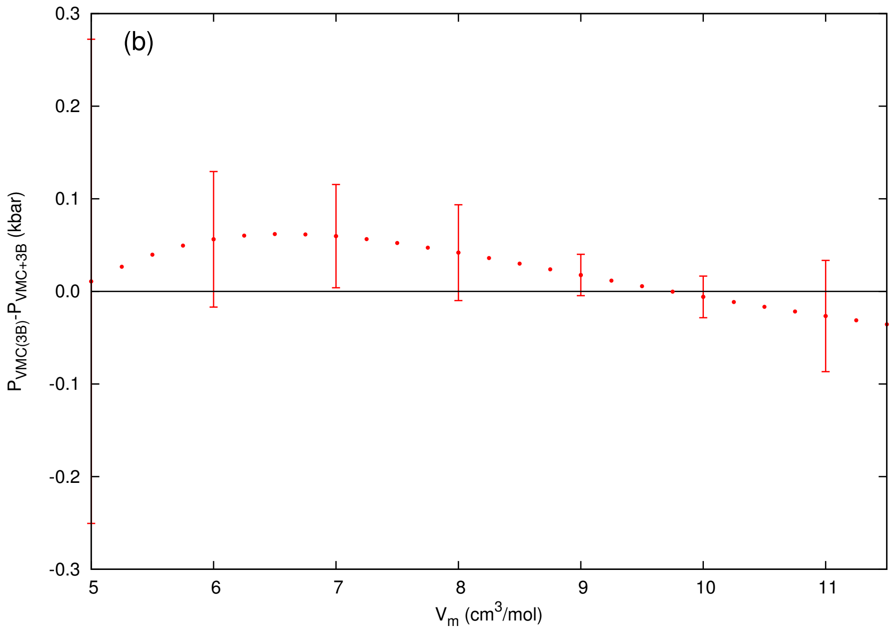


Figure 4.8: *Continued.*

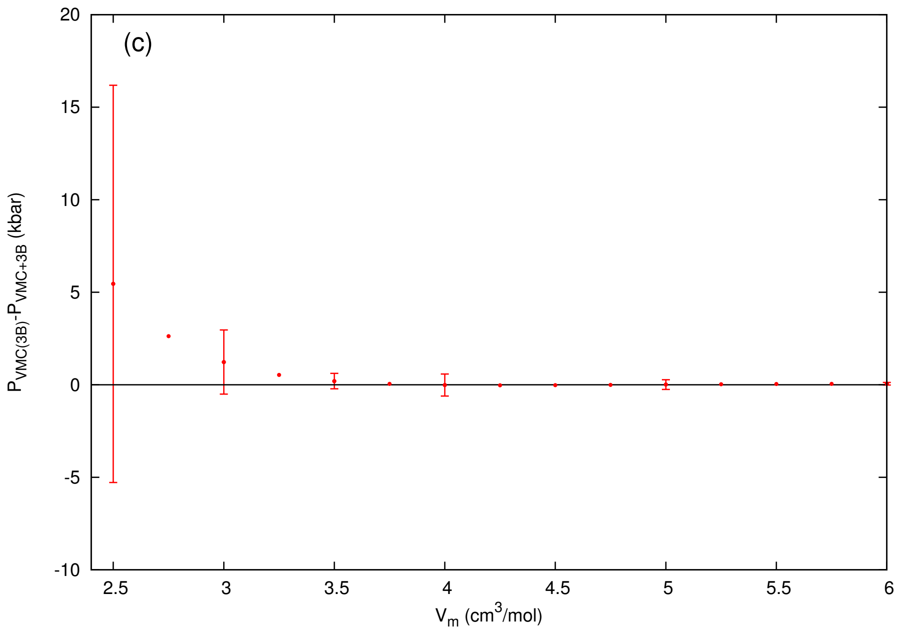


Figure 4.8: *Continued.*

Although the absolute errors in the predicted pressures are similar at low densities for simulations with and without three-body interactions, the difference in curvature of the pressure-volume relationships can affect the elastic properties of the system. For example, bulk compressibilities K calculated from the VPI-2B or VPI(3B) EOS (Table 4.7) differ by 3-5% in the molar volume range from $V_m = 16.22$ cm^3/mol to 20.78 cm^3/mol. At lower molar volumes the calculated K values are shown to differ by as much as 17.9% at $V_m = 7.88$ cm^3/mol. This, however, does not indicate the need for a computationally expensive incorporation of the Cencek potential throughout the simulations. From Fig. 4.8, we can conclude that the more efficient perturbative implementation is sufficient to accurately describe the ^{4}He system in the studied density range without significantly increasing computational cost.

Overall, the proposed perturbative treatment of three-body interactions utilizing the Cencek *et al.* three-body potential[10] has been shown to provide an efficient and reliable description of the ground state properties of hcp solid ^{4}He at lower molar volumes than can be described by two-body models. The equations of state derived here from three-body simulations are in closer agreement with experimental data than either equations of state based on two-body simulations or the DFT-corrected EOS reported by Cazorla and Boronat[5], except at the highest densities. The better agreement from the Cazorla and Boronat EOS at these densities indicates that at molar volumes below 4.0 cm^3/mol, higher many-body interactions are becoming important, and therefore we would need to go beyond the three-body model in order to accurately describe the hcp ^{4}He system at higher densities.

Table 4.7: Bulk compressibility K (bar) calculated from either the VPI-2B or VPI(3B) EOS at selected molar volumes.

V_m (cm^3/mol)	VPI-2B	VPI(3B)	% Difference
20.78	310.3	294.9	5.089
19.36	457.7	442.3	3.422
18.13	668.0	646.7	3.240
16.22	1264	1210	4.365
13.73	3106	2907	6.619
11.90	6321	5812	8.390
9.69	16010	14604	9.185
7.88	43514	36360	17.910

4.5 References

[1] D. M. Ceperley, *Rev. Mod. Phys.* **67**, 279 (1995). 84, 87, 90

[2] S. Ujevic and S. A. Vitiello, *Journal of Physics: Condensed Matter* **19**, 116212 (2007). 85

[3] M. J. Cohen and J. N. Murrell, *Chemical Physics Letters* **260**, 371 (1996). 85

[4] S.-Y. Chang and M. Boninsegni, *The Journal of Chemical Physics* **115**, 2629 (2001). 85

[5] C. Cazorla and J. Boronat, *Journal of Physics: Condensed Matter* **20**, 015223 (2008). 85, 97, 99, 107, 109, 120

[6] S. Grimme, *Journal of Computational Chemistry* **27**, 1787 (2006). 85

[7] L. W. Bruch and I. J. McGee, *The Journal of Chemical Physics* **59**, 409 (1973). 85

[8] C. Cazorla and J. Boronat, *Phys. Rev. B* **92**, 224113 (2015). 86

[9] R. A. Aziz, F. R. McCourt, and C. C. Wong, *Molecular Physics* **61**, 1487 (1987). 86, 88, 98

[10] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 86, 87, 92, 98, 111, 112, 116, 120

[11] G. Garberoglio and A. H. Harvey, *Journal of Research of the National Institute of Standards and Technology* **114**, 249 (2009). 86

[12] G. Garberoglio, M. R. Moldover, and A. H. Harvey, *Journal of research of the National Institute of Standards and Technology* **116**, 729 (2011). 86

[13] G. Garberoglio and A. H. Harvey, *The Journal of chemical physics* **134**, 134106 (2011). 86

[14] K. R. S. Shaul, A. J. Schultz, and D. A. Kofke, *The Journal of Chemical Physics* **137**, 184101 (2012). 86

[15] K. R. Shaul, A. J. Schultz, D. A. Kofke, and M. R. Moldover, *Chemical Physics Letters* **531**, 11 (2012). 86

[16] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **144**, 084505 (2016). 86, 87, 88, 101, 106

[17] J.-P. Hansen and D. Levesque, *Physical Review* **165**, 293 (1968). 87

[18] R. J. Hinde, *Computer Physics Communications* **182**, 2339 (2011). 88, 89, 90

[19] A. Sarsa, K. E. Schmidt, and W. R. Magro, *The Journal of Chemical Physics* **113**, 1366 (2000). 88, 90

[20] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *The Journal of Chemical Physics* **21**, 1087 (1953). 89

[21] H. F. Trotter, *Proceedings of the American Mathematical Society* **10**, 545 (1959). 89

[22] R. J. Hinde, *Chemical Physics Letters* **460**, 141 (2008). 93

[23] T. Williams, C. Kelley, and many others, Gnuplot 4.2: an interactive plotting program, http://gnuplot.sourceforge.net/, 2009. 96

[24] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 96, 107, 109, 110, 116

[25] J. J. Moré, The Levenberg-Marquardt algorithm: Implementation and theory, in *Numerical Analysis*, edited by G. Watson, volume 630 of *Lecture Notes in Mathematics*, pp. 105–116, Springer Berlin Heidelberg, 1978. 96

[26] D. O. Edwards and R. C. Pandorf, *Phys. Rev.* **140**, A816 (1965). 97, 99

4.6 Appendix

Table 4.8: Optimized wavefunction parameters from VMC-2B and VMC+3B simulations.

	VMC-2B			VMC+3B			% Difference		
V_m (cm^3/mol)	a_{xy} (a_o^{-2})	a_z (a_o^{-2})	b (a_o)	a_{xy} (a_o^{-2})	a_z (a_o^{-2})	b (a_o)	a_{xy} (%)	a_z (%)	b (%)
21.30	0.1481	0.1476	5.42202	0.1502	0.1494	5.40018	1.408	1.212	0.404
20.78	0.1555	0.1549	5.41794	0.1578	0.1572	5.39434	1.468	1.474	0.437
20.29	0.1630	0.1624	5.41374	0.1655	0.1650	5.38778	1.522	1.588	0.481
19.81	0.1706	0.1701	5.40969	0.1736	0.1730	5.38161	1.743	1.690	0.520
19.36	0.1785	0.1779	5.40542	0.1818	0.1811	5.37616	1.832	1.783	0.543
18.93	0.1864	0.1857	5.40214	0.1901	0.1894	5.36994	1.965	1.973	0.598
18.52	0.1945	0.1937	5.39881	0.1987	0.1981	5.36448	2.136	2.246	0.638
18.13	0.2026	0.2021	5.39492	0.2075	0.2072	5.35875	2.390	2.492	0.673
17.75	0.2114	0.2109	5.39169	0.2164	0.2157	5.35342	2.338	2.250	0.712
16.22	0.2595	0.2589	5.33359	0.2625	0.2621	5.31010	1.149	1.228	0.441
14.87	0.3090	0.3082	5.31036	0.3130	0.3125	5.28218	1.286	1.386	0.532
13.73	0.3620	0.3617	5.28892	0.3672	0.3667	5.25619	1.426	1.373	0.621
12.75	0.4186	0.4180	5.26881	0.4252	0.4247	5.23152	1.564	1.590	0.710
11.90	0.4791	0.4786	5.24982	0.4870	0.4862	5.20764	1.635	1.575	0.807
11.02	0.5556	0.5548	5.22638	0.5522	0.5516	5.21126	0.614	0.578	0.290
9.69	0.7046	0.7040	5.19194	0.6994	0.6986	5.17165	0.741	0.770	0.392
8.53	0.8824	0.8819	5.15733	0.8739	0.8733	5.13124	0.968	0.980	0.507
7.88	1.0100	1.0090	5.13252	0.9985	0.9972	5.10224	1.145	1.176	0.592
6.00	1.6841	1.6821	4.93175	1.6823	1.6788	4.86071	0.107	0.196	1.451
5.00	2.2082	2.2057	4.83922	2.2121	2.2068	4.74770	0.176	0.050	1.909
4.00	2.9683	2.9668	4.71054	3.0068	2.9986	4.59011	1.289	1.066	2.590
3.00	4.1245	4.1228	4.51362	4.2840	4.2747	4.35534	3.794	3.618	3.569
2.50	5.4508	5.4476	4.27625	5.5501	5.5329	4.13400	1.805	1.554	3.383

Table 4.9: Summary of energies calculated from VMC and VPI simulations. Uncertainties represent the 95% confidence interval.

V_m (cm^3/mol)	$E_{\text{VMC-2B}}$ (K/atom)	$E_{\text{VMC(3B)}}$ (K/atom)	$E_{\text{VMC+3B}}$ (K/atom)	$E_{\text{VPI-2B}}$ (K/atom)	$E_{\text{VPI(3B)}}$ (K/atom)	$E_{\text{VPI+3B}}$ (K/atom)
21.30	-5.255 $\pm$ 0.001	-5.135 $\pm$ 0.001	-5.142 $\pm$ 0.001	-6.264 $\pm$ 0.032	-6.161 $\pm$ 0.032	
20.78	-5.102 $\pm$ 0.001	-4.979 $\pm$ 0.001	-4.988 $\pm$ 0.001	-6.134 $\pm$ 0.035	-6.032 $\pm$ 0.035	
20.29	-4.913 $\pm$ 0.001	-4.790 $\pm$ 0.001	-4.799 $\pm$ 0.001	-5.951 $\pm$ 0.035	-5.849 $\pm$ 0.035	
19.81	-4.685 $\pm$ 0.001	-4.561 $\pm$ 0.001	-4.574 $\pm$ 0.001	-5.734 $\pm$ 0.035	-5.634 $\pm$ 0.035	
19.36	-4.416 $\pm$ 0.001	-4.294 $\pm$ 0.001	-4.309 $\pm$ 0.001	-5.476 $\pm$ 0.036	-5.379 $\pm$ 0.036	
18.93	-4.106 $\pm$ 0.001	-3.985 $\pm$ 0.001	-4.003 $\pm$ 0.001	-5.172 $\pm$ 0.037	-5.079 $\pm$ 0.037	
18.52	-3.750 $\pm$ 0.001	-3.633 $\pm$ 0.001	-3.654 $\pm$ 0.001	-4.823 $\pm$ 0.037	-4.737 $\pm$ 0.037	
18.13	-3.348 $\pm$ 0.001	-3.236 $\pm$ 0.002	-3.259 $\pm$ 0.001	-4.435 $\pm$ 0.038	-4.356 $\pm$ 0.038	
17.75	-2.897 $\pm$ 0.001	-2.791 $\pm$ 0.002	-2.818 $\pm$ 0.001	-3.992 $\pm$ 0.039	-3.923 $\pm$ 0.039	
16.22	-0.214 $\pm$ 0.001	-0.174 $\pm$ 0.002	-0.187 $\pm$ 0.001	-1.351 $\pm$ 0.040	-1.354 $\pm$ 0.040	
14.87	4.035 $\pm$ 0.001	3.941 $\pm$ 0.003	3.920 $\pm$ 0.001	2.839 $\pm$ 0.045	2.689 $\pm$ 0.045	
13.73	9.980 $\pm$ 0.001	9.657 $\pm$ 0.004	9.621 $\pm$ 0.001	8.712 $\pm$ 0.047	8.312 $\pm$ 0.048	
12.75	17.906 $\pm$ 0.001	17.214 $\pm$ 0.005	17.160 $\pm$ 0.001	16.564 $\pm$ 0.051	15.774 $\pm$ 0.052	
11.90	28.100 $\pm$ 0.001	26.848 $\pm$ 0.007	26.778 $\pm$ 0.001	26.670 $\pm$ 0.055	25.302 $\pm$ 0.056	
11.02	43.688 $\pm$ 0.001	41.458 $\pm$ 0.009	41.397 $\pm$ 0.001	42.139 $\pm$ 0.086	39.754 $\pm$ 0.089	39.701 $\pm$ 0.100
9.69	84.105 $\pm$ 0.001	78.741 $\pm$ 0.015	78.634 $\pm$ 0.001	82.291 $\pm$ 0.107	76.699 $\pm$ 0.109	
8.53	150.887 $\pm$ 0.002	139.033 $\pm$ 0.025	138.826 $\pm$ 0.002	148.678 $\pm$ 0.132	136.459 $\pm$ 0.136	
7.88	211.024 $\pm$ 0.002	192.258 $\pm$ 0.034	191.942 $\pm$ 0.002	208.586 $\pm$ 0.131	189.338 $\pm$ 0.135	189.137 $\pm$ 0.117
6.00	616.937 $\pm$ 0.002	533.159 $\pm$ 0.100	532.345 $\pm$ 0.002	613.489 $\pm$ 0.135	529.165 $\pm$ 0.153	528.619 $\pm$ 0.178
5.00	1183.929 $\pm$ 0.003	976.864 $\pm$ 0.196	974.933 $\pm$ 0.003	1179.240 $\pm$ 0.183	971.176 $\pm$ 0.219	
4.00	2477.490 $\pm$ 0.004	1900.717 $\pm$ 0.427	1896.283 $\pm$ 0.005	2470.558 $\pm$ 0.254	1892.415 $\pm$ 0.333	1887.88 $\pm$ 0.303
3.00	5888.382 $\pm$ 0.007	3986.885 $\pm$ 1.072	3975.837 $\pm$ 0.009	5877.281 $\pm$ 0.413	3974.841 $\pm$ 0.577	
2.50	9733.440 $\pm$ 0.007	5949.746 $\pm$ 1.852	5935.463 $\pm$ 0.011	9718.597 $\pm$ 0.428	5941.117 $\pm$ 0.714	

Table 4.10: Summary of three-body energies calculated from VMC(3B), VMC+3B, VPI(3B), and VPI+3B simulations. Uncertainties represent the 95% confidence interval.

V_m (cm^3/mol)	$V_{3,\mathrm{VMC(3B)}}$ (K/atom)	$V_{3,\mathrm{VMC+3B}}$ (K/atom)	$V_{3,\mathrm{VPI(3B)}}$ (K/atom)	$V_{3,\mathrm{VPI+3B}}$ (K/atom)
21.30	0.1203 ± 0.0009	0.1192 ± 0.0001	0.1027 ± 0.0003	
20.78	0.1227 ± 0.0009	0.1209 ± 0.0001	0.1029 ± 0.0004	
20.29	0.1229 ± 0.0010	0.1215 ± 0.0001	0.1021 ± 0.0004	
19.81	0.1240 ± 0.0010	0.1212 ± 0.0001	0.1002 ± 0.0004	
19.36	0.1219 ± 0.0011	0.1197 ± 0.0001	0.0971 ± 0.0005	
18.93	0.1209 ± 0.0012	0.1169 ± 0.0001	0.0926 ± 0.0005	
18.52	0.1166 ± 0.0013	0.1126 ± 0.0001	0.0866 ± 0.0005	
18.13	0.1123 ± 0.0014	0.1068 ± 0.0001	0.0789 ± 0.0006	
17.75	0.1057 ± 0.0015	0.0990 ± 0.0001	0.0692 ± 0.0006	
16.22	0.0406 ± 0.0020	0.0332 ± 0.0001	-0.0035 ± 0.0008	
14.87	-0.0936 ± 0.0028	-0.1040 ± 0.0001	-0.1499 ± 0.0011	
13.73	-0.3226 ± 0.0038	-0.3433 ± 0.0001	-0.3995 ± 0.0014	
12.75	-0.6926 ± 0.0050	-0.7240 ± 0.0001	-0.7897 ± 0.0019	
11.90	-1.2515 ± 0.0065	-1.2925 ± 0.0001	-1.3684 ± 0.0024	
11.02	-2.2295 ± 0.0088	-2.2781 ± 0.0002	-2.3840 ± 0.0137	-2.4391 ± 0.0159
9.69	-5.3636 ± 0.0147	-5.4443 ± 0.0002	-5.5900 ± 0.0048	
8.53	-11.8542 ± 0.0246	-12.0114 ± 0.0004	-12.2186 ± 0.0337	
7.88	-18.7659 ± 0.0336	-19.0148 ± 0.0005	-19.2482 ± 0.0096	-19.5856 ± 0.0104
6.00	-83.7784 ± 0.0998	-84.7215 ± 0.0013	-84.3239 ± 0.0228	-85.5579 ± 0.0254
5.00	-207.065 ± 0.196	-209.265 ± 0.002	-208.064 ± 0.038	
4.00	-576.773 ± 0.427	-581.699 ± 0.004	-578.143 ± 0.068	-584.905 ± 0.0796
3.00	-1901.50 ± 1.07	-1913.43 ± 0.01	-1902.44 ± 0.13	
2.50	-3783.69 ± 1.85	-3802.16 ± 0.01	-3777.48 ± 0.18	

Chapter 5

Three-Body Interactions and the Elastic Constants of hcp Solid ^{4}He

Abstract

The effect of three-body interactions on the elastic properties of hexagonal close packed (hcp) solid ^{4}He is investigated using variational path integral Monte Carlo (VPI) simulations. The nonzero elastic constants are calculated, at $T = 0$ K and for a range of molar volumes from 7.88 cm^3/mol to 20.78 cm^3/mol, from the bulk modulus and the three pure shear constants C_0, C_{66}, and C_{44}. Three-body interactions are accounted for using our recently reported perturbative treatment with the Cencek nonadditive three-body potential. Previous studies have attempted to account for the effect of three-body interactions on the elastic properties of solid ^{4}He; however, these calculations have treated zero point motions using either the Einstein or Debye approximations, which are insufficient in the molar volume range where solid ^{4}He is characterized as a quantum solid. Our VPI calculations allow for a more accurate treatment of the zero point motions which include atomic correlation. From these calculations we find that agreement with the experimental bulk modulus is significantly improved when three-body interactions are considered. In addition, three-body interactions result in non-negligible differences in the calculated pure shear constants and nonzero elastic constants, particularly at higher densities, where differences of up to 26.5% are observed when three-body interactions are included. We compare to available experimental data and find that our results are generally in as good or better agreement with experiment as previous theoretical investigations.

5.1 Introduction

For many decades, solid ^{4}He has been a material of interest in the quantum chemical community due to the highly quantum nature of the ^{4}He atoms. The lightness of the ^{4}He atoms results in large zero point motions which dominate the material's low-temperature properties. These zero point motions are responsible for the persistence of the ^{4}He liquid phase at absolute zero and atmospheric pressure, and for an expanded yet highly compressible lattice in the hexagonal close packed (hcp) solid phase[1] which exists at pressures above 25.2 bar[2]. It follows that accurate treatment of these zero point motions is necessary in order to predict reliable elastic properties of this system. Previous theoretical investigations have employed quantum Monte Carlo (QMC) methods such as variational QMC (VMC)[3, 4] and diffusion QMC (DMC)[5] in order to calculate the elastic constants at $T = 0$ K. In each of these studies the potential energy was assumed to be pairwise additive. However, recently we have shown that three-body interactions play an important role in the zero-temperature equation of state (EOS), which can in turn greatly affect the compressibility and elastic constants of the system[6]. Although these effects were most dramatic at high densities, the curvature of the EOS was noticeably different in the low density region for simulations with and without three-body interactions. Properties which depend on the derivative of pressure with respect to volume, such as the bulk modulus, are therefore influenced by three-body interactions even at low densities.

In light of the demonstrated influence of three-body interactions on the system's response to isotropic compression, it is of interest to determine what effect three-body interactions may have when other types of stress are applied. Recently, a DMC investigation was reported by Cazorla and Boronat[7] in which effective three-body potentials parameterized using density functional theory (DFT) were fully incorporated into the DMC simulations. These simulations were used to calculate the zero temperature EOS and the bulk modulus, among other properties. Although incorporation of the three-body potentials improved agreement with experimental data compared to two-body simulations, the calculated properties varied depending on whether the potential was parameterized from atomic forces or energies, with different potentials performing better for different properties. The same effective three-body potentials were also used to calculate the classical shear modulus (C_{44}) of the system in the absence of zero point motion. The authors compared the classical C_{44} values to quantum values calculated from the DMC simulations (though the quantum C_{44} values were not reported) and found that at a molar volume of approximately 4.5

cm^3/mol this difference amounted to about 2 GPa, or roughly 5% of the reported classical value. However, at higher densities the crystal becomes more classical, and therefore it is expected that a classical approach would result in larger errors at lower densities.

In the same year, Grechnev and coworkers published a study in which semiempirical calculations utilizing a reparameterized version of the three-body potential from Bruch and McGee[8] were used to derive the elastic constants at pressures as high as 100 GPa[9]. This study, however, treated the zero point motions within the Debye approximation which has been shown to be unreliable in the modeling of solid ^{4}He[10]. Similar studies by the same group calculated the theoretical sound velocities, as well as the elastic response of hcp ^{4}He to anisotropic compression, using the same reparameterized three-body potential while treating zero point motion within the Einstein model[11, 12]. The results of these investigations suggested that three-body interactions do substantially impact the elastic properties of solid ^{4}He beyond what is predicted when only two-body interactions are considered. However, to truly quantify this effect at lower densities where ^{4}He behaves as a quantum crystal, a more accurate treatment of the three-body interactions which dominate the crystal's properties must be implemented.

It is our aim to calculate the elastic constants using methods which accurately treat the zero point motions as well as three-body interactions. In our previous investigation[6] we demonstrated that accurate ground state properties can be calculated using variational path integral Monte Carlo (VPI)[2] two-body energies with a perturbative three-body correction calculated from the nonadditive three-body potential reported by Cencek, *et al.*[13]. We will therefore utilize this approach in order to efficiently incorporate three-body interactions into calculations of the elastic properties of hcp solid ^{4}He while accurately treating zero point motion. This investigation constitutes the first implementation of the Cencek three-body potential in the calculation of the elastic properties of hcp ^{4}He.

In the following section we review the relationships between the bulk modulus and nonzero elastic constants in hcp solid ^{4}He as well as the QMC simulation methods employed in this study. Next, the bulk moduli, pure shear constants, and elastic constants calculated with and without three-body contributions are evaluated against one another as well as against previous experimental and theoretical calculations.

5.2 Computational Methods

5.2.1 Definition of the Elastic Constants for hcp ^{4}He

The zero-temperature elastic constants of hcp ^{4}He were calculated following the method prescribed by Cazorla and Boronat[5]. For an hcp crystal, there are five nonzero elastic constants: C_{11}, C_{12}, C_{13}, C_{33}, and C_{44} (the shear modulus). Each of the nonzero elastic constants can be determined through calculation of the pure shear constants (C_0, C_{66}, and C_{44}) along with the bulk modulus, K, and the c/a ratio dependence on molar volume. Previous experimental and theoretical investigations have concluded that the optimal c/a ratio in hcp ^{4}He is independent of molar volume and very close to the ideal value of $c/a = 1.633$ for an hcp lattice[5, 11, 14]. Therefore, $\left(\frac{\delta \ln c/a}{\delta V}\right)_{V=V_0} = \frac{C_{33}-C_{11}-C_{12}+C_{13}}{C_0} = 0$, where here and elsewhere V_0 refers to the equilibrium geometry of the system with a given molar volume. Taking advantage of previously determined relationships between the elastic constants in an hcp lattice[5, 14, 15], the remaining nonzero elastic constants can be calculated according to Eqs. 5.1-5.4[5].

$$C_{11} = K + C_{66} + \frac{1}{18}C_0 \tag{5.1}$$

$$C_{12} = K - C_{66} + \frac{1}{18}C_0 \tag{5.2}$$

$$C_{13} = K - \frac{1}{9}C_0 \tag{5.3}$$

$$C_{33} = K + \frac{2}{9}C_0 \tag{5.4}$$

K is calculated in a straightforward manner from the previously reported pressure-volume EOS[6] following Eq. 5.5:

$$K = -V\left(\frac{\delta P}{\delta V}\right)_{V=V_0} = \frac{C_{33}(C_{11}+C_{12}) - 2C_{13}^2}{C_{11}+C_{12}+2C_{33}-4C_{13}}. \tag{5.5}$$

The three pure shear constants quantify the response of the hcp ^{4}He crystal to changes in the c/a ratio (C_0), the angle between the x- and y-axes in the basal plane (C_{66}), and the angle between the z-axis and the basal plane (C_{44}). They are calculated from the second derivatives of the internal energy with respect to the deformation parameters η, γ, and ϵ (also refered to as heterogeneous strain variables[5]) applied to the primitive lattice vectors of the crystal, respectively. In the undistorted crystal, the primitive lattice vectors are given by

$$\begin{aligned} \mathbf{a}_1 &= a(\tfrac{\sqrt{3}}{2}\mathbf{i} + \tfrac{1}{2}\mathbf{j}) \\ \mathbf{a}_2 &= a(\tfrac{\sqrt{3}}{2}\mathbf{i} - \tfrac{1}{2}\mathbf{j}) \\ \mathbf{a}_3 &= c\mathbf{k}. \end{aligned} \tag{5.6}$$

where a and c are the hcp lattice parameters. We note that our primitive lattice vectors differ from those reported by Cazorla and Boronat due to differences in the orientation of our unit cell. These vectors can be generalized to the form shown below:

$$\begin{aligned} \mathbf{a}_1 &= a\phi^{-1}\gamma^{1/2}(\gamma^{-1}\tfrac{\sqrt{3}}{2}\mathbf{i} + \tfrac{1}{2}\mathbf{j} + \tfrac{\epsilon}{2}\mathbf{k}) \\ \mathbf{a}_2 &= a\phi^{-1}\gamma^{1/2}(\gamma^{-1}\tfrac{\sqrt{3}}{2}\mathbf{i} - \tfrac{1}{2}\mathbf{j} + \tfrac{\epsilon}{2}\mathbf{k}) \\ \mathbf{a}_3 &= c\phi^2\mathbf{k}. \end{aligned} \tag{5.7}$$

where $\phi = \sqrt{(1+\eta)}$. This construction of the primitive lattice vectors maintains constant volume of the unit cell when the deformation parameters are changed. The system is in the equilibrium geometry when $\eta = \epsilon = 0$ and $\gamma = 1$. In the calculation of the pure shears, only one parameter is allowed to vary from equilibrium in a given simulation. The values of the pure shear constants can then be determined according the relationships in Eqs. 5.8-5.10.

$$C_0 = \frac{2}{V_0}(\delta^2 E/\delta\eta^2)_{V=V_0} = C_{11} + C_{12} + 2C_{33} - 4C_{13} \tag{5.8}$$

$$C_{66} = \frac{1}{V_0}(\delta^2 E/\delta\gamma^2)_{V=V_0} = \frac{1}{2}(C_{11} - C_{12}) \tag{5.9}$$

$$C_{44} = \frac{1}{V_0}(\delta^2 E/\delta\epsilon^2)_{V=V_0} \tag{5.10}$$

5.2.2 Energy Calculations in the Distorted Lattices

5.2.2.1 VMC

Approximate ground state wavefunctions for a range of molar volumes are obtained from VMC optimizations following the previously reported procedure[16] in which a Jastrow-McMillan style trial wavefunction (Eq. 5.11 below) is optimized by determining the values of a_{xy}, a_z, and b which minimize the total energy per atom.

$$\Psi = A \prod_i e^{-a_{xy}(s_{i,x}^2 + s_{i,y}^2)} e^{-a_z s_{i,z}^2} \prod_{(i,j)\in \mathrm{IPs}} e^{-\frac{1}{2}(b/R_{ij})^5} \tag{5.11}$$

In the above equation, A is a normalization factor, $\vec{s}_i$ is the displacement vector of atom i from its average lattice site, and R_{ij} is the instantaneous interatomic distance between atoms i and j. IPs in Eq. 5.11 denotes the set of interacting pairs of atoms, defined to be those atoms whose average lattice positions in the ideal lattice are separated by less than $2.05R_{nn}$. Only atomic pairs in the set of IPs are accounted for in the potential energy calculations throughout the VMC simulations. Neighbor lists are constructed from the lattice in its equilibrium geometry before changing η, γ, or ϵ, thereby ensuring that the same interacting pairs are employed in each simulation.

Throughout the VMC simulation, new atomic coordinates are selected for each atom using Metropolis Monte Carlo moves[17] by sampling the probability density of the trial wavefunction. The pairwise-additive potential energy is evaluated for all interacting pairs every 50 Monte Carlo cycles (MCCs) as shown in Eq. 5.12,

$$V = \sum_{(i,j)\in\mathrm{IPs}} V_2(R_{ij}), \tag{5.12}$$

where V_2 is taken to be the Aziz HFD-B(He) pair potential[18]. In these simulations one MCC represents an attempt to move each atom in the lattice once in sequential fashion. The kinetic energy and total energy are also recorded every 50 MCCs, along with snapshots of the atomic positions. We have previously shown that this snapshot interval is useful for eliminating correlation between sequential snapshots, allowing for a simplified calculation of the statistical uncertainties of the average energies. For a sufficiently long simulation, the average energy converges to the expectation value for the trial wavefunction within statistical uncertainty. For this investigation, 3.2x10^7 MCCs are performed, corresponding to 6.4x10^5 snapshots which are then utilized in a reweighting procedure[16] in order to allow for a more precise determination of the optimal wavefunction parameters. This is a smaller number of snapshots than was utilized in the previous study in which both VMC and VPI energies were used to derive pressure-volume equations of state (EOSs)[6], and therefore leads to more uncertainty in the optimized wavefunction parameters. However, for this investigation, the VMC optimized wavefunctions are used solely as starting trial wavefunctions for VPI, which is relatively insensitive to small changes in the trial wavefunction parameters. Therefore we can reduce the number of snapshots used in our VMC optimizations without greatly affecting convergence in the VPI simulations.

Wavefunctions are optimized for hcp ^{4}He systems consisting of 448 atoms with molar volumes ranging from 7.88 cm^3/mol to 20.78 cm^3/mol with periodic boundary conditions applied in all directions. In addition to the equilibrium geometry, wavefunctions are optimized for 4 to 6 different values of η, γ, and ϵ at each molar volume. Optimal parameters from ideal lattices at each molar volume are used to initialize the trial wavefunctions of the distorted systems. We then alternate between optimization of the a_i parameters and optimization of the b parameter until the estimated next change in b is less than 0.1%. Linear interpolation of the optimized a_{xy}, a_z, and b variational parameters as a function of the deformation parameters is also used to obtain approximate optimized wavefunctions for a total of 8 different values of each of the deformation parameters at each density.

5.2.2.2 VPI

The approximate ground state wavefunctions from VMC are used as starting trial wavefunctions for VPI simulations[2] (also referred to as the path integral ground state method, or PIGS[19]). This is an exact method which eliminates error due to the variational principle in our ground state energies. In this method, the hcp ^{4}He system is modeled as a p-bead polymer chain where each bead represents a replica of the full N_{VPI} atom system. Progression down the chain in either direction corresponds to evolution of the trial wavefunction Ψ_{tr} in imaginary time. Specifically, links between the beads in the chain correspond to evolution of the wavefunction through $\Delta\tau$ units of imaginary time via application of the imaginary time propagator, $\exp[-\hat{H}\Delta\tau/\hbar]$[20]. This method relies on the fact that for a given Hamiltonian, $\hat{H}$, any Ψ_{tr} can be written as a linear combination of the eigenfunctions of $\hat{H}$. Given a sufficiently long chain (large p) and small $\Delta\tau$, the imaginary time propagator projects out the lowest energy eigenstate from this linear combination, corresponding to the exact ground state wavefunction Ψ_{gs}, provided that the overlap between Ψ_{tr} and Ψ_{gs} is not negligible.

Our VPI simulations are carried out using the QSATS code[20] which has been modified to use independent a_{xy} and a_z parameters as well as nonequilibrium values of η, γ, and ϵ. This program utilizes Metropolis Monte Carlo moves to generate atomic configurations in the system by sampling the probability density for each bead according to Eq. 5.13,

$$P(C) = A\Psi_{tr}(\mathbf{Q}_1)\Psi_{tr}(\mathbf{Q}_p)exp\Big(-\frac{\Delta\tau}{\hbar}\sum_{j=1}^{p-1}F(\mathbf{Q}_j,\mathbf{Q}_{j+1})\Big), \tag{5.13}$$

where $C = \mathbf{Q}_1, \mathbf{Q}_2, ..., \mathbf{Q}_p$ is the configuration of the entire polymer chain and $\mathbf{Q}_j$ represents the configuration in the jth replica[20]. The function $F(\mathbf{Q}_j, \mathbf{Q}_{j+1})$ in Eq. 5.13 is related to the Trotter factorization[21] of the imaginary time propagator and depends on the potential energy of the N_{VPI} atom system, the mass of each atom, and the displacement of each particle from bead j to $j+1$[20]. Eq. 5.13 indicates that as the number of beads p increases and the imaginary time step $\Delta\tau$ decreases, the probability density of the interior beads converges to $|\Psi_{gs}|^2$ while the terminal beads sample $|\Psi_{tr}\Psi_{gs}|$. From the distribution of the interior beads we can sample the ground state values of any coordinate-space observables. For a more in depth description of this method and the QSATS code, the reader is directed to Refs. [20], [2], and [6].

Due to the greater computational cost of VPI simulations, the lattice is reduced to 180 ^{4}He atoms. A single MCC in these simulations now refers to an attempt to move each of the atoms in each replica once, sequentially. For each attempted move, the QSATS algorithm automatically incorporates all contributions to the F function in Eq. 5.13 except for contributions from the potential energy, V. The move is accepted if the new potential energy is lower than in the previous configuration, otherwise it is conditionally rejected with the probability $e^{-\Delta V \Delta\tau/\hbar}$. The simulations reported here make use of $p = 430$ replicas and a time step $\Delta\tau = 200$ a.u., along with the same Aziz pair potential used above. Earlier investigations of the hcp ^{4}He system using the QSATS code determined that the calculated properties of the system agreed within their statistical uncertainties when 200 a.u. $\leq \Delta\tau \leq$ 500 a.u., and therefore the observables are independent of the time step in this range. In addition, we have previously shown that the use of VMC-optimized wavefunctions in our VPI simulations results in rapid convergence to the ground state wavefunction[6]. The average two-body potential energy was found to converge at most densities after only 23 replicas, corresponding to an imaginary time of 4600 a.u. Our 430 replica (86000 a.u.) simulations are therefore sufficient to achieve convergence to the ground state. Average energies for each replica are calculated from 6000 snapshots which were obtained every 1000 MCCs (after a 1x10^5 MCC warmup) in order to eliminate serial correlation in the configurations. In addition, to reduce correlation between sequential replicas, the average observables are only calculated for every 11th replica.

As in VMC, potential energy calculations from the VPI snapshots consider only those atomic pairs considered to be interacting pairs according to the cutoff criterion defined above. Contributions from atomic pairs separated by greater distances are accounted for in the VPI simulations using a previously reported long-range correction (LRC) procedure[16]. As before, atoms within the 180 ^{4}He atom crystal but beyond the interacting pair cutoff (region 1) are treated using Gaussian quadrature. For these distorted lattices, however, it is necessary to allow for an independent Gaussian distribution along each Cartesian direction. The parameters describing these Gaussian distributions are chosen so that the distributions reproduce the mean squared displacements $\langle u_j^2 \rangle$ along each direction.

The infinite number of atoms beyond the 180 represented in the periodic boundary conditions (region 2) are treated in the same manner as before by subtracting the contributions from the interacting pairs and region 1 atoms from the infinite lattice sum S_6^*/R_{nn}^6, where the value of $S_6^* = \sum_{i=2}^{\infty} \frac{R_{nn}^6}{R_{1i}^6}$ has been previously reported for an ideal hcp lattice[22]. For the distorted hcp lattices in this study, S_6^* is calculated for each value of the deformation parameters according to the procedure described in Ref. [16]. In these simulations, R_{nn} refers to the nearest neighbor distance in the undistorted lattice. Multiplication of the region 2 contribution to S_6^*/R_{nn}^6 by the C_6 parameter of the Aziz potential gives the region 2 LRC.

Previous calculations of elastic constants by Cazorla and Boronat[5] did not include long-range corrections in the calculation of C_0, C_{66}, and C_{44}. This was justified by the fact that the distortion imposed by the heterogeneous strain variables maintained the volume of the unit cell and therefore kept the solid's density constant. They concluded that the long-range correction would be constant and would not affect the second derivative from which the pure shear constants are calculated. We tested this assumption by calculating the potential energy of our system with a molar volume of 7.88 cm^3/mol in the absence of zero point motion using various cutoff distances while changing ϵ. The potential energies calculated from each cutoff distance as well as the potential energy containing the long-range correction are shown in Fig. 5.1. For these calculations, a simulation cell consisting of 7920 ^{4}He atoms was used.

These data are found to be well described by the function $V(\epsilon) = a + b\epsilon^2$. The second derivative of $V(\epsilon)$ for each cutoff distance considered is provided in Table 5.1. Using a cutoff distance of $2.05R_{nn}$ introduces an error in $\delta^2V/\delta\epsilon^2$ of approximately 1.3% when the atoms are localized to their lattice sites. Even at the largest cutoff distance considered, $4.05R_{nn}$, $\delta^2V/\delta\epsilon^2$ differs from the long-range correction value by more than the statistical uncertainty. It is possible that in the presence of zero point motion these differences could have a bigger impact on the calculated pure shear constants, and therefore we include long-range corrections to the potential energy in all of our calculations.

5.2.2.3 Three-Body Interactions

Three-body contributions to the potential energy are calculated as a perturbative correction to the total energy obtained from the two-body VPI simulations following our previously reported method[6]. This correction is calculated by evaluating the nonadditive three-body potential reported by Cencek *et al.*[13] at each recorded VPI snapshot for all trimers formed by an atom and two of its nearest neighbors (referred to as "interacting trimers"). Using this method, trimers composed of three nearest neighbor pairs will appear in the interacting trimer list three times, and therefore care is taken to avoid triple counting. We have previously used this method to calculate the zero-temperature EOS for hcp solid ^{4}He using both VMC and VPI simulations, and found that this perturbative treatment produced EOSs in much closer agreement with the experimental pressure-volume data from Driessen, *et al.*[6, 23]. The results from this perturbative treatment were also in very good agreement with those obtained from fully incorporating the Cencek potential into the many-body potential energy function throughout the simulations[6], and therefore we do not expect any significant

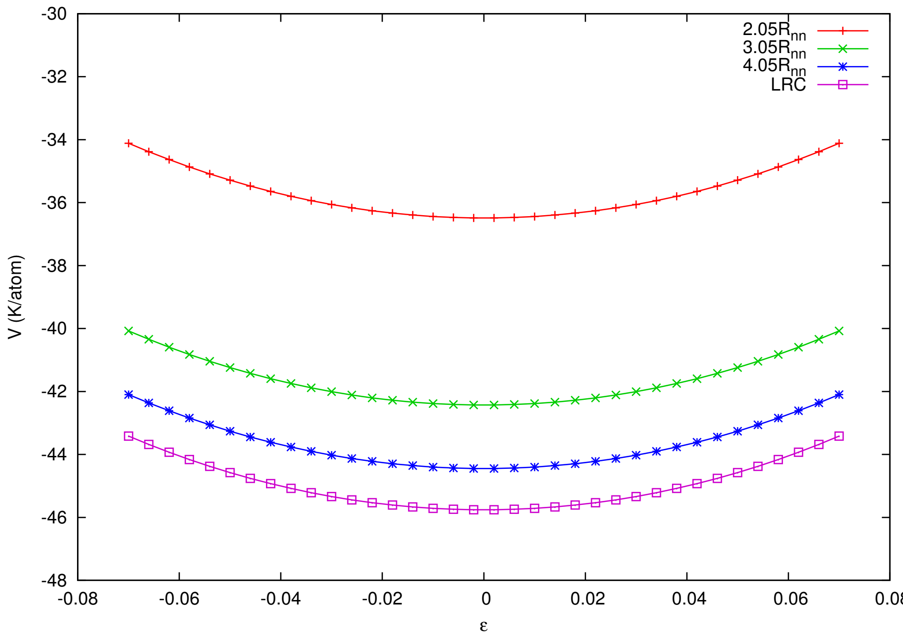


Figure 5.1: Potential energy vs. ϵ considering interacting pairs defined by cutoff distances of $2.05R_{nn}$, $3.05R_{nn}$, and $4.05R_{nn}$ in the absence of zero point motion in a simulation cell of 7920 ^{4}He atoms with a molar volume $V_m = 7.88$ cm^3/mol. The potential energy including the full long-range correction is also shown.

Table 5.1: Second derivative of $V(\epsilon)$ with respect to ϵ at $\epsilon = 0$ in the absence of zero point motion for various interacting pair cutoff distances in a 7920 atom simulation cell with a molar volume $V_m = 7.88$ cm^3/mol. Units are in K/atom.

Cutoff Distance	$\delta^2 V/\delta\epsilon^2$
$2.05 R_{nn}$	966.68 ± 0.42
$3.05 R_{nn}$	958.93 ± 0.42
$4.05 R_{nn}$	955.90 ± 0.42
LRC	954.41 ± 0.42

loss of accuracy in our calculation of the elastic constants using energies calculated from this computationally efficient treatment of three-body interactions. As before, the average three-body correction is found to converge after approximately 55 replicas or 11000 a.u. time in most simulations. However, to ensure that the three-body correction is truly representative of the ground state, only those values from the two innermost replicas (corresponding to replicas 210 and 221 after the 11-replica interval is applied) are used to calculate the final three-body correction.

5.3 Results and Discussion

The following sections compare the elastic properties of the hcp solid ^{4}He system calculated from the VPI simulations described above with and without perturbatively corrected three-body interactions. For simplicity, we will refer to results from simulations without three-body interactions as VPI-2B results, and those with three-body interactions as VPI(3B) results.

5.3.1 The Bulk Modulus

The bulk modulus is calculated from the previously reported VPI-2B and VPI(3B) pressure-volume EOSs[6] according to Eq. 5.5, above. The values of the bulk modulus from both EOSs are reported in Table 5.2, along with the bulk modulus calculated from the experimental EOS reported by Driessen, *et al.*[23].

From Table 5.2, we see that even at the lowest densities investigated, the incorporation of three-body interactions has a significant effect on the bulk modulus. This impact amounts to a 3-5% difference at molar volumes above 16 cm^3/mol, and as much as 17.9% at the lowest molar volume investigated here, 7.88 cm^3/mol. Recalling Eqs. 5.1-5.4, four of the five nonzero elastic constants depend on K and therefore this influence of three-body interactions on the bulk modulus cannot be ignored. In addition, the VPI(3B) bulk moduli are typically in better agreement

Table 5.2: Bulk moduli K (bar) calculated from either the VPI-2B or VPI(3B) EOS at selected molar volumes, along with the corresponding bulk moduli calculated from the Driessen experimental EOS[23].

V_m (cm^3/mol)	VPI-2B bar	VPI(3B) bar	% Difference	Driessen bar
20.78	310.27 ± 8.64	294.93 ± 9.06	5.07	288.4
19.36	457.25 ± 5.01	441.87 ± 5.25	3.42	445.2
18.13	668.60 ± 1.64	647.20 ± 1.71	3.25	652.3
16.22	1262.21 ± 3.17	1208.88 ± 1.71	4.32	1212
13.73	3107.58 ± 2.03	2908.11 ± 1.71	6.63	2938
11.90	6325.28 ± 19.92	5815.84 ± 20.97	8.39	6110
9.69	16028.80 ± 440.97	14620.32 ± 120.39	9.19	16180
7.88	43433.61 ± 409.33	36298.91 ± 101.43	17.90	35424

with the experimental values than the corresponding VPI-2B bulk moduli. The exception to this occurs at molar volumes of 11.90 cm^3/mol and 9.69 cm^3/mol where the VPI-2B results are found to be in better agreement with experiment. There is no physical interpretation which can explain why a two-body model would result in better agreement with experimental data than a three-body model over this small range of molar volumes. Instead, this behavior likely results from the fact that our theoretical EOSs and the experimental EOS of Driessen, *et al.*[23] were parameterized independently in the high and low density regions, with the transition from low density to high density regions occuring at 11.02 cm^3/mol for our theoretical EOSs. Attempts to refit our VPI energy-volume data in the middle density region in order to remove any discontinuity in the EOSs did not result in a significant change in the calculated bulk moduli reported here. In addition, we note that the experimental EOS of Ref. [23] was parameterized to reproduce experimental pressure-volume data extrapolated to $T = 0$ K, rather than experimental bulk moduli. We therefore suggest that the anomalous better agreement of the VPI-2B data with experiment near the transtion molar volume may be due to the piecewise fitting of the experimental EOS or possibly due to the extrapolation of experimental pressure-volume data to absolute zero.

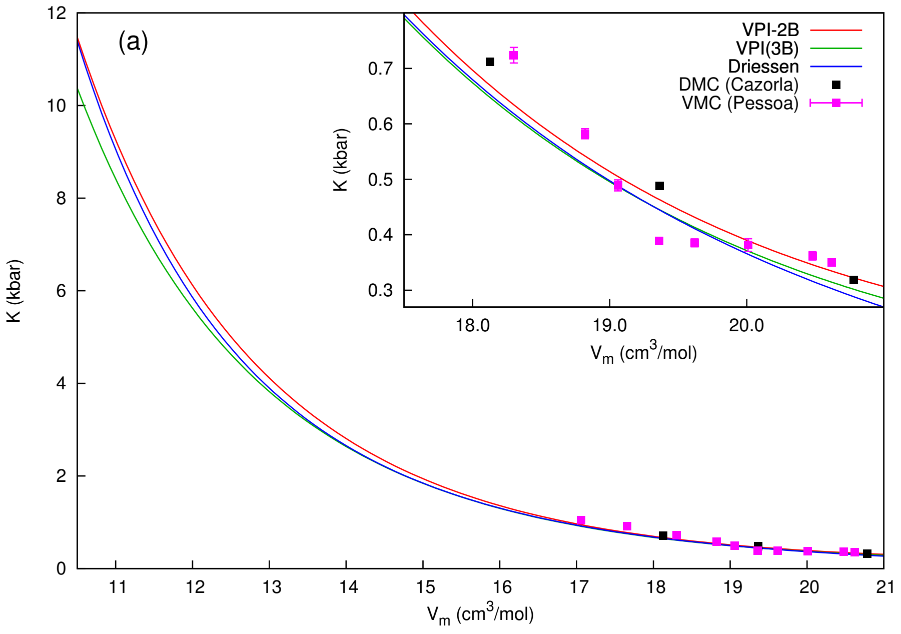
Fig. 5.2 provides a visual comparison of the bulk modulus functions derived from our VPI-2B and VPI(3B) EOSs compared to the Driessen bulk modulus, where we see the experimental bulk modulus diverging from the VPI(3B) results near the transition from the high to low molar volume region. However, at molar volumes below about 9.0 cm^3/mol, we once again see significantly better agreement with experiment from the VPI(3B) results. At higher molar volumes, we also compare the bulk moduli used in this investigation to those calculated from the EOS utilized by Cazorla and Boronat[5, 24], as well as the bulk modulus calculated from the VMC study of Pessoa, *et al.*[4] (Fig. 5.2a inset). In Ref. [4], all of the nonzero elastic constants were calculated independently rather than from the bulk modulus and pure shear constants, and therefore the bulk modulus from Pessoa, *et al.* plotted here was obtained from the reported elastic constants using the relationship in Eq. 5.5[5].

This figure shows that, in general, the bulk moduli used in this investigation from either VPI-2B or VPI(3B) simulations are in closer agreement with the experimental data from Driessen, *et al.*[23] than are the values from either Cazorla and Boronat or Pessoa, *et al.* In addition, the values calculated from the Pessoa VMC results using Eq. 5.5 do not appear to follow a clear relationship with respect to molar volume.

5.3.2 Calculation of Pure Shear Constants

Ground state energies, including long-range corrections, are obtained from VPI simulations using eight non-equilibrium values of each deformation parameter at every molar volume. The range of parameter values used for each molar volume can be found in Table 5.3. Smaller ranges of η and ϵ parameters are used at higher densities because it was determined that imposing greater distortion resulted in VPI calculations which would not converge to a ground state energy, suggesting that these highly distorted configurations are unstable at higher densities.

Fig. 5.3 shows the dependence of the VPI-2B and VPI(3B) energy on each of the three heterogeneous strain variables at a molar volume of 13.73 cm^3/mol. Similar



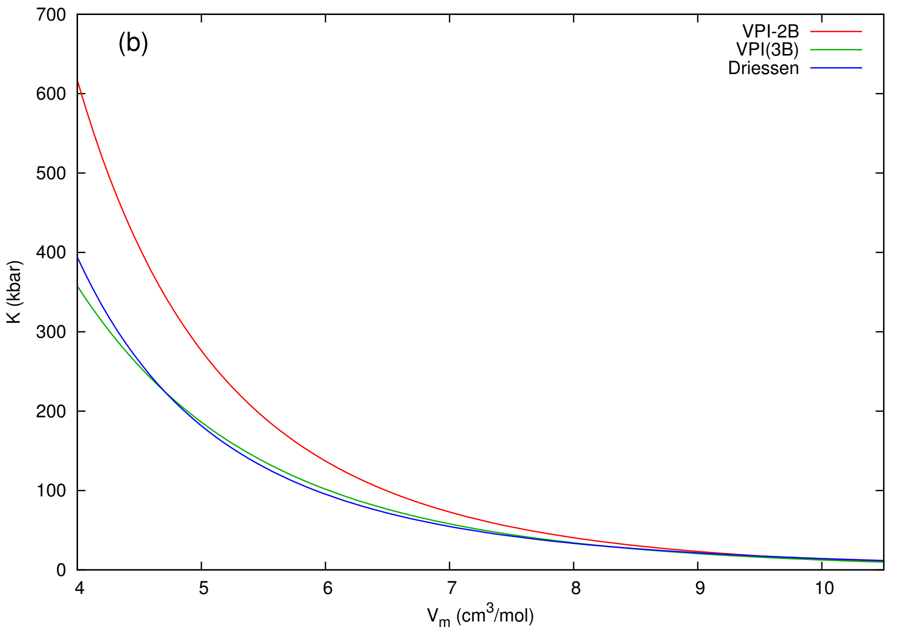
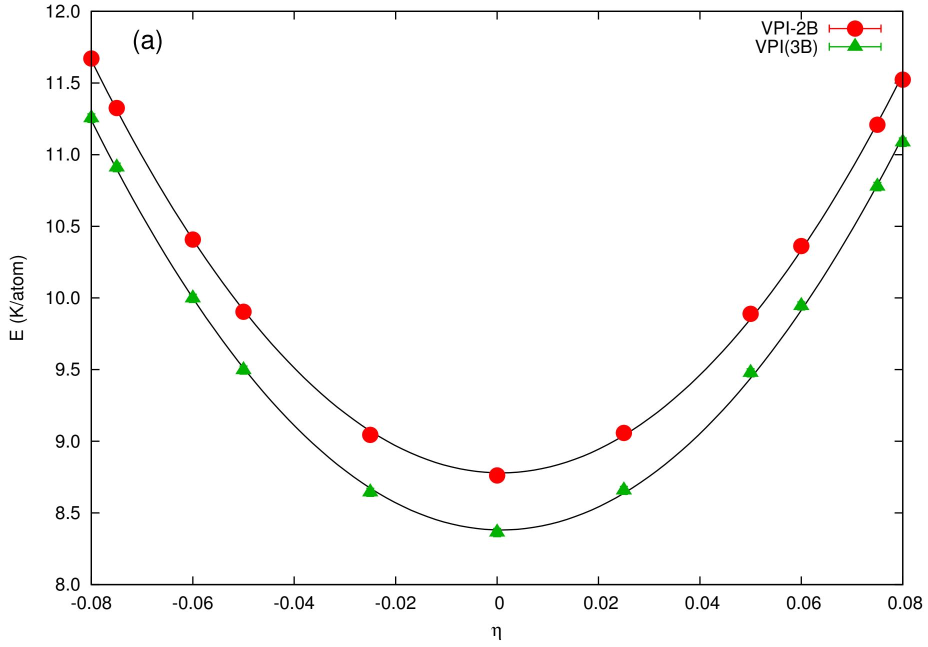


Figure 5.2: Comparison of bulk modulus functions calculated from the VPI-2B (red line) and VPI(3B) (green line) EOSs used in the current study[6] to the bulk modulus function from the experimental EOS reported by Driessen, *et al.*[23] (blue line) in the (a) low denstiy and (b) high density regions. In the low density region, theoretical bulk moduli from the Cazorla and Boronat DMC studies[5] (black squares) and the Pessoa, *et al.*[4] VMC studies (pink squares) are also included.

Table 5.3: Ranges of η, γ, and ϵ values used to calculate the pure shear constants.

V_m (cm^3/mol)	η	γ	ϵ
20.78	-0.15, 0.15	0.90, 1.10	-0.15, 0.15
19.36	-0.15, 0.15	0.90, 1.10	-0.15, 0.15
18.13	-0.10, 0.10	0.90, 1.10	-0.15, 0.15
16.22	-0.10, 0.10	0.90, 1.10	-0.15, 0.15
13.73	-0.08, 0.08	0.90, 1.10	-0.10, 0.10
11.90	-0.08, 0.08	0.90, 1.10	-0.08, 0.08
9.69	-0.065, 0.065	0.90, 1.10	-0.065, 0.065
7.88	-0.065, 0.065	0.90, 1.10	-0.065, 0.065

Figure 5.3: Energy vs. η (a), γ (b), and ϵ (c) at a molar volume of 13.73 cm^3/mol. Results from VPI-2B (red circles) and VPI(3B) (green triangles) are shown along with their best fit equations (solid and dashed black lines, respectively). *Continued on next page.*



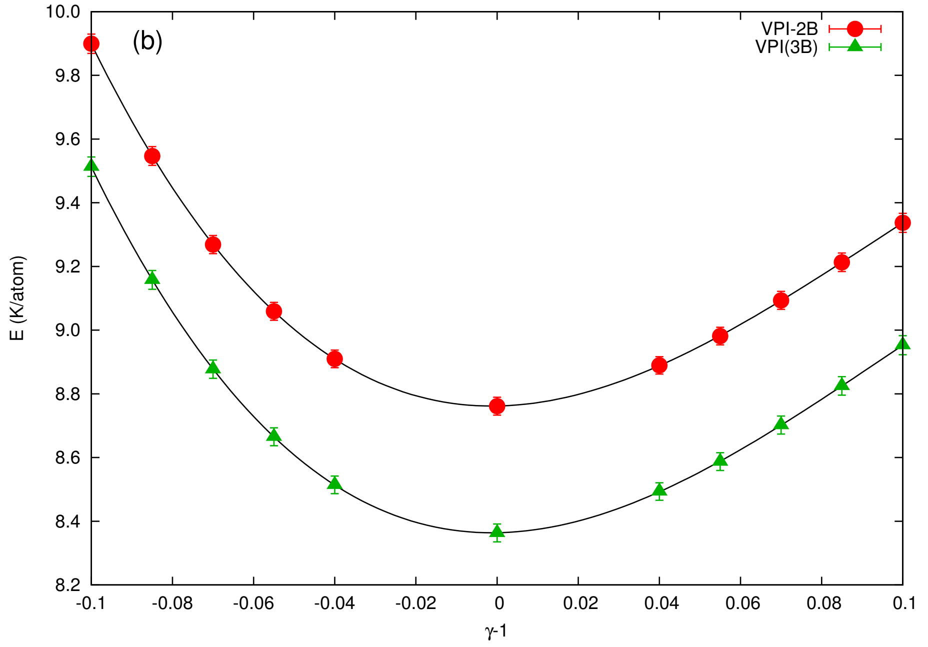


Figure 5.3: *Continued.*

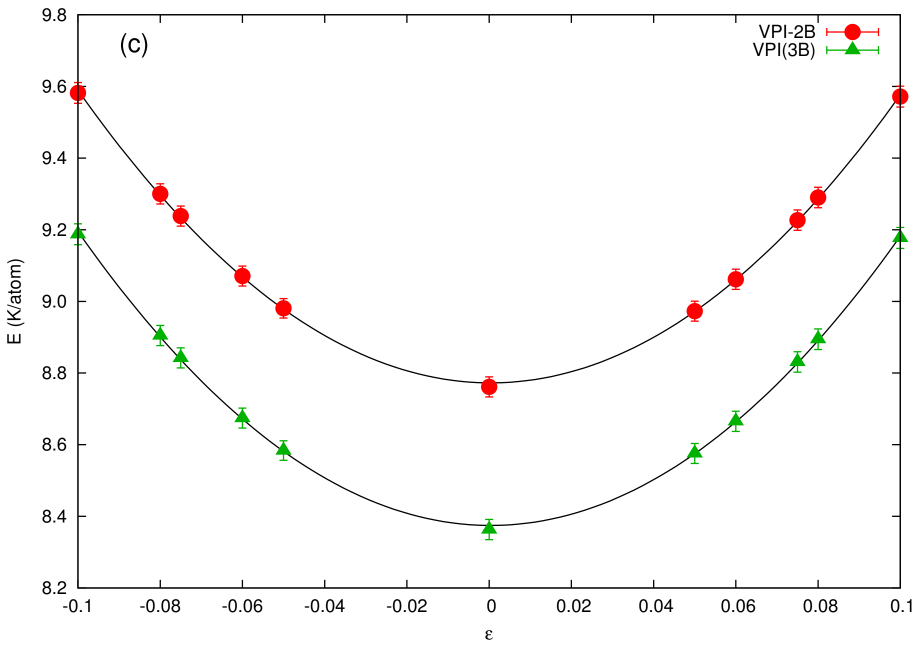


Figure 5.3: *Continued.*

relationships are observed at every molar volume. At molar volumes above 13.73 cm^3/mol, the total energy as a function of η and ϵ is best described by a quartic equation of the form $E(x) = a + bx^2 + cx^3 + dx^4$ where $x = \eta$ or ϵ. However at lower molar volumes, the value of the quartic term is less than the uncertainty in the total energies, and therefore the corresponding fitting parameter cannot be accurately determined. In these cases, a quadratic equation of the form $E(x) = a + bx + cx^2$ is found to fit the data best. The same results are obtained at both high and low densities when the odd-powered terms in $E(x)$ are omitted, however the uncertainty in the fitted parameters and the residuals improve when these terms are included. At all molar volumes, $E(\gamma)$ is best described by the cubic equation $E(\gamma) = a+b\gamma+c\gamma^2+d\gamma^3$.

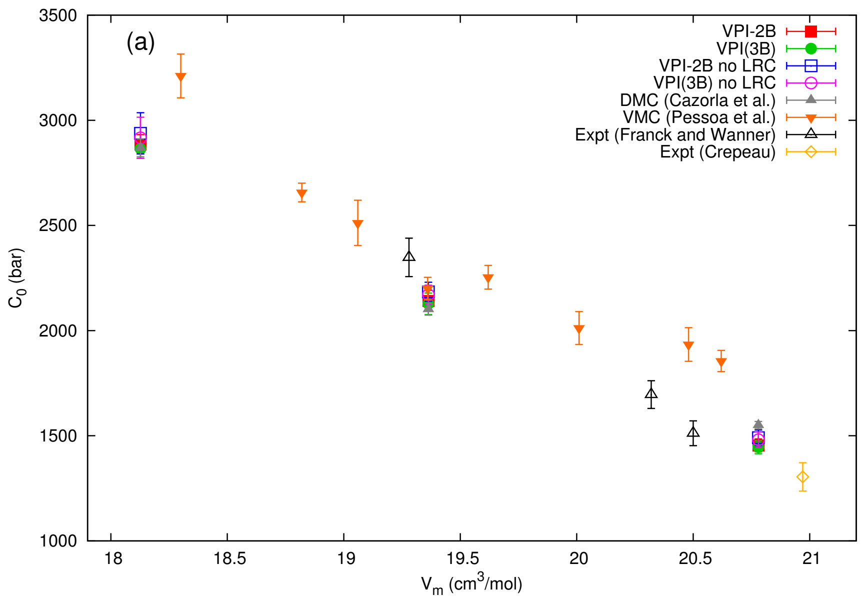
The pure shear constants C_0, C_{66}, and C_{44} are calculated from the second derivatives of these best-fit functions according to Eqs. 5.8-5.10 and are tabulated in Table 5.4. Errors in the pure shear constants are determined from errors in the fitting parameters. In Fig. 5.4 we compare the results obtained from both VPI-2B and VPI(3B) simulations to previous results from Cazorla and Boronat[5] and Pessoa, *et al.*[4]. In addition, experimental data from Franck and Wanner[14], Crepeau[25], and Greywall[26] are included where available.

Fig. 5.4 considers pure shear constants calculated with and without long-range corrections to the two-body potential energy in the low density range where previous theoretical results have been reported. At these low densities, the pure shear constants are not strongly influenced by the three-body interactions. However, it is clear that the incorporation of the long-range corrections has a non-negligible impact on the calculated pure shear constants. In many cases, omitting these long-range corrections brings our results into closer agreement with previous theoretical results, however this is not strictly true for the full set of data. In addition, we find that the values of C_{66} calculated in this study are in much closer agreement with the VMC results from Pessoa, *et al.* and experimental values than the values reported by Cazorla and Boronat. As this pure shear constant is used to calculate two of the four remaining

Table 5.4: Pure shear constants C_0, C_{66}, and C_{44} (bar) calculated from VPI-2B and VPI(3B) energies.

V_m	C_0 (bar)			C_{66} (bar)			C_{44} (bar)		
(cm^3/mol)	VPI-2B	VPI(3B)	% Difference	VPI-2B	VPI(3B)	% Difference	VPI-2B	VPI(3B)	% Difference
20.78	1454.64 $\pm$ 30.37	1445.25 $\pm$ 30.81	0.65	114.54 $\pm$ 0.68	114.93 $\pm$ 0.67	0.34	128.86 $\pm$ 0.70	128.75 $\pm$ 0.70	0.08
19.36	2141.59 $\pm$ 49.78	2127.41 $\pm$ 50.91	0.66	171.36 $\pm$ 0.76	172.25 $\pm$ 0.76	0.52	192.41 $\pm$ 1.36	192.46 $\pm$ 1.36	0.02
18.13	2885.59 $\pm$ 45.95	2864.29 $\pm$ 45.75	0.74	245.98 $\pm$ 0.79	247.69 $\pm$ 0.78	0.69	273.17 $\pm$ 0.80	273.49 $\pm$ 0.82	0.12
16.22	4979.05 $\pm$ 51.82	4938.08 $\pm$ 51.85	0.83	442.37 $\pm$ 1.24	446.83 $\pm$ 1.22	1.00	489.78 $\pm$ 8.94	491.25 $\pm$ 8.99	0.30
13.73	10710.76 $\pm$ 83.77	10604.76 $\pm$ 90.65	0.99	1036.70 $\pm$ 0.98	1052.43 $\pm$ 0.96	1.51	981.54 $\pm$ 7.39	985.83 $\pm$ 7.54	0.44
11.90	20865.33 $\pm$ 131.36	20615.10 $\pm$ 149.10	1.21	2036.20 $\pm$ 4.10	2078.56 $\pm$ 3.92	2.06	1937.79 $\pm$ 4.84	1952.26 $\pm$ 5.00	0.74
9.69	52974.26 $\pm$ 150.68	52146.51 $\pm$ 202.32	1.57	5076.92 $\pm$ 21.76	5236.33 $\pm$ 21.16	3.09	4810.86 $\pm$ 11.90	4871.87 $\pm$ 12.05	1.26
7.88	126469.28 $\pm$ 216.71	124058.95 $\pm$ 361.74	1.92	11890.79 $\pm$ 57.19	12447.42 $\pm$ 55.62	4.57	11176.89 $\pm$ 18.46	11402.79 $\pm$ 18.79	2.00

Figure 5.4: (a) C_0, (b) C_{66}, and (c) C_{44} vs. V_m calculated from VPI-2B and VPI(3B) simulations with and without long-range corrections. Values from DMC simulations by Cazorla and Boronat[5] and VMC simulations from Pessoa, *et al.*[4] are shown as solid triangles. Where available, experimental data from Refs. [14], [25], and [26] are also shown. *Continued on next page.*



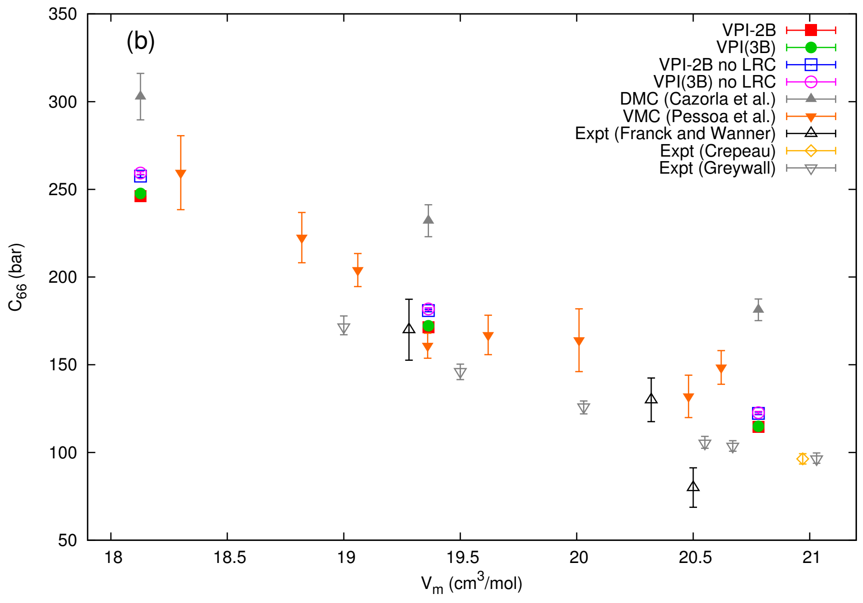


Figure 5.4: *Continued.*

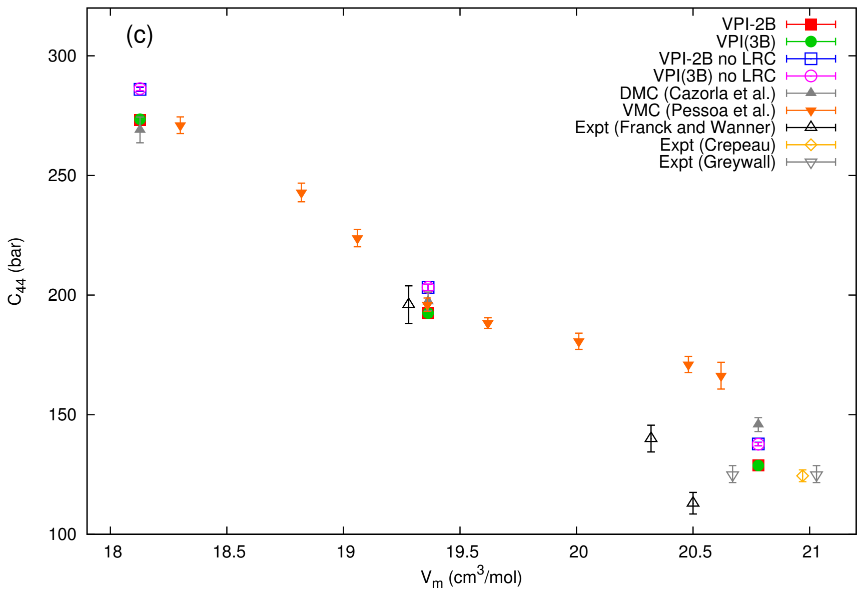


Figure 5.4: *Continued.*

nonzero elastic constants, we expect our calculated values of C_{11} and C_{12} to differ from those reported in Ref. [5].

5.3.3 Dependence of Three-Body Energy on the Heterogeneous Strain Variables

It is also of interest to consider the effect of the three deformation parameters on the average three-body energy. The change in the three-body energy with each parameter is shown in Fig. 5.5 at a molar volume of 13.73 cm^3/mol where $\Delta\alpha = 0$ represents the equilibrium value for each parameter. From this figure it is clear that the three-body energy is most strongly influenced by the η parameter, and therefore we would expect to see the biggest difference between the VPI-2B and VPI(3B) results when the C_0 pure shear constant is evaluated. The γ parameter also causes changes in the three-body energy greater than the uncertainty in the V_3 at the equilibrium geometry, indicating that C_{66} should also be influenced by the addition of the three-body correction, though to a lesser extent than C_0. It is only in the ϵ parameter that we see changes in V_3 which are within the statistical uncertainty of V_3 at the equilibrium geometry, and therefore little difference is expected between the VPI-2B and VPI(3B) calculated values of C_{44}.

The greater effect of the η parameter on the three-body energy compared to γ and ϵ is made clearer by considering the effect of this distortion on the equilateral trimers. Equilateral trimers make a higher contribution to the total three-body energy than any other trimer geometry by an order of magnitude or more, and therefore changes in their conformations and energies will dominate the three-body response to each of the strain variables. For reference, the Cencek three-body potential energy as a function of side length R_{nn} of an equilateral trimer is provided in Fig. 4.6. Table 5.5 summarizes the change in the conformation and energy of an equilateral trimer oriented parallel to the basal plane of the crystal (in-plane) and perpendicular to the basal plane (out-of-plane) when one of the heterogeneous strain variables is fixed at

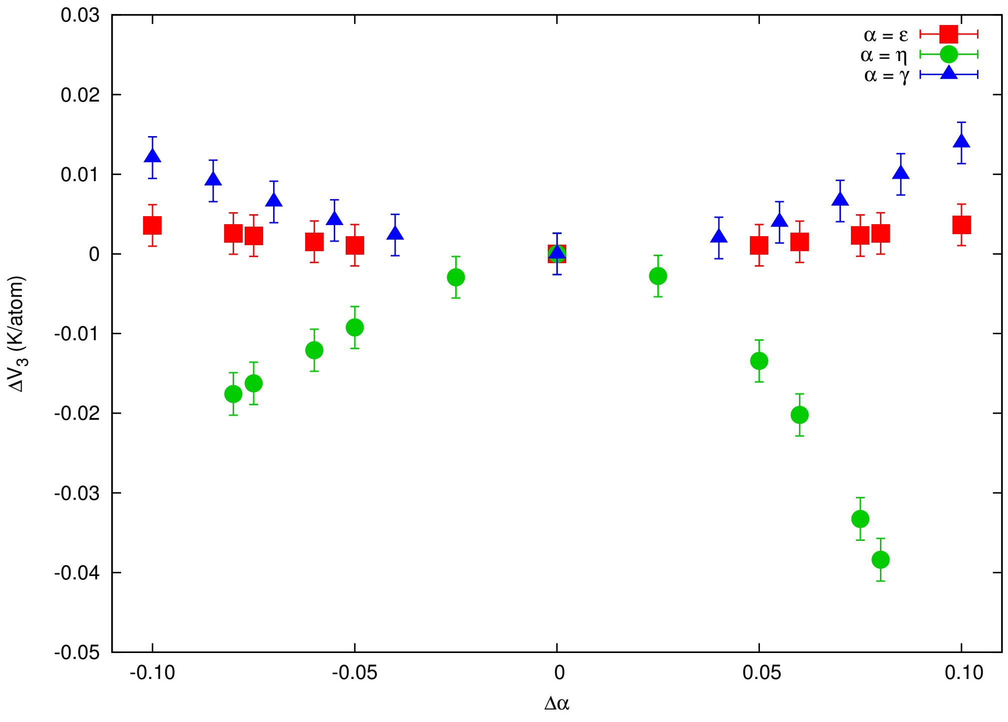


Figure 5.5: Change in the three-body correction (ΔV_3) vs. change in the η (green circles), γ (red squares), and ϵ (blue triangles) parameters at a molar volume of 13.73 cm^3/mol. Similar trends were observed at every molar volume.

Table 5.5: Change in trimer geometry and energy (V_3) with change in one of the heterogeneous strain variables for an in-plane and out-of-plane equilateral trimer at a molar volume of 13.73 cm^3/mol. All other variables are fixed at the equilibrium values. The central angle θ is reported in degrees. Side lengths R_1, R_2, and R_3 are provided in units of R_{nn}. At equilibrium, all side lengths are equal to $1.0R_{nn}$ and both in-plane and out-of-plane equilateral trimers have energies of 2.14x10^{-2} K/trimer and make contributions of 4.28x10^{-2} K/atom and 12.85x10^{-2} K/atom to the total three-body energy, respectively.

In-plane trimer							
Parameter	θ (°)	R_1 (R_{nn})	R_2 (R_{nn})	R_3 (R_{nn})	V_3 (K/trimer)	% Change	$V_{3,\text{tot}}$ (K/atom)
η = -0.08	60.00	1.04257	1.04257	1.04257	2.89x10^{-2}	34.93	5.78x10^{-2}
η = 0.08	60.00	0.96225	0.96225	0.96225	-1.26x10^{-2}	-158.78	-2.52x10^{-2}
γ = 0.90	62.54	1.02875	0.94868	1.02875	2.34x10^{-2}	9.11	4.67x10^{-2}
γ = 1.10	57.58	0.97817	1.04881	0.97817	2.28x10^{-2}	6.33	4.55x10^{-2}
ϵ = -0.10	59.88	1.00125	1.00499	1.00125	2.25x10^{-2}	4.86	4.49x10^{-2}
ϵ = 0.10	59.88	1.00125	1.00499	1.00125	2.25x10^{-2}	4.86	4.49x10^{-2}
Out-of-plane trimer							
Parameter	θ (°)	R_1 (R_{nn})	R_2 (R_{nn})	R_3 (R_{nn})	V_3 (K/trimer)	% Change	$V_{3,\text{tot}}$ (K/atom)
η = -0.08	57.21	1.04257	0.96259	0.96259	1.66x10^{-2}	-22.37	9.97x10^{-2}
η = 0.08	62.51	0.96225	1.04223	1.04223	2.72x10^{-2}	27.14	16.34x10^{-2}
γ = 0.90	57.97	1.02875	1.01835	0.99210	2.59x10^{-2}	21.11	11.73x10^{-2}
γ = 1.10	61.84	0.97817	0.98473	1.00868	1.66x10^{-2}	-22.69	12.16x10^{-2}
ϵ = -0.10	62.70	1.00125	1.00000	1.04123	2.62x10^{-2}	22.52	12.75x10^{-2}
ϵ = 0.10	57.31	1.00125	1.00000	0.95961	1.44x10^{-2}	-32.78	12.75x10^{-2}

the maximum or minimum value. In addition, the total contribution to the three-body energy from all in-plane and out-of-plane trimers is provided in the final column in order to account for orientation-dependent changes in the three-body energy. As in the perturbative calculation, these energies have been divided by three to avoid triple counting. The results in Table 5.5 are obtained from calculations at a molar volume of 13.73 cm^3/mol in the absence of zero point motion. At this molar volume, R_{nn} = 6.014 a_o, which is very close to the distance at which the three-body energy becomes attractive in an equilateral trimer.

Table 5.5 shows a fairly consistent, non-negligible response to each of the strain variables from the out-of-plane equilateral trimer; however, it is clear that the in-plane equilateral trimer is influenced much more strongly by the η parameter. Changing η changes the c/a ratio of the crystal by compressing or expanding the lattice spacing along the z-axis with a corresponding compensation in the lattice spacing in the x, y-plane to maintain constant volume. Therefore, for the in-plane equilateral trimer, increasing η results in isotropic compression of the equilateral trimer, simultaneously bringing all three atoms into closer contact. Noting the steep decline in Fig. 4.6, we see that a small compression in the equilateral trimer results in a much larger change in V_3 than an equivalent expansion in the trimer, accounting for the anisotropic response of ΔV_3 to the η parameter in Fig. 5.5. Although the energy is seen to increase in the in-plane trimer when η = -0.08, the opposite effect is seen in the out-of-plane trimer. Because there are three out-of-plane equilateral trimers for every in-plane equilateral trimer, the net response to any change in η is a more attractive V_3. This explains the consistently lower VPI(3B) values of C_0 in Table 5.4.

Analysis of the responses to γ and ϵ is more complicated because these will result in different responses depending on the orientation of the selected trimer. In the two representative trimers selected for Table 5.5, the γ parameter results in greater changes in V_3 for the in-plane equilateral trimer, while ϵ has a stronger impact on the out-of-plane trimer. Due to the greater number of out-of-plane equilateral trimers, it would seem that not only should ΔV_3 show a greater dependence on ϵ, but that

ΔV_3 should become more attractive for positive values of ϵ and more repulsive for negative values. Neither of these are shown to be true in Fig. 5.5, and therefore it is helpful to consider the change in the total energy for all in-plane and out-of-plane equilateral trimers. From the $V_{3,\text{tot}}$ values in Table 5.5, it is clearer that the changes in the total three-body contribution from the two types of equilateral trimers considered here are significantly lower when γ and ϵ are changed than when η is changed, which manifests in smaller absolute differences between the VPI-2B and VPI(3B) C_{66} and C_{44} constants in comparison to the differences in C_0 reported in Table 5.4. The changes in the total three-body energy contribution from the in-plane and out-of-plane trimers also cancel out to an extent, with the in-plane $V_{3,\text{tot}}$ becoming more repulsive when the out-of-plane contribution becomes more attractive. Although these exact energies will change when zero point motion is accounted for, this helps to explain the lower (though not insignificant) dependence of ΔV_3 on γ and ϵ.

5.3.4 Remaining Nonzero Elastic Constants

The remaining four nonzero elastic constants C_{11}, C_{12}, C_{13}, and C_{33} are calculated from K, C_0, and C_{66} using Eqs. 5.1-5.4 above. The results of these calculations are tabulated in Table 5.6. In Fig. 5.6, we compare our elastic constants to previous experimental and theoretical results. As expected, differences in the bulk moduli and C_{66} constant result in disagreement between our calculated elastic constants and those of Ref. [5]. However, in general our results are in similar or better agreement with the experimental data from Refs. [14] and [26] compared to the results from Cazorla and Boronat. The variability and uncertainty in the experimental data makes it difficult to determine whether or not the incorporation of three-body interactions results in better agreement with experiment in this molar volume range. In addition, impurities and thermal contributions in the finite-temperature experiments, as well as uncertainty in the determination of the molar volume from the experimental pressure, can easily lead to differences between experimental and theoretical results. However,

Table 5.6: Nonzero elastic constants C_{11}, C_{12}, C_{13}, and C_{33} (bar) calculated from VPI-2B and VPI(3B) pure shear constants and bulk moduli.

V_m	C_{13} (bar)		C_{12} (bar)		C_{13} (bar)		C_{33} (bar)	
(cm^3/mol)	VPI-2B	VPI(3B)	VPI-2B	VPI(3B)	VPI-2B	VPI(3B)	VPI-2B	VPI(3B)
20.78	505.62 ± 8.83	490.15 ± 9.24	276.55 ± 8.83	260.29 ± 9.24	148.64 ± 9.28	134.35 ± 9.68	633.52 ± 10.96	616.10 ± 11.35
19.36	747.58 ± 5.78	732.31 ± 6.01	404.87 ± 5.78	387.80 ± 6.01	219.29 ± 7.46	205.49 ± 7.72	933.16 ± 12.14	914.62 ± 12.47
18.13	1074.89 ± 3.13	1054.02 ± 3.16	582.93 ± 3.13	558.64 ± 3.16	347.98 ± 5.36	328.95 ± 5.36	1309.84 ± 10.34	1283.71 ± 10.31
16.22	1981.19 ± 4.46	1930.04 ± 3.56	1096.46 ± 4.46	1036.39 ± 3.56	708.99 ± 6.57	660.20 ± 6.01	2368.67 ± 11.94	2306.23 ± 11.65
13.73	4739.32 ± 5.17	4549.69 ± 5.40	2665.93 ± 5.17	2444.84 ± 5.40	1917.50 ± 9.53	1729.81 ± 10.22	5487.75 ± 18.73	5264.73 ± 20.22
11.90	9520.67 ± 21.61	9039.68 ± 22.89	5448.27 ± 21.61	4882.56 ± 22.89	4006.91 ± 24.70	3525.27 ± 26.73	10962.02 ± 35.34	10396.97 ± 39.21
9.69	24048.73 ± 441.59	22753.68 ± 122.75	13894.89 ± 441.59	12281.01 ± 122.75	10142.77 ± 441.29	8826.26 ± 122.47	27800.86 ± 442.24	26208.43 ± 128.51
7.88	62350.48 ± 413.48	55638.50 ± 117.41	38568.89 ± 413.48	30743.66 ± 117.41	29381.47 ± 410.04	22514.58 ± 109.10	71537.90 ± 412.16	63867.57 ± 129.42

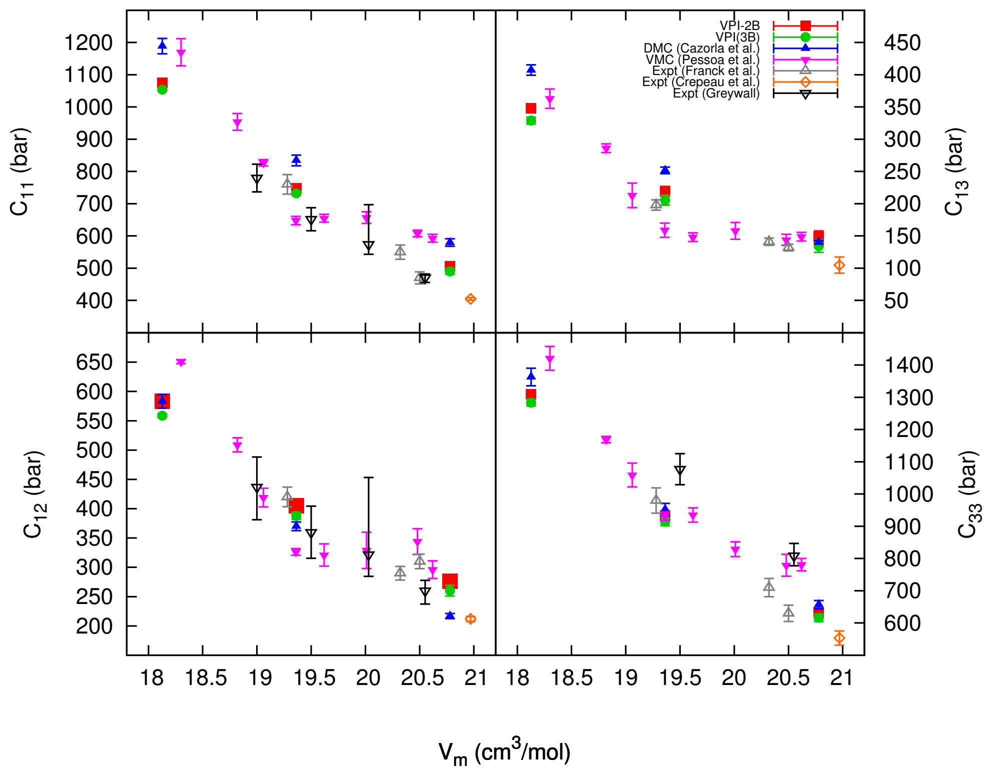


Figure 5.6: Comparison of the low density values of C_{11}, C_{12}, C_{13}, and C_{33} elastic constants calculated from VPI-2B and VPI(3B) simulations. In many cases, error bars are smaller than the symbol size. Existing experimental[14, 25, 26] and theoretical[4, 5] data is included where available.

we can conclude that even at the high molar volumes, the incorporation of three-body interactions has a non-negligible effect on the calculated elastic constants. At molar volumes below 18.13 cm^3/mol, the VPI-2B and VPI(3B) calculated results no longer agree within their statistical uncertainties for any of the elastic constants in Table 5.6, and the differences in C_{11} and C_{12} are statistically significant even at higher molar volumes.

The effect of three-body interactions becomes much more pronounced at lower molar volumes (see Fig. 5.7). At the lowest molar volume studied here, the difference in the elastic constants ranges from 11.3-26.5%. This is a significant contribution which should not be ignored. However, to our knowledge, experimental low-temperature elastic constant measurements have not been reported at these densities, and therefore a direct comparison to experimental results is not currently possible. More high-pressure, low-temperature experimental determinations of the elastic constants are necessary to assess the accuracy of the current study.

5.4 Summary and Conclusion

Elastic constants for hcp solid ^{4}He have been calculated at $T = 0$ K using VPI simulations with and without perturbative three-body corrections. These calculations also include long-range corrections to the two-body potential energy which were not accounted for in a previous DMC study[5]. The nonzero elastic constants were calculated from the bulk modulus and pure shear constants C_0, C_{66}, and C_{44} at each molar volume of interest. The bulk modulus was calculated from our previously reported pressure-volume EOSs[6] and it is determined that for all but the molar volumes closest to the transition from the high to low density regions, much better agreement with the experimental bulk modulus is obtained from the VPI(3B) simulations. For the most part, omission of the long-range correction brings our calculated C_0, C_{66} and C_{44} values into better agreement with the previously reported theoretical values from Ref. [5], however this was not strictly true. In the case of

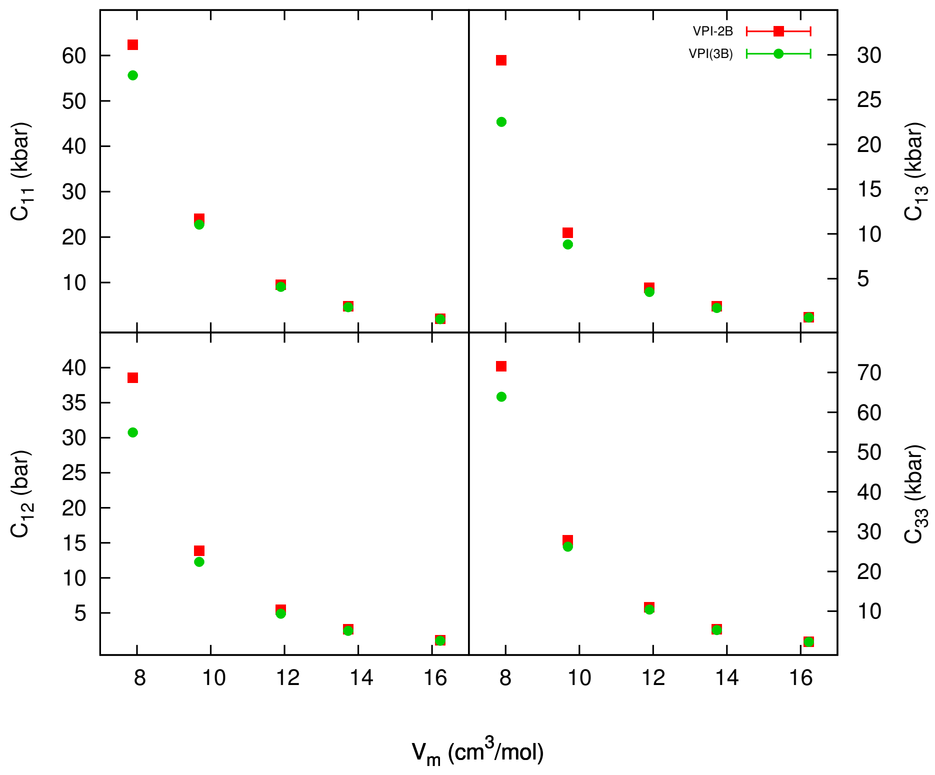


Figure 5.7: Comparison of the high density values of C_{11}, C_{12}, C_{13}, and C_{33} elastic constants calculated from VPI-2B and VPI(3B) simulations. Error bars are smaller than the symbol size.

C_{66}, neither set of values (with or without the long-range corrections) agrees well with Ref. [5], though better agreement with experiment is obtained from the current study. Analysis of the change in the three-body correction with each of the pure shear constants demonstrates that three-body interactions play a more significant role in the C_0 and C_{66} constants than in the C_{44} shear modulus. These effects can largely be understood by considering the deformation of the in-plane and out-of-plane equilateral trimers which make the largest contribution to the three-body correction.

C_{11}, C_{12}, C_{13}, and C_{33} constants calculated from the pure shear constants and K are found to agree qualitatively with the available theoretical and experimental values. Although low-temperature experimental data is lacking at high densities, we can conclude that three-body interactions have a non-negligible effect on the calculated elastic constants, particularly at lower molar volumes. Our results suggest that three-body interactions cannot be ignored in the calculation of the bulk modulus or the elastic constants at high densities. We hope that the results of this investigation will inspire further experimental determinations of the low-temperature elastic constants of high-density hcp solid ^{4}He in order to further validate the results obtained here.

5.5 References

[1] E. Polturak and N. Gov, *Contemporary Physics* **44**, 145 (2003). 129

[2] D. M. Ceperley, *Rev. Mod. Phys.* **67**, 279 (1995). 129, 130, 135, 136

[3] R. Pessoa, M. de Koning, and S. A. Vitiello, *arXiv:1203.0456v1 [cond-matt.other]* , 1 (2012). 129

[4] R. Pessoa, M. De Koning, and S. A. Vitiello, *Journal of Low Temperature Physics* **173**, 143 (2013). 129, 143, 144, 146, 151, 161

[5] C. Cazorla, Y. Lutsyshyn, and J. Boronat, *Phys. Rev. B* **85**, 024101 (2012). 129, 131, 132, 137, 143, 144, 146, 151, 154, 158, 159, 161, 163

[6] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **146**, 094510 (2017). 129, 130, 132, 134, 136, 138, 141, 144, 159

[7] C. Cazorla and J. Boronat, *Phys. Rev. B* **92**, 224113 (2015). 129

[8] L. W. Bruch and I. J. McGee, *The Journal of Chemical Physics* **59**, 409 (1973). 130

[9] A. Grechnev, S. M. Tretyak, Y. A. Freiman, A. F. Goncharov, and E. Gregoryanz, *Physical Review B* **92**, 024102 (2015). 130

[10] C. Cazorla and J. Boronat, *Physical Review B - Condensed Matter and Materials Physics* **91**, 1 (2015). 130

[11] Y. A. Freiman, S. M. Tretyak, A. Grechnev, A. F. Goncharov, J. S. Tse, D. Errandonea, H. K. Mao, and R. J. Hemley, *Physical Review B - Condensed Matter and Materials Physics* **80**, 3 (2009). 130, 131

[12] Y. A. Freiman, A. Grechnev, S. M. Tretyak, A. F. Goncharov, C. S. Zha, and R. J. Hemley, *Physical Review B - Condensed Matter and Materials Physics* **88**, 1 (2013). 130

[13] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 130, 138

[14] J. P. Franck and R. Wanner, *Physical Review Letters* **25**, 345 (1970). 131, 146, 151, 158, 161

[15] W. F. King and P. H. Cutler, *J. Phys. Chem. Solids* **32**, 761 (1971). 131

[16] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **144**, 084505 (2016). 133, 134, 137

[17] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *The Journal of Chemical Physics* **21**, 1087 (1953). 133

[18] R. A. Aziz, F. R. McCourt, and C. C. Wong, *Molecular Physics* **61**, 1487 (1987). 134

[19] A. Sarsa, K. E. Schmidt, and W. R. Magro, *The Journal of Chemical Physics* **113**, 1366 (2000). 135

[20] R. J. Hinde, *Computer Physics Communications* **182**, 2339 (2011). 135, 136

[21] H. F. Trotter, *Proceedings of the American Mathematical Society* **10**, 545 (1959). 136

[22] J. Hirschfelder, C. Curtiss, and R. Bird, *Molecular theory of gases and liquids*, Structure of matter series, Wiley, 1954. 137

[23] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 138, 141, 142, 143, 144

[24] Y. Lutsyshyn, C. Cazorla, G. E. Astrakharchik, and J. Boronat, *Physical Review B - Condensed Matter and Materials Physics* **82**, 3 (2010). 143

[25] R. H. Crepeau, O. Heybey, D. M. Lee, and S. A. Strauss, *Physical Review A* **3**, 1162 (1971). 146, 151, 161

[26] D. S. Greywall, *Physical Review B* **16**, 5127 (1977). 146, 151, 158, 161

Chapter 6

Conclusion

Taken as a whole, the results of these investigations further confirm that three-body interactions do make a significant contribution to the $T = 0$ K properties of hcp solid ^{4}He, even at low densities, and should not be ignored. Although the incorporation of a three-body potential into quantum Monte Carlo simulations could not corroborate the anomolous anisotropy in the Debye-Waller factors reported by Blackburn *et al.*[1], simulations which accounted for both variational error and three-body interactions produced ground state energies in closer agreement with experimental energies from Edwards and Pandorf[2] than two-body simulations at low densities. Moreover, the pressure-volume equations of state derived from both VMC and VPI three-body simulations have been shown to agree much better with the experimental EOS reported by Driessen, *et al.*[3]. Differences in the two-body and three-body EOSs also impact the calculated elastic properties of the system, as four of the five nonzero elastic constants depend on the bulk modulus, which is related to the derivative of the pressure-volume EOS. Beyond this, we have shown that three-body interactions also make a non-negligible contribution to the three pure shear constants, C_0, C_{66}, and C_{44}, a contribution that increases steadily with increasing density. One of the fundamental characteristics of a quantum solid is its high compressibility, and therefore accurate calculations of the elastic properties of this system are important to understanding the properties of this and other quantum solids.

In addition to the evidence we have presented to support the importance of three-body interactions, we have also demonstrated that, in many cases, these interactions can be accounted for using a simple perturbative approach which allows us to maintain computational efficiency while increasing the accuracy of our model compared to two-body simulations. The applicability of this perturbative treatment hinges on the small contribution of three-body interactions to the ground state wavefunction, such that the zero point motions of the ^{4}He atoms are not strongly affected by their incorporation. The close agreement between the mean squared displacements calculated from the VMC-2B and VMC+3B simulations first suggested that three-body interactions may be treated using the perturbative approach[4] (Chapter 3).

This hypothesis was further supported when comparing the pressure-volume EOS derived from the VMC(3B) and VMC+3B energies which were shown to agree within their statistical uncertainties at most points throughout the full molar volume region. A closer inspection of the two-body and three-body contributions to the potential energy in the four VPI+3B simulations and their corresponding VPI(3B) results, however, suggests that at higher densities the individual contribution of the potential energy to the total energy is underestimated in the perturbative treatment while the kinetic contribution is overestimated[5].

It is therefore important to determine which properties and what molar volume ranges are of interest when choosing between the perturbative and full-incorporation treatments. Properties which depend only on the total energy, such as the elastic constants, appear to be well characterized by the perturbative treatment. In addition, at low molar volumes, we have shown that three-body interactions make a small contribution to the pure shear constants, and therefore it is possible that accurate elastic properties of hcp ^{4}He can be obtained by considering three-body interactions only in the equilibrium geometry, making the incorporation of three-body interactions into these calculations even more efficient.

It stands to reason that the reliability of this perturbative treatment also depends on the accuracy of the three-body potential employed. Previous perturbative treatments of three-body interactions utilizing the Bruch-McGee[6] or Cohen and Murrell[7] potentials did not result in significantly improved agreement with experimental energies or pressures, and indeed, often worse agreement was produced if phenomenological scaling factors were not introduced[8, 9, 10]. The nonadditive three-body potential reported by Cencek, *et al.*[11] implemented in this investigation resulted in a significant improvement in the pressure-volume EOS, ground state energies, and bulk moduli without the need for further parameterization at molar volumes as low as 4.0 cm^3/mol. We therefore find the perturbative incorporation of this three-body potential to be a computationally efficient and reliable means of accounting for three-body interactions in the hcp ^{4}He system.

Future work in this area might investigate the use of a reweighing method such as that proposed (though not implemented) in Ref. [8] in order to account for the effect of three-body interactions on the potential and kinetic energies, among other observables, within the framework of a perturbative treatment. This could help to eliminate the discrepancy between the V_2 potential energies calculated from VPI+3B and VPI(3B) simulations, thereby extending the region of validity for our perturbative treatment. Further verification of the Cencek three-body potential in finite-temperature simulations would also be useful in order to allow for a more direct comparison to experimental results. Additionally, it remains to be determined what effect three-body interactions might have on the dynamical properties of hcp ^{4}He, and therefore the calculation of the experimentally well-studied Raman spectrum of hcp ^{4}He[12, 13, 14] using the Cencek potential would prove valuable in further understanding the role of many-body interactions in this system. Beyond this, more experimental investigations in the $T < 1$ K range achieved by Blackburn, *et al.*[1] would be very helpful in the continued evaluation and improvement of theoretical models such as the ones presented here.

6.1 References

[1] E. Blackburn, J. M. Goodkind, S. K. Sinha, J. Hudis, C. Broholm, J. van Duijn, C. D. Frost, O. Kirichek, and R. B. E. Down, *Phys. Rev. B* **76**, 024523 (2007). 167, 169

[2] D. O. Edwards and R. C. Pandorf, *Phys. Rev.* **140**, A816 (1965). 167

[3] A. Driessen, E. van der Poll, and I. F. Silvera, *Phys. Rev. B* **33**, 3269 (1986). 167

[4] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **144**, 084505 (2016). 167

[5] A. L. Barnes and R. J. Hinde, *The Journal of Chemical Physics* **146**, 094510 (2017). 168

[6] L. W. Bruch and I. J. McGee, *The Journal of Chemical Physics* **59**, 409 (1973). 168

[7] M. J. Cohen and J. N. Murrell, *Chemical Physics Letters* **260**, 371 (1996). 168

[8] S.-Y. Chang and M. Boninsegni, *The Journal of Chemical Physics* **115**, 2629 (2001). 168, 169

[9] S. Ujevic and S. A. Vitiello, *Physical Review B - Condensed Matter and Materials Physics* **71**, 1 (2005). 168

[10] S. Ujevic and S. A. Vitiello, *Journal of Physics: Condensed Matter* **19**, 116212 (2007). 168

[11] W. Cencek, K. Patkowski, and K. Szalewicz, *The Journal of Chemical Physics* **131**, 064105 (2009). 168

[12] N. Ogita, M. Udagawa, and K. Ohbayashi, *Phys. Rev. B* **47**, 11810 (1993). 169

[13] R. E. Slusher and C. M. Surko, *Phys. Rev. Lett.* **27**, 1699 (1971). 169

[14] C. M. Surko and R. E. Slusher, *Phys. Rev. B* **13**, 1095 (1976). 169

Appendix

Appendix A

VMC Programs

A.1 VMC 2-Body Program (VMC-2B)

The VMC program files used to sample fixed trial wavefunctions as well as calculate the reweighted energies from wavefunctions with different parameters are included below. The parent-child setup was largely adapted from the QSATS code (Robert J. Hinde, *Computer Physics Communications*, **182**(11), 2339 (2011)). A number of the subroutines are the same as the QSATS code and have therefore not been included. This program implements MPI parallelization and requires the following files for compilation:

main.f	Main program that determines whether a node is a parent or child node. It is responsible for initializing and terminating the full VMC code.
cmrg.f	Random number generator which is called to randomly generate new atomic positions from the trial wavefunction and advances the random number generator state vector.
rsetup.f	Initializes the random number generator state vector for each child.
parent.f	Parent process which runs on node 0 and sets up the ^{4}He lattice, interacting pair list, and potential energy surface. It also sends tasks to the child processes and calculates the final reweighted energies when the simulation is complete.
input.f	Reads in the input and output file names, debugging level, and simulation parameters.
vinit.f	Sets up the linear interpolation arrays for the Aziz HFD-B(He) pair potential.

child.f	Child process that performs the Metropolis Monte Carlo moves and evaluates the instantaneous kinetic, potential, and total energies every 50 MCCs. This information is sent back to the parent along with the snapshot of the atomic positions.
allrep.f	Calls the subroutine that sends data to the child from the parent and receives data from the child. This subroutine calculates the running averages of the potential, kinetic, and total energies and writes these energies and the atomic snapshots to the output files. It also calculates the reweighted kinetic, potential, and total energies.
send.f	Sends the old atomic configuration and random number generator state vector to the designated child.
kinetic-rw.f	Calculates the reweighted kinetic energy for a given atomic configuration and set of wavefunction parameters.
paramest.f	Estimates initial values of the parameters of the fitting functions for the reweighted energies from either the a_{xy} and a_z reweighting calculations or the b reweighting calculations and generates a gnuplot script which can be used to generate an accurate fit.
tstamp.f	Generates output that states when the file was last compiled. Requires the file tstamp.master.
sizes.h	Contains fixed parameters of the system including the number of atoms, number of interacting pairs, child processes allowed, MCCs assigned to each child process with each send statement from the parent, etc.
vmc-448.com	Contains the common block variables used by the parent and all parent-called subroutines. Child subroutines contain their own versions of these variables.

Program files that can be found in the QSATS code and are not reproduced here: **main.f**, **cmrg.f**, **vinit.f**, **send.f**, **tstamp.f**

In addition, an input file containg the file names read in input.f and a parameter file containing the parameter values read in input.f are required to run the job, along with a lattice file which specifies the total number of atoms, lengths of the simulation cell in the x, y, and z directions, and the atomic coordinates in the simulation cell. These are not provided.

rsetup.f

```
c ----------------------------------------------------------------------

c     this subroutine initializes the pseudo random number generators
c     for the replicas.  it also initializes the value of the rscale
c     variable, which is needed to convert integer pseudo random
c     numbers, which are the raw output of the generators, to floating
c     point pseudo random numbers.

c ----------------------------------------------------------------------

      subroutine rsetup

      implicit double precision (a-h, o-z)
      include 'sizes.h'
      include 'vmc-448.com'

      dimension rseed(6)

      rscale=1.0d0/4294967088.0d0
      write (6, 6000)
6000  format ('INITIALIZING random number seeds'/)

      do i=1, 6
         rseed(i)=12345.0d0
      end do

c --- for each child, skip ahead in the random number stream using rskip
      do i=1, NCH
         do j=1, 6
            rstatv(j, i)=rseed(j)
         end do
         rstatv(7, i)=-1.0d0
         rstatv(8, i)=0.0d0
         call rskip(rseed)
      end do

c --- write out debugging info
      if (idebug.ge.3) then

         write (6, 6001)
6001     format  ('rstatv(1) values:'/)

         do i=1, NCH
            write (6, 6100) i, rstatv(1, i)
6100        format (i5, 1x, f20.1)
         end do

         write (6, *) ''

      end if

      return
      end
```

parent.f

```
c ----------------------------------------------------------------------

c     this is the parent process that runs on node 0.

c     errchk is a subroutine called after every MPI subroutine that
c     checks the MPI error code and reports any errors.

c ----------------------------------------------------------------------

      subroutine parent(ierror)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      include 'mpif.h'

c --- istat = MPI status array
c --- imsg = array of integer values to send to child
```

```
c --- fmsg = array of floating point values to send to child
c --- rw_sums = array of reweighted energies for parameter estimation
c --- rstate = random number state vector

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(9), fmsg(7)
      dimension isent(NCHUNKS), psi(NATOM3)
      dimension rw_sums(4, 9)
      dimension rstate(8)

c --- hart = conversion from hartree to K/atom
      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)
      parameter (hart=315774.65d0)

c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================

      ierror=0

c --- read input file.
      write (6, *) "parent calling input"
      call input

      write (6, 6100) ltfile, dump, dfile, ofile, rwfile
6100  format ('lattice file name  = ', a16/,
     +        'snapshot file name = ', a16/,
     +        'dfile file name     = ', a16/,
     +        'ofile file name     = ', a16/,
     +        'rwfile file name    = ', a16)

      if (idebug.eq.0) write (6, 6110) idebug, 'NONE'
      if (idebug.eq.1) write (6, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (6, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (6, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (6, 6110) idebug, 'HIGH'

6110  format ('debug level = ', i1,' or ', a8/)
c --- calculate phi distortion parameter from eta
c --- this changes the c/a ratio
       phi = sqrt(1.0d0+eta)

c --- read the potential energy curve.

      call vinit(r2min, bin)

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a16/)

      open (8, file=ltfile, status='old', err=901)

      read (8, *, err=902) nlpts

      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         call quit
      end if

c --- read the edge lengths of the supercell.
```

```
      read (8, *, err=903) xlen, ylen, zlen

      den0=dble(NATOMS)/(xlen*ylen*zlen)

      xlen = xlen
      ylen = ylen
      zlen = zlen
c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *, err=904) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

c --- this helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6310) rnnmin
6310  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

      write (6, 6330) xlen, ylen, zlen
6330  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs from the atomic positions of the
c     undistorted lattice

      do i=1, NATOMS
         npair(i)=0
      end do

      nvpair=0

      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)
```

```
c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.
c --------- we determine the interacting pairs from the undistorted
c           then use our values of eta (phi), gamma, and epsilon to
c           impose the distortions for the elastic constant
c           calculations.
            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

c ------------ these transformations impose the lattice distortions
c              They reduce to dx, dy, and dz for eta = 0 (phi = 1),
c              gamma = 1 and epsilon = 0.

               vpvec(1, nvpair)=dx/(sqrt(gam)*phi)
               vpvec(2, nvpair)=dy*sqrt(gam)/phi
               vpvec(3, nvpair)=dz*phi**2+dy*eps

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair

            end if

         end if

      end do
      end do

c --- Now loop back through the coordinates in the xtal array and
c     transform them appropriately

      do i=1, NATOMS

         xtal(i, 1)=xtal(i, 1)/(sqrt(gam)*phi)
         xtal(i, 2)=xtal(i, 2)*sqrt(gam)/phi
         xtal(i, 3)=xtal(i, 3)*phi**2+eps*xtal(i, 2)

      end do

c --- write out the interacting pair information

      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +       'total number of interacting pairs = ', i6)

      if (idebug.ge.2) then

         write (6, 6401)
6401     format (/'interaction pair vectors for atom 1 ',
     +           '[NN distances]:'/)

         do i=1, npair(1)
            ip=ipairs(i, 1)
            d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +        rnnmin
            write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                     vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410        format ('vector # ', i3, ' to atom ', i4, ': ',
     +              3(1x, f9.5), ' length = ', f8.5)
         end do
      end if

c --- set the displacement vectors for all children to zero.

      write (6, 6500)
6500  format (/'SETTING initial configuration to zero'/)

      do j=1, NCH
      do i=1, NATOM3
         path(i,j)=0.0d0
      end do
      end do
c --- initialize random number generator.
      call rsetup
c --- this is the output file where snapshots of the atoms will be
c     stored for analysis by another program.

      open (10, file=dump, form='unformatted')
c --- this is the output file where the instantaneous potential
c     energy and running averages of all energies are stored.

c --- initialize MPI.

      MPI_R=MPI_DOUBLE_PRECISION

      call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)

      call errchk(0, ierr, 100000)

      write (6, 6600) ntasks-1
6600  format ('number of child processes = ', i3/)

c --- now see if there is an old set of displacement vectors from a
c     previous run.  if not, jump head to line 200.

      if (irrst.eq.1) then
         open (8, file=dfile, form='unformatted', status='old', err=200)

         write (6, 6510) dfile
6510     format ('READING initial configuration from ', a16/)

         read (8) nchildren ! make sure we have the same number as before
         do k = 1, nchildren
            read (8) (rstatv(i, k), i=1, 8)
            read (8) (path(i,k), i=1, NATOM3)
         end do
         close (8)
      end if

200   if (idebug.ge.3) then

         write (6, 6170)
6170     format ('x(1) and rstatv(1) values:'/)

         do k = 1, ntasks-1
            write (6, 6180) path(1,k), rstatv(1,k)
6180     format (1x, f15.9, 1x, f20.1)
         end do

         write (6, *) ''
      end if

      if(irrst.eq.1) then
         if(nchildren.ne.ntasks-1) then
         write (6, 6605) nchildren+1
6605     format ('attempting to start calculation from previous run'/
     +           'with a different number of child processors than'/
     +           'the current run. To start from these snapshots, use',
     +           1x, i3, 1x, 'processors.')
         end if
      end if

      if (ntasks-1.gt.NCH) then

         write (6, 6610)
6610     format ('too many child processes; expand the iwork, path, and
     +rstatv arrays.'/
     +           'also note that write statements for HIGH '
     +           'debugging level may fail on some systems.')

         call quit

      end if
c --- this array just counts how evenly the workload was spread among
c     the child processes.

      do i=1, ntasks-1
         iwork(i)=0
      end do
c --- broadcast integer constants to all child processes.

      imsg(1)=NATOMS
      imsg(2)=NATOM3
      imsg(3)=NATOM6
      imsg(4)=NATOM7
      imsg(5)=NIP
      imsg(6)=NPAIRS
      imsg(7)=NVBINS
      imsg(8)=idebug
      imsg(9)=nprint

      do itask=1, ntasks-1

         call MPI_SEND(imsg,
     +                 9,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0101,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100101)

      end do

      if (idebug.ne.0) open (9, file='debug.log')

      if (idebug.eq.1) write (9, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (9, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (9, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (9, 6110) idebug, 'HIGH'
      call flush(9)
c --- broadcast floating-point constants to all child processes.

      fmsg(1)=den
      fmsg(2)=bin
      fmsg(3)=r2min
      fmsg(4)=aaxy
      fmsg(5)=aaz
      fmsg(6)=bb
      fmsg(7)=zmhe

      do itask=1, ntasks-1

         call MPI_SEND(fmsg,
     +                 7,
     +                 MPI_R,
     +                 itask,
     +                 0102,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100102)

      end do
c --- broadcast the interacting-pair vectors to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(vpvec,
     +                 3*NPAIRS,
     +                 MPI_R,
     +                 itask,
     +                 0103,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100103)

      end do
c --- broadcast the list of atom id numbers for the interacting pairs
c     to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ivpair,
     +                 2*NPAIRS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0104,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100104)

      end do
c --- broadcast the size of each stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1

         call MPI_SEND(npair,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0105,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100105)

      end do
c --- broadcast the list of interacting pair id numbers that define the
```

```
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ipairs,
     +                 NIP*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0106,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100106)

      end do
c --- broadcast the potential energy curve V(R) to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(v,
     +                 2*NVBINS,
     +                 MPI_R,
     +                 itask,
     +                 0107,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100107)

      end do
      if (idebug.gt.0) write (9, *) 'end parent PART ONE'
      if (idebug.gt.0) write (9, *) ''
      call flush(9)
c =======================================================================
c     PART TWO: PERFORMING THE SIMULATION
c =======================================================================

c --- open the output files: ofile, dump, and rwfile

      open (10, file=dump, form='unformatted')

      write (10) bb, aaxy, aaz, eta, gam, eps
      open (11, file=ofile)
      open (12, file=rwfile)

c --- Check that NCHUNKS evenly divides nloop/nprint
      ncalc = nloop/nprint
      if(mod(ncalc,NCHUNKS).ne.0) then
          write (6, 5000)
5000     format ('MCCs not divisble by NCHUNKS. Change NCHUNKS'
     +           'or nloops')
         call quit
      end if

c --- initialization of various progress counters.

      denom = 0.0d0
      vsum = 0.0d0
      v2sum = 0.0d0
      esum = 0.0d0
      e2sum = 0.0d0
      tsum = 0.0d0
      t2sum = 0.0d0

c --- also need to keep counters for the reweighting calculations
```

```
c --- these values are required for error estimation in reweighting
      do i = 1, 9
         rw_vsum(i) = 0.0d0
         rw_tsum(i) = 0.0d0
         rw_esum(i) = 0.0d0
         rw_wsum(i) = 0.0d0
         rw_wsqsum(i) = 0.0d0
         rw_vwsum(i) = 0.0d0
         rw_ewsum(i) = 0.0d0
         rw_twsum(i) = 0.0d0
         rw_vvsum(i) = 0.0d0
         rw_eesum(i) = 0.0d0
         rw_ttsum(i) = 0.0d0
      end do

c --- this is how many iterations we have done.
c --- for the vmc program, all loop counting is essentially handled
c     in allrep.f where snapshots are received. This just initializes
c     loop for us.

      loop=0

c --- these tell us about the acceptance ratio for the atom moves.

      ztacc=0.0d0
      ztrej=0.0d0

c --- these counters make sure that we don't lose a chunk of snapshots somewhere
c     in the ether. we use them to count how many chunks have been sent and
c     received.

300   nsent=0
      nrcvd=0

c --- this is a list of flags that are zero for chunks that haven't yet
c     been sent to a child for processing, positive for chunks that have
c     been sent, and negative for chunks that have been processed and
c     returned to the parent.

c     isent(n) is set to the (positive) task id of the receiving child
c     process when a chunk is sent.  this is basically leaving a trail
c     of crumbs so that we can track down the chunks and ask the children
c     to return them to us.

      do nchunk=1, NCHUNKS
         isent(nchunk)=0
      end do
c     allrep distributes chunks of snapshots to children. Once this command has
c     has been called and returns, all loops will have been performed,
c     so all chunks should have been sent and received.

      call allrep(nsent, nrcvd, loops, MPI_R)
      loop = loop + loops
c --- check for lost chunks.

      if (nsent.ne.NCHUNKS.or.nrcvd.ne.NCHUNKS) then
         write (6, *) 'chunks have been lost!'
         write (6, *) 'nsent = ', nsent
         write (6, *) 'nrcvd = ', nrcvd
         ierror=1
      end if

c --- Want to save a checkpoint every 1000 chunks in case job stops
c     before completion. If we need to run more passes, go back to line
c     300.
```

```
      if(loop.lt.nloop) then

         open(8, file=dfile, form='unformatted')

         write (8) loop
         do k=1, ntasks-1
            write (8) (rstatv(i, k), i=1, 8)
            write (8) (path(i, k), i=1, NATOM3)
         end do
         close(8)

         goto 300

      else if(loop.eq.nloop) then

          write (6, 6810) dfile
6810      format ('SAVING final configuration to ', a16/)
          open(8, file=dfile, form='unformatted')

         write (8) loop
         do k=1, ntasks-1
            write (8) (rstatv(i, k), i=1, 8)
            write (8) (path(i, k), i=1, NATOM3)
         end do
         close(8)
c ------ write out reweighting results
c ------ Calculate sums for standard deviation of each energy.
c        Ref: A.M. Ferrenberg, et. al. Phys. Rev. E 51, 5092 (1995).
         rwaxy = aaxy-da
         rwaz = aaz-da
         rwb = bb-3.0d0*db
         do i = 1, 9

           vsum2 = rw_vsum(i)*rw_vsum(i)
           tsum2 = rw_tsum(i)*rw_tsum(i)
           esum2 = rw_esum(i)*rw_esum(i)
           wsum2 = rw_wsum(i)*rw_wsum(i)

           vsumw = rw_vsum(i)*rw_wsum(i)
           tsumw = rw_tsum(i)*rw_wsum(i)
           esumw = rw_esum(i)*rw_wsum(i)

           rw_uavg = rw_vsum(i)/rw_wsum(i)
           rw_tavg = rw_tsum(i)/rw_wsum(i)
           rw_eavg = rw_esum(i)/rw_wsum(i)

           rw_uavgsq = rw_vvsum(i)/rw_wsum(i)
           rw_tavgsq = rw_ttsum(i)/rw_wsum(i)
           rw_eavgsq = rw_eesum(i)/rw_wsum(i)

           rw_uavgvar = denom*((rw_vvsum(i)/vsum2)+(rw_wsqsum(i)/wsum2)
     +                      -2*(rw_vwsum(i)/vsumw))*rw_uavg*rw_uavg
           rw_tavgvar = denom*((rw_ttsum(i)/tsum2)+(rw_wsqsum(i)/wsum2)
     +                      -2*(rw_twsum(i)/tsumw))*rw_tavg*rw_tavg
           rw_eavgvar = denom*((rw_eesum(i)/esum2)+(rw_wsqsum(i)/wsum2)
     +                      -2*(rw_ewsum(i)/esumw))*rw_eavg*rw_eavg

           rw_uavgsd = sqrt(rw_uavgvar)
           rw_tavgsd = sqrt(rw_tavgvar)
           rw_eavgsd = sqrt(rw_eavgvar)
c -------- Save necessary information for parameter estimation to
c          rw_sum
           rw_sums(1, i) = rwb
           rw_sums(2, i) = rwaxy
          rw_sums(3, i) = rwaz
          rw_sums(4, i) = rw_eavg*hart/dble(NATOMS)
          write (12, 900) rwb, rwaxy, rwaz,
     +                         rw_uavg*hart/dble(NATOMS),
     +                         rw_uavgsd*hart/dble(NATOMS),
     +                         rw_eavg*hart/dble(NATOMS),
     +                         rw_eavgsd*hart/dble(NATOMS),
     +                         rw_tavg*hart/dble(NATOMS),
     +                         rw_tavgsd*hart/dble(NATOMS)

900      format (3(1x, F10.8), 6(1x, 1pe20.13))

         call  flush (12)
          if (mod(i,3).eq.0) then
             rwaxy = rwaxy + da
             rwaz = aaz-da
          else
             rwaz = rwaz+da
          end if
          rwb = rwb+db
        end do
        if(da.eq.0.0d0) then
          call param_est_bb(rw_sums)
        else if(db.eq.0.0d0) then
          call param_est_aa(rw_sums)
        else
          write (6, *) "No paramest program called"
        end if
      end if

c --- Now all chunks have run

      if (idebug.gt.0) then
         write (9, *) ''
         write (9, *) 'QSATS is done!'
         write (9, *) ''
      end if

c --- close output files

      close(10)
      close(11)

c --- show how much work every child did.

      if (idebug.gt.0) then
         do i=1, ntasks-1
            write (9, 9100) i, iwork(i)
9100        format ('task ', i3, ' received ', i9, ' chunks')
         end do
      end if

c --- tell the children we're all done.

      do itask=1, ntasks-1

         imsg(1)=0

         call MPI_SEND(imsg,
     +                1,
     +                MPI_INTEGER,
     +                itask,
     +                0204,
     +                MPI_COMM_WORLD,
     +                ierr)
```

```
         call errchk(0, ierr, 100204)

      end do

      write (6, 6900) ztacc
6900  format ('total number of accepted moves = ', f20.1)

      write (6, 6901) ztrej
6901  format ('total number of rejected moves = ', f20.1/)

      if (idebug.gt.0) write (9, *) ''
      if (idebug.gt.0) write (9, *) 'end parent PART TWO'

      return

901   write (6, *) 'error opening lattice file'
      goto 999

902   write (6, *) 'error reading number of atoms from lattice file'
      goto 999

903   write (6, *) 'error reading (unscaled) supercell edge lengths'
      goto 999

904   write (6, *) 'error reading atom number ', i
      goto 999

999   call quit

      return
      end
```

input.f

```
c ---------------------------------------------------------------------
c     this inputs the names of various I/O files and also reads in the
c     parameters for the simulation.
c ---------------------------------------------------------------------
      subroutine input

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      character*8 inword

c --- read in filenames.
c     ltfile = lattice file containing total # atoms and x,y,z
c             coordinates
c     pfile = parameter file, read next
c     ofile = energy output file
c     dfile = snapshot checkpoint file
c     dump = snapshot file
c     rwfile = reweighted energy output file

      read (5, 5000, err=922) ltfile
5000  format (a20)
      read (5, 5000, err=923) pfile
      read (5, 5000, err=924) ofile
      read (5, 5000, err=925) dfile
      read (5, 5000, err=926) dump
      read (5, 5000, err=927) rwfile
```

```
c --- set debug level.

      read (5, 5001, err=931) inword
5001  format (a8)

      if (inword.eq.'NONE') then
         idebug=0
      else if (inword.eq.'MINIMAL') then
         idebug=1
      else if (inword.eq.'LOW') then
         idebug=2
      else if (inword.eq.'MEDIUM') then
         idebug=3
      else if (inword.eq.'HIGH') then
         idebug=4
      else
         write (6, *) 'invalid debug level'
      end if

c --- define some masses.  amu = the unified mass unit in terms of
c     atomic units.
c     zmh and zmhe are the hydrogen and helium atomic masses.

      zmh=1837.1526d0
      amu=zmh/1.007825d0
      zmhe=4.0026d0*amu

c --- read in the simulation parameters.
c     nloop = total # of MCCS
c     nprint = snapshot interval
c     den = number density in atoms per cubic bohr
c     bb = trial wavefunction b parameter
c     aaxy = trial wavefunction a_xy parameter
c     aaz = trial wavefunction a_z parameter
c     irrst = restart variable
c     eta = deformation parameter for C0 (c/a ratio)
c     gam = deformation parameter for C66
c     eps = deformation parameter for C44
c     da = a_i parameter interval used for reweighting
c     db = b parameter interval used for reweighting

      open (7, file=pfile)
      read (7, *, err=901) nloop
      read (7, *, err=902) nprint
      read (7, *, err=903) den
      read (7, *, err=904) bb
      read (7, *, err=905) aaxy
      read (7, *, err=906) aaz
      read (7, *, err=907) irrst
      read (7, *, err=908) eta
      read (7, *, err=909) gam
      read (7, *, err=910) eps
      read (7, *, err=911) da
      read (7, *, err=912) db

      write (6, 6000) NATOMS
6000  format ('REPEATING input parameters'//,
     +        'atom count    = ', i6/)

      write (6, 6001) den, aaxy, aaz, bb, eta, gam, eps
6001  format ('density          = ', f14.7, ' atoms per cubic bohr'/,
     +        'a_xy parameter   = ', f14.7, ' bohr**(-2)'/,
     +        'a_z parameter    = ', f14.7, ' bohr**(-2)'/,
     +        'B parameter      = ', f14.7, ' bohr'/,
     +        'eta factor       = ', f14.7, /,
     +        'gamma factor     = ', f14.7, /,
     +        'epsilon          = ', f14.7, /)

      write (6, 6002) nloop, nprint
6002  format ('number of simulation steps = ', i8/,
     +        'snapshot interval          = ', i8/)

      return

c --- error handling
901   write (6, *) 'error reading number of loops'
      goto 999
902   write (6, *) 'error reading nprint'
      goto 999
903   write (6, *) 'error reading density'
      goto 999
904   write (6, *) 'error reading bb'
      goto 999
905   write (6, *) 'error reading axy'
      goto 999
906   write (6, *) 'error reading az'
      goto 999
907   write (6, *) 'error reading irrst value'
      goto 999
908   write (6, *) 'error reading eta value'
      goto 999
909   write (6, *) 'error reading gamma value'
      goto 999
910   write (6, *) 'error reading eps value'
      goto 999
911   write (6, *) 'error reading da value'
      goto 999
912   write (6, *) 'error reading db value'
      goto 999
921   write (6, *) 'error reading RNG file name'
      goto 999
922   write (6, *) 'error reading lattice file name'
      goto 999
923   write (6, *) 'error reading parameter file name'
      goto 999
924   write (6, *) 'error reading ofile file name'
      goto 999
925   write (6, *) 'error reading dfile file name'
      goto 999
926   write (6, *) 'error reading dump file name'
      goto 999
927   write (6, *) 'error reading rwfile file name'
      goto 999
931   write (6, *) 'error reading debug level'
      goto 999
932   write (6, *) 'error reading RNG initialization mode'
      goto 999
999   call quit

      return
      end
```

child.f

```
c ----------------------------------------------------------------------
c     this is the child process that runs on all nodes except node 0
c     (which is running the parent process).
c ----------------------------------------------------------------------

      subroutine child(MPI_R)

      implicit double precision (a-h, o-z)

      include 'mpif.h'
      include 'sizes.h'

c --- child processes don't include common block, all variables are
c     local and must be defined below. Prevents child processes from
c     overwriting global variables

      common /rancm1/ rscale

      dimension psi(NATOM6), npair(NATOMS), rv(NATOM3)
      dimension istat(MPI_STATUS_SIZE)
      dimension ipairs(NIP, NATOMS)
      dimension vpvec(3, NPAIRS)
      dimension ivpair(2, NPAIRS)
      dimension r2old(NATOMS), r2new(NATOMS), v1(NATOMS), v2(NATOMS)
      dimension v(2, NVBINS)
      dimension imsg(9), fmsg(7), emsg(2), imsg2(3)
      dimension rstate(8), qsave(3)
      dimension dlng(NATOM3), d2lng(NATOM3)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)
c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================
      MPI_R=MPI_DOUBLE_PRECISION

c --- numerical factor for random number generator.

      rscale=1.0d0/4294967088.0d0

c --- determine which process this is and store it in myid.

      call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)

c --- receive all of the information that is broadcast by the parent
c     process.

c --- first receive some integer constants.  these are primarily used to
c     check that the arrays are properly dimensioned.

      call MPI_RECV(imsg,
     +              9,
     +              MPI_INTEGER,
     +              0,
     +              0101,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)
      call errchk(myid, ierr, 200101)

      istop=0
      if (imsg(1).ne.NATOMS) then
         write (6, *) 'size mismatch 1: ', imsg(1)
         istop=1
      end if
      if (imsg(2).ne.NATOM3) then
         write (6, *) 'size mismatch 2: ', imsg(2)
         istop=1
      end if
      if (imsg(3).ne.NATOM6) then
         write (6, *) 'size mismatch 3: ', imsg(3)
         istop=1
      end if
      if (imsg(4).ne.NATOM7) then
         write (6, *) 'size mismatch 4: ', imsg(4)
         istop=1
      end if
      if (imsg(5).ne.NIP) then
         write (6, *) 'size mismatch 5: ', imsg(5)
         istop=1
      end if
      if (imsg(6).ne.NPAIRS) then
         write (6, *) 'size mismatch 6: ', imsg(6)
         istop=1
      end if
      if (imsg(7).ne.NVBINS) then
         write (6, *) 'size mismatch 7: ', imsg(7)
         istop=1
      end if

      if (istop.eq.1) call quit


      nvpair = imsg(6)
      idebug=imsg(8)
      nprint = imsg(9)

c --- debugging output.

      if (idebug.eq.4) write (30+myid, *) 'idebug = ', idebug
      call flush(30+myid)
c --- next receive some floating-point constants.

      call MPI_RECV(fmsg,
     +              7,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0102,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200102)

      den=fmsg(1)
      bin=fmsg(2)
      r2min=fmsg(3)
      aaxy=fmsg(4)
      aaz=fmsg(5)
      bb=fmsg(6)
      zmhe=fmsg(7)

      if (idebug.eq.4) then
         write (30+myid, *) 'den = ', den
         write (30+myid, *) 'bin = ', bin
         write (30+myid, *) 'r2min = ', r2min
```

```
           write (30+myid, *) 'aaxy = ', aaxy
           write (30+myid, *) 'aaz = ', aaz
           write (30+myid, *) 'bb = ', bb
        end if
        call flush(30+myid)
c --- compute the inverse of the potential energy V(R) bin width, to
c     avoid unnecessary divisions.

        binvrs=one/bin

c --- next receive the vectors that connect pairs of atoms in a stencil.

        call MPI_RECV(vpvec,
     +              3*NPAIRS,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0103,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

        call errchk(myid, ierr, 200103)

c --- next receive the list of pairs of atoms.

        call MPI_RECV(ivpair,
     +              2*NPAIRS,
     +              MPI_INTEGER,
     +              0,
     +              0104,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

        call errchk(myid, ierr, 200104)

c --- next receive the number of atoms that belong to each atom's stencil.
c     this should really be the same for every atom for a regular crystal
c     lattice, but we treat it as a variable.

        call MPI_RECV(npair,
     +              NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0105,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

        call errchk(myid, ierr, 200105)
c --- next receive the pairs that constitute each atom's stencil.

        call MPI_RECV(ipairs,
     +              NIP*NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0106,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

        call errchk(myid, ierr, 200106)
c --- next receive the potential energy curve V(R) for interpolation.

        call MPI_RECV(v,
     +              2*NVBINS,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0107,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200107)
      if (idebug.eq.4) then
         write (30+myid, *) 'child moving to PART TWO'
         call flush(30+myid)
      end if


c ======================================================================
c     PART TWO: PERFORMING THE SIMULATION
c ======================================================================
c --- initialize running totals
100   idchunk=0
      nacc=0
      nrej=0
      pot = 0.0d0
      tloc = 0.0d0
c --- send request for data (message type 1201) to parent.  the first
c     time through, or if we are waiting for all children to sync up,
c     there are no results to send back to the parent, so we indicate
c     this by setting idchunk=0 just above, and then sending this to
c     the parent in imsg2(1).

200   imsg2(1)=idchunk
      imsg2(2)=nacc
      imsg2(3)=nrej

      emsg(1) = pot
      emsg(2) = tloc
      call MPI_SEND(imsg2,
     +              3,
     +              MPI_INTEGER,
     +              0,
     +              1201,
     +              MPI_COMM_WORLD,
     +              ierr)

      call errchk(myid, ierr, 201201)
c --- on the other hand, if there are results to send back, then we
c     do so here.

      if (idchunk.gt.0) then
c ------ first we send a message of type 1202 that contains the atoms'
c        new positions.

         call MPI_SEND(psi,
     +                 NATOM3,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1202,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201202)
c ------ then we send a message of type 1203 that contains the updated
c        random number generator state vector.

         call MPI_SEND(rstate,
     +                 8,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1203,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201203)
c ------ and a message of type 1204 that contains the instantaneous
c        potential and kinetic energies (2-body only)
         call MPI_SEND(emsg,
     +                 2,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 201204)
      end if

c --- wait for acknowledgement (message type 0204) from parent.  the
c     parent also uses this to signal the child that more input will
c     be sent.

c     if imsg(1) is positive, it is a idchunk number that represents the
c     next chunk of snapshots that this child should process.
c     if imsg(1) is negative, then this child needs to wait for the
c     other children to sync up, and so the child goes back to the top
c     of PART TWO.

c     if imsg(1) is zero, there is no more work to be done.

      call MPI_RECV(imsg2,
     +              1,
     +              MPI_INTEGER,
     +              0,
     +              0204,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200204)

c --- loop back and wait for more input if instructed by parent.

      if (imsg2(1).lt.0) goto 100

c --- terminate if the simulation is complete.

      if (imsg2(1).eq.0) then
         if (idebug.eq.4) write (30+myid, *) 'child is done!'
         call flush(30+myid)
         return
      end if

c --- if there is a new chunk to process, then receive data from
c     the parent.
c --- we need to save the replica number that we are about to work on.

      idchunk=imsg2(1)
c --- next receive the old atomic coordinates in a message of type 0205.

      call MPI_RECV(psi,
     +              NATOM3,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0205,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200205)
c --- next receive the random number generator state vector, in
c     a message of type 0206.

      call MPI_RECV(rstate,
     +              8,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0206,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200206)
c ==================================================================
c --- this is the actual VMC simulation
c ==================================================================
c --- this counts the simulation loop that we're on. For every chunk
c     received by the child, we run through nprint loops
      loop = 0

c --- this is the number of accepted (nacc) and rejected (nrej) moves
c     in the current set of nprint loops.

      nacc=0
      nrej=0

c --- this is the denominator that we will use to compute the average
c     potential energy for each set of nprint loops.

      denom=0.0

c --- energy adjustment loop.

300   loop=loop+1

c --- try to move each atom once.

      do k=1, NATOMS
c ----- compute ln of the trial wave function squared when the
c       atom is in its current location. (in the serial version
c       of the code these were in separate subroutines)

         glog=0.0d0
         do nn=1, npair(k)

            n=ipairs(nn, k)
            i=ivpair(1, n)
            j=ivpair(2, n)

            dx=(psi(3*j-2))+(-psi(3*i-2))+
     +         vpvec(1, n)
            dy=(psi(3*j-1))+(-psi(3*i-1))+
     +         vpvec(2, n)
            dz=(psi(3*j))  +(-psi(3*i))+
     +         vpvec(3, n)

            r2=dx*dx+dy*dy+dz*dz
            br2=bb*bb/r2
            br5=br2*br2*sqrt(br2)
            glog=glog-0.5d0*br5

         end do
c ------ multiplying the ln by 2 is like computing the ln of the square.

         g1=2.0d0*glog

c ----- save the old position of this atom.

        qsave(1)=psi(3*k-2)
        qsave(2)=psi(3*k-1)
        qsave(3)=psi(3*k)

c ----- pick three gaussian random numbers and scale them.

         call gstep(rstate, gauss, rscale)
         psi(3*k-2)=gauss/sqrt(2.0*2.0*aaxy)

         call gstep(rstate, gauss, rscale)
         psi(3*k-1)=gauss/sqrt(2.0*2.0*aaxy)

         call gstep(rstate, gauss, rscale)
         psi(3*k)=gauss/sqrt(2.0*2.0*aaz)

c ----- compute ln of the trial wave function squared after the atom moves
c       to its new location.
         glog=0.0d0
         do nn=1, npair(k)

            n=ipairs(nn, k)

            i=ivpair(1, n)
            j=ivpair(2, n)
            dx=(psi(3*j-2))+(-psi(3*i-2))+
     +         vpvec(1, n)
            dy=(psi(3*j-1))+(-psi(3*i-1))+
     +         vpvec(2, n)
            dz=(psi(3*j))  +(-psi(3*i))+
     +         vpvec(3, n)

            r2=dx*dx+dy*dy+dz*dz
            br2=bb*bb/r2
            br5=br2*br2*sqrt(br2)
            glog=glog-0.5d0*br5

         end do
c ------ multiplying the ln by 2 is like computing the ln of the square.

         g2=2.0d0*glog

c ----- decide whether to accept or reject the move.

c ----- if the new trial wave function is lower than the old, we
c       conditionally accept the move.
        if (g2.lt.g1) then
           gratio=exp(g2-g1)
           call rstep(rstate, z, rscale)

           if (z.lt.gratio) then
              nacc = nacc+1
           else

              psi(3*k-2)=qsave(1)
              psi(3*k-1)=qsave(2)
              psi(3*k)=qsave(3)

              nrej = nrej+1
           end if

c --- if the new trial wave function is larger, we always accept the
c     move.

        else
           nacc = nacc+1
        end if
      end do
c --- check whether it's time to calculate energies and
c     send info back to the parent
      if (loop.eq.nprint) then

c ======================================================================
c       Calculate the energy
c ======================================================================
c ----- pot is the instantaneous "snapshot" potential energy

        potl=0.0d0

c ----- loop over all of the interacting pairs.
        do n=1, nvpair
           i=ivpair(1, n)
           j=ivpair(2, n)

           dx=(psi(3*j-2))+(-psi(3*i-2))+
     +        vpvec(1, n)
           dy=(psi(3*j-1))+(-psi(3*i-1))+
     +        vpvec(2, n)
           dz=(psi(3*j))  +(-psi(3*i))+
     +        vpvec(3, n)

           r2=dx*dx+dy*dy+dz*dz
c -------- compute the potential energy by interpolating between two grid
c          points.

           ibin=int((r2-r2min)*binvrs)+1

           if (ibin.gt.0) then
              dr=(r2-r2min)-bin*dble(ibin-1)
              p= v(1, ibin)+ v(2, ibin)*dr
              potl=potl+p
           else
              potl=potl+v(1, 1)
           end if

        end do
c ----- divide by 2 to get energy per atom
        pot = potl*0.5d0

c ----- tloc is the instantaneous "snapshot" kinetic energy

        do i=1, NATOM3
           dlng(i)=0.0
           d2lng(i)=0.0
        end do

c ----- first compute the one-atom contributions to the kinetic energy.
        do i=1, NATOMS

           xx=psi(3*i-2)
           yy=psi(3*i-1)
           zz=psi(3*i)

           dlng(3*i-2)= dlng(3*i-2)-2.0*aaxy*xx
           dlng(3*i-1)= dlng(3*i-1)-2.0*aaxy*yy
           dlng(3*i)  = dlng(3*i)-2.0*aaz*zz

           d2lng(3*i-2)= d2lng(3*i-2)-2.0*aaxy
           d2lng(3*i-1)= d2lng(3*i-1)-2.0*aaxy
           d2lng(3*i)  = d2lng(3*i)-2.0*aaz

        end do

c ----- loop over all interacting pairs.

        do n=1, nvpair

           i=ivpair(1, n)
           j=ivpair(2, n)

           dx=-((psi(3*j-2))+vpvec(1, n)+(-psi(3*i-2)))
           dy=-((psi(3*j-1))+vpvec(2, n)+(-psi(3*i-1)))
           dz=-((psi(3*j))  +vpvec(3, n)+(-psi(3*i))  )

           r2=dx*dx+dy*dy+dz*dz

           if (r2.le.0.0) write (6, *) 'i, j, r2 = ', i, j, r2

           br2=bb*bb/r2

           br5=br2*br2*sqrt(br2)

           dlng(3*i-2)=dlng(3*i-2)+2.5*br5*dx/r2
           dlng(3*i-1)=dlng(3*i-1)+2.5*br5*dy/r2
           dlng(3*i)  =dlng(3*i)  +2.5*br5*dz/r2

           d2lng(3*i-2)=d2lng(3*i-2)+2.5*br5*
     *                           (1.0-7.0*dx**2/r2)/r2
           d2lng(3*i-1)=d2lng(3*i-1)+2.5*br5*
     *                           (1.0-7.0*dy**2/r2)/r2
           d2lng(3*i)  =d2lng(3*i)  +2.5*br5*
     *                           (1.0-7.0*dz**2/r2)/r2

        end do
c ----- now add up all of the contributions to the kinetic energy.
        tloc=0.0

        do i=1, NATOM3
           tloc=tloc+d2lng(i)+dlng(i)**2
        end do

c ----- divide by (two times the mass) and negate the result.  this is
c       minus hbar squared divided by twice the mass

        tloc=-0.5*tloc/zmhe

        goto 200
      else
        goto 300
      end if

      end
```

allrep.f

```
c ----------------------------------------------------------------------

c     this subroutine distributes chunks of iterations to the child
c     processes, waits for them to be processed, and then returns
c     control to the main parent subroutine. Running averages are
c     also calculated and printed here, along with the snapshots.

c     errchk is a subroutine called after every MPI subroutine that
c     checks the MPI error code and reports any errors.

c ----------------------------------------------------------------------

      subroutine allrep(nsent, nrcvd, loops, MPI_R)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      include 'mpif.h'

      parameter (hart=315774.65d0)

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(3), emsg(2)
      dimension isent(NCHUNKS), psi(NATOM3)
      dimension rstate(8)

c --- loop over all chunks.

      MPI_R=MPI_DOUBLE_PRECISION

      loops = 0
      do nchunk=1, NCHUNKS

         if (idebug.eq.4)
     +      write (9, *) 'finding child who can receive chunk= ', nchunk
            call flush(9)
c ------ wait for data request from a child.

         call MPI_PROBE(MPI_ANY_SOURCE,
     +                  1201,
     +                  MPI_COMM_WORLD,
     +                  istat,
     +                  ierr)

         call errchk(0, ierr, 111201)

         nchild=istat(MPI_SOURCE)
         call MPI_RECV(imsg,
     +                 3,
     +                 MPI_INTEGER,
     +                 nchild,
     +                 1201,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(0, ierr, 151201)
         if (idebug.eq.4)
     +      write (9, *) 'sending chunk = ', nchunk, ' to ', nchild
            call flush(9)
c ------ check whether the child is returning results.  if so, then
c        receive the results.
```

```
         if (imsg(1).gt.0) then
            idchunk=imsg(1)
            if (idebug.eq.4)
     +         write (9, *) 'child ', nchild, ' returning chunk ',
     +                      idchunk
               call flush(9)
c --------- keep track of acceptances and rejections.

            ztacc=ztacc+imsg(2)
            ztrej=ztrej+imsg(3)
            loops = loops+nprint ! whenever a child returns, 50 loops done
            denom = denom+1.0d0
            call MPI_RECV(psi,
     +                    NATOM3,
     +                    MPI_R,
     +                    nchild,
     +                    1202,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111202)

c --------- receive updated random number state vector and energies
            call MPI_RECV(rstate,
     +                    8,
     +                    MPI_DOUBLE_PRECISION,
     +                    nchild,
     +                    1203,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111203)

            call MPI_RECV(emsg,
     +                    2,
     +                    MPI_DOUBLE_PRECISION,
     +                    nchild,
     +                    1204,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111204)
c --------- here, print out the snapshots as we get them, update running
c           average energies, and print out the energies to the ofile

            write (10) (psi(i), i=1, NATOM3)

            potl = emsg(1)
            tloc = emsg(2)

            vsum = vsum+potl
            v2sum = v2sum+potl*potl
            esum = esum+potl+tloc
            e2sum = e2sum+(potl+tloc)**2
            tsum = tsum+tloc
            t2sum = t2sum+tloc*tloc
c --------- calculate current averages and standard deviations

            uavg = vsum/denom
            u2avg = v2sum/denom
            usd = sqrt(u2avg-uavg*uavg)
            eavg = esum/denom
            e2avg = e2sum/denom
            esd = sqrt(e2avg-eavg*eavg)

            tavg = tsum/denom
            t2avg = t2sum/denom
            tsd = sqrt(t2avg-tavg*tavg)

c --------- Print energies for this snapshot to the ofile (also want to
c           inclue nacc, nrej, and nchild for this chunk)

            write(11, 8400) potl*hart/dble(NATOMS),
     +                     uavg*hart/dble(NATOMS),
     +                     usd*hart/dble(NATOMS),
     +                     eavg*hart/dble(NATOMS),
     +                     esd*hart/dble(NATOMS),
     +                     tavg*hart/dble(NATOMS),
     +                     tsd*hart/dble(NATOMS),
     +                     imsg(2), imsg(3), nchild

8400       format (7(1x, 1pe13.6), 1x, i7, 1x, i7, 1x, i3)

c --------- perform the calculations for reweighting
            nrw = 0
            rwaxy = aaxy-da
            rwaz = aaz-da
            rwb = bb-3.0d0*db

            do ii = 1, 9
               nrw = nrw+1
c ------------ Calculate 2*log(psi') and 2*log(psi) (called pnewsq
c              and poldsq, respectively).
               dx2sum=0.0d0
               dy2sum=0.0d0
               dz2sum=0.0d0
               do l=1, NATOMS
                  dx=psi(3*l-2)
                  dy=psi(3*l-1)
                  dz=psi(3*l)
                  dx2sum=dx2sum+dx*dx
                  dy2sum=dy2sum+dy*dy
                  dz2sum=dz2sum+dz*dz
               end do
c ------------ This is the one body contribution to 2*log(psi') and
c              2*log(psi):

               p1oldsq=-2.0d0*(aaxy*dx2sum+aaxy*dy2sum+aaz*dz2sum)
               p1newsq=-2.0d0*(rwaxy*dx2sum+rwaxy*dy2sum+rwaz*dz2sum)

c ------------ calcilate the same for the two body term:
               psi2old = 0.0d0
               psi2new = 0.0d0
               do n= 1, nvpair
                  j=ivpair(1,n)
                  m=ivpair(2,n)
                  if (m.gt.j) then
                     dx = (psi(3*m-2))+vpvec(1, n) +(-psi(3*j-2))
                     dy = (psi(3*m-1))+vpvec(2, n) +(-psi(3*j-1))
                     dz = (psi(3*m))+vpvec(3, n) +(-psi(3*j))

                     r2 = dx*dx+dy*dy+dz*dz
                     br2old = bb*bb/r2
                     br2new = rwb*rwb/r2
                     br5old = br2old*br2old*sqrt(br2old)
                     br5new = br2new*br2new*sqrt(br2new)

                     psi2old = psi2old - 0.5d0*br5old
                     psi2new = psi2new - 0.5d0*br5new
                  end if
               end do

               psi2oldsq = 2.0d0*psi2old
               psi2newsq = 2.0d0*psi2new
c ------------ Add one and two body terms for total 2*log(psi) and
c              2*log(psi')

               poldsq = p1oldsq + psi2oldsq
               pnewsq = p1newsq + psi2newsq

c ------------ calculate the reweighting factor, w = |psi'^2|/|psi^2|
c              = exp|2*log(psi') - 2*log(psi)|
               w=exp(pnewsq-poldsq)
               rw_wsum(ii) = rw_wsum(ii)+w
               rw_wsqsum(ii) = rw_wsqsum(ii) + (w*w)

c ------------ calculate the reweighted potential
               pot=w*potl
               rw_vsum(ii) = rw_vsum(ii)+pot
               rw_vwsum(ii) = rw_vwsum(ii) + (pot*w)
               rw_vvsum(ii) = rw_vvsum(ii) + (pot*pot)

c ----------- The subroutine kinrw(psi, tloc) calculates the kinetic
c             energy using the new parameters of psi prime.
              call kinrw(psi, rwaxy, rwaz, rwb, tloc)

              tloc = w*tloc
              rw_tsum(ii) = rw_tsum(ii)+tloc
              rw_twsum(ii) = rw_twsum(ii) + (tloc*w)
              rw_ttsum(ii) = rw_ttsum(ii) + (tloc*tloc)

c ----------- Calculate the total energy from the sum of potential and
c             kinetic.
              etot = pot + tloc
              rw_esum(ii) = rw_esum(ii) + etot
              rw_ewsum(ii) = rw_ewsum(ii) + (etot*w)
              rw_eesum(ii) = rw_eesum(ii) + (etot*etot)

c ----------- increment reweighting parameters (rwaxy incremeted in if
c             statement above)
              rwb = rwb + db
              if (mod(ii,3).eq.0) then
                  rwaxy = rwaxy + da
                  rwaz = aaz-da
              else
                  rwaz = rwaz+da
              end if
            end do

c --------- update the random number generator state vector for this
c           child.
            do i=1, 8
               rstatv(i, nchild)=rstate(i)
            end do

c --------- update the atom positions.

            do i=1, NATOM3
               path(i, nchild)=psi(i)
            end do

c --------- update the number of received chunks.

            nrcvd=nrcvd+1

c --------- indicate that this chunk has been processed and returned.

            isent(idchunk)=-nchild

         end if

c ------ send a new chunk to child.  first tell the child which chunk
c        it is going to receive.

         imsg(1)=nchunk

         call MPI_SEND(imsg,
     +                 1,
     +                 MPI_INTEGER,
     +                 nchild,
     +                 0204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 110204)

c ------ send the chunk.

         if (idebug.eq.4)
     +      write (9, *) 'calling send for child ', nchild
            call flush(9)
         call send(nchild, MPI_R)

         if (idebug.eq.4)
     +      write (9, *) 'chunk ', nchunk, ' sent to child ', nchild
            call flush(9)
c ------ update how many chunks have been sent.

         nsent=nsent+1

c ------ leave the trail of crumbs!

         isent(nchunk)=nchild

c ------ update how much work has been sent to this child.

         iwork(nchild)=iwork(nchild)+1

      end do

c --- at this point we don't have any more iterations to send to the
c     children, but we need to retrieve any processed iterations that the
c     children are still holding to send back to the parent.  this
c     flushes out all of those chunks.

      do i=1, NCHUNKS

         if (isent(i).gt.0) then

            nchild=isent(i)

            call MPI_RECV(imsg,
     +                    3,
     +                    MPI_INTEGER,
     +                    nchild,
     +                    1201,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 121201)

c --------- check whether the child is returning results.  if so, get
c           the results and update the atomic positions.

            if (imsg(1).gt.0) then

               idchunk=imsg(1)

               if (idebug.eq.4)
     +            write (9, *) 'child ', nchild,
     +                         ' returning chunk ', idchunk

c ------------ keep track of acceptances and rejections.

               ztacc=ztacc+imsg(2)
               ztrej=ztrej+imsg(3)
               loops  = loops+nprint
               denom = denom+1.0d0

               call MPI_RECV(psi,
     +                       NATOM3,
     +                       MPI_R,
     +                       nchild,
     +                       1202,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121202)

               call MPI_RECV(rstate,
     +                       8,
     +                       MPI_DOUBLE_PRECISION,
     +                       nchild,
     +                       1203,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121203)

               call MPI_RECV(emsg,
     +                       2,
     +                       MPI_DOUBLE_PRECISION,
     +                       nchild,
     +                       1204,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121204)

c --------- here, print out the snapshots as we get them, update running
c           average energies, and print out the energies to the ofile

            write (10) (psi(ii), ii=1, NATOM3)

            potl = emsg(1)
            tloc = emsg(2)
            vsum = vsum+potl
            v2sum = v2sum+potl*potl
            esum = esum+potl+tloc
            e2sum = e2sum+(potl+tloc)**2
            tsum = tsum+tloc
            t2sum = t2sum+tloc*tloc
c --------- calculate current averages and standard deviations

            uavg = vsum/denom
            u2avg = v2sum/denom
            usd = sqrt(u2avg-uavg*uavg)

            eavg = esum/denom
            e2avg = e2sum/denom
            esd = sqrt(e2avg-eavg*eavg)

            tavg = tsum/denom
            t2avg = t2sum/denom
            tsd = sqrt(t2avg-tavg*tavg)

c --------- Print energies for this snapshot to the ofile (also want to
c           inclue nacc, nrej, and nchild for this chunk)

            write(11, 8400) potl*hart/dble(NATOMS),
     +                     uavg*hart/dble(NATOMS),
     +                     usd*hart/dble(NATOMS),
     +                     eavg*hart/dble(NATOMS),
     +                     esd*hart/dble(NATOMS),
     +                     tavg*hart/dble(NATOMS),
     +                     tsd*hart/dble(NATOMS),
     +                     imsg(2), imsg(3), nchild

c --------- perform the calculations for reweighting
            rwaxy = aaxy-da
            rwaz = aaz-da
            rwb = bb-3.0d0*db

            do ii = 1, 9
               nrw = nrw+1
c ------------ Calculate 2*log(psi') and 2*log(psi) (called pnewsq
c              and poldsq, respectively).
               dx2sum=0.0d0
               dy2sum=0.0d0
               dz2sum=0.0d0
               do l=1, NATOMS
                  dx=psi(3*l-2)
                  dy=psi(3*l-1)
                  dz=psi(3*l)
                  dx2sum=dx2sum+dx*dx
                  dy2sum=dy2sum+dy*dy
                  dz2sum=dz2sum+dz*dz
               end do
c ------------ This is the one body contribution to 2*log(psi') and
c              2*log(psi):

               p1oldsq=-2.0d0*(aaxy*dx2sum+aaxy*dy2sum+aaz*dz2sum)
               p1newsq=-2.0d0*(rwaxy*dx2sum+rwaxy*dy2sum+rwaz*dz2sum)

c ------------ calcilate the same for the two body term:
               psi2old = 0.0d0
               psi2new = 0.0d0
               do n= 1, nvpair
                  j=ivpair(1,n)
                  m=ivpair(2,n)
                  if (m.gt.j) then
                     dx = (psi(3*m-2))+vpvec(1, n) +(-psi(3*j-2))
                     dy = (psi(3*m-1))+vpvec(2, n) +(-psi(3*j-1))
                     dz = (psi(3*m))+vpvec(3, n) +(-psi(3*j))

                     r2 = dx*dx+dy*dy+dz*dz
                     br2old = bb*bb/r2
                     br2new = rwb*rwb/r2

                     br5old = br2old*br2old*sqrt(br2old)
                     br5new = br2new*br2new*sqrt(br2new)

                     psi2old = psi2old - 0.5d0*br5old
                     psi2new = psi2new - 0.5d0*br5new
                  end if
               end do

               psi2oldsq = 2.0d0*psi2old
               psi2newsq = 2.0d0*psi2new
c ------------ Add one and two body terms for total 2*log(psi) and
c              2*log(psi')

               poldsq = p1oldsq + psi2oldsq
               pnewsq = p1newsq + psi2newsq

c ------------ calculate the reweighting factor, w = |psi'^2|/|psi^2|
c              = exp|2*log(psi') - 2*log(psi)|
               w=exp(pnewsq-poldsq)
               rw_wsum(ii) = rw_wsum(ii)+w
               rw_wsqsum(ii) = rw_wsqsum(ii) + (w*w)

c ------------ calculate the reweighted potential
               pot=w*potl
               rw_vsum(ii) = rw_vsum(ii)+pot
               rw_vwsum(ii) = rw_vwsum(ii) + (pot*w)
               rw_vvsum(ii) = rw_vvsum(ii) + (pot*pot)

c ----------- The subroutine kinrw(psi, tloc) calculates the kinetic
c             energy using the new parameters of psi prime.
              call kinrw(psi, rwaxy, rwaz, rwb, tloc)

              tloc = w*tloc
              rw_tsum(ii) = rw_tsum(ii)+tloc
              rw_twsum(ii) = rw_twsum(ii) + (tloc*w)
              rw_ttsum(ii) = rw_ttsum(ii) + (tloc*tloc)

c ----------- Calculate the total energy from the sum of potential and
c             kinetic.
              etot = pot + tloc
              rw_esum(ii) = rw_esum(ii) + etot
              rw_ewsum(ii) = rw_ewsum(ii) + (etot*w)
              rw_eesum(ii) = rw_eesum(ii) + (etot*etot)

c ----------- increment reweighting parameters (rwaxy incremeted in if
c             statement above)
              rwb = rwb + db
              if (mod(ii,3).eq.0) then
                  rwaxy = rwaxy + da
                  rwaz = aaz-da
               else
                  rwaz = rwaz+da
               end if
              end do

c ------------ update the random number generator state vector for this
```

```
c              child.

               do k=1, 8
                   rstatv(k, nchild)=rstate(k)
               end do

c ------------ update the atom positions for this child.

               do n=1, NATOM3
                  path(n, nchild)=psi(n)
               end do

c ------------ update the number of received chunks.

               nrcvd=nrcvd+1

c ------------ indicate that this chunk has been processed and returned.

               isent(idchunk)=-nchild
            end if

c --------- now tell the child to wait until all of the children are done
c           and more work is available.

            imsg(1)=-1

            call MPI_SEND(imsg,
     +                    1,
     +                    MPI_INTEGER,
     +                    nchild,
     +                    0204,
     +                    MPI_COMM_WORLD,
     +                    ierr)

            call errchk(0, ierr, 121204)

         end if

      end do
      return
      end
```

kinetic-rw.f

```
c ---------------------------------------------------------------------
c     This subroutine calculates the new kinetic energy of the new
c     wavefunction with a new set of axy, az, and b parameters using the
c     displacements from the old wavefunction, psi
c ---------------------------------------------------------------------
      subroutine kinrw(psi, axy, az, b, tloc)

      implicit real*8 (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'

      dimension psi(NATOM3), dlng(NATOM3), d2lng(NATOM3)

      do i=1, NATOM3
         dlng(i)=0.0
         d2lng(i)=0.0
      end do
```

```
c --- first compute the one-atom contributions to the kinetic energy.

      do i=1, NATOMS

         xx=psi(3*i-2)
         yy=psi(3*i-1)
         zz=psi(3*i)

         dlng(3*i-2)=dlng(3*i-2)-2.0*axy*xx
         dlng(3*i-1)=dlng(3*i-1)-2.0*axy*yy
         dlng(3*i)  =dlng(3*i)  -2.0*az*zz

         d2lng(3*i-2)=d2lng(3*i-2)-2.0*axy
         d2lng(3*i-1)=d2lng(3*i-1)-2.0*axy
         d2lng(3*i)  =d2lng(3*i)  -2.0*az

      end do

c --- loop over all interacting pairs.

      do n=1, nvpair

         i=ivpair(1, n)
         j=ivpair(2, n)

         dx=-((psi(3*j-2))+vpvec(1, n)+(-psi(3*i-2)))
         dy=-((psi(3*j-1))+vpvec(2, n)+(-psi(3*i-1)))
         dz=-((psi(3*j))  +vpvec(3, n)+(-psi(3*i))  )

         r2=dx*dx+dy*dy+dz*dz

         br2=b*b/r2

         br5=br2*br2*sqrt(br2)

         dlng(3*i-2)=dlng(3*i-2)+2.5*br5*dx/r2
         dlng(3*i-1)=dlng(3*i-1)+2.5*br5*dy/r2
         dlng(3*i)  =dlng(3*i)  +2.5*br5*dz/r2

         d2lng(3*i-2)=d2lng(3*i-2)+2.5*br5*
     *                          (1.0-7.0*dx**2/r2)/r2
         d2lng(3*i-1)=d2lng(3*i-1)+2.5*br5*
     *                          (1.0-7.0*dy**2/r2)/r2
         d2lng(3*i)  =d2lng(3*i)  +2.5*br5*
     *                          (1.0-7.0*dz**2/r2)/r2

      end do

c --- now add up all of the contributions to the kinetic energy.
      tloc=0.0

      do i=1, NATOM3
         tloc=tloc+d2lng(i)+dlng(i)**2
      end do

c --- divide by (two times the mass) and negate the result.  this is
c     minus hbar squared divided by twice the mass
      tloc=-0.5*tloc/zmhe

      return
      end
```

paramest.f

```
c=============================================================
c
c      This program determines the 6 unknown parameters
c      which define the predicted contour plots from
c      reweighting: Cxx, Cyy, Cxy, x, y, and E(x,y)
c      where x and y are the optimal aaxy and aaz values,
c      respectively.
c      The output is a gnuplot ".p" file that can be loaded
c      and used to set initial parameter values for fitting

c=============================================================
      subroutine param_est_aa(rw_sums)

      implicit real*8 (a-h, o-z)

      dimension rw_sums(4, 9)
      dimension p11(3), p12(3), p13(3), p21(3), p22(3), p23(3)
      dimension p31(3), p32(3), p33(3)
      dimension system(2,3)
c -----------------------------------------------------------

c --- This is set up to read the two paramester values and energy from a
c     3x3 (or 3x4) grid (4th column being std. dev. or confidence interval,
c     which is not included in these calculations). The two parameter values
c     are in the first two columns (corresponding to x and y) and the energy
c     is in the third.

      do i = 1, 3
         p11(i) = rw_sums(i+1, 1)
         p12(i) = rw_sums(i+1, 2)
         p13(i) = rw_sums(i+1, 3)
         p21(i) = rw_sums(i+1, 4)
         p22(i) = rw_sums(i+1, 5)
         p23(i) = rw_sums(i+1, 6)
         p31(i) = rw_sums(i+1, 7)
         p32(i) = rw_sums(i+1, 8)
         p33(i) = rw_sums(i+1, 9)
      end do
c --- Define fitting function for gnuplot
      open(17, file='paramest.p')
      write(17,*) "f(x,y) = cxx*(x-x0)**2+2*cxy*(x-x0)*(y-y0)+",
     +"cyy*(y-y0)**2+E0"

c --- store data to appropriate variables
      dEdx = (p32(3)-p12(3))/(p32(1)-p12(1))
      dEdy = (p23(3)-p21(3))/(p23(2)-p21(2))

      dEdx1 = (p32(3)-p22(3))/(p32(1)-p22(1))
      dEdx2 = (p22(3)-p12(3))/(p22(1)-p12(1))
      d2Edx2 =2.0d0*(dEdx1-dEdx2)/(p32(1)-p12(1))

      dEdy1 = (p23(3)-p22(3))/(p23(2)-p22(2))
      dEdy2 = (p22(3)-p21(3))/(p22(2)-p21(2))
      d2Edy2 =2.0d0*(dEdy1-dEdy2)/(p23(2)-p21(2))

      dEdx32 = (p33(3)-p13(3))/(p33(1)-p13(1))
      dEdx12 = (p31(3)-p11(3))/(p31(1)-p11(1))
      d2Edxdy = (dEdx32-dEdx12)/(p23(2)-p21(2))

      cxx = 0.5d0*d2Edx2
      cyy = 0.5d0*d2Edy2
      cxy = 0.5d0*d2Edxdy
      write(17, 700) cxx
      write(17, 710) cxy
      write(17, 720) cyy
700   format('cxx = ', f12.2)
710   format('cxy = ', f12.2)
720   format('cyy = ', f12.2)


c --- set up 2x2 system of equations as augmented matrix system and solve

      system(1,1) = 2.0d0*cxx
      system(1,2) = 2.0d0*cxy
      system(1,3) = 2.0d0*cxx*p22(1)+2.0d0*cxy*p22(2)-dEdx
      system(2,1) = 2.0d0*cxy
      system(2,2) = 2.0d0*cyy
      system(2,3) = 2.0d0*cxy*p22(1)+2.0d0*cyy*p22(2)-dEdy

c --- solve matrix system using gaussian elimination and backward substitution

c --- Gaussian Elimination:
      r = system(2,1)/system(1,1)
      system(2,2) = system(2,2)-r*system(1,2)
      system(2,3) = system(2,3)-r*system(1,3)

c --- Back substitution

      y = system(2,3)/system(2,2)
      x = (system(1,3)-system(1,2)*y)/system(1,1)

      write(17,750) x
      write(17,760) y
750   format('x0= ', f15.8)
760   format('y0= ', f15.8)
c --- Solve for the final unknown, E(x,y)

      a = p22(1)-x
      b = p22(2)-y

      Exy = p22(3)-cxx*a*a-cyy*b*b-2.0d0*cxy*a*b

      write(17, 800)  Exy
800   format('E0= ', 1pe13.6)
      close(17)
      end

c=============================================================
c
c      This program determines the 3 unknown parameters
c      which define the predicted quadratic fit from bb
c      reweighting: a, b, c
c      where b is the estimated new b-parameter
c      The output is a gnuplot ".p" file that can be loaded
c      and used to set initial parameter values for fitting

c=============================================================
      subroutine param_est_bb(rw_sums)

      implicit real*8 (a-h, o-z)

      dimension rw_sums(4, 9)
      dimension p1(2), p2(2), p3(2)

c --- Read in three data points from reweighting.

      p1(1) = rw_sums(1, 4)
      p2(1) = rw_sums(1, 5)
      p3(1) = rw_sums(1, 6)
      p1(2) = rw_sums(4, 4)
      p2(2) = rw_sums(4, 5)
      p3(2) = rw_sums(4, 6)

c --- Calculate first and second derivatives

      E = p2(2)
      dEdx = (p1(2)-p3(2))/(p1(1)-p3(1))

c --- second derivative:

      dEdx1 = (p1(2)-p2(2))/(p1(1)-p2(1))
      dEdx2 = (p2(2)-p3(2))/(p2(1)-p3(1))

      d2Edx2 = 2.0d0*(dEdx1-dEdx2)/(p1(1)-p3(1))

c --- Solve equations for function parameters

      a = d2Edx2/2.0d0

      b = (-dEdx+2.0d0*a*p2(1))/(2.0d0*a)

      c = p2(2) - a*((p2(1)-b)**2)

c --- Write out the results
      open(17, file='paramest.p')
      write (17, *) "f(x) = a*(x-b)**2+c"
      write (17, *) "a = ", a
      write (17, *) "b = ", b
      write (17, *) "c = ", c
      close(17)
      end
```

sizes.h

```
c --- number of atoms in the system.

      parameter (NATOMS=448)

c --- various multiples of NATOMS.

      parameter (NATOM3=NATOMS*3)
      parameter (NATOM6=NATOMS*6)
      parameter (NATOM7=NATOMS*7)

c --- number of points on the interatomic potential energy curve, for
c     linear interpolation of the potential energy function.

      parameter (NVBINS=20000)

c --- "radius" of the interacting-pair region, in nearest-neighbor distances.

      parameter (RATIO=2.05)

c --- number of interacting pairs for each atom.

      parameter (NIP=56)

c --- number of interacting trimers for each atom.
```

```
      parameter (NIT = 66)

c --- total number of interacting pairs in the simulation box.

      parameter (NPAIRS=NATOMS*NIP)

c --- Maximum number of child processors allowed at time of compilation

      parameter (NCH = 64)

c --- Number of chunks of iterations sent with each pass of the parent loop

      parameter (NCHUNKS = 1000)
```

vmc-448.com

```
c --- internal units are atomic units.
c --- hartrees per electron volt.

      parameter (evconv=3.67495735d-2)

c --- hartrees per wavenumber.

      parameter (cmconv=4.55636866d-6)

c --- QMC variables.

      common /monte/  ztacc, ztrej,
     +                zmh, zmhe, den, step,
     +                nloop, nequil, nprint,  nrst

      common /param/  bb, aaxy, aaz, dzscale, phi,
     +                eta, gam, eps, da, db, idebug

c --- random number variables.

      double precision zm1, zm2, rm1, rm2, rscale, rstatv

      common /moduli/ zm1, zm2, rm1, rm2

      common /rancom/ rstatv(8, NCH), rscale, irrst, nrskip

c --- potential energy curve

      common /potcom/ v(2, NVBINS)

c --- filenames.

      character*20 ltfile, pfile, sfile, ofile, dfile, dump,
     +             rwfile

      common /files/  ltfile, pfile, sfile, ofile, dfile, dump,
     +                rwfile

c --- crystal lattice.

      common /crystl/ xtal(NATOMS, 3), path(NATOM3, NCH),
     +                npair(NATOMS), ipairs(NIP, NATOMS)

      common /vpairs/ vpvec(3, NPAIRS), ivpair(2, NPAIRS),
     +                nvpair

c --- energy sums.
```

```
      common /energy_vec/ rw_vsum(9), rw_tsum(9), rw_esum(9),
     +                    rw_wsqsum(9), rw_vwsum(9), rw_ewsum(9),
     +                    rw_twsum(9), rw_vvsum(9), rw_eesum(9),
     +                    rw_ttsum(9), rw_wsum(9)

      common /energy/ vsum, v2sum, esum, e2sum, tsum, t2sum, denom

c --- counters to monitor load balancing.

      common /parcom/ iwork(NCH)
```

A.1.1 VMC Long-Range Correction Program

The program files necessary to compute the long-range correction to the potential energy as well as the mean squared displacement are included below. This version of the program takes the density, deformation parameters, number of snapshot files, number of MCCs used to generate each file, and the name of each snapshot file as input. In order to allow for the possibility of a distorted lattice, independent Gaussian parameters are calculated in the x and y directions, though for an ideal lattice they should be approximately the same. The program files required to compile and run this program are as follows:

msd.f	Main program that reads in the input data and the atomic snapshots and calculates the mean squared displacements and Gaussian parameters of the atomic distributions. This program also calls the lrc subroutine which calculates the long-range correction to the total energy.
lrc-3d-sub.f	Subroutine which calculates the contribution of region 1 and region 2 atoms to the long-range correction in an ideal or distorted lattice. Calculating the region 2 contribution requires calling the c6sub subroutine in c6-sub.f.
c6-sub.f	Contains the c6sub subroutine which calculates the infinite lattice sum S_6^* from the nearest neighbor distance and the deformation parameters. For an ideal lattice this returns the value published by Hirschfelder, Curtiss, and Bird. This calculation requires the file 'lattice-file-7920' which is a file containing the lattice positions of 7920 atoms in an hcp configuration.
Gauss-Hermite.dat	Data file containing the weights and abcissas for Gaussian quadrature calculations for different numbers of nodes.
sizes.h	See Sec. A.1.
vmc-448.com	See Sec. A.1.

msd.f

```
c =====================================================
c         This program calculates the mean
c         square displacement from the lattic
c         position for each atom in each direction
c
c         It also calculated the gaussian parameters
c         of the atomic probability densities and
c         calculates the long range correction to
c         the potential energy
c =====================================================

      implicit real*8 (a-h, o-z)
      real*8 lrctot
      include 'sizes.h'
      include 'vmc-448.com'

      dimension u(NATOM3), ux(NATOMS), uy(NATOMS), uz(NATOMS)
      parameter (bohr=0.529177249d0)
      character*30 file1, file2, file3, file4, file5

      common /files/  file1, file2, file3, file4, file5
c --- Read in density, number of files, number of MCCs per file
c     and distortion parameters eta gam and eps
      read(5, *) den
      read(5, *) eta
      read(5, *) gam
      read(5, *) eps
      read(5, *) nfiles
      read(5, *) nloops
55    format(a30)

      phi = sqrt(1.0d0+eta)
      ncalc = nloops/50
c --- use lattice parameters to calculate nearest neighbor distance for
c     long-range correction
      open (1, file="lattice-file-448")
      read (1, *) nat

      read (1, *) xlen, ylen, zlen

      den0=dble(NATOMS)/(xlen*ylen*zlen)

      xlen = xlen
      ylen = ylen
      zlen = zlen

c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=0.50d0*xlen
      dymax=0.50d0*ylen
      dzmax=0.50d0*zlen

      do i=1, NATOMS

         read (1, *) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (1)

c --- this helps us remember the nearest-neighbor distance.
c --- notice that the nearest neighbor distance is calculated
c     from the undistorted lattice as in the main program.
      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do
c --- compute interacting pairs from the atomic positions of the
c     undistorted lattice

      do i=1, NATOMS
         npair(i)=0
      end do

      nvpair=0

      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.
```

```
            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.
c --------- we determine the interacting pairs from the undistorted
c           then use our values of eta (phi), gamma, and epsilon to
c           impose the distortions for the elastic constant
c           calculations.
            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

c ------------ these transformations impose the lattice distortions
c              They reduce to dx, dy, and dz for eta = 0 (phi = 1),
c              gamma = 1 and epsilon = 0.

               vpvec(1, nvpair)=dx/(sqrt(gam)*phi)
               vpvec(2, nvpair)=dy*sqrt(gam)/phi
               vpvec(3, nvpair)=dz*phi**2+dy*eps

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair

            end if

         end if

      end do
      end do

c --- Now loop back through the coordinates in the xtal array and
c     transform them appropriately

      do i=1, NATOMS

         xtal(i, 1)=xtal(i, 1)/(sqrt(gam)*phi)
         xtal(i, 2)=xtal(i, 2)*sqrt(gam)/phi
         xtal(i, 3)=xtal(i, 3)*phi**2+eps*xtal(i, 2)

      end do
```

```
c --- initilize running totals
      u2xtot = 0.0d0
      u2ytot = 0.0d0
      u2xtot = 0.0d0

      u4xtot = 0.0d0
      u4ytot = 0.0d0
      u4ztot = 0.0d0


c --- Read in lattice positions from the snapshot files
      do k = 1, nfiles
         read (5, 55) file1
            open (1, file=file1, form='unformatted', status='old',
     +      access='sequential', recl=NATOM3*8)

         read(1) bb, aaxy, aaz


         u2x = 0.0d0
         u2y = 0.0d0
         u2z = 0.0d0

         u4x = 0.0d0
         u4y = 0.0d0
         u4z = 0.0d0

         do i = 1, ncalc
            read(1) (u(j), j=1, NATOM3)

            do l = 1, NATOMS
               u2x = u2x + (u(3*l-2)**2)
               u4x = u4x + (u(3*l-2)**4)

               u2y = u2y + (u(3*l-1)**2)
               u4y = u4y + (u(3*l-1)**4)

               u2z = u2z + (u(3*l)**2)
               u4z = u4z + (u(3*l)**4)
            end do
         end do

         u2xtot = u2xtot + u2x
         u2ytot = u2ytot + u2y
         u2ztot = u2ztot + u2z

         u4xtot = u4xtot + u4x
         u4ytot = u4ytot + u4y
         u4ztot = u4ztot + u4z

         close(1)
      end do

      u2xavg = u2xtot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))
      u4xavg = u4xtot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))

      u2yavg = u2ytot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))
      u4yavg = u4ytot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))

      u2zavg = u2ztot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))
      u4zavg = u4ztot/(dble(nfiles)*dble(ncalc)*dble(NATOMS))

      u2xsd = sqrt(u4xavg - (u2xavg)**2)
      u2ysd = sqrt(u4yavg - (u2yavg)**2)
      u2zsd = sqrt(u4zavg - (u2zavg)**2)

      u2xCI = 1.96d0*u2xsd/sqrt(dble(nfiles)*dble(ncalc)*dble(NATOMS))
      u2yCI = 1.96d0*u2ysd/sqrt(dble(nfiles)*dble(ncalc)*dble(NATOMS))
      u2zCI = 1.96d0*u2zsd/sqrt(dble(nfiles)*dble(ncalc)*dble(NATOMS))

c --- Use mean squared displacement to calculate the apparent gaussian
c     parameters and their uncertainties using propagation of error

      ax = 1.0d0/(4.0d0*u2xavg)
      ay = 1.0d0/(4.0d0*u2yavg)
      az = 1.0d0/(4.0d0*u2zavg)

      axsd = (1.0d0/(4.0d0*u2xavg*u2xavg))*u2xsd
      aysd = (1.0d0/(4.0d0*u2yavg*u2yavg))*u2ysd
      azsd = (1.0d0/(4.0d0*u2zavg*u2zavg))*u2zsd

c --- in order to allow for distorted lattice we allow ax and ay to
c     be used independently rather than combining them as before:
c      axy = (ax+ay)/2.0d0
c      axysd = sqrt((axsd**2+aysd**2)/2.0d0)

      write(6, 650) NATOMS, aaxy, aaz, bb, den, ax,
     +              axsd*1.96d0/sqrt(dble(ncalc*nfiles*NATOMS)),
     +              ay, aysd*1.96d0/sqrt(dble(ncalc*nfiles*NATOMS)),
     +              az, azsd*1.96d0/sqrt(dble(ncalc*nfiles*NATOMS))
650   format(1x, i3, 3(1x, f9.6), 1x, f10.8,
     +       3(4x, f10.7, 1x, '+/-', 1x, f10.7))

      b2 = bohr*bohr

      write(6, 300) u2xavg*b2, u2xsd*b2, u2xCI*b2
      write(6, 310) u2yavg*b2, u2ysd*b2, u2yCI*b2
      write(6, 320) u2zavg*b2, u2zsd*b2, u2zCI*b2

      write(6, 330) u4xavg/(u2xavg*u2yavg)
      write(6, 340) u4yavg/(u2yavg*u2yavg)
      write(6, 350) u4zavg/(u2zavg*u2zavg)

300   format('x: ', 3(1x, 1pe13.6), " Ang**2")
310   format('y: ', 3(1x, 1pe13.6), " Ang**2")
320   format('z: ', 3(1x, 1pe13.6), " Ang**2")
330   format('u4x/(u2x)^2 = ', 1pe13.6)
340   format('u4y/(u2y)^2 = ', 1pe13.6)
350   format('u4z/(u2z)^2 = ', 1pe13.6)

      call lrc(ax, ay, az, rnnmin, lrctot)
      write(6, 360) lrctot
360   format('V_lrc (K/atom) = ', 1pe13.6)

      end
```

lrc-3d-sub.f

```
c =========================================================
c    This subroutine takes the ax, ay, az, and rnnmin
c    parameters as arguments and returns the total long
c    range correction to the potential energy.
c =========================================================
      subroutine lrc(ax, ay, az, rnnmin, lrctot)

      implicit real*8 (a-h, o-z)
      real*8 lrctot
      include 'sizes.h'
      include 'vmc-448.com'

      dimension x(20), w(20)
      dimension znpvec(3, NATOMS), nonvec(NATOMS)

      parameter (hart = 315774.65d0)
      parameter (c6 = 1.461d0)
      parameter (c8 = 14.11d0)
      parameter (c10 = 183.5d0)
      parameter (pi = 3.14159265358979323846d0)

      phi = sqrt(1.0d0+eta)
c --- define number of nodes to use for gaussian quadrature. This was arrived at by
c     considering 2 up to 20 nodes and determining where the change becaem negligible

      nodes = 8
      nlskip = 29
      nread = 4

c --- Calculate the infinite lattice sum needed to calculate the
c     contribution from region 2 atoms.
      call c6sub(den, eta, gam, eps, c6hcb)
c --- read positions and weights from Guass-Hermite.dat

      open(1, file="Gauss-Hermite.dat", status="old")
      do i=1, nlskip
         read(1,*)
      end do

      do i=1, nread
         read(1,*) x(i), w(i)
         x(i+nread) = -1.0d0*x(i)
         w(i+nread) = w(i)
      end do
      close(1)
c --- from the lattice file, read in number of atoms, calculate which are
c     non-interacting, and store displacement vector. For this calculation,
c     we are centering on atom 1, so we need only to calculate the displacement
c     of all atoms from atom 1.

      nonpair = 0
      do j=2, NATOMS
            i=1
            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)
c --------- If we are dealing with a distorted lattice, we need to
c           "undistort" the dx, dy, and dz to get the proper noninteracting pairs.

            dx = dx*sqrt(gam)*phi
            dy = dy*phi/sqrt(gam)
            dz = (dz-dy*eps)/phi**2

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz
            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount, defined as RATIO times the nearest-
c           neighbor distance.

            if (r/rnnmin.ge.RATIO) then
               nonpair = nonpair+1
               nonvec(nonpair) = j
               znpvec(1, nonpair)=dx/(sqrt(gam)*phi)
               znpvec(2, nonpair)=dy*sqrt(gam)/phi
               znpvec(3, nonpair)=dz*phi**2+dy*eps
            end if
         end do

c --- Now we have our vector components from atom 1 to all of the remaining 448 atoms
c     with which it has only long range interactions. We can cycle through these to
c     calculate the total long range correction

c --- set up the seven nested do-loops. These will loop over u1x, u1y, u1z
c     u2x, u2y, and u2z. For each one we will calculate the C6/R^6, C8/R^8,
c     and C10/R^10 terms separately.

c --- calculate scaling factors for xy and z coordinates:
      factor = 1.0d0/(pi**3.0d0)
      ysc = 1.0d0/sqrt(2.0d0*ax)
      ysc = 1.0d0/sqrt(2.0d0*ay)
      zsc = 1.0d0/sqrt(2.0d0*az)
      ghvtot = 0.0d0
      R6nonpair = 0.0d0

      do ii = 1, nonpair
      Rx = znpvec(1, ii)
      Ry = znpvec(2, ii)
      Rz = znpvec(3, ii)

      id = nonvec(ii)

      fr6 = 0.0d0
      fr8 = 0.0d0
      fr10 = 0.0d0

      r2latt = Rx**2.0d0+Ry**2.0d0+Rz**2.0d0
      R6nonpair = R6nonpair + c6/(r2latt**3.0d0)
      dist = sqrt(r2latt)
      do i = 1, nodes
         u1x = x(i)
         w1x = w(i)
         do j = 1, nodes
            u2x = x(j)
            w2x = w(j)
            do k = 1, nodes
               u1y = x(k)
               w1y = w(k)
```

```
                  do l = 1, nodes
                     u2y = x(l)
                     w2y = w(l)
                     do m = 1, nodes
                        u1z = x(m)
                        w1z = w(m)
                        do n = 1, nodes
                           u2z = x(n)
                           w2z = w(n)

                           r2x = (Rx+u2x*xsc-u1x*xsc)**2.0d0
                           r2y = (Ry+u2y*ysc-u1y*ysc)**2.0d0
                           r2z = (Rz+u2z*zsc-u1z*zsc)**2.0d0

                           rr2 = r2x+r2y+r2z
                           r6 = rr2*rr2*rr2
                           r8 = r6*rr2
                           r10 = r8*rr2
                           wtot = w1x*w2x*w1y*w2y*w1z*w2z

                           fr6 = fr6+wtot*c6/r6
                           fr8 = fr8+wtot*c8/r8
                           fr10 = fr10+wtot*c10/r10
                         end do
                     end do
                 end do
             end do
         end do
      end do
      vcor = factor*(fr6+fr8+fr10)
      ghvtot = ghvtot+vcor
      end do

c --- to calculate LRC beyond the 448 atoms, we reference Hirschfelder, Curtiss and Bird
c     who tabulated C6 contribution for the potential energy of a crystal as the sum over
c     an infinite number of atoms. The contribution from the 448 atoms we have considered
c     in the initial energy calculation and the gauss-hermite integration above must be
c     subtracted from this term.
      r6ipairs = 0.0d0

      do i = 1, npair(1)
         rxnew = vpvec(1, i)
         rynew = vpvec(2, i)
         rznew = vpvec(3, i)
         r2lattnew = rxnew**2.0d0+rynew**2.0d0+rznew**2.0d0
         r6ipairs = r6ipairs+1.0d0/(r2lattnew*r2lattnew*r2lattnew)
      end do

      region1 = (r6ipairs*c6)+R6nonpair
      vcoradd = c6hcb*c6/(rnnmin**6.0d0)-region1
c --- This is works for an ideal crystal only. For non-ideal c/a ratios, we need to
c     adjust c6hcb.
      lrctot = -0.5d0*(ghvtot+vcoradd)*hart
      return
      end
```

c6-sub.f

```
c ======================================================

c   This program calculates the sum of the 1/R^6 terms
c   of all pairs contained within a distorted sphere of a
c   rad=60.2 centered on atom 1. The truncated sums
```

```
c   are corrected to the hypothesized exact value using the
c   previously determined corrections from the ideal crystal
c   (based on the exact sum from Hirschfelder, Curtiss, and Bird).

c   This version accounts for the three typs of distortion
c   needed to calculated the elastic constants, governed by
c   epsilon, gamma, and eta.

c =======================================================
      subroutine c6sub (den, eta, gam, eps, c6_param)
      implicit real*8 (a-h, o-z)

c --- set up a larger xtal array than is used in the main program
c     this allows us to use the 7920 atom lattice file
      real*8 xtal(7920, 3)

c --- Scale the radius of interest (should still give us 3838 pairs)
      den1 = 0.0041896d0 !reference density where 60.2 radius was used
      radius = 60.2d0*exp(dlog(den1/den)/3.0d0)
c --- initialize some values we will need
      phi = sqrt(1.0d0+eta)

      error = 0.063761d0/100.0d0 !error from ideal study at this radius
      corr = 1.0d0/(1.0d0-error) !correction factor

c --- read the edge lengths of the supercell and scale them
c     based on the density provided.
      open (1, file="lattice-file-7920")
      read (1, *) NATOMS

      read (1, *, err=903) xlen, ylen, zlen

      den0=dble(NATOMS)/(xlen*ylen*zlen)

      xlen = xlen
      ylen = ylen
      zlen = zlen

c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=0.50d0*xlen
      dymax=0.50d0*ylen
      dzmax=0.50d0*zlen

      do i=1, NATOMS

         read (1, *, err=904) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (1)

c --- this helps us remember the nearest-neighbor distance.
c --- notice that the nearest neighbor distance is calculated
c     from the undistorted lattice as in the main program.
```

```
      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do
      rnn2 = rnnmin*rnnmin
      rnn6 = rnn2*rnn2*rnn2
c --- compute interacting pairs.
      nvpair = 0
      c6_param = 0.0d0
      do j=2, NATOMS
         i=1

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- we determine the interacting pairs from the undistorted lat.
c           then use our values of eta (phi), gamma, and epsilon to
c           impose the distortions for the elastic constant
c           calculations.
             if (r.lt.radius) then
               nvpair = nvpair+1
               dxnew=dx/(sqrt(gam)*phi)
               dynew=dy*sqrt(gam)/phi
               dznew=dz*phi**2+dy*eps

               r2 = dxnew*dxnew+dynew*dynew+dznew*dznew
               r6 = r2*r2*r2
               r6i = 1.0d0/r6
               c6_param = c6_param+rnn6*r6i
             end if
          end if
      end do

      if(nvpair.ne.3838) goto 905

      c6_param = c6_param*corr
      return

903   write(6, *) "Error reading side lengths"
904   write(6, *) "Error reading atomic positions"
905   write(6, *) "Error: Not enough pairs considered in C6 calc"
      return
      end
```

Gauss-Hermite.dat

```
        +/- x_i                        w_i
n=2
        0.707106781186548        8.862269254528E-01

n=3
        0.000000000000000        1.181635900604E+00
        1.224744871391589        2.954089751509E-01

n=4
        0.524647623275290        8.049140900055E-01
        1.650680123885785        8.131283544725E-02

n=5
        0.000000000000000        9.453087204829E-01
        0.958572464613819        3.936193231522E-01
        2.020182870456086        1.995324205905E-02

n=6
        0.436077411927617        7.246295952244E-01
        1.335849074013697        1.570673203229E-01
        2.350604973674492        4.530009905509E-03

n=7
        0.000000000000000        8.102646175568E-01
        0.816287882858965        4.256072526101E-01
        1.673551628767471        5.451558281913E-02
        2.651961356835233        9.717812450995E-04

n=8
        0.381186990207322        6.611470125582E-01
        1.157193712446780        2.078023258149E-01
        1.981656756695843        1.707798300741E-02
        2.930637420257244        1.996040722114E-04

n=9
        0.000000000000000        7.202352156061E-01
        0.723551018752838        4.326515590026E-01
        1.468553289216668        8.847452739438E-02
        2.266580584531843        4.943624274437E-03
        3.190993201781528        3.960697726326E-05

n=10
        0.342901327223705        6.108626337353E-01
        1.036610829789514        2.401386110823E-01
        1.756683649299882        3.387439445548E-02
        2.532731674232790        1.343645746781E-03
        3.436159118837738        7.640432855233E-06

n=12
        0.314240376254359        5.701352362625E-01
        0.947788391240164        2.604923102642E-01
        1.597682635152605        5.160798561588E-02
        2.279507080501060        3.905390584629E-03
        3.020637025120890        8.573687043588E-05
        3.889724897869782        2.658551684356E-07

n=16
        0.27348104613815        5.079294790166E-01
        0.82295144914466        2.806474585285E-01
        1.38025853919888        8.381004139899E-02
        1.95178799091625        1.288031153551E-02
        2.54620215784748        9.322840086242E-04
        3.17699916197996        2.711860092538E-05
        3.86944790486012        2.320980844865E-07
        4.68873893930582        2.654807474011E-10

n=20
        0.2453407083009        4.622436696006E-01
        0.7374737285454        2.866755053628E-01
        1.2340762153953        1.090172060200E-01
        1.7385377121166        2.481052088746E-02
        2.2549740020893        3.243773342238E-03
        2.7888060584281        2.283386360163E-04
        3.3478545673832        7.802556478532E-06
        3.9447640401156        1.086069370769E-07
        4.6036824495507        4.399340992273E-10
        5.3874808900112        2.229393645534E-13
```

A.2 VMC Perturbative 3-body Correction Program (VMC(3B))

The VMC(3B) program reported below calculates the perturbative three-body correction to the VMC-2B energies from six representative ^{4}He trimers. This approach assumes an ideal crystal. In the case of distorted lattices, each of the 66 trimers formed by a central atom and two of its nearest neighbors should be accounted for individually. The three-body correction calculation reads in atomic snapshots from the unformatted VMC snapshot files, taking the density, number of files, MCCs per file, and names of the snapshot files as input. This is a serial program that requires the following files to compile and run:

3body-cencek.f	Performs the VMC perturbative three-body energy calculation by calculating the new trimer side lengths of each of the six representative trimers from the VMC snapshots. The three-body energy is calculated by calling the He3 subroutine in he3fci.f.
he3fci.f	Calculates the Cencek nonadditive potential and can be found in the supplementary material of the original publication (Wojciech Cencek, Konrad Patkowski, and Krzysztof Szalewicz, *J. Chem. Phys.*, 131(6), 2009). It calls on data in the file 'E3.dat' which is also found in the supplementary material. These files are not reproduced below.
trimers.dat	Data file containing the atoms in each representative trimer in the first 3 columns, and an integer corresponding to the central angle in the fourth column.
sizes.h	See Sec. A.1.
vmc-448.com	See Sec. A.1.

3body-cencek.f

```
c ==============================================
c   This program takes an unformatted snapshot
c   file and calculates the three body energy by
c   averaging over one representation of each of
c   the six nearest neighbor trimer geometries
c   using atom 1 as the central atom
c
c   The point of this program is to calculate the
c   perturbative three-body energy
c ==============================================

      implicit real*8 (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'

      dimension psi(NATOM3), ntrimer(6, 4)
      dimension vect(6, 6)
      parameter (hart=315774.65d0)
c --- define common block files names

      character*30 snapfile
      common /files/ snapfile

c --- Read in central atom, input file names, one with x,y,z vectors from
c     central atom, one with trimers and central angles.

      read (5, *) den
      read (5, *) nfiles ! number of snapshot files to read
      read (5, *) nloop !number of MCCs used in VMC simulation
100   format(a30)

      ncalc = nloop/50

c --- First define the trimers we are using for each geometry
c     to simplify we will specify each geometry by an integer
c     based on the central angle: 60 = 1, 90 = 2, 109.4 = 3,
c     120 = 4, 146.4 = 5, 180 = 6.
c     we read these in from a timer file. The first three columns
c     in the file are the atoms in the trimer, the next is the
c     integer specifying the central angle

      open(1, file='trimers.dat')
      do i=1, 6
         read(1, *) (ntrimer(i, j), j=1,4)
      end do

c --- Open the lattice file, scale, and read in atomic positions.

      open(2, file = "lattice-file-448", status="old")
      read(2, *) nlpts
      read(2, *) xlen, ylen, zlen

      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      scale=exp(dlog(den/den0)/3.0d0)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

c --- these are the maximum distance in the x, y, or z directions before
c     we have to invoke periodic boundary conditions.

      dxmax=0.5d0*xlen
      dymax=0.5d0*ylen
      dzmax=0.5d0*zlen

c --- read in the lattice points and scale them.

      do i=1, NATOMS

         read (2, *) xtal(i, 1), xtal(i, 2), xtal(i, 3)
             xtal(i, 1)=xtal(i, 1)/scale
             xtal(i, 2)=xtal(i, 2)/scale
             xtal(i, 3)=xtal(i, 3)/scale
      end do

      close (2)

c --- this helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

c ------ update the minimum nearest-neighbor distance if needed.

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

c --- Calculate vectors from atom 1 to each atom in each trimer

      do i=1, 6
         j = ntrimer(i, 2)
         k = ntrimer(i, 3)

         dx1=xtal(j, 1)-xtal(1, 1)
         dy1=xtal(j, 2)-xtal(1, 2)
         dz1=xtal(j, 3)-xtal(1, 3)

         dx2=xtal(k, 1)-xtal(1, 1)
         dy2=xtal(k, 2)-xtal(1, 2)
         dz2=xtal(k, 3)-xtal(1, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx1.gt.dxmax) then
               dx1=dx1-xlen
            else if (dx1.lt.-dxmax) then
               dx1=dx1+xlen
            end if
            if (dy1.gt.dymax) then
               dy1=dy1-ylen
            else if (dy1.lt.-dymax) then
               dy1=dy1+ylen
            end if

            if (dz1.gt.dzmax) then
               dz1=dz1-zlen
            else if (dz1.lt.-dzmax) then
               dz1=dz1+zlen
            end if


            if (dx2.gt.dxmax) then
               dx2=dx2-xlen
            else if (dx2.lt.-dxmax) then
               dx2=dx2+xlen
            end if

            if (dy2.gt.dymax) then
               dy2=dy2-ylen
            else if (dy2.lt.-dymax) then
               dy2=dy2+ylen
            end if

            if (dz2.gt.dzmax) then
               dz2=dz2-zlen
            else if (dz2.lt.-dzmax) then
               dz2=dz2+zlen
            end if

c ------ Save the vectors to trimer atoms in the vect vector

         vect(i, 1) = dx1
         vect(i, 2) = dy1
         vect(i, 3) = dz1
         vect(i, 4) = dx2
         vect(i, 5) = dy2
         vect(i, 6) = dz2

      end do

c --- initialize running totals
      potl3b = 0.0d0
      potl3bsq = 0.0d0

      do ii = 1, nfiles
         read (5, 100) snapfile
         open (3, file=snapfile, form='unformatted',
     +      status='old', access='sequential', recl=NATOM3*8)
         read(3) bb, aaxy, aaz
c --- now for each snapshot, loop over all trimers, calculate new
c     distance and get three body energy
       do i = 1, ncalc
         read(3) (psi(j), j = 1, NATOM3)
c --- For each snapshot, reset the running total for each trimer type
c     but not for the total 3body energy.
         tri1=0.0d0
         tri2=0.0d0
         tri3=0.0d0
         tri4=0.0d0
         tri5=0.0d0
         tri6=0.0d0
         do l = 1, 6
```

```
            n1 = ntrimer(l, 2)
            n2 = ntrimer(l, 3)

c --- Now that the three atoms have been identified
c     (the central atom, ncent, was specified in the
c     input), we can calculate the new side lengths
c     starting from the vectors. Distances between
c     central atoms and nearest neighbors are labeled
c     with 01 or 02, third distance is labeled 12

              dx01 = vect(l, 1)+psi(3*n1-2)-psi(1)
              dy01 = vect(l, 2)+psi(3*n1-1)-psi(2)
              dz01 = vect(l, 3)+psi(3*n1)-psi(3)
              side1 = sqrt(dx01*dx01+dy01*dy01+dz01*dz01)

              dx02 = vect(l, 4)+psi(3*n2-2)-psi(1)
              dy02 = vect(l, 5)+psi(3*n2-1)-psi(2)
              dz02 = vect(l, 6)+psi(3*n2)-psi(3)
              side2 = sqrt(dx02*dx02+dy02*dy02+dz02*dz02)

              dx12 = vect(l, 4)+psi(3*n2-2)-vect(l, 1)-psi(3*n1-2)
              dy12 = vect(l, 5)+psi(3*n2-1)-vect(l, 2)-psi(3*n1-1)
              dz12 = vect(l, 6)+psi(3*n2)-vect(l, 3)-psi(3*n1)
              side3 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)

c ------------ calculate instantaneous triangles
               call He3(side1, side2, side3, E3)
               if(ntrimer(l, 4).eq.1) tri1=tri1+E3
               if(ntrimer(l, 4).eq.2) tri2 = tri2+E3
               if(ntrimer(l, 4).eq.3) tri3 = tri3+E3
               if(ntrimer(l, 4).eq.4) tri4 = tri4+E3
               if(ntrimer(l, 4).eq.5) tri5 = tri5+E3
               if(ntrimer(l, 4).eq.6) tri6 = tri6+E3
         end do
c ------ There are 24 tri1, 12 tri2, 3 tri3, 18 tri4,
c        6 tri5, and 3 tri6. We need to also divide the energy
c        of each equilateral trimer by 3 bc of triple counting.

         pot = 8.0d0*tri1+12.0d0*tri2+3.0d0*tri3+18.0d0*tri4+6.0d0*tri5+
     +         3.0d0*tri6
         pot2 = pot*pot

c ------ keep track of the running total of the 3 body energy and the
c        <3BE^2> to calculate standard deviation
         potl3b = potl3b+pot
         potl3bsq = potl3bsq+pot2
      end do
      close(3)
      end do

      potl3b = potl3b*hart/dble(nfiles*ncalc)
      potl3bsq = potl3bsq*hart*hart/dble(nfiles*ncalc)

      sd = sqrt(potl3bsq-potl3b*potl3b)
      write(6, *) 'Average three-body energy from central atom : ',
     +  '1 at density: ', den
      write(6, *)
      write(6, 9500) potl3b, sd, 1.96*sd/sqrt(dble(4*ncalc))
9500  format('Avg 3B energy: ', 1pe13.6, 1x, 'St. Dev.: ',
     +      1pe13.6, 1x, ' 95% CI: +/- ', 1pe13.6, 1x)

999   stop
      end
```

A.3 VMC Fully-Incorporated 3-body Program (VMC+3B)

The VMC+3B version of the VMC program incorporates three-body interactions by calculating the three-body potential from all interacting trimers whenever snapshots of atomic positions are recorded and was adapted from the VMC-2B program described above. Many of the subroutines are unchanged. The current version of this code does not allow for non-ideal lattice geometries, however it can be easily modified to accept and implement nonequilibrium values of η, γ, and ϵ following the implementation in the VMC-2B code. The files required to compile and run this program are as follows:

main.f	See Sec. A.1.
cmrg.f	See Sec. A.1.
rsetup.f	See Sec. A.1.
parent-3b.f	Replaces parent.f in Sec. A.1. In addition to the tasks in parent.f, parent-3b.f also sets up the interacting trimer list.
input-3b.f	Replaces input.f in Sec. A.1. Reads in the input and output file names, debugging level, and simulation parameters, but is not currently formatted to read deformation parameters.
vinit.f	See Sec. A.1.
he3fci.f	See Sec. A.2.
child-3b.f	Replaces child.f in Sec. A.1. In addition to the tasks of child.f, child-3b.f calculates the instantaneous three-body energy and sends this back to the parent.
allrep-3b.f	Replaces allrep.f in Sec. A.1. In addition to the tasks of allrep.f, allrep-3b.f includes the three-body potential energy in the total energy and reweighting calculations.
send.f	See Sec. A.1.
kinetic-rw.f	See Sec. A.1.
paramest.f	See Sec. A.1.
tstamp.f	See Sec. A.1.
sizes.h	Replaces sizes.h in Sec. A.1. This version of sizes.h also includes parameters necessary for calculating the total number of interacting trimers.
vmc-448.com	Replaces vmc-448.com in Sec. A.1. This version of vmc-448.com also includes all parameters related to the interacting trimers.

Program files that can be found in the QSATS code and are not reproduced here: **main.f**, **cmrg.f**, **vinit.f**, **send.f**, **tstamp.f**

Each subroutine that differs from the VMC-2B program in Sec. A.1 is reproduced in its entirety below.

parent-3b.f

```
c ----------------------------------------------------------------------
c     this is the parent process that runs on node 0.

c     errchk is a subroutine called after every MPI subroutine that
c     checks the MPI error code and reports any errors.
c     This version sets up the interacting trimer list to account for
c     three-body interactions.
c ----------------------------------------------------------------------

      subroutine parent(ierror)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      include 'mpif.h'

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(11), fmsg(7)
      dimension isent(NCHUNKS), psi(NATOM3)
      dimension rw_sums(4, 9)
      dimension rstate(8)

c --- hart = conversion from hartree to K/atom
      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)
      parameter (hart=315774.65d0)
c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================
      ierror=0

c --- read input file.
      write (6, *) "parent calling input"
      call input

      write (6, 6100) ltfile, dump, dfile, ofile
6100  format ('lattice file name  = ', a16/,
     +        'snapshot file name = ', a16/,
     +        'dfile file name     = ', a16/,
     +        'ofile file name     = ', a16/)

      if (idebug.eq.0) write (6, 6110) idebug, 'NONE'
      if (idebug.eq.1) write (6, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (6, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (6, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (6, 6110) idebug, 'HIGH'

6110  format ('debug level = ', i1,' or ', a8/)

c --- read the potential energy curve.

      call vinit(r2min, bin)

c --- read crystal lattice points.
```

```
      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a16/)

      open (8, file=ltfile, status='old', err=901)
      read (8, *, err=902) nlpts

      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         call quit
      end if

c --- read the edge lengths of the supercell.

      read (8, *, err=903) xlen, ylen, zlen
      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *, err=904) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

c --- this helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6310) rnnmin
6310  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

      write (6, 6330) xlen, ylen, zlen
6330  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
      end do

      nvpair=0
      nvpair1=0
      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if
            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.

            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

               vpvec(1, nvpair)=dx
               vpvec(2, nvpair)=dy
               vpvec(3, nvpair)=dz

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair
c ------------ for three-body calculations, keep track of only first
c              nearest neighbors.
               if (r/rnnmin.lt.1.05) then

               nvpair1=nvpair1+1

c ------------ store information about this pair (i->j) in arrays.

               ivpair1(1, nvpair1)=i
               ivpair1(2, nvpair1)=j

               vpvec1(1, nvpair1)=dx
               vpvec1(2, nvpair1)=dy
               vpvec1(3, nvpair1)=dz

               npair1(i)=npair1(i)+1

               ipairs1(npair1(i), i)=nvpair1
               end if

            end if

         end if

      end do
      end do

c --- To save time later, we are going to calculate all of the first
c     nearest neighbor trimers now, along with angles and vectors
      nvtrim=0

      do i = 1, NATOMS
        ntrim(i) = 0
        do j = 1, npair1(i)-1
        do k = j+1, npair1(i)
c--------- keep running count of all trimers
          nvtrim = nvtrim+1
c -------- record vector from central atom to two neighbors
```

```
           npairA = ipairs1(j, i) !tells us the nvpair reference number
           npairB = ipairs1(k, i) ! for each of the atoms in the trimer
c--------- keep record of atoms in trimer
           ivtrim(1, nvtrim) = i !central atom
           ivtrim(2, nvtrim) = ivpair1(2, npairA)
           ivtrim(3, nvtrim) = ivpair1(2, npairB)

           vpvectri(1, nvtrim) = vpvec1(1, npairA)
           vpvectri(2, nvtrim) = vpvec1(2, npairA)
           vpvectri(3, nvtrim) = vpvec1(3, npairA)

           vpvectri(4, nvtrim) = vpvec1(1, npairB)
           vpvectri(5, nvtrim) = vpvec1(2, npairB)
           vpvectri(6, nvtrim) = vpvec1(3, npairB)
c -------- calculate and store side lenghts and central angle
           dx1 = vpvectri(1, nvtrim)
           dy1 = vpvectri(2, nvtrim)
           dz1 = vpvectri(3, nvtrim)

           dx2 = vpvectri(4, nvtrim)
           dy2 = vpvectri(5, nvtrim)
           dz2 = vpvectri(6, nvtrim)

           dx12 = dx2-dx1
           dy12 = dy2-dy1
           dz12 = dz2-dz1

           side1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
           side2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
           side3 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)

           vpvectri(7, nvtrim) = side1
           vpvectri(8, nvtrim) = side2
           vpvectri(9, nvtrim) = side3
c -------- We know sides 1 and 2 = Rnn. If side 3 is lt or
c          equal to Rnn, trimer will be triple counted
           ivtrim(4, nvtrim) = 1
           if((side3/Rnnmin).lt.1.05) ivtrim(4, nvtrim) = 3

c -------- Update number of trimers for given central atom
           ntrim(i) = ntrim(i)+1

           itrims(ntrim(i), i) = nvtrim
        end do
        end do
      end do

c --- write out interacting pair and interacting trimer information
      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +        'total number of interacting pairs = ', i6)

      write (6, 6403) ntrim(1), nvtrim
6403  format ('atom 1 forms ', i3, ' trimers with interacting
     + neighbors'//, ' total number of interacting trimers = ', i9)

      if (idebug.ge.2) then

         write (6, 6401)
6401     format (/'interaction pair vectors for atom 1 ',
     +            '[NN distances]:'/)

         do i=1, npair(1)
            ip=ipairs(i, 1)
            d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +        rnnmin
            write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                      vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410        format ('vector # ', i3, ' to atom ', i4, ': ',
     +              3(1x, f9.5), ' length = ', f8.5)
         end do

         write (6, 6402)
6402     format (/'interaction trimer side lengths for atom 1 ',
     +           '[NN distances]:'/)
         do i=1, ntrim(1)
            itri = itrims(i, 1)
            write(6, 6411) itri, ivtrim(2, itri), ivtrim(3, itri),
     +                     vpvectri(7, itri)/rnnmin, vpvectri(8,
     +                     itri)/rnnmin, vpvectri(9, itri)/rnnmin,
     +                     ivtrim(4, itri)
6411        format('trimer # ', i3, 'incuding atoms ', i4, 1x, i4,': ',
     +           'side lengths: ', 3(1x, f8.5), ' counted: ',
     +            i1, 1x, 'times')
         end do

      end if

c --- set the displacement vectors for all children to zero.

      write (6, 6500)
6500  format (/'SETTING initial configuration to zero'/)

      do j=1, NCH
      do i=1, NATOM3
         path(i,j)=0.0d0
      end do
      end do
c --- initialize random number generator.
      call rsetup
c --- this is the output file where snapshots of the atoms will be
c     stored for analysis by another program.

      open (10, file=dump, form='unformatted')
c --- this is the output file where the instantaneous potential
c     energy and running averages of all energies are stored.

c --- initialize MPI.

      MPI_R=MPI_DOUBLE_PRECISION

      call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)

      call errchk(0, ierr, 100000)

      write (6, 6600) ntasks-1
6600  format ('number of child processes = ', i3/)

c --- now see if there is an old set of displacement vectors from a
c     previous run.  if not, jump head to line 200.

      if (irrst.eq.1) then
         open (8, file=dfile, form='unformatted', status='old', err=200)

         write (6, 6510) dfile
6510     format ('READING initial configuration from ', a16/)

         read (8) nchildren ! make sure we have the same number as before
         do k = 1, nchildren
```

```
            read (8) (rstatv(i, k), i=1, 8)
            read (8) (path(i,k), i=1, NATOM3)
         end do
         close (8)
      end if

200   if (idebug.ge.3) then

         write (6, 6170)
6170     format ('x(1) and rstatv(1) values:'/)

         do k = 1, ntasks-1
            write (6, 6180) path(1,k), rstatv(1,k)
6180     format (1x, f15.9, 1x, f20.1)

         end do

         write (6, *) ''

      end if

      if(irrst.eq.1) then
         if(nchildren.ne.ntasks-1) then

         write (6, 6605) nchildren+1
6605     format ('attempting to start calculation from previous run'/
     +           'with a different number of child processors than'/
     +           'the current run. To start from these snapshots, use',
     +           1x, i3, 1x, 'processors.')
         end if
      end if

      if (ntasks-1.gt.NCH) then

         write (6, 6610)
6610     format ('too many child processes; expand the iwork, path, and
     +rstatv arrays.'/
     +           'also note that write statements for HIGH '
     +           'debugging level may fail on some systems.')

         call quit

      end if
c --- this array just counts how evenly the workload was spread among
c     the child processes.

      do i=1, ntasks-1
         iwork(i)=0
      end do
c --- broadcast integer constants to all child processes.

      imsg(1)=NATOMS
      imsg(2)=NATOM3
      imsg(3)=NATOM6
      imsg(4)=NATOM7
      imsg(5)=NIP
      imsg(6)=NPAIRS
      imsg(7)=NIT
      imsg(8)=NTRIMS
      imsg(9)=NVBINS
      imsg(10)=idebug
      imsg(11)=nprint

      do itask=1, ntasks-1
```

```
         call MPI_SEND(imsg,
     +                 11,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0101,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100101)

      end do

      if (idebug.ne.0) open (9, file='debug.log')

      if (idebug.eq.1) write (9, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (9, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (9, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (9, 6110) idebug, 'HIGH'
      call flush(9)
c --- broadcast floating-point constants to all child processes.

      fmsg(1)=den
      fmsg(2)=bin
      fmsg(3)=r2min
      fmsg(4)=aaxy
      fmsg(5)=aaz
      fmsg(6)=bb
      fmsg(7)=zmhe


      do itask=1, ntasks-1

         call MPI_SEND(fmsg,
     +                 7,
     +                 MPI_R,
     +                 itask,
     +                 0102,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100102)

      end do
c --- broadcast the interacting-pair vectors to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(vpvec,
     +                 3*NPAIRS,
     +                 MPI_R,
     +                 itask,
     +                 0103,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100103)

      end do

c --- broadcast the interacting-trimer vectors to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(vpvectri,
     +                 9*NTRIMS,
     +                 MPI_R,
     +                 itask,
     +                 0113,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100113)

      end do

c --- broadcast the list of atom id numbers for the interacting pairs
c     to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ivpair,
     +                2*NPAIRS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0104,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100104)

      end do

c --- broadcast the list of atom id numbers for the interacting trimers
c     to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ivtrim,
     +                4*NTRIMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0114,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100114)

      end do

c --- broadcast the size of each stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1

         call MPI_SEND(npair,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0105,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100105)

      end do

c --- broadcast the size of each trimer stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1
```

```
         call MPI_SEND(ntrim,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0115,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100115)

      end do
c --- broadcast the list of interacting pair id numbers that define the
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ipairs,
     +                 NIP*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0106,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100106)

      end do

c --- broadcast the list of interacting trimer id numbers that define the
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(itrims,
     +                 NIT*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0116,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100116)

      end do
c --- broadcast the potential energy curve V(R) to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(v,
     +                 2*NVBINS,
     +                 MPI_R,
     +                 itask,
     +                 0107,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100107)

      end do
      if (idebug.gt.0) write (9, *) 'end parent PART ONE'
      if (idebug.gt.0) write (9, *) ''
      call flush(9)
c =======================================================================
c     PART TWO: PERFORMING THE SIMULATION
```

```
c ======================================================================

c --- open the output files: ofile, dump, and rwfile

      open (10, file=dump, form='unformatted')

      write (10) bb, aaxy, aaz
      open (11, file=ofile)
      open (12, file=rwfile)

c --- Check that NCHUNKS evenly divides nloop/nprint
      ncalc = nloop/nprint
      if(mod(ncalc,NCHUNKS).ne.0) then

          write (6, 5000)
5000     format ('MCCs not divisble by NCHUNKS. Change NCHUNKS'
     +           'or nloops')

         call quit

      end if
c --- initialization of various progress counters.

      denom = 0.0d0
      vsum = 0.0d0
      v2sum = 0.0d0
      esum = 0.0d0
      e2sum = 0.0d0
      tsum = 0.0d0
      t2sum = 0.0d0

c --- these values are required for error estimation in reweighting
      do i = 1, 9
         rw_vsum(i) = 0.0d0
         rw_tsum(i) = 0.0d0
         rw_esum(i) = 0.0d0
         rw_wsum(i) = 0.0d0
         rw_wsqsum(i) = 0.0d0
         rw_vwsum(i) = 0.0d0
         rw_ewsum(i) = 0.0d0
         rw_twsum(i) = 0.0d0
         rw_vvsum(i) = 0.0d0
         rw_eesum(i) = 0.0d0
         rw_ttsum(i) = 0.0d0
      end do
c --- this is how many iterations we have done.
c --- for the vmc program, all loop counting is essentially handled
c     in allrep.f where snapshots are received. This just initializes
c     loop for us.

      loop=0

c --- these tell us about the acceptance ratio for the atom moves.

      ztacc=0.0d0
      ztrej=0.0d0

c --- these counters make sure that we don't lose a chunk of snapshots somewhere in
c     the ether. we use them to count how many chunks have been sent and
c     received.

300   nsent=0
      nrcvd=0

c --- this is a list of flags that are zero for chunks that haven't yet
```

```
c     been sent to a child for processing, positive for chunks that have
c     been sent, and negative for chunks that have been processed and
c     returned to the parent.

c     isent(n) is set to the (positive) task id of the receiving child
c     process when a chunk is sent.  this is basically leaving a trail
c     of crumbs so that we can track down the chunks and ask the children
c     to return them to us.

      do nchunk=1, NCHUNKS
         isent(nchunk)=0
      end do
c     allrep distributes chunks to children. Once this command has been called
c     and returns, all loops will have been performed, so all
c     chunks should have been sent and received.

      call allrep(nsent, nrcvd, loops, MPI_R)
      loop = loop + loops
c --- check for lost chunks.

      if (nsent.ne.NCHUNKS.or.nrcvd.ne.NCHUNKS) then
         write (6, *) 'chunks have been lost!'
         write (6, *) 'nsent = ', nsent
         write (6, *) 'nrcvd = ', nrcvd
         ierror=1
      end if

c --- Want to save a checkpoint every 1000 chunks in case job stops
c     before completion. If we need to run more passes, go back to line
c     300.

      if(loop.lt.nloop) then

         open(8, file=dfile, form='unformatted')

         write (8) loop
         do k=1, ntasks-1
            write (8) (rstatv(i, k), i=1, 8)
            write (8) (path(i, k), i=1, NATOM3)
         end do
         close(8)

         goto 300

      else if(loop.eq.nloop) then

          write (6, 6810) dfile
6810      format ('SAVING final configuration to ', a16/)
          open(8, file=dfile, form='unformatted')

         write (8) loop
         do k=1, ntasks-1
            write (8) (rstatv(i, k), i=1, 8)
            write (8) (path(i, k), i=1, NATOM3)
         end do
         close(8)

c ------ Calculate sums for standard deviation of each energy.
c        Ref: A.M. Ferrenberg, et. al. Phys. Rev. E 51, 5092 (1995).
         rwaxy = aaxy-da
         rwaz = aaz-da
         rwb = bb-3.0d0*db
         do i = 1, 9

           vsum2 = rw_vsum(i)*rw_vsum(i)
           tsum2 = rw_tsum(i)*rw_tsum(i)
           esum2 = rw_esum(i)*rw_esum(i)
           wsum2 = rw_wsum(i)*rw_wsum(i)

           vsumw = rw_vsum(i)*rw_wsum(i)
           tsumw = rw_tsum(i)*rw_wsum(i)
           esumw = rw_esum(i)*rw_wsum(i)

           rw_uavg = rw_vsum(i)/rw_wsum(i)
           rw_tavg = rw_tsum(i)/rw_wsum(i)
           rw_eavg = rw_esum(i)/rw_wsum(i)

           rw_uavgsq = rw_vvsum(i)/rw_wsum(i)
           rw_tavgsq = rw_ttsum(i)/rw_wsum(i)
           rw_eavgsq = rw_eesum(i)/rw_wsum(i)

           rw_uavgvar = denom*((rw_vvsum(i)/vsum2)+(rw_wsqsum(i)/wsum2)
     +                   -2*(rw_vwsum(i)/vsumw))*rw_uavg*rw_uavg
           rw_tavgvar = denom*((rw_ttsum(i)/tsum2)+(rw_wsqsum(i)/wsum2)
     +                   -2*(rw_twsum(i)/tsumw))*rw_tavg*rw_tavg
           rw_eavgvar = denom*((rw_eesum(i)/esum2)+(rw_wsqsum(i)/wsum2)
     +                   -2*(rw_ewsum(i)/esumw))*rw_eavg*rw_eavg

           rw_uavgsd = sqrt(rw_uavgvar)
           rw_tavgsd = sqrt(rw_tavgvar)
           rw_eavgsd = sqrt(rw_eavgvar)
c -------- Save necessary information for parameter estimation to
c          rw_sum
           rw_sums(1, i) = rwb
           rw_sums(2, i) = rwaxy
           rw_sums(3, i) = rwaz
           rw_sums(4, i) = rw_eavg*hart/dble(NATOMS)

c -------- write out the reweighted energies to rwfile
           write (12, 900) rwb, rwaxy, rwaz,
     +                        rw_uavg*hart/dble(NATOMS),
     +                        rw_uavgsd*hart/dble(NATOMS),
     +                        rw_eavg*hart/dble(NATOMS),
     +                        rw_eavgsd*hart/dble(NATOMS),
     +                        rw_tavg*hart/dble(NATOMS),
     +                        rw_tavgsd*hart/dble(NATOMS)

900      format (3(1x, F10.8), 6(1x, 1pe20.13))

         call  flush (12)
           if (mod(i,3).eq.0) then
              rwaxy = rwaxy + da
              rwaz = aaz-da
           else
              rwaz = rwaz+da
           end if
           rwb = rwb+db
        end do
        if(da.eq.0.0d0) then
           call param_est_bb(rw_sums)
        else if(db.eq.0.0d0) then
           call param_est_aa(rw_sums)
        else
           write (6, *) "No paramest program called"
        end if
      end if

c --- Now all chunks have run

      if (idebug.gt.0) then
```

```
            write (9, *) ''
            write (9, *) 'QSATS is done!'
            write (9, *) ''
         end if

c --- close output files

         close(10)
         close(11)

c --- show how much work every child did.

         if (idebug.gt.0) then
            do i=1, ntasks-1
               write (9, 9100) i, iwork(i)
9100           format ('task ', i3, ' received ', i9, ' chunks')
            end do
         end if

c --- tell the children we're all done.

         do itask=1, ntasks-1

            imsg(1)=0

            call MPI_SEND(imsg,
     +                 1,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

            call errchk(0, ierr, 100204)

         end do

         write (6, 6900) ztacc
6900     format ('total number of accepted moves = ', f20.1)

         write (6, 6901) ztrej
6901     format ('total number of rejected moves = ', f20.1/)

         if (idebug.gt.0) write (9, *) ''
         if (idebug.gt.0) write (9, *) 'end parent PART TWO'

         return

c --- error handling
901      write (6, *) 'error opening lattice file'
         goto 999
902      write (6, *) 'error reading number of atoms from lattice file'
         goto 999
903      write (6, *) 'error reading (unscaled) supercell edge lengths'
         goto 999
904      write (6, *) 'error reading atom number ', i
         goto 999

999      call quit

         return
         end
```

input-3b.f

```
c ------------------------------------------------------------------------
c     this inputs the names of various I/O files and also reads in the
c     parameters for the simulation.
c     This program is not currently set up to accept deformation parameters
c ------------------------------------------------------------------------
      subroutine input

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      character*8 inword
c --- read in filenames.
c     ltfile = lattice file containing total # atoms and coordinates
c     pfile = parameter file, read next
c     ofile = energy output file
c     dfile = snapshot checkpoint file
c     dump = snapshot file
c     rwfile = reweighted energy output file

      read (5, 5000, err=922) ltfile
5000  format (a20)
      read (5, 5000, err=923) pfile
      read (5, 5000, err=924) ofile
      read (5, 5000, err=925) dfile
      read (5, 5000, err=926) dump
      read (5, 5000, err=927) rwfile
c --- set debug level.
      read (5, 5001, err=931) inword
5001  format (a8)
      if (inword.eq.'NONE') then
         idebug=0
      else if (inword.eq.'MINIMAL') then
         idebug=1
      else if (inword.eq.'LOW') then
         idebug=2
      else if (inword.eq.'MEDIUM') then
         idebug=3
      else if (inword.eq.'HIGH') then
         idebug=4
      else
         write (6, *) 'invalid debug level'
      end if

c --- define some masses. amu = the unified mass unit in terms of atomic units.
c     zmh and zmhe are the hydrogen and helium atomic masses.
      zmh=1837.1526d0
      amu=zmh/1.007825d0
      zmhe=4.0026d0*amu

c --- read in the simulation parameters.
c     nloop = total # of MCCS
c     nprint = snapshot interval
c     den = number density in atoms per cubic bohr
c     bb = trial wavefunction b parameter
c     aaxy = trial wavefunction a_xy parameter
c     aaz = trial wavefunction a_z parameter
c     irrst = restart variable
c     da = a_i parameter interval used for reweighting
c     db = b parameter interval used for reweighting

      open (7, file=pfile)
      read (7, *, err=901) nloop
      read (7, *, err=902) nprint
      read (7, *, err=903) den
      read (7, *, err=904) bb
      read (7, *, err=905) aaxy
      read (7, *, err=906) aaz
      read (7, *, err=907) irrst
      read (7, *, err=909) da
      read (7, *, err=910) db

      write (6, 6000) NATOMS
6000  format ('REPEATING input parameters'//,
     +        'atom count    = ', i6/)

      write (6, 6001) den, aaxy, aaz, bb, dzscale
6001  format ('density          = ', f14.7, ' atoms per cubic bohr'/,
     +        'a_xy parameter   = ', f14.7, ' bohr**(-2)'/,
     +        'a_z parameter    = ', f14.7, ' bohr**(-2)'/,
     +        'B parameter      = ', f14.7, ' bohr'/)

      write (6, 6002) nloop, nprint
6002  format ('number of simulation steps = ', i9/,
     +        'snapshot interval          = ', i8/)
      return

901   write (6, *) 'error reading number of loops'
      goto 999
902   write (6, *) 'error reading nprint'
      goto 999
903   write (6, *) 'error reading density'
      goto 999
904   write (6, *) 'error reading bb'
      goto 999
905   write (6, *) 'error reading axy'
      goto 999
906   write (6, *) 'error reading az'
      goto 999
907   write (6, *) 'error reading irrst value'
      goto 999
908   write (6, *) 'error reading dzscale value'
      goto 999
909   write (6, *) 'error reading da value'
      goto 999
910   write (6, *) 'error reading db value'
      goto 999
921   write (6, *) 'error reading RNG file name'
      goto 999
922   write (6, *) 'error reading lattice file name'
      goto 999
923   write (6, *) 'error reading parameter file name'
      goto 999
924   write (6, *) 'error reading ofile file name'
      goto 999
925   write (6, *) 'error reading dfile file name'
      goto 999
926   write (6, *) 'error reading dump file name'
      goto 999
927   write (6, *) 'error reading reweighting file name'
      goto 999
931   write (6, *) 'error reading debug level'
      goto 999
932   write (6, *) 'error reading RNG initialization mode'
      goto 999
999   call quit
      return
      end
```

child-3b.f

```
c ---------------------------------------------------------------------
c     this is the child process that runs on all nodes except node 0
c     (which is running the parent process).
c ---------------------------------------------------------------------

      subroutine child(MPI_R)

      implicit double precision (a-h, o-z)

      include 'mpif.h'
      include 'sizes.h'


c --- child processes don't include common block, all variables are
c     local and must be defined below. Prevents child processes from
c     overwriting global variables

      common /rancm1/ rscale

      dimension psi(NATOM6), npair(NATOMS), rv(NATOM3)
      dimension istat(MPI_STATUS_SIZE)
      dimension ipairs(NIP, NATOMS)
      dimension itrims(NIT, NATOMS)
      dimension vpvec(3, NPAIRS)
      dimension ivpair(2, NPAIRS)
      dimension ivtrim(4, NTRIMS)
      dimension vpvectri(9, NTRIMS)
      dimension ntrim(NATOMS)

      dimension r2old(NATOMS), r2new(NATOMS), v1(NATOMS), v2(NATOMS)
      dimension v(2, NVBINS)
      dimension imsg(11), fmsg(7), emsg(3), imsg2(3)
      dimension rstate(8), qsave(3)
      dimension dlng(NATOM3), d2lng(NATOM3)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)
c =====================================================================
c     PART ONE: INITIALIZATION
c =====================================================================
      MPI_R=MPI_DOUBLE_PRECISION

c --- numerical factor for random number generator.

      rscale=1.0d0/4294967088.0d0

c --- determine which process this is and store it in myid.

      call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)

c --- receive all of the information that is broadcast by the parent
c     process.

c --- first receive some integer constants.  these are primarily used to
c     check that the arrays are properly dimensioned.

      call MPI_RECV(imsg,
     +              11,
     +              MPI_INTEGER,
     +              0,
     +              0101,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200101)

      istop=0

      if (imsg(1).ne.NATOMS) then
         write (6, *) 'size mismatch 1: ', imsg(1)
         istop=1
      end if

      if (imsg(2).ne.NATOM3) then
         write (6, *) 'size mismatch 2: ', imsg(2)
         istop=1
      end if

      if (imsg(3).ne.NATOM6) then
         write (6, *) 'size mismatch 3: ', imsg(3)
         istop=1
      end if

      if (imsg(4).ne.NATOM7) then
         write (6, *) 'size mismatch 4: ', imsg(4)
         istop=1
      end if

      if (imsg(5).ne.NIP) then
         write (6, *) 'size mismatch 5: ', imsg(5)
         istop=1
      end if

      if (imsg(6).ne.NPAIRS) then
         write (6, *) 'size mismatch 6: ', imsg(6)
         istop=1
      end if

      if (imsg(7).ne.NIT) then
         write (6, *) 'size mismatch 5: ', imsg(5)
         istop=1
      end if

      if (imsg(8).ne.NTRIMS) then
         write (6, *) 'size mismatch 6: ', imsg(6)
         istop=1
      end if

      if (imsg(9).ne.NVBINS) then
         write (6, *) 'size mismatch 7: ', imsg(7)
         istop=1
      end if

      if (istop.eq.1) call quit


      nvpair = imsg(6)
      idebug=imsg(10)
      nprint = imsg(11)

c --- debugging output.

      if (idebug.eq.4) write (30+myid, *) 'idebug = ', idebug
      call flush(30+myid)
c --- next receive some floating-point constants.
```

```
      call MPI_RECV(fmsg,
     +              7,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0102,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200102)

      den=fmsg(1)
      bin=fmsg(2)
      r2min=fmsg(3)
      aaxy=fmsg(4)
      aaz=fmsg(5)
      bb=fmsg(6)
      zmhe=fmsg(7)

      if (idebug.eq.4) then
         write (30+myid, *) 'den = ', den
         write (30+myid, *) 'bin = ', bin
         write (30+myid, *) 'r2min = ', r2min
         write (30+myid, *) 'aaxy = ', aaxy
         write (30+myid, *) 'aaz = ', aaz
         write (30+myid, *) 'bb = ', bb
      end if
      call flush(30+myid)
c --- compute the inverse of the potential energy V(R) bin width, to
c     avoid unnecessary divisions.

      binvrs=one/bin

c --- next receive the vectors that connect pairs of atoms in a stencil.

      call MPI_RECV(vpvec,
     +              3*NPAIRS,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0103,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200103)

c --- receive the interacting-trimer vectors to all child processes.

         call MPI_RECV(vpvectri,
     +                 9*NTRIMS,
     +                 MPI_R,
     +                 0,
     +                 0113,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(myid, ierr, 200113)

c --- next receive the list of pairs of atoms.

      call MPI_RECV(ivpair,
     +              2*NPAIRS,
     +              MPI_INTEGER,
     +              0,
     +              0104,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200104)

c --- receive the list of atom id numbers for the interacting trimers
c     to all child processes.

         call MPI_RECV(ivtrim,
     +                 4*NTRIMS,
     +                 MPI_INTEGER,
     +                 0,
     +                 0114,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(myid, ierr, 200114)


c --- next receive the number of atoms that belong to each atom's stencil.
c     this should really be the same for every atom for a regular crystal
c     lattice, but we treat it as a variable.

      call MPI_RECV(npair,
     +              NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0105,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200105)

c --- receive the size of each trimer stencil to all child processes.all
c     stencils should be the same size, but we treat this as a variable.

         call MPI_RECV(ntrim,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 0,
     +                 0115,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(myid, ierr, 200115)

c --- next receive the pairs that constitute each atom's stencil.

      call MPI_RECV(ipairs,
     +              NIP*NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0106,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200106)

c --- receive the list of interacting trimer id numbers that define
```

```
c     the stencils to all child processes.

         call MPI_RECV(itrims,
     +                 NIT*NATOMS,
     +                 MPI_INTEGER,
     +                 0,
     +                 0116,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(myid, ierr, 200116)

c --- next receive the potential energy curve V(R) for interpolation.

      call MPI_RECV(v,
     +              2*NVBINS,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0107,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200107)
      if (idebug.eq.4) then
         write (30+myid, *) 'child moving to PART TWO'
         call flush(30+myid)
      end if


c ======================================================================
c     PART TWO: PERFORMING THE SIMULATION
c ======================================================================

100   idchunk=0

      nacc=0
      nrej=0
      pot = 0.0d0
      tloc = 0.0d0
c --- send request for data (message type 1201) to parent.  the first
c     time through, or if we are waiting for all children to sync up,
c     there are no results to send back to the parent, so we indicate
c     this by setting idchunk=0 just above, and then sending this to
c     the parent in imsg2(1).

200   imsg2(1)=idchunk

      imsg2(2)=nacc
      imsg2(3)=nrej

      emsg(1) = pot
      emsg(2) = tloc
      emsg(3) = potl3b
      call MPI_SEND(imsg2,
     +              3,
     +              MPI_INTEGER,
     +              0,
     +              1201,
     +              MPI_COMM_WORLD,
     +              ierr)

      call errchk(myid, ierr, 201201)
c --- on the other hand, if there are results to send back, then we
```

```
c     do so here.

      if (idchunk.gt.0) then
c ------ first we send a message of type 1202 that contains the atoms'
c        new positions.

         call MPI_SEND(psi,
     +                 NATOM3,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1202,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201202)
c ------ then we send a message of type 1203 that contains the updated
c        random number generator state vector.

         call MPI_SEND(rstate,
     +                 8,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1203,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201203)
         call MPI_SEND(emsg,
     +                 3,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 201204)
      end if

c --- wait for acknowledgement (message type 0204) from parent.  the
c     parent also uses this to signal the child that more input will
c     be sent.

c     if imsg(1) is positive, it is a replica number that represents the
c     next replica that this child should process.

c     if imsg(1) is negative, then this child needs to wait for the
c     other children to sync up, and so the child goes back to the top
c     of PART TWO.

c     if imsg(1) is zero, there is no more work to be done.

      call MPI_RECV(imsg2,
     +              1,
     +              MPI_INTEGER,
     +              0,
     +              0204,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200204)

c --- loop back and wait for more input if instructed by parent.

      if (imsg2(1).lt.0) goto 100
```

```
c --- terminate if the simulation is complete.

      if (imsg2(1).eq.0) then
         if (idebug.eq.4) write (30+myid, *) 'child is done!'
         call flush(30+myid)
         return
      end if

c --- if there is a new replica to process, then receive data from
c     the parent.

c --- we need to save the replica number that we are about to work on.

      idchunk=imsg2(1)
c --- next receive the old atomic coordinates in a message of type 0205.

      call MPI_RECV(psi,
     +              NATOM3,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0205,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200205)
c --- next receive the random number generator state vector, in
c     a message of type 0206.

      call MPI_RECV(rstate,
     +              8,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0206,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200206)
c ================================================================
c --- this is the actual VMC simulation
c ================================================================
c --- this counts the simulation loop that we're on. For every chunk
c     received by the child, we run through nprint loops
      loop = 0

c --- this is the number of accepted (nacc) and rejected (nrej) moves
c     in the current set of nprint loops.

      nacc=0
      nrej=0

c --- this is the denominator that we will use to compute the average
c     potential energy for each set of nprint loops.

      denom=0.0

c --- energy adjustment loop.

300   loop=loop+1

c --- try to move each atom once.

      do k=1, NATOMS
```

```
c ----- this replaces calling pick.f
c ----- compute ln of the trial wave function squared when the
c       atom is in its current location. (replaces call trial)

         glog=0.0d0
         do nn=1, npair(k)

            n=ipairs(nn, k)

            i=ivpair(1, n)
            j=ivpair(2, n)

            dx=(psi(3*j-2))+(-psi(3*i-2))+
     +         vpvec(1, n)
            dy=(psi(3*j-1))+(-psi(3*i-1))+
     +         vpvec(2, n)
            dz=(psi(3*j))  +(-psi(3*i))+
     +         vpvec(3, n)

            r2=dx*dx+dy*dy+dz*dz

            br2=bb*bb/r2

            br5=br2*br2*sqrt(br2)
            glog=glog-0.5d0*br5

         end do
c ------ multiplying the ln by 2 is like computing the ln of the square.

         g1=2.0d0*glog

c ----- save the old position of this atom.

        qsave(1)=psi(3*k-2)
        qsave(2)=psi(3*k-1)
        qsave(3)=psi(3*k)

c ----- pick three gaussian random numbers and scale them.
         call gstep(rstate, gauss, rscale)

         psi(3*k-2)=gauss/sqrt(2.0*2.0*aaxy)

         call gstep(rstate, gauss, rscale)

         psi(3*k-1)=gauss/sqrt(2.0*2.0*aaxy)

         call gstep(rstate, gauss, rscale)

         psi(3*k)=gauss/sqrt(2.0*2.0*aaz)

c ----- compute ln of the trial wave function squared after the atom moves
c       to its new location.
c       this replaces the second "call trial"
         glog=0.0d0
         do nn=1, npair(k)

            n=ipairs(nn, k)

            i=ivpair(1, n)
            j=ivpair(2, n)
            dx=(psi(3*j-2))+(-psi(3*i-2))+
     +         vpvec(1, n)
            dy=(psi(3*j-1))+(-psi(3*i-1))+
     +         vpvec(2, n)
            dz=(psi(3*j))  +(-psi(3*i))+
     +         vpvec(3, n)

            r2=dx*dx+dy*dy+dz*dz
            br2=bb*bb/r2

            br5=br2*br2*sqrt(br2)
            glog=glog-0.5d0*br5

         end do
c ------ multiplying the ln by 2 is like computing the ln of the square.

         g2=2.0d0*glog

c ----- decide whether to accept or reject the move.

c ----- if the new trial wave function is lower than the old, we
conditionally
c       accept the move.
        if (g2.lt.g1) then

           gratio=exp(g2-g1)

           call rstep(rstate, z, rscale)

           if (z.lt.gratio) then

              nacc = nacc+1

           else

              psi(3*k-2)=qsave(1)
              psi(3*k-1)=qsave(2)
              psi(3*k)=qsave(3)

              nrej = nrej+1

           end if

c --- if the new trial wave function is larger, we always accept the
c     move.

        else

           nacc = nacc+1

        end if
      end do
c --- check whether it’s time to calculate energies and
c     send info back to the parent
      if (loop.eq.nprint) then

c ======================================================================
c       Calculate the energy
c ======================================================================
c ----- pot is the instantaneous "snapshot" potential energy

        potl=0.0d0

c ----- loop over all of the interacting pairs.
        do n=1, nvpair
           i=ivpair(1, n)
           j=ivpair(2, n)

           dx=(psi(3*j-2))+(-psi(3*i-2))+
     +        vpvec(1, n)
```

```
           dy=(psi(3*j-1))+(-psi(3*i-1))+
     +        vpvec(2, n)
           dz=(psi(3*j))  +(-psi(3*i))+
     +        vpvec(3, n)

           r2=dx*dx+dy*dy+dz*dz
c -------- compute the 2body potential energy by interpolating between two grid
c          points.

           ibin=int((r2-r2min)*binvrs)+1

           if (ibin.gt.0) then
              dr=(r2-r2min)-bin*dble(ibin-1)
              p= v(1, ibin)+ v(2, ibin)*dr
              potl=potl+p
           else
              potl=potl+v(1, 1)
           end if

        end do
        pot = potl*0.5d0
c ----- compute the 3body potential energy

         potl3b = 0.0d0

c ------ loop over all nearest neighbor trimers

         do n=1, NTRIMS
            i= ivtrim(1, n)
            j= ivtrim(2, n)
            k= ivtrim(3, n)
            ndiv = ivtrim(4, n)

            dx1 = vpvectri(1, n)+psi(3*j-2)-psi(3*i-2)
            dy1 = vpvectri(2, n)+psi(3*j-1)-psi(3*i-1)
            dz1 = vpvectri(3, n)+psi(3*j)-psi(3*i)

            dx2 = vpvectri(4, n)+psi(3*k-2)-psi(3*i-2)
            dy2 = vpvectri(5, n)+psi(3*k-1)-psi(3*i-1)
            dz2 = vpvectri(6, n)+psi(3*k)-psi(3*i)

            dx12 = dx2-dx1
            dy12 = dy2-dy1
            dz12 = dz2-dz1

            r1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
            r2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
            r12 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)
c --------- get 3B energy and add to correct total
            call He3(r1, r2, r12, E3)
            E3 = E3/dble(ndiv)
            potl3b = potl3b+E3
        end do
c ----- tloc is the instantaneous "snapshot" kinetic energy

        do i=1, NATOM3
           dlng(i)=0.0
           d2lng(i)=0.0
        end do

c ----- first compute the one-atom contributions to the kinetic energy.
        do i=1, NATOMS

           xx=psi(3*i-2)
           yy=psi(3*i-1)
           zz=psi(3*i)

           dlng(3*i-2)= dlng(3*i-2)-2.0*aaxy*xx
           dlng(3*i-1)= dlng(3*i-1)-2.0*aaxy*yy
           dlng(3*i)  = dlng(3*i)-2.0*aaz*zz

           d2lng(3*i-2)= d2lng(3*i-2)-2.0*aaxy
           d2lng(3*i-1)= d2lng(3*i-1)-2.0*aaxy
           d2lng(3*i)  = d2lng(3*i)-2.0*aaz

        end do

c ----- loop over all interacting pairs.

        do n=1, nvpair

           i=ivpair(1, n)
           j=ivpair(2, n)

           dx=-((psi(3*j-2))+vpvec(1, n)+(-psi(3*i-2)))
           dy=-((psi(3*j-1))+vpvec(2, n)+(-psi(3*i-1)))
           dz=-((psi(3*j))  +vpvec(3, n)+(-psi(3*i))  )

           r2=dx*dx+dy*dy+dz*dz

           if (r2.le.0.0) write (6, *) 'i, j, r2 = ', i, j, r2

           br2=bb*bb/r2

           br5=br2*br2*sqrt(br2)

           dlng(3*i-2)=dlng(3*i-2)+2.5*br5*dx/r2
           dlng(3*i-1)=dlng(3*i-1)+2.5*br5*dy/r2
           dlng(3*i)  =dlng(3*i)  +2.5*br5*dz/r2

           d2lng(3*i-2)=d2lng(3*i-2)+2.5*br5*
     *                              (1.0-7.0*dx**2/r2)/r2
           d2lng(3*i-1)=d2lng(3*i-1)+2.5*br5*
     *                              (1.0-7.0*dy**2/r2)/r2
           d2lng(3*i)  =d2lng(3*i)  +2.5*br5*
     *                              (1.0-7.0*dz**2/r2)/r2

        end do

c ----- now add up all of the contributions to the kinetic energy.
        tloc=0.0

        do i=1, NATOM3

           tloc=tloc+d2lng(i)+dlng(i)**2

        end do

c ----- divide by (two times the mass) and negate the result.  this is
c       minus hbar squared divided by twice the mass...

        tloc=-0.5*tloc/zmhe

        goto 200
      else
        goto 300
      end if

      end
```

allrep-3b.f

```
c ----------------------------------------------------------------------
c     this subroutine distributes chunks of iterations to the child
c     processes, waits for them to be processed, and then returns
c     control to the main parent subroutine. Running averages are
c     also calculated and printed here, along with the snapshots.
c     The running average of the potential and total energies
c     include contributions from first nearest neighbor trimers.
c ----------------------------------------------------------------------

      subroutine allrep(nsent, nrcvd, loops, MPI_R)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'vmc-448.com'
      include 'mpif.h'

      parameter (hart=315774.65d0)

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(3), emsg(3)
      dimension isent(NCHUNKS), psi(NATOM3)
      dimension rstate(8)

c --- loop over all chunks.

      MPI_R=MPI_DOUBLE_PRECISION

      loops = 0
      do nchunk=1, NCHUNKS

         if (idebug.eq.4)
     +      write (9, *) 'finding child who can receive chunk= ', nchunk
            call flush(9)
c ------ wait for data request from a child.

         call MPI_PROBE(MPI_ANY_SOURCE,
     +                  1201,
     +                  MPI_COMM_WORLD,
     +                  istat,
     +                  ierr)

         call errchk(0, ierr, 111201)

         nchild=istat(MPI_SOURCE)
         call MPI_RECV(imsg,
     +                 3,
     +                 MPI_INTEGER,
     +                 nchild,
     +                 1201,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(0, ierr, 151201)
         if (idebug.eq.4)
     +      write (9, *) 'sending chunk = ', nchunk, ' to ', nchild
            call flush(9)
c ------ check whether the child is returning results.  if so, then
c        receive the results.

         if (imsg(1).gt.0) then
            idchunk=imsg(1)
            if (idebug.eq.4)
     +         write (9, *) 'child ', nchild, ' returning chunk ',
     +                      idchunk
               call flush(9)
c --------- keep track of acceptances and rejections.

            ztacc=ztacc+imsg(2)
            ztrej=ztrej+imsg(3)
            loops = loops+nprint ! whenever a child returns, 50 loops done
            denom = denom+1.0d0
c --------- Receive new configurations, rstate vectors, and energies
            call MPI_RECV(psi,
     +                    NATOM3,
     +                    MPI_R,
     +                    nchild,
     +                    1202,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111202)


            call MPI_RECV(rstate,
     +                    8,
     +                    MPI_DOUBLE_PRECISION,
     +                    nchild,
     +                    1203,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111203)


            call MPI_RECV(emsg,
     +                    3,
     +                    MPI_DOUBLE_PRECISION,
     +                    nchild,
     +                    1204,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 111204)
c --------- here, print out the snapshots as we get them, update running
c           average energies, and print out the energies to the ofile

            write (10) (psi(i), i=1, NATOM3)

            potl = emsg(1)
            tloc = emsg(2)
            pot3b = emsg(3)

            vsum = vsum+potl+pot3b
            v2sum = v2sum+(potl+pot3b)**2
            esum = esum+potl+tloc+pot3b
            e2sum = e2sum+(potl+pot3b+tloc)**2
            tsum = tsum+tloc
            t2sum = t2sum+tloc*tloc
c --------- calculate current averages and standard deviations

            uavg = vsum/denom
            u2avg = v2sum/denom
            usd = sqrt(u2avg-uavg*uavg)
            eavg = esum/denom
            e2avg = e2sum/denom
            esd = sqrt(e2avg-eavg*eavg)

            tavg = tsum/denom
            t2avg = t2sum/denom
            tsd = sqrt(t2avg-tavg*tavg)

c --------- Print energies for this snapshot to the ofile (also want to
c           inclue nacc, nrej, and nchild for this chunk)

            write(11, 8400) potl*hart/dble(NATOMS),
     +                     pot3b*hart/dble(NATOMS),
     +                     uavg*hart/dble(NATOMS),
     +                     usd*hart/dble(NATOMS),
     +                     eavg*hart/dble(NATOMS),
     +                     esd*hart/dble(NATOMS),
     +                     tavg*hart/dble(NATOMS),
     +                     tsd*hart/dble(NATOMS),
     +                     imsg(2), imsg(3), nchild

8400       format (8(1x, 1pe13.6), 1x, i7, 1x, i7, 1x, i3)

c --------- perform the calculations for reweighting
            rwaxy = aaxy-da
            rwaz = aaz-da
            rwb = bb-3.0d0*db

            do ii = 1, 9
c ------------ Calculate 2*log(psi') and 2*log(psi) (called pnewsq
c               and poldsq, respectively).
               dx2sum=0.0d0
               dy2sum=0.0d0
               dz2sum=0.0d0
               do l=1, NATOMS
                  dx=psi(3*l-2)
                  dy=psi(3*l-1)
                  dz=psi(3*l)
                  dx2sum=dx2sum+dx*dx
                  dy2sum=dy2sum+dy*dy
                  dz2sum=dz2sum+dz*dz
               end do
c ------------ This is the one body contribution to 2*log(psi') and
c               2*log(psi):

               p1oldsq=-2.0d0*(aaxy*dx2sum+aaxy*dy2sum+aaz*dz2sum)
               p1newsq=-2.0d0*(rwaxy*dx2sum+rwaxy*dy2sum+rwaz*dz2sum)

c ------------ calculate the same for the two body term:
               psi2old = 0.0d0
               psi2new = 0.0d0
               do n= 1, nvpair
                  j=ivpair(1,n)
                  m=ivpair(2,n)
                  if (m.gt.j) then
                     dx = (psi(3*m-2))+vpvec(1, n) +(-psi(3*j-2))
                     dy = (psi(3*m-1))+vpvec(2, n) +(-psi(3*j-1))
                     dz = (psi(3*m))+vpvec(3, n) +(-psi(3*j))

                     r2 = dx*dx+dy*dy+dz*dz
                     br2old = bb*bb/r2
                     br2new = rwb*rwb/r2

                     br5old = br2old*br2old*sqrt(br2old)
                     br5new = br2new*br2new*sqrt(br2new)

                     psi2old = psi2old - 0.5d0*br5old
                     psi2new = psi2new - 0.5d0*br5new
                  end if
               end do

               psi2oldsq = 2.0d0*psi2old
               psi2newsq = 2.0d0*psi2new
c ------------ Add one and two body terms for total 2*log(psi) and
c             2*log(psi')

               poldsq = p1oldsq + psi2oldsq
               pnewsq = p1newsq + psi2newsq

c ------------ calculate the reweighting factor, w = |psi'^2|/|psi^2|
c             = exp|2*log(psi') - 2*log(psi)|
               w=exp(pnewsq-poldsq)
               rw_wsum(ii) = rw_wsum(ii)+w
               rw_wsqsum(ii) = rw_wsqsum(ii) + (w*w)

c ------------ calculate the reweighted potential
               pot=w*(potl+pot3b)
               rw_vsum(ii) = rw_vsum(ii)+pot
               rw_vwsum(ii) = rw_vwsum(ii) + (pot*w)
               rw_vvsum(ii) = rw_vvsum(ii) + (pot*pot)

c ----------- The subroutine kinrw(psi, tloc) calculates the kinetic
c            energy using the new parameters of psi prime.
              call kinrw(psi, rwaxy, rwaz, rwb, tloc)

              tloc = w*tloc
              rw_tsum(ii) = rw_tsum(ii)+tloc
              rw_twsum(ii) = rw_twsum(ii) + (tloc*w)
              rw_ttsum(ii) = rw_ttsum(ii) + (tloc*tloc)

c ----------- Calculate the total energy from the sum of potential and
c            kinetic.
              etot = pot + tloc
              rw_esum(ii) = rw_esum(ii) + etot
              rw_ewsum(ii) = rw_ewsum(ii) + (etot*w)
              rw_eesum(ii) = rw_eesum(ii) + (etot*etot)
c ----------- increment reweighting parameters (rwaxy incremeted in if
c            statement above)
              rwb = rwb + db
              if (mod(ii,3).eq.0) then
                  rwaxy = rwaxy + da
                  rwaz = aaz-da
              else
                  rwaz = rwaz+da
              end if
            end do

c --------- update the random number generator state vector for this
c          child.
            do i=1, 8
               rstatv(i, nchild)=rstate(i)
            end do

c --------- update the atom positions.

            do i=1, NATOM3
               path(i, nchild)=psi(i)
            end do
```

```
c --------- update the number of received chunks.

            nrcvd=nrcvd+1

c --------- indicate that this chunk has been processed and returned.

            isent(idchunk)=-nchild

         end if

c ------ send a new chunk to child.  first tell the child which chunk
c        it is going to receive.

         imsg(1)=nchunk

         call MPI_SEND(imsg,
     +                 1,
     +                 MPI_INTEGER,
     +                 nchild,
     +                 0204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 110204)

c ------ send the chunk.

         if (idebug.eq.4)
     +      write (9, *) 'calling send for child ', nchild
            call flush(9)
         call send(nchild, MPI_R)

         if (idebug.eq.4)
     +      write (9, *) 'chunk ', nchunk, ' sent to child ', nchild
            call flush(9)
c ------ update how many chunks have been sent.

         nsent=nsent+1

c ------ leave the trail of crumbs!

         isent(nchunk)=nchild

c ------ update how much work has been sent to this child.

         iwork(nchild)=iwork(nchild)+1

      end do

c --- at this point we don't have any more iterations to send to the
c     children, but we need to retrieve any processed iterations that the
c     children are still holding to send back to the parent.  this
c     flushes out all of those chunks.

      do i=1, NCHUNKS

         if (isent(i).gt.0) then

            nchild=isent(i)

            call MPI_RECV(imsg,
     +                    3,
     +                    MPI_INTEGER,
     +                    nchild,
     +                    1201,
     +                    MPI_COMM_WORLD,
     +                    istat,
     +                    ierr)

            call errchk(0, ierr, 121201)

c --------- check whether the child is returning results.  if so, get
c           the results and update the atomic positions.

            if (imsg(1).gt.0) then

               idchunk=imsg(1)

               if (idebug.eq.4)
     +            write (9, *) 'child ', nchild,
     +                         ' returning chunk ', idchunk

c ------------ keep track of acceptances and rejections.

               ztacc=ztacc+imsg(2)
               ztrej=ztrej+imsg(3)
               loops  = loops+nprint
               denom = denom+1.0d0

               call MPI_RECV(psi,
     +                       NATOM3,
     +                       MPI_R,
     +                       nchild,
     +                       1202,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121202)

               call MPI_RECV(rstate,
     +                       8,
     +                       MPI_DOUBLE_PRECISION,
     +                       nchild,
     +                       1203,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121203)

               call MPI_RECV(emsg,
     +                       3,
     +                       MPI_DOUBLE_PRECISION,
     +                       nchild,
     +                       1204,
     +                       MPI_COMM_WORLD,
     +                       istat,
     +                       ierr)

               call errchk(0, ierr, 121204)

c --------- here, print out the snapshots as we get them, update running
c           average energies, and print out the energies to the ofile

            write (10) (psi(ii), ii=1, NATOM3)

            potl = emsg(1)
            tloc = emsg(2)
            pot3b = emsg(3)
            vsum = vsum+potl+pot3b
            v2sum = v2sum+(potl+pot3b)**2
            esum = esum+potl+tloc+pot3b
            e2sum = e2sum+(potl+pot3b+tloc)**2
            tsum = tsum+tloc
            t2sum = t2sum+tloc*tloc
c --------- calculate current averages and standard deviations

            uavg = vsum/denom
            u2avg = v2sum/denom
            usd = sqrt(u2avg-uavg*uavg)

            eavg = esum/denom
            e2avg = e2sum/denom
            esd = sqrt(e2avg-eavg*eavg)

            tavg = tsum/denom
            t2avg = t2sum/denom
            tsd = sqrt(t2avg-tavg*tavg)

c --------- Print energies for this snapshot to the ofile (also want to
c           inclue nacc, nrej, and nchild for this chunk)

            write(11, 8400) potl*hart/dble(NATOMS),
     +                     pot3b*hart/dble(NATOMS),
     +                     uavg*hart/dble(NATOMS),
     +                     usd*hart/dble(NATOMS),
     +                     eavg*hart/dble(NATOMS),
     +                     esd*hart/dble(NATOMS),
     +                     tavg*hart/dble(NATOMS),
     +                     tsd*hart/dble(NATOMS),
     +                     imsg(2), imsg(3), nchild

c --------- perform the calculations for reweighting
            rwaxy = aaxy-da
            rwaz = aaz-da
            rwb = bb-3.0d0*db

            do ii = 1, 9
c ------------ Calculate 2*log(psi') and 2*log(psi) (called pnewsq
c              and poldsq, respectively).
               dx2sum=0.0d0
               dy2sum=0.0d0
               dz2sum=0.0d0
               do l=1, NATOMS
                  dx=psi(3*l-2)
                  dy=psi(3*l-1)
                  dz=psi(3*l)
                  dx2sum=dx2sum+dx*dx
                  dy2sum=dy2sum+dy*dy
                  dz2sum=dz2sum+dz*dz
               end do
c ------------ This is the one body contribution to 2*log(psi') and
c              2*log(psi):

               p1oldsq=-2.0d0*(aaxy*dx2sum+aaxy*dy2sum+aaz*dz2sum)
               p1newsq=-2.0d0*(rwaxy*dx2sum+rwaxy*dy2sum+rwaz*dz2sum)

c ------------ calculate the same for the two body term:
               psi2old = 0.0d0
               psi2new = 0.0d0
               do n= 1, nvpair
                  j=ivpair(1,n)
                  m=ivpair(2,n)
                  if (m.gt.j) then
                     dx = (psi(3*m-2))+vpvec(1, n) +(-psi(3*j-2))
                     dy = (psi(3*m-1))+vpvec(2, n) +(-psi(3*j-1))
                     dz = (psi(3*m))+vpvec(3, n) +(-psi(3*j))

                     r2 = dx*dx+dy*dy+dz*dz
                     br2old = bb*bb/r2
                     br2new = rwb*rwb/r2

                     br5old = br2old*br2old*sqrt(br2old)
                     br5new = br2new*br2new*sqrt(br2new)

                     psi2old = psi2old - 0.5d0*br5old
                     psi2new = psi2new - 0.5d0*br5new
                  end if
               end do

               psi2oldsq = 2.0d0*psi2old
               psi2newsq = 2.0d0*psi2new
c ------------ Add one and two body terms for total 2*log(psi) and
c              2*log(psi')

               poldsq = p1oldsq + psi2oldsq
               pnewsq = p1newsq + psi2newsq

c ------------ calculate the reweighting factor, w = |psi'^2|/|psi^2|
c              = exp|2*log(psi') - 2*log(psi)|
               w=exp(pnewsq-poldsq)
               rw_wsum(ii) = rw_wsum(ii)+w
               rw_wsqsum(ii) = rw_wsqsum(ii) + (w*w)

c ------------ calculate the reweighted potential
               pot=w*(potl+pot3b)
               rw_vsum(ii) = rw_vsum(ii)+pot
               rw_vwsum(ii) = rw_vwsum(ii) + (pot*w)
               rw_vvsum(ii) = rw_vvsum(ii) + (pot*pot)

c ----------- The subroutine kinrw(psi, tloc) calculates the kinetic
c             energy using the new parameters of psi prime.
              call kinrw(psi, rwaxy, rwaz, rwb, tloc)

              tloc = w*tloc
              rw_tsum(ii) = rw_tsum(ii)+tloc
              rw_twsum(ii) = rw_twsum(ii) + (tloc*w)
              rw_ttsum(ii) = rw_ttsum(ii) + (tloc*tloc)

c ----------- Calculate the total energy from the sum of potential and
c             kinetic.
              etot = pot + tloc
              rw_esum(ii) = rw_esum(ii) + etot
              rw_ewsum(ii) = rw_ewsum(ii) + (etot*w)
              rw_eesum(ii) = rw_eesum(ii) + (etot*etot)
c ----------- increment reweighting parameters (rwaxy incremeted in if
c             statement above)
              rwb = rwb + db
              if (mod(ii,3).eq.0) then
                  rwaxy = rwaxy + da
                  rwaz = aaz-da
              else
                  rwaz = rwaz+da
              end if
            end do

c ------------ update the random number generator state vector for this
c              child.
```

```
               do k=1, 8
                   rstatv(k, nchild)=rstate(k)
               end do

c ------------ update the atom positions for this child.

               do n=1, NATOM3
                  path(n, nchild)=psi(n)
               end do

c ------------ update the number of received chunks.

               nrcvd=nrcvd+1

c ------------ indicate that this chunk has been processed and returned.

               isent(idchunk)=-nchild

            end if

c --------- now tell the child to wait until all of the children are done
c           and more work is available.

            imsg(1)=-1

            call MPI_SEND(imsg,
     +                    1,
     +                    MPI_INTEGER,
     +                    nchild,
     +                    0204,
     +                    MPI_COMM_WORLD,
     +                    ierr)

            call errchk(0, ierr, 121204)

         end if

      end do

      return
      end
```

sizes.h

```
c --- number of atoms in the system.

      parameter (NATOMS=448)

c --- various multiples of NATOMS.

      parameter (NATOM3=NATOMS*3)
      parameter (NATOM6=NATOMS*6)
      parameter (NATOM7=NATOMS*7)

c --- number of points on the interatomic potential energy curve, for
c     linear interpolation of the potential energy function.

      parameter (NVBINS=20000)

c --- "radius" of the interacting-pair region, in nearest-neighbor distances.

      parameter (RATIO=2.05)
```

```
c --- number of interacting pairs for each atom.

      parameter (NIP=56)

c --- number of first nearest neighbors for each atom.

      parameter (NIP1=12)

c --- number of interacting trimers for each atom.

      parameter (NIT = 66)

c --- total number of interacting pairs in the simulation box.

      parameter (NPAIRS=NATOMS*NIP)

c --- total number of nearest neighbor pairs (for calculating trimers)

      parameter (NPAIRS1=NATOMS*NIP1)

c --- total number of nearest neighbor trimers

      parameter (NTRIMS = NIT*NATOMS)

c --- Maximum number of child processors allowed at time of compilation

      parameter (NCH = 64)

c --- Number of chunks of iterations sent with each pass of the parent loop

      parameter (NCHUNKS = 10)
```

vmc-448.com

```
c --- internal units are atomic units.
c --- hartrees per electron volt.

      parameter (evconv=3.67495735d-2)

c --- hartrees per wavenumber.

      parameter (cmconv=4.55636866d-6)

c --- QMC variables.

      common /monte/  ztacc, ztrej,
     +                zmh, zmhe, den, step,
     +                nloop, nequil, nprint,  nrst

      common /param/  bb, aaxy, aaz, dzscale, da, db, idebug

c --- random number variables.

      double precision zm1, zm2, rm1, rm2, rscale, rstatv

      common /moduli/ zm1, zm2, rm1, rm2

      common /rancom/ rstatv(8, NCH), rscale, irrst, nrskip

c --- potential energy curve

      common /potcom/ v(2, NVBINS)
```

```
c --- filenames.

      character*20 ltfile, pfile, sfile, ofile, dfile, dump,
     +             rwfile
      common /files/  ltfile, pfile, sfile, ofile, dfile, dump,
     +                rwfile

c --- crystal lattice.

      common /crystl/ xtal(NATOMS, 3), path(NATOM3, NCH),
     +                npair(NATOMS), ipairs(NIP, NATOMS),
     +                npair1(NATOMS), ntrim(NATOMS),
     +                ipairs1(NIP1, NATOMS), itrims(NIT, NATOMS)

      common /vpairs/ vpvec(3, NPAIRS), vpvec1(3, NPAIRS1),
     +                vpvectri(9, NTRIMS), ivtrim(4, NTRIMS),
     +                ivpair(2, NPAIRS), ivpair1(2, NPAIRS1),
     +                nvpair, nvpair1, nvtrim


c --- energy sums.

      common /energy/ vsum, v2sum, esum, e2sum, tsum, t2sum, denom

c --- reweighting energy sums.

      common /energy_vec/ rw_vsum(9), rw_tsum(9), rw_esum(9),
     +                    rw_wsqsum(9), rw_vwsum(9), rw_ewsum(9),
     +                    rw_twsum(9), rw_vvsum(9), rw_eesum(9),
     +                    rw_ttsum(9), rw_wsum(9)

c --- counters to monitor load balancing.

      common /parcom/ iwork(NCH)
```

Appendix B

VPI Programs

B.1 VPI 2-Body Program (VPI-2B)

VPI 2-body simulations were performed using a modified version of the QSATS code (Robert J. Hinde, *Computer Physics Communications*, **182**(11), 2339 (2011)). Only those subroutines which differ from the published QSATS code are included below. This version allows for independent a_{xy} and a_z trial wavefunction parameters as well as nonequilibrium values of η, γ, and ϵ. The files required to compile and run the code are as follows:

main.f	Main program that determines whether the processor is the parent or a child processor.
cmrg.f	See Sec. A.1.
rsetup.f	See Sec. A.1.
parent-distortion.f	Parent process which runs on node 0. This sets up the interacting pair list, accounts for distortion effects on the lattice postions, and sends tasks to the child processes.
input-distortion.f	Reads in important file names as well as simulation parameters, including deformation parameters.
vinit.f	See Sec. A.1.
child.f	Child process that generates new configurations for the atoms in each replica using Metropolis Monte Carlo moves. The potential energy is calculated after each move after which an accept/reject decision is made and information is sent back to the parent.
even.f	Divides up even numbered replicas among the child processes.
odd.f	Divides up odd numbered replicas among the child processes.
rpsend.f	Sends a replica to a child proccess to update the atomic configuration.

tstamp.f	See Sec. A.1.
sizes.h	Contains fixed parameters for the simulation including number of atoms, interacting pairs, and the maximum number of child processors.
qsats.h	Sets up the common block variables for the VPI simulation.

Program files that can be found in the QSATS code and are not reproduced here: **main.f**, **cmrg.f**, **rsetup.f**, **vinit.f**, **even.f**, **odd.f**, **rpsend.f**, **tstamp.f**, **sizes.f**

The snapshots generated from the VPI-2B simulation are then used to calculate the average potential (2-body), kinetic, and total energies for each replica, along with the long-range correction to the two-body energy. This is accomplished using eloc-distortion.f program which allows for distorted lattices and calls on many of the same subroutines as the VPI main program. The files required to compile and run this program are listed below:

eloc-distortion.f	Main program which calls the input subroutine and calculates the average energies for each replica from the VPI snapshot file.
input-distortion.f	See **input-distortion.f** above.
vinit.f	See Sec. A.1.
lrc-3d-sub.f	See Sec. A.1.1. This subroutine is identical to the VMC subroutine, except that the line “include ‘vmc-448.com”’ is replaced by “include ‘qsats.h”’.
c6-sub.f	See Sec. A.1.1. This subroutine is identical to the VMC subroutine, except that the line “include ‘vmc-448.com”’ is replaced by “include ‘qsats.h”’.
tstamp.f	See Sec. A.1.
Gauss-Hermite.dat	See Sec. A.1.1.
sizes.h	See **sizes.h** above.
qsats.h	See **qsats.h** above.

parent-distortion.f

```
c ----------------------------------------------------------------------
c     this is the parent process tha
c     errchk is a subroutine called after every MPI subroutine that
c     checks the MPI error code and reports any errors.
c     This version allows for non-equilibrium values of the three
c     distortion parameters eta, gam, and epsil
c ----------------------------------------------------------------------

      subroutine parent(ierror)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'
      include 'mpif.h'

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(9), fmsg(7)
      dimension isent(NREPS), ikeep(NATOMS), replic(NATOM7)
      dimension rstate(8)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)

c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================

      ierror=0

c --- read input file.

      call input

      write (6, 6100) ltfile, spfile, svfile
6100  format ('lattice file name  = ', a17/,
     +        'snapshot file name = ', a17/,
     +        'save file name     = ', a17/)

      if (idebug.eq.0) write (6, 6110) idebug, 'NONE'
      if (idebug.eq.1) write (6, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (6, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (6, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (6, 6110) idebug, 'HIGH'

6110  format ('debug level = ', i1,' or ', a8/)

      phi = sqrt(1.0d0+eta)

c --- read the potential energy curve.

      call vinit(r2min, bin)

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a17/)

      open (8, file=ltfile, status='old', err=901)

      read (8, *, err=902) nlpts
      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         call quit
      end if

c --- read the edge lengths of the supercell.

      read (8, *, err=903) xlen, ylen, zlen
      den0=dble(NATOMS)/(xlen*ylen*zlen)

      xlen = xlen
      ylen = ylen
      zlen = zlen
c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)
      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *, err=904) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)
c --- this helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6310) rnnmin
```

```
6310  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

      write (6, 6330) xlen, ylen, zlen
6330  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
         ntrim(i)=0
      end do

      nvpair=0

      do i=1, NATOMS
      do j=1, NATOMS
         if (j.ne.i) then
            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.
c --------- we determine the interacting pairs from the undistorted
c           then use our values of eta (phi), gamma, and epsilon to
c           impose the distortions for the elastic constant
c           calculations.
            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j
```

```
c ------------ these transformations impose the lattice distortions
c               They reduce to dx, dy, and dz for eta = 0 (phi = 1),
c               gamma = 1 and epsilon = 0.

                vpvec(1, nvpair)=dx/(sqrt(gam)*phi)
                vpvec(2, nvpair)=dy*sqrt(gam)/phi
                vpvec(3, nvpair)=dz*phi**2+dy*epsil

                npair(i)=npair(i)+1
                ipairs(npair(i), i)=nvpair

             end if
          end if
       end do
       end do

c --- Now loop back through the coordinates in the xtal array and
c     transform them appropriately

       do i=1, NATOMS
          xtal(i, 1)=xtal(i, 1)/(sqrt(gam)*phi)
          xtal(i, 2)=xtal(i, 2)*sqrt(gam)/phi
          xtal(i, 3)=xtal(i, 3)*phi**2+epsil*xtal(i, 2)
       end do

       write (6, 6400) npair(1), nvpair
6400   format ('atom 1 interacts with ', i3, ' other atoms'//,
     +         'total number of interacting pairs = ', i6)

          write (6, 6401)
6401      format (/'interaction pair vectors for atom 1 ',
     +             '[NN distances]:'/)

          do i=1, npair(1)
             ip=ipairs(i, 1)
             d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +         rnnmin
             write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                       vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410         format ('vector # ', i3, ' to atom ', i4, ': ',
     +               3(1x, f9.5), ' length = ', f8.5)
          end do


c --- set the displacement vectors for all replicas to zero.

       write (6, 6500)
6500   format (/'SETTING initial configuration to zero'/)

       do j=1, NREPS
       do i=1, NATOM3
          path(i, j)=0.0
       end do
       end do

c --- initialize random number generator.

       call rsetup

c --- now see if there is an old set of displacement vectors from a
c     previous run.  if not, jump head to line 200.

       open (8, file=svfile, form='unformatted', status='old', err=200)
```

```
      write (6, 6510) svfile
6510  format ('READING initial configuration from ', a17/)

      do j=1, NREPS
         read (8) (rstatv(i, j), i=1, 8)
         read (8) (path(i, j), i=1, NATOM3)
      end do

      close (8)

200   if (idebug.ge.3) then

         write (6, 6170)
6170     format ('x(1) and rstatv(1) values for each replica:'/)

         do j=1, NREPS
            write (6, 6180) j, path(1, j), rstatv(1, j)
6180        format (i5, 1x, f15.9, 1x, f20.1)
         end do

         write (6, *) ''

      end if

c --- this is the output file where snapshots of the replicas will be
c     stored for analysis by another program.

      open (10, file=spfile, form='unformatted')

c --- initialize MPI.

      MPI_R=MPI_DOUBLE_PRECISION

      call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)

      call errchk(0, ierr, 100000)

      write (6, 6600) ntasks-1
6600  format ('number of child processes = ', i3/)

      if (ntasks-1.gt.130) then
         write (6, 6610)
6610     format ('too many child processes; expand the iwork array.'/
     +           'also note that write statements for HIGH '
     +           'debugging level may fail on some systems.')
         call quit
      end if

c --- this array just counts how evenly the workload was spread among
c     the child processes.

      do i=1, ntasks-1
         iwork(i)=0
      end do

c --- broadcast integer constants to all child processes.

      imsg(1)=NATOMS
      imsg(2)=NATOM3
      imsg(3)=NATOM6
      imsg(4)=NATOM7
      imsg(5)=NREPS
      imsg(6)=NIP
      imsg(7)=NPAIRS
      imsg(8)=NVBINS
      imsg(9)=idebug

      do itask=1, ntasks-1

         call MPI_SEND(imsg,
     +                 9,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0101,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100101)

      end do

      if (idebug.gt.0) open (9, file='debug.log')

      if (idebug.eq.1) write (9, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (9, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (9, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (9, 6110) idebug, 'HIGH'

c --- broadcast floating-point constants to all child processes.

      fmsg(1)=tau
      fmsg(2)=bin
      fmsg(3)=r2min
      fmsg(4)=amass
      fmsg(5)=aaxy
      fmsg(6)=aaz
      fmsg(7)=bb

      do itask=1, ntasks-1

         call MPI_SEND(fmsg,
     +                 7,
     +                 MPI_R,
     +                 itask,
     +                 0102,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100102)

      end do

c --- broadcast the interacting-pair vectors to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(vpvec,
     +                 3*NPAIRS,
     +                 MPI_R,
     +                 itask,
     +                 0103,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100103)

      end do

c --- broadcast the list of atom id numbers for the interacting pairs
c     to all child processes.
```

```
      do itask=1, ntasks-1

         call MPI_SEND(ivpair,
     +                 2*NPAIRS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0104,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100104)

      end do

c --- broadcast the size of each stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1

         call MPI_SEND(npair,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0105,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100105)

      end do

c --- broadcast the list of interacting pair id numbers that define the
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ipairs,
     +                 NIP*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0106,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100106)

      end do

c --- broadcast the potential energy curve V(R) to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(v,
     +                 2*NVBINS,
     +                 MPI_R,
     +                 itask,
     +                 0107,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100107)

      end do
```

```
c --- Now begin broadcasting interacting trimer information

      if (idebug.gt.0) write (9, *) 'end parent PART ONE'
      if (idebug.gt.0) write (9, *) ''
c ======================================================================
c     PART TWO: PERFORMING THE SIMULATION
c ======================================================================

c --- initialization of various progress counters.

c --- this is how many iterations we have done.

      loop=0

c --- these tell us about the acceptance ratio for the atom moves.

      ztacc=0.0d0
      ztrej=0.0d0

      ztacc0=0.0d0
      ztrej0=0.0d0

300   loop=loop+1

c --- these counters make sure that we don't lose a replica somewhere in
c     the ether. we use them to count how many replicas have been sent and
c     received.

      nsent=0
      nrcvd=0

c --- this is a list of flags that are zero for replicas that haven't yet
c     been sent to a child for processing, positive for replicas that have
c     been sent, and negative for replicas that have been processed and
c     returned to the parent.

c     isent(n) is set to the (positive) task id of the receiving child
c     process when a replica is sent.  this is basically leaving a trail
c     of crumbs so that we can track down the replicas and ask the children
c     to return them to us.

      do nrep=1, NREPS
         isent(nrep)=0
      end do

c --- first do all odd replicas.

      call oddrep(loop, nsent, nrcvd, MPI_R)

c --- then do all even replicas.

      call evnrep(loop, nsent, nrcvd, MPI_R)

c --- check for lost replicas.

      if (nsent.ne.NREPS.or.nrcvd.ne.NREPS) then
         write (6, *) 'replicas have been lost!'
         write (6, *) 'nsent = ', nsent
         write (6, *) 'nrcvd = ', nrcvd
         ierror=1
      end if

c --- take a snapshot every so often.

      if (mod(loop, nprint).eq.0) then
```

```
         zacc=ztacc-ztacc0
         zrej=ztrej-ztrej0

         ztacc0=ztacc
         ztrej0=ztrej

         if (idebug.gt.0) then
            write (9, 9400) zacc, zrej, 100.0d0*zacc/(zacc+zrej)
9400        format ('accepted = ', f11.0, 1x,
     +              'rejected = ', f11.0, 3x,
     +              '% accepted = ', f6.2)
            call flush(9)
         end if

c ------ we only actually take snapshots of every 11th replica.

         do k=1, NREPS, 11
            write (10) (path(i, k), i=1, NATOM3)
         end do

      end if

c --- do the next loop if needed.

      if (loop.lt.nloop) goto 300

c --- otherwise save a checkpoint file.

      write (6, 6810) svfile
6810  format ('SAVING final configuration to ', a17/)

      open (8, file=svfile, form='unformatted')

      do k=1, NREPS
         write (8) (rstatv(i, k), i=1, 8)
         write (8) (path(i, k), i=1, NATOM3)
      end do

      if (idebug.ge.3) then

         write (6, 6170)

         do k=1, NREPS
            write (6, 6180) k, path(1, k), rstatv(1, k)
         end do

         write (6, *) ''

      end if

      close (8)

      close (10)

      if (idebug.gt.0) then
         write (9, *) ''
         write (9, *) 'QSATS is done!'
         write (9, *) ''
      end if

c --- show how much work every child did.

      if (idebug.gt.0) then
         do i=1, ntasks-1
```

```
            write (9, 9100) i, iwork(i)
9100        format ('task ', i3, ' received ', i9, ' replicas')
         end do
      end if

c --- tell the children we're all done.

      do itask=1, ntasks-1

         imsg(1)=0

         call MPI_SEND(imsg,
     +                 1,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0204,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100204)

      end do

      write (6, 6900) ztacc
6900  format ('total number of accepted moves = ', f20.1)

      write (6, 6901) ztrej
6901  format ('total number of rejected moves = ', f20.1/)

      if (idebug.gt.0) write (9, *) ''
      if (idebug.gt.0) write (9, *) 'end parent PART TWO'

      return

901   write (6, *) 'error opening lattice file'
      goto 999
902   write (6, *) 'error reading number of atoms from lattice file'
      goto 999
903   write (6, *) 'error reading (unscaled) supercell edge lengths'
      goto 999
904   write (6, *) 'error reading atom number ', i
      goto 999
999   call quit

      return
      end
```

input-distortion.f

```
c ---------------------------------------------------------------------
c     this inputs the names of various I/O files and also reads in the
c     parameters for the simulation.
c     This version has been modified to accept nonequilibrium values
c     of the distortion parameters eta, gam, and epsil
c ---------------------------------------------------------------------
      subroutine input

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'

      character*8 inword
```

```
c --- read in filenames.
c     spfile = snapshot file
c     svfile = checkpoint "save" file in the main VPI program
c            = tabulated output datafile in the eloc file
c     ltfile = lattice file

      read (5, 5000, err=922) spfile
5000  format (a17)
      read (5, 5000, err=923) svfile
      read (5, 5000, err=924) ltfile

c --- set debug level.
      read (5, 5001, err=931) inword
5001  format (a8)

      if (inword.eq.'NONE') then
         idebug=0
      else if (inword.eq.'MINIMAL') then
         idebug=1
      else if (inword.eq.'LOW') then
         idebug=2
      else if (inword.eq.'MEDIUM') then
         idebug=3
      else if (inword.eq.'HIGH') then
         idebug=4
      else
         write (6, *) 'invalid debug level'
      end if

c --- read in the simulation parameters.
c     tau = imaginary time step in a.u.
c     den = number density in atoms per cubic bohr
c     amass = atomic mass of the He-4 atoms
c     aaxy = trial wavefunction a_xy parameter
c     aaz = trial wavefunction a_z parameter
c     bb = trial wavefunction b parameter
c     nloop = total MCCs
c     nprint = snapshot interval
c     eta = distortion parameter for C0 (change c/a ratio)
c     gam = distortion parameter for C66
c     epsil = distortion parameter for C44

      read (5, *, err=901) tau
      read (5, *, err=902) den
      read (5, *, err=903) amass
      read (5, *, err=904) aaxy
      read (5, *, err=904) aaz
      read (5, *, err=905) bb
      read (5, *, err=906) nloop
      read (5, *, err=907) nprint
      read (5, *, err=909) eta
      read (5, *, err=910) gam
      read (5, *, err=911) epsil

      write (6, 6000) NATOMS, NREPS
6000  format ('REPEATING input parameters'//,
     +        'atom count    = ', i6/,
     +        'replica count = ', i6/)

      write (6, 6001) tau, den, amass, aaxy, aaz, bb, eta, gam,
     +                epsil
6001  format ('tau                = ', f14.7, ' au time'/,
     +        'density            = ', f14.7, ' atoms per cubic bohr'/,
     +        'atomic mass        = ', f14.7, ' electron masses'/,
     +        'alpha-xy parameter = ', f14.7, ' bohr**(-2)'/,
     +        'alpha-z parameter  = ', f14.7, ' bohr**(-2)'/,
     +        'B parameter        = ', f14.7, ' bohr'/,
     +        'eta value          = ', f5.2, /,
     +        'gamma value        = ', f5.2, /,
     +        'epsilon value      = ', f5.2, /)

      write (6, 6002) nloop, nprint
6002  format ('number of simulation steps = ', i8/,
     +        'snapshot interval          = ', i8/)
      return

901   write (6, *) 'error reading time step value'
      goto 999
902   write (6, *) 'error reading density value'
      goto 999
903   write (6, *) 'error reading atomic mass value'
      goto 999
904   write (6, *) 'error reading aa value'
      goto 999
905   write (6, *) 'error reading bb value'
      goto 999
906   write (6, *) 'error reading nloop value'
      goto 999
907   write (6, *) 'error reading nprint value'
      goto 999
908   write (6, *) 'error reading dzscale value'
      goto 999
909   write (6, *) 'error reading eta value'
      goto 999
910   write (6, *) 'error reading gamma value'
      goto 999
911   write (6, *) 'error reading epsilon value'
      goto 999
921   write (6, *) 'error reading RNG file name'
      goto 999
922   write (6, *) 'error reading snapshot file name'
      goto 999
923   write (6, *) 'error reading save file name'
      goto 999
924   write (6, *) 'error reading lattice file name'
      goto 999
931   write (6, *) 'error reading debug level'
      goto 999
932   write (6, *) 'error reading RNG initialization mode'
      goto 999
999   call quit
      return
      end
```

child.f

```
c ---------------------------------------------------------------------
c     this is the child process that runs on all nodes except node 0
c     (which is running the parent process).
c ---------------------------------------------------------------------

      subroutine child(MPI_R)

      implicit double precision (a-h, o-z)

      include 'mpif.h'
      include 'sizes.h'
```

```
c --- child processes don't include common block, all variables are
c     local and must be defined below. Prevents child processes from
c     overwriting global variables

      common /rancm1/ rscale

      dimension replic(NATOM6), npair(NATOMS), rv(NATOM3)
      dimension istat(MPI_STATUS_SIZE)
      dimension vpvec(3, NPAIRS)
      dimension ivpair(2, NPAIRS)
      dimension ipairs(NIP, NATOMS)
      dimension xx(NATOMS), yy(NATOMS), zz(NATOMS)
      dimension r2old(NATOMS), r2new(NATOMS), v1(NATOMS), v2(NATOMS)
      dimension v(2, NVBINS)
      dimension imsg(9), fmsg(7)
      dimension rstate(8)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)

c =====================================================================
c     PART ONE: INITIALIZATION
c =====================================================================

c --- numerical factor for random number generator.

      rscale=1.0d0/4294967088.0d0

c --- determine which process this is and store it in myid.

      call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)

c --- receive all of the information that is broadcast by the parent
c     process.

c --- first receive some integer constants.  these are primarily used to
c     check that the arrays are properly dimensioned.

      call MPI_RECV(imsg,
     +              9,
     +              MPI_INTEGER,
     +              0,
     +              0101,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200101)

      istop=0

      if (imsg(1).ne.NATOMS) then
         write (6, *) 'size mismatch 1: ', imsg(1)
         istop=1
      end if
      if (imsg(2).ne.NATOM3) then
         write (6, *) 'size mismatch 2: ', imsg(2)
         istop=1
      end if
      if (imsg(3).ne.NATOM6) then
         write (6, *) 'size mismatch 3: ', imsg(3)
         istop=1
      end if
      if (imsg(4).ne.NATOM7) then
```

```
            write (6, *) 'size mismatch 4: ', imsg(4)
            istop=1
         end if
         if (imsg(5).ne.NREPS) then
            write (6, *) 'size mismatch 5: ', imsg(5)
            istop=1
         end if
         if (imsg(6).ne.NIP) then
            write (6, *) 'size mismatch 6: ', imsg(6)
            istop=1
         end if
         if (imsg(7).ne.NPAIRS) then
            write (6, *) 'size mismatch 7: ', imsg(7)
            istop=1
         end if
         if (imsg(8).ne.NVBINS) then
            write (6, *) 'size mismatch 8: ', imsg(8)
            istop=1
         end if

         if (istop.eq.1) call quit

         idebug=imsg(9)

   c --- debugging output.

         if (idebug.eq.4) write (30+myid, *) 'idebug = ', idebug

   c --- next receive some floating-point constants.

         call MPI_RECV(fmsg,
        +              7,
        +              MPI_R,
        +              0,
        +              0102,
        +              MPI_COMM_WORLD,
        +              istat,
        +              ierr)

         call errchk(myid, ierr, 200102)

         tau=fmsg(1)
         bin=fmsg(2)
         r2min=fmsg(3)
         amass=fmsg(4)
         aaxy=fmsg(5)
         aaz=fmsg(6)
         bb=fmsg(7)

         if (idebug.eq.4) then
            write (30+myid, *) 'tau = ', tau
            write (30+myid, *) 'bin = ', bin
            write (30+myid, *) 'r2min = ', r2min
            write (30+myid, *) 'amass = ', amass
            write (30+myid, *) 'aaxy = ', aaxy
            write (30+myid, *) 'aaz = ', aaz
            write (30+myid, *) 'bb = ', bb
         end if

   c --- compute the inverse of the potential energy V(R) bin width, to
   c     avoid unnecessary divisions.

         binvrs=one/bin

   c --- compute gaussian scaling parameters.
```

```
      gscale=sqrt(half*tau/amass)
      gscal2=sqrt(tau/amass)

c --- next receive the vectors that connect pairs of atoms in a stencil.

      call MPI_RECV(vpvec,
     +              3*NPAIRS,
     +              MPI_R,
     +              0,
     +              0103,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200103)

c --- next receive the list of pairs of atoms.

      call MPI_RECV(ivpair,
     +              2*NPAIRS,
     +              MPI_INTEGER,
     +              0,
     +              0104,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200104)

c --- next receive the number of atoms that belong to each atom's stencil.
c     this should really be the same for every atom for a regular crystal
c     lattice, but we treat it as a variable.

      call MPI_RECV(npair,
     +              NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0105,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200105)

c --- next receive the pairs that constitute each atom's stencil.

      call MPI_RECV(ipairs,
     +              NIP*NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0106,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200106)

c --- next receive the potential energy curve V(R) for interpolation.

      call MPI_RECV(v,
     +              2*NVBINS,
     +              MPI_R,
     +              0,
     +              0107,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200107)

      if (idebug.eq.4) then
         write (30+myid, *) 'child moving to PART TWO'
         call flush(30+myid)
      end if
c ======================================================================
c     PART TWO: PERFORMING THE SIMULATION
c ======================================================================

100   idrep=0

      nacc=0
      nrej=0

c --- send request for data (message type 1201) to parent.  the first
c     time through, or if we are waiting for all children to sync up,
c     there are no results to send back to the parent, so we indicate
c     this by setting idrep=0 just above, and then sending this to
c     the parent in imsg(1).

200   imsg(1)=idrep

      imsg(2)=nacc
      imsg(3)=nrej

      call MPI_SEND(imsg,
     +              3,
     +              MPI_INTEGER,
     +              0,
     +              1201,
     +              MPI_COMM_WORLD,
     +              ierr)

      call errchk(myid, ierr, 201201)

c --- on the other hand, if there are results to send back, then we
c     do so here.

      if (idrep.gt.0) then

c ------ first we send a message of type 1202 that contains the atoms'
c        new positions.

         call MPI_SEND(replic,
     +                 NATOM3,
     +                 MPI_R,
     +                 0,
     +                 1202,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201202)

c ------ then we send a message of type 1203 that contains the updated
c        random number generator state vector.

         call MPI_SEND(rstate,
     +                 8,
     +                 MPI_DOUBLE_PRECISION,
     +                 0,
     +                 1203,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(myid, ierr, 201203)

      end if

c --- wait for acknowledgement (message type 0204) from parent.  the
c     parent also uses this to signal the child that more input will
c     be sent.

c     if imsg(1) is positive, it is a replica number that represents the
c     next replica that this child should process.

c     if imsg(1) is negative, then this child needs to wait for the
c     other children to sync up, and so the child goes back to the top
c     of PART TWO.

c     if imsg(1) is zero, there is no more work to be done.

      call MPI_RECV(imsg,
     +              1,
     +              MPI_INTEGER,
     +              0,
     +              0204,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200204)

c --- loop back and wait for more input if instructed by parent.

      if (imsg(1).lt.0) goto 100

c --- terminate if the simulation is complete.

      if (imsg(1).eq.0) then
         if (idebug.eq.4) write (30+myid, *) 'child is done!'
         return
      end if

c --- if there is a new replica to process, then receive data from
c     the parent.

c --- we need to save the replica number that we are about to work on.

      idrep=imsg(1)

c --- first receive the loop number, in a message of type 0207.

      call MPI_RECV(loop,
     +              1,
     +              MPI_INTEGER,
     +              0,
     +              0207,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200207)

c --- next receive the old atomic coordinates and the means of the
c     neighboring replicas' coordinates, in a message of type 0205.
```

```
      call MPI_RECV(replic,
     +              NATOM6,
     +              MPI_R,
     +              0,
     +              0205,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200205)

c --- next receive the random number generator state vector, in
c     a message of type 0206.

      call MPI_RECV(rstate,
     +              8,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0206,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200206)

c --- generate provisional new atomic positions by adding gaussian
c     displacements.

c --- first choose the appropriate gaussian scaling factor.

      if (idrep.eq.1.or.idrep.eq.NREPS) then
         gsc=gscal2
      else
         gsc=gscale
      end if

c --- then add the gaussian displacements.

      do nn=1, NATOM3
         call gstep(rstate, gg, rscale)
         replic(NATOM3+nn)=replic(NATOM3+nn)+gg*gsc
      end do

c --- attempt to move each atom in turn.

      nacc=0
      nrej=0

      do nn=1, NATOMS

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'moving atom 1'
               call flush(30+myid)
            end if
         end if

c ------ set up the coordinates of the atoms that are in this atom's
c        stencil.

         do i=1, npair(nn)
```

```
            ip=ipairs(i, nn)
            j=ivpair(2, ip)

            xx(i)=replic(3*j-2)+vpvec(1, ip)
            yy(i)=replic(3*j-1)+vpvec(2, ip)
            zz(i)=replic(3*j-0)+vpvec(3, ip)

         end do

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'after do loop, xx(1) = ', xx(1)
               call flush(30+myid)
            end if
         end if

c ------ get the old and new coordinates of the atom that we're about
c        to try to move.

         xold=replic(3*nn-2)
         yold=replic(3*nn-1)
         zold=replic(3*nn-0)

         xnew=replic(3*nn-2+NATOM3)
         ynew=replic(3*nn-1+NATOM3)
         znew=replic(3*nn-0+NATOM3)

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'xold, xnew = ', xold, xnew
               call flush(30+myid)
            end if
         end if

c ------ compute the old and new distances between this atom and all
c        of the atoms in the stencil.

c ------ the do loops are split up to promote vectorization, although
c        i'm not sure this is necessary.

         do i=1, npair(nn)
            r2old(i)=(xx(i)-xold)**2
         end do

         do i=1, npair(nn)
            r2old(i)=r2old(i)+(yy(i)-yold)**2
         end do

         do i=1, npair(nn)
            r2old(i)=r2old(i)+(zz(i)-zold)**2
         end do

         do i=1, npair(nn)
            r2new(i)=(xx(i)-xnew)**2
         end do

         do i=1, npair(nn)
            r2new(i)=r2new(i)+(yy(i)-ynew)**2
         end do

         do i=1, npair(nn)
```

```
            r2new(i)=r2new(i)+(zz(i)-znew)**2
         end do

c ------ compute the change in potential energy.

         do i=1, npair(nn)

c --------- use linear interpolation.

            ibin1=int((r2old(i)-r2min)*binvrs)+1
            ibin2=int((r2new(i)-r2min)*binvrs)+1

            if (ibin1.gt.0) then
               dr1=(r2old(i)-r2min)-bin*dble(ibin1-1)
               v1(i)=v(1, ibin1)+v(2, ibin1)*dr1
            else
               v1(i)=v(1, 1)
            end if

            if (ibin2.gt.0) then
               dr2=(r2new(i)-r2min)-bin*dble(ibin2-1)
               v2(i)=v(1, ibin2)+v(2, ibin2)*dr2
            else
               v2(i)=v(1, 1)
            end if

         end do

         dv=0.0

         do i=1, npair(nn)
            dv=dv+v1(i)-v2(i)
         end do

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'dv = ', dv
               call flush(30+myid)
            end if
         end if

         dv=dv*tau

c ------ deal with trial function for first and last replicas.

         if (idrep.eq.1.or.idrep.eq.NREPS) then

            dpsi=0.0

            do i=1, npair(nn)
               dpsi=dpsi+
     +              (1.0d0/sqrt(r2old(i)))**5-
     -              (1.0d0/sqrt(r2new(i)))**5
            end do

            soldxy=xold**2+yold**2
            snewxy=xnew**2+ynew**2

            dpsi=0.5d0*bb**5*dpsi+aaxy*(soldxy-snewxy)+
     +           aaz*(zold**2-znew**2)

c --------- debugging output.
```

```
            if (nn.eq.1) then
               if (idebug.eq.4) then
                  write (30+myid, *) 'evaluating trial function'
                  write (30+myid, *) 'dpsi = ', dpsi
                  call flush(30+myid)
               end if
            end if

c --------- also remember to scale the change in potential energy by
c           one-half for the end replicas.

            dv=half*dv+dpsi
         end if

c ------ choose whether to accept the new position.

         call rstep(rstate, zran, rscale)
         if (dv.ge.0.0) then

c --------- accept this move.

            replic(3*nn-2)=xnew
            replic(3*nn-1)=ynew
            replic(3*nn-0)=znew

            nacc=nacc+1

         else if (zran.lt.exp(dv)) then

c --------- accept this move.

            replic(3*nn-2)=xnew
            replic(3*nn-1)=ynew
            replic(3*nn-0)=znew

            nacc=nacc+1
         else

c --------- reject this move.

            nrej=nrej+1
         end if

c --- end of loop over atoms.

      end do

c --- go back to send these results back to the parent.

      goto 200
      end
```

qsats.h

```
c -----------------------------------------------------------------------

c     this file contains the common blocks used by QSATS.

c -----------------------------------------------------------------------

c --- parameters and counters for the VPI simulation.

      common /monte/  zaccv(NREPS, NATOMS), zrejv(NREPS, NATOMS),
     +                naccv(NATOMS), nrejv(NATOMS), zm,
     +                tau, den, scale, amass, ztacc, ztrej,
     +                nph2, nloop, nprint, nacc, nrej, irrst, idebug

c --- trial wave function parameters.

      common /psitri/ aaxy, aaz, bb, eta, gam, epsil, phi

c --- random number generator variables.

      double precision zm1, zm2, rm1, rm2, rscale, rstatv

      common /moduli/ zm1, zm2, rm1, rm2

      common /rancm1/ rscale

      common /rancm2/ rstatv(8, NREPS)

c --- potential energy lookup table.

      common /potcom/ v(2, NVBINS)

c --- VPI replicas and atomic masses.

      common /vpi/    path(NATOM3, NREPS),
     +                pathnu(NATOM3, NREPS),
     +                zmass(NATOM3)

c --- filenames.

      character*17 spfile, svfile, ltfile

      common /files/  spfile, svfile, ltfile

c --- description of the crystal lattice.

      common /crystl/ xtal(NATOMS, 3)

      common /box/    xlen, ylen, zlen, dxmax, dymax, dzmax

c --- arrays dealing with interacting pairs and trimers.

      common /vpairs/ vpvec(3, NPAIRS),
     +                ivpair(2, NPAIRS), ivpair1(2, NPAIRS),
     +                ipairs(NIP, NATOMS),
     +                npair(NATOMS),
     +                nvpair

c --- counters to monitor load balancing.

      common /parcom/ iwork(130)
```

eloc-distortion.f

```
c ----------------------------------------------------------------------
c     this computes the total energy and the expectation value of the
c     potential energy from the snapshots recorded by QSATS.
c     This version of the code accounts for strain applied to the
c     lattice as determined by the eta, gamma, and epsilon parameters
c     (see Cazorla, Phys. Rev. B 85, 024101 (2012)) for the purpose
c     of calculating elastic constants.
c ----------------------------------------------------------------------

      program eloc

      implicit double precision (a-h, o-z)
      real*8 lrctot

      include 'sizes.h'
      include 'qsats.h'

      parameter (bohr=0.529177249d0)
c --- this common block is used to enable interpolation in the potential
c     energy lookup table in the subroutine local below.

      common /bincom/ bin, binvrs, r2min

      dimension q(NATOM3), vtavg(NREPS), vtavg2(NREPS),
     +          etavg(NREPS), etavg2(NREPS), u2xavg(NREPS),
     +          u2yavg(NREPS), u2zavg(NREPS), u4xavg(NREPS),
     +          u4yavg(NREPS), u4zavg(NREPS)

      parameter (half=0.5d0)
      parameter (one=1.0d0)

c --- initialization.

      call tstamp

      write (6, 6001) NREPS, NATOMS, NATOM3, NATOM6, NATOM7,
     +                NVBINS, RATIO, NIP, NPAIRS
6001  format ('compile-time parameters:'//,
     +       'NREPS  = ', i6/,
     +       'NATOMS = ', i6/,
     +       'NATOM3 = ', i6/,
     +       'NATOM6 = ', i6/,
     +       'NATOM7 = ', i6/,
     +       'NVBINS = ', i6/,
     +       'RATIO  = ', f6.4/,
     +       'NIP    = ', i6/,
     +       'NPAIRS = ', i6/)

      call input

      call vinit(r2min, bin)

      binvrs=one/bin

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a16/)

      open (8, file=ltfile, status='old')

      read (8, *) nlpts
      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         stop
      end if

c --- define strain parameter phi from eta (read in input)

      phi = sqrt(1+eta)

c --- read the edge lengths of the supercell.

      read (8, *) xlen, ylen, zlen

c --- compute a distance scaling factor.
      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      write (6, 6310) xlen, ylen, zlen
6310  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

c --- this variable helps us remember the nearest-neighbor distance.
c --- The nearest neighbor distance and interacting pairs are determined
c     from the undistorted lattice and then distortion is applied after.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)
```

```
c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6330) rnnmin
6330  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
      end do

      nvpair=0

      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if
            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.

c --------- we determine the interacting pairs from the undistorted
c           then use our values of eta (phi), gamma, and epsilon to
c           impose the distortions for the elastic constant
c           calculations.
            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

c ------------ these transformations impose the lattice distortions
c              They reduce to dx, dy, and dz for eta = 0 (phi = 1),
c              gamma = 1 and epsilon = 0.

               vpvec(1, nvpair)=dx/(sqrt(gam)*phi)
               vpvec(2, nvpair)=dy*sqrt(gam)/phi
               vpvec(3, nvpair)=dz*phi**2+dy*epsil

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair

            end if

         end if

      end do
      end do

c --- Now loop back through the coordinates in the xtal array and
c     transform them appropriately

      do i=1, NATOMS

         xtal(i, 1)=xtal(i, 1)/(sqrt(gam)*phi)
         xtal(i, 2)=xtal(i, 2)*sqrt(gam)/phi
         xtal(i, 3)=xtal(i, 3)*phi**2+epsil*xtal(i, 2)

      end do

c --- write out interacting pair information

      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +        'total number of interacting pairs = ', i6/)

c --- initialization.

      loop=0
      do k=1, NREPS
         vtavg(k)=0.0d0
         etavg(k)=0.0d0
         vtavg2(k)=0.0d0
         etavg2(k)=0.0d0
         u2xavg(k)=0.0d0
         u2yavg(k)=0.0d0
         u2zavg(k)=0.0d0
         u4xavg(k)=0.0d0
         u4yavg(k)=0.0d0
         u4zavg(k)=0.0d0
      end do

      open (10, file=spfile, form='unformatted')

c --- this loops reads the snapshots saved by QSATS.

300   loop=loop+1

      do k=1, NREPS, 11

         read (10, end=600) (path(i, k), i=1, NATOM3)

c ------ compute the local energy and the potential energy.

         do i=1, NATOM3
            q(i)=path(i, k)
         end do

         call local(q, tloc, vloc)

c ------ convert to kelvin per atom.

         tloc=tloc/(3.1668513d-6*dble(NATOMS))
         vloc=vloc/(3.1668513d-6*dble(NATOMS))

c ------ accumulate the results.

         vtavg(k)=vtavg(k)+vloc
         vtavg2(k)=vtavg2(k)+(vloc)**2
         etavg(k)=etavg(k)+tloc+vloc
         etavg2(k)=etavg2(k)+(tloc+vloc)**2

c ------ compute <u^2> in all three directions

         call msd(q, u2x, u2y, u2z, u4x, u4y, u4z)
         u2xavg(k) = u2xavg(k)+u2x
         u2yavg(k) = u2yavg(k)+u2y
         u2zavg(k) = u2zavg(k)+u2z

         u4xavg(k) = u4xavg(k)+u4x
         u4yavg(k) = u4yavg(k)+u4y
         u4zavg(k) = u4zavg(k)+u4z

350      continue

      end do

      goto 300

c --- account for overshooting.

600   loop=loop-1

      write (6, 6600) loop
6600  format ('number of snapshots = ', i6/)
```

```
c --- compute the averages and standard deviations.

      b2 = bohr*bohr
      open (4, file=svfile)
      do k=1, NREPS, 11

      vtavg(k)=vtavg(k)/dble(loop)
      vtavg2(k)=vtavg2(k)/dble(loop)
      etavg(k)=etavg(k)/dble(loop)
      etavg2(k)=etavg2(k)/dble(loop)

      vsd=sqrt(vtavg2(k)-vtavg(k)**2)
      esd=sqrt(etavg2(k)-etavg(k)**2)

      u2xavg(k)=u2xavg(k)/dble(loop)
      u2yavg(k)=u2yavg(k)/dble(loop)
      u2zavg(k)=u2zavg(k)/dble(loop)

      u4xavg(k)=u4xavg(k)/dble(loop)
      u4yavg(k)=u4yavg(k)/dble(loop)
      u4zavg(k)=u4zavg(k)/dble(loop)

      u2xsd=sqrt(u4xavg(k)-u2xavg(k)**2)
      u2ysd=sqrt(u4yavg(k)-u2yavg(k)**2)
      u2zsd=sqrt(u4zavg(k)-u2zavg(k)**2)

c --- calculate gaussian parameters for each replica for use in
c     the long-range correction calculation

      axprm = b2/(4.0d0*u2xavg(k))
      ayprm = b2/(4.0d0*u2yavg(k))
      azprm = b2/(4.0d0*u2zavg(k))

      axsd = b2*sqrt((1.0d0/(16.0d0*u2xavg(k)**2))*u2xsd**2)
      aysd = b2*sqrt((1.0d0/(16.0d0*u2yavg(k)**2))*u2ysd**2)
      azsd = b2*sqrt((1.0d0/(16.0d0*u2zavg(k)**2))*u2zsd**2)

      call lrc(axprm, ayprm, azprm, rnnmin, vlrc)

      write (6, 6610) k, 'VAVG = ', vtavg(k)
6610  format ('replica ', i3, 1x, a9, f10.5, ' Kelvin')
6620  format ('replica ', i3, 1x, a9, 1pe13.6, ' Angstrom**2')

      write (6, 6610) k, 'V SD =   ', vsd

      write (6, 6610) k, 'EAVG =   ', etavg(k)

      write (6, 6610) k, 'E SD =   ', esd

      write (6, 6620) k, 'u2x  =   ', u2xavg(k)

      write (6, 6620) k, 'u2x sd = ', u2xsd

      write (6, 6620) k, 'u2y  =   ', u2yavg(k)

      write (6, 6620) k, 'u2y sd = ', u2ysd

      write (6, 6620) k, 'u2z  =   ', u2zavg(k)

      write (6, 6620) k, 'u2z sd = ', u2zsd

      write (4, 6630) k, vtavg(k), vsd,
     +                etavg(k), esd, u2xavg(k), u2xsd,
     +                u2yavg(k), u2ysd, u2zavg(k), u2zsd,
     +                axprm, axsd, ayprm, aysd, azprm, azsd, vlrc
```

```
6630  format(1x, i3, 4(1x, 1pe13.6), 6(1x, 1pe13.6), 6(1x, 1pe13.6),
     +       1x, 1pe13.6)

      end do

      stop
      end

c ----------------------------------------------------------------------

c     this subroutine computes the local energy and potential energy
c     of a configuration.

c ----------------------------------------------------------------------

      subroutine local(q, tloc, vloc)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'

      common /bincom/ bin, binvrs, r2min

c --- alpha is the exponential parameter in psi:

c     psi = N * exp(-alpha*(r-r0)**2) * Jastrow

c --- bb is the exponential parameter in Jastrow:

c     ln Jastrow(ij) = -0.5 * (bb/rij)**5

      dimension q(NATOM3), dlng(NATOM3), d2lng(NATOM3)

      do i=1, NATOM3
         dlng(i)=0.0d0
         d2lng(i)=0.0d0
      end do

      do i=1, NATOMS

         xx=q(3*i-2)
         yy=q(3*i-1)
         zz=q(3*i)

         dlng(3*i-2)=dlng(3*i-2)-2.0d0*aaxy*xx
         dlng(3*i-1)=dlng(3*i-1)-2.0d0*aaxy*yy
         dlng(3*i)  =dlng(3*i)  -2.0d0*aaz*zz

         d2lng(3*i-2)=d2lng(3*i-2)-2.0d0*aaxy
         d2lng(3*i-1)=d2lng(3*i-1)-2.0d0*aaxy
         d2lng(3*i)  =d2lng(3*i)  -2.0d0*aaz

      end do

c --- loop over all interacting pairs.

      vloc=0.0d0
      tloc=0.0d0

      do n=1, nvpair

         i=ivpair(1, n)
         j=ivpair(2, n)
         dx=-((q(3*j-2))+vpvec(1, n)+(-q(3*i-2)))
         dy=-((q(3*j-1))+vpvec(2, n)+(-q(3*i-1)))
         dz=-((q(3*j))  +vpvec(3, n)+(-q(3*i))  )

         r2=dx*dx+dy*dy+dz*dz

         ibin=int((r2-r2min)*binvrs)+1

         if (ibin.gt.0) then
            dr=(r2-r2min)-bin*dble(ibin-1)
            vloc=vloc+v(1, ibin)+v(2, ibin)*dr
         else
            vloc=vloc+v(1, 1)
         end if

         br2=bb*bb/r2

         br5=br2*br2*sqrt(br2)

         br52=br5/r2

         dlng(3*i-2)=dlng(3*i-2)+2.5d0*br52*dx
         dlng(3*i-1)=dlng(3*i-1)+2.5d0*br52*dy
         dlng(3*i)  =dlng(3*i)  +2.5d0*br52*dz

         d2lng(3*i-2)=d2lng(3*i-2)+2.5d0*br52*
     *                            (1.0d0-7.0d0*dx**2/r2)
         d2lng(3*i-1)=d2lng(3*i-1)+2.5d0*br52*
     *                            (1.0d0-7.0d0*dy**2/r2)
         d2lng(3*i)  =d2lng(3*i)  +2.5d0*br52*
     *                            (1.0d0-7.0d0*dz**2/r2)

      end do

c --- now sum up the kinetic energy components.

      do i=1, NATOM3
         tloc=tloc+d2lng(i)+dlng(i)**2
      end do

c --- account for mass factor and for double-counting of pairs.

      tloc=-0.5d0*tloc/amass
      vloc=0.5d0*vloc

      return
      end

c ----------------------------------------------------------------------

c     msd is a subroutine that calculates the mean squared displacement
c     in all three directions from the snapshots.

c ----------------------------------------------------------------------

       subroutine msd(q, u2x, u2y, u2z, u4x, u4y, u4z)

       implicit real*8 (a-h, o-z)

       include 'sizes.h'

       include 'qsats.h'

       dimension q(NATOM3)
       parameter (bohr=0.529177249d0)
```

```
        u2x = 0.0d0
        u2y = 0.0d0
        u2z = 0.0d0

        u4x = 0.0d0
        u4y = 0.0d0
        u4z = 0.0d0

        do l = 1, NATOMS
           u2x = u2x + (q(3*l-2)**2)
           u4x = u4x + (q(3*l-2)**4)

           u2y = u2y + (q(3*l-1)**2)
           u4y = u4y + (q(3*l-1)**4)

           u2z = u2z + (q(3*l)**2)
           u4z = u4z + (q(3*l)**4)
       end do

c --- conversion factor from bohr**2 to angstrom**2

      b2=bohr*bohr

      u2x = u2x*b2/dble(NATOMS)
      u2y = u2y*b2/dble(NATOMS)
      u2z = u2z*b2/dble(NATOMS)

      u4x = u4x*b2*b2/dble(NATOMS)
      u4y = u4y*b2*b2/dble(NATOMS)
      u4z = u4z*b2*b2/dble(NATOMS)

      return
      end
c ----------------------------------------------------------------------

c     quit is a subroutine used to terminate execution if there is
c     an error.

c     it is needed here because the subroutine that reads the parameters
c     (subroutine input) may call it.

c ----------------------------------------------------------------------

      subroutine quit

      write (6, *) 'termination via subroutine quit'

      stop

      return
      end
```

B.2 VPI Perturbative 3-body Correction Program (VPI-3B)

The perturbative three-body correction to the VPI-2B energy is calculated using the VPI-3B program described below. This replaces the eloc-distortion.f program associated with the VPI-2B program above and uses the same input. The files required to compile and run this program are listed below:

eloc-3b-distortion.f	Main program which calls the input subroutine and calculates the average energies for each replica from the VPI snapshot file.
input-distortion.f	See **input-distortion.f** above.
vinit.f	See Sec. A.1.
he3fci.f	See Sec. A.2.
lrc-3d-sub.f	See Sec. B.1.
c6-sub.f	See Sec. B.1.
tstamp.f	See Sec. A.1.
Gauss-Hermite.dat	See Sec. A.1.1.
sizes.h	Replaces sizes.h in Sec. B.1. This version of sizes.h also includes parameters necessary for calculating the total number of interacting trimers.
qsats.h	Replaces qsats.h in Sec. B.1. This version of qsats.h also includes all parameters related to the interacting trimers.

eloc-3b-distortion.f

```
c ----------------------------------------------------------------------

c     this computes the total energy and the expectation value of the
c     potential energy from the snapshots recorded by QSATS.
c     The three-body energy is calculated here considering only those
c     trimers formed by first nearest neighbors.
c     The pairwise additive potential energy and total energy are
c     reported in the absence of three-body interactions, but the
c     perturbative three-body contribution is also reported.

c     This version of the code accounts for strain applied to the
c     lattice as determined by the eta, gamma, and epsilon parameters
c     (see Cazorla, Phys. Rev. B 85, 024101 (2012)) for the purpose
c     of calculating elastic constants. The dzscale parameter is no
c     longer used in this version of the code, though it is still read
c     into input.
c ----------------------------------------------------------------------

      program eloc3bdistortion

      implicit double precision (a-h, o-z)
      real*8 lrctot
      include 'sizes.h'
      include 'qsats.h'

      parameter (bohr=0.529177249d0)

c --- this common block is used to enable interpolation in the potential
c     energy lookup table in the subroutine local below.

      common /bincom/ bin, binvrs, r2min

c --- set up arrays to calculate potential (2 and 3 body), kinetic,
c     and total energy per atom for each replica, as well as mean
c     squared displacement and all uncertainties.
      dimension q(NATOM3), vtavg(NREPS), vtavg2(NREPS),
     +          etavg(NREPS), etavg2(NREPS), v3avg(NREPS),
     +          v3avg2(NREPS), u2xavg(NREPS),
     +          u2yavg(NREPS), u2zavg(NREPS), u4xavg(NREPS),
     +          u4yavg(NREPS), u4zavg(NREPS)

      parameter (half=0.5d0)
      parameter (one=1.0d0)

c --- initialization.

      call tstamp

      write (6, 6001) NREPS, NATOMS, NATOM3, NATOM6, NATOM7,
     +                NVBINS, RATIO, NIP, NPAIRS
6001  format ('compile-time parameters:'//,
     +        'NREPS  = ', i6/,
     +        'NATOMS = ', i6/,
     +        'NATOM3 = ', i6/,
     +        'NATOM6 = ', i6/,
     +        'NATOM7 = ', i6/,
     +        'NVBINS = ', i6/,
     +        'RATIO  = ', f6.4/,
     +        'NIP    = ', i6/,
     +        'NPAIRS = ', i6/)

      call input

      call vinit(r2min, bin)

      binvrs=one/bin

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a17/)

      open (8, file=ltfile, status='old')

      read (8, *) nlpts

      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         stop
      end if

c --- calculate distortion parameter phi from eta
      phi = sqrt(1+eta)

c --- read the edge lengths of the supercell.
```

```
      read (8, *) xlen, ylen, zlen

c --- compute a distance scaling factor.
      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      write (6, 6310) xlen, ylen, zlen
6310  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

c --- this variable helps us remember the nearest-neighbor distance.
c --- The nearest neighbor distance and interacting pairs/trimers are determined
c     from the undistorted lattice and then distortion is applied after.
      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.
         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6330) rnnmin
6330  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
      end do

      nvpair=0

      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount. Nearest neighbors are determined
c           before distortion is applied.
```

```
            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j
c ----------- account for distortion in displacement vectors
               vpvec(1, nvpair)=dx/(sqrt(gam)*phi)
               vpvec(2, nvpair)=dy*sqrt(gam)/phi
               vpvec(3, nvpair)=dz*phi**2+dy*epsil

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair
c ------------ for three-body calculations, keep track of only first
c              nearest neighbors.

               if (r/rnnmin.lt.1.05) then

               nvpair1=nvpair1+1

c ------------ store information about this pair (i->j) in arrays.

               ivpair1(1, nvpair1)=i
               ivpair1(2, nvpair1)=j

               vpvec1(1, nvpair1)=dx/(sqrt(gam)*phi)
               vpvec1(2, nvpair1)=dy*sqrt(gam)/phi
               vpvec1(3, nvpair1)=dz*phi**2+dy*epsil

               npair1(i)=npair1(i)+1

               ipairs1(npair1(i), i)=nvpair1

            end if
          end if
         end if

      end do
      end do

c --- Now loop back through the coordinates in the xtal array and
c     transform them appropriately

      do i=1, NATOMS

         xtal(i, 1)=xtal(i, 1)/(sqrt(gam)*phi)
         xtal(i, 2)=xtal(i, 2)*sqrt(gam)/phi
         xtal(i, 3)=xtal(i, 3)*phi**2+epsil*xtal(i, 2)

      end do

c --- write out interacting pair information

      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +        'total number of interacting pairs = ', i6/)

c --- To save time later, we are going to calculate all of the first
c     nearest neighbor trimers now, along with angles and vectors.
      nvtrim=0

      do i = 1, NATOMS
        do j = 1, npair1(i)-1
        do k = j+1, npair1(i)
c--------- keep running count of all trimers
           nvtrim = nvtrim+1
c -------- record vector from central atom to two neighbors

           npairA = ipairs1(j, i) !tells us the nvpair reference number
           npairB = ipairs1(k, i) ! for each of the atoms in the trimer

c--------- keep record of atoms in trimer
           ivtrim(1, nvtrim) = i !central atom
           ivtrim(2, nvtrim) = ivpair1(2, npairA)
           ivtrim(3, nvtrim) = ivpair1(2, npairB)

           vpvectri(1, nvtrim) = vpvec1(1, npairA)
           vpvectri(2, nvtrim) = vpvec1(2, npairA)
           vpvectri(3, nvtrim) = vpvec1(3, npairA)

           vpvectri(4, nvtrim) = vpvec1(1, npairB)
           vpvectri(5, nvtrim) = vpvec1(2, npairB)
           vpvectri(6, nvtrim) = vpvec1(3, npairB)
c -------- calculate and store side lenghts and central angle
           dx1 = vpvectri(1, nvtrim)
           dy1 = vpvectri(2, nvtrim)
           dz1 = vpvectri(3, nvtrim)

           dx2 = vpvectri(4, nvtrim)
           dy2 = vpvectri(5, nvtrim)
           dz2 = vpvectri(6, nvtrim)

           dx12 = dx2-dx1
           dy12 = dy2-dy1
           dz12 = dz2-dz1

           side1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
           side2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
           side3 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)

           vpvectri(7, nvtrim) = side1
           vpvectri(8, nvtrim) = side2
           vpvectri(9, nvtrim) = side3
c -------- We know sides 1 and 2 = Rnn. If side 3 is lt or
c          equal to Rnn, trimer will be triple counted
c -------- first calculate undistorted side 3 length

           dx12_und = dx12*phi*sqrt(gam)
           dy12_und = dy12*phi/sqrt(gam)
           dz12_und = (dz12-dy12_und*epsil)/(phi**2)

           side3_und = sqrt(dx12_und*dx12_und+dy12_und*dy12_und+
     +                     dz12_und*dz12_und)
c -------- Test side 3 to see if it is less than 1.05*Rnn (approx = Rnn)
c          If it is, then we have an equilateral timer in the
c          undistorted lattice which will be triple counted.
           ivtrim(4, nvtrim) = 1

           if (side3_und/rnnmin.lt.1.05) then
              ivtrim(4, nvtrim) = 3
           end if

c -------- Update number of trimers for given central atom
           ntrim(i) = ntrim(i)+1

           itrims(ntrim(i), i) = nvtrim
        end do
        end do
```

```
      end do

      write (6, 6403) ntrim(1), nvtrim
6403  format ('atom 1 forms ', i3, 'trimers with interacting
     + neighbors'//, 'total number of interacting trimers = ', i9)

      do i=1, npair(1)
         ip=ipairs(i, 1)
         d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +     rnnmin
         write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                   vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410     format ('vector # ', i3, ' to atom ', i4, ': ',
     +           3(1x, f9.5), ' length = ', f8.5)
      end do

         write (6, 6402)
6402     format (/'interaction trimer side lengths for atom 1 ',
     +           '[NN distances]:'/)
         do i=1, ntrim(1)
            itri = itrims(i, 1)
            write(6, 6411) itri, ivtrim(2, itri), ivtrim(3, itri),
     +                     vpvectri(7, itri)/rnnmin, vpvectri(8,
     +                     itri)/rnnmin, vpvectri(9, itri)/rnnmin,
     +                     ivtrim(4, itri)
6411        format('trimer # ', i3, 'incuding atoms ', i4, 1x, i4, ':',
     +           'side lengths: ', 3(1x, f8.5), ' counted: ',
     +            i1, 1x, 'times')
         end do

c --- initialization.

      loop=0
      do k=1, NREPS
         vtavg(k)=0.0d0
         etavg(k)=0.0d0
         vtavg2(k)=0.0d0
         etavg2(k)=0.0d0
         u2xavg(k)=0.0d0
         u2yavg(k)=0.0d0
         u2zavg(k)=0.0d0
         u4xavg(k)=0.0d0
         u4yavg(k)=0.0d0
         u4zavg(k)=0.0d0
      end do

      open (10, file=spfile, form='unformatted')

c --- this loops reads the snapshots saved by QSATS.

300   loop=loop+1

      do k=1, NREPS, 11

         read (10, end=600) (path(i, k), i=1, NATOM3)

c ------ compute the local energy and the potential energy.

         do i=1, NATOM3
            q(i)=path(i, k)
         end do

         call local(q, tloc, vloc, pot3b)

c ------ convert to kelvin per atom.
```

```
         tloc=tloc/(3.1668513d-6*dble(NATOMS))
         vloc=vloc/(3.1668513d-6*dble(NATOMS))
         pot3b=pot3b/(3.1668513d-6*dble(NATOMS))

c ------ accumulate the results.
c ------ note, vloc does not include pot3b
         v3avg(k)=v3avg(k)+pot3b
         v3avg2(k)=v3avg2(k)+(pot3b)**2
c ------ note: these following energies do not contain three-body
c        contributions

         vtavg(k)=vtavg(k)+vloc
         vtavg2(k)=vtavg2(k)+(vloc)**2
         etavg(k)=etavg(k)+tloc+vloc
         etavg2(k)=etavg2(k)+(tloc+vloc)**2

c ------ compute <u^2> in all three directions

         call msd(q, u2x, u2y, u2z, u4x, u4y, u4z)

         u2xavg(k) = u2xavg(k)+u2x
         u2yavg(k) = u2yavg(k)+u2y
         u2zavg(k) = u2zavg(k)+u2z

         u4xavg(k) = u4xavg(k)+u4x
         u4yavg(k) = u4yavg(k)+u4y
         u4zavg(k) = u4zavg(k)+u4z

350      continue

      end do

      goto 300

c --- account for overshooting.

600   loop=loop-1

      write (6, 6600) loop
6600  format ('number of snapshots = ', i6/)

c --- compute the averages and standard deviations.
      b2 = bohr*bohr
      open (4, file=svfile)
      do k=1, NREPS, 11

      v3avg(k)=v3avg(k)/dble(loop)
      v3avg2(k)=v3avg2(k)/dble(loop)
      vtavg(k)=vtavg(k)/dble(loop)
      vtavg2(k)=vtavg2(k)/dble(loop)
      etavg(k)=etavg(k)/dble(loop)
      etavg2(k)=etavg2(k)/dble(loop)

      v3sd=sqrt(v3avg2(k)-v3avg(k)**2)
      vsd=sqrt(vtavg2(k)-vtavg(k)**2)
      esd=sqrt(etavg2(k)-etavg(k)**2)

      u2xavg(k)=u2xavg(k)/dble(loop)
      u2yavg(k)=u2yavg(k)/dble(loop)
      u2zavg(k)=u2zavg(k)/dble(loop)

      u4xavg(k)=u4xavg(k)/dble(loop)
      u4yavg(k)=u4yavg(k)/dble(loop)
      u4zavg(k)=u4zavg(k)/dble(loop)
      u2xsd=sqrt(u4xavg(k)-u2xavg(k)**2)
      u2ysd=sqrt(u4yavg(k)-u2yavg(k)**2)
      u2zsd=sqrt(u4zavg(k)-u2zavg(k)**2)

c --- calculate apparent gaussian parameters for each replica

      axprm = b2*1.0d0/(4.0d0*u2xavg(k))
      ayprm = b2*1.0d0/(4.0d0*u2yavg(k))
      azprm = b2*1.0d0/(4.0d0*u2zavg(k))

      axsd = b2*sqrt((1.0d0/(16.0d0*u2xavg(k)**2))*u2xsd**2)
      aysd = b2*sqrt((1.0d0/(16.0d0*u2yavg(k)**2))*u2ysd**2)
      azsd = b2*sqrt((1.0d0/(16.0d0*u2zavg(k)**2))*u2zsd**2)

      call lrc(axprm, ayprm, azprm, rnnmin, lrctot)
      print *, lrctot

      write (6, 6610) k, 'V2AVG = ', vtavg(k)
6610  format ('replica ', i3, 1x, a9, f10.5, ' Kelvin')
6620  format ('replica ', i3, 1x, a9, 1pe13.6, ' Angstrom**2')
      write (6, 6610) k, 'V2 SD = ', vsd

      write (6, 6610) k, 'V3AVG = ', v3avg(k)

      write (6, 6610) k, 'V3 SD = ', v3sd

      write (6, 6610) k, 'E2AVG = ', etavg(k)

      write (6, 6610) k, 'E2 SD = ', esd

      write (6, 6620) k, 'u2x  =   ', u2xavg(k)

      write (6, 6620) k, 'u2x sd = ', u2xsd

      write (6, 6620) k, 'u2y  =   ', u2yavg(k)

      write (6, 6620) k, 'u2y sd = ', u2ysd

      write (6, 6620) k, 'u2z  =   ', u2zavg(k)

      write (6, 6620) k, 'u2z sd = ', u2zsd

      write (4, 6630) k, vtavg(k), vsd, v3avg(k), v3sd,
     +                etavg(k), esd, u2xavg(k), u2xsd,
     +                u2yavg(k), u2ysd, u2zavg(k), u2zsd,
     +                axprm, axsd, ayprm, aysd, azprm, azsd, lrctot,
     +                loop

6630  format(1x, i3, 6(1x, 1pe13.6), 6(1x, 1pe13.6), 6(1x, 1pe13.6)
     +       1x, 1pe13.6, 1x, i5)
      end do

      flush (4)
      stop
      end

c ----------------------------------------------------------------------

c     this subroutine computes the local energy and potential energy
c     of a configuration.

c ----------------------------------------------------------------------

      subroutine local(q, tloc, vloc, pot3b)
      implicit double precision (a-h, o-z)

      include 'sizes.h'

      include 'qsats.h'

      common /bincom/ bin, binvrs, r2min

c --- alpha is the exponential parameter in psi:

c     psi = N * exp(-alpha*(r-r0)**2) * Jastrow

c --- bb is the exponential parameter in Jastrow:

c     ln Jastrow(ij) = -0.5 * (bb/rij)**5

      dimension q(NATOM3), dlng(NATOM3), d2lng(NATOM3)

      do i=1, NATOM3
         dlng(i)=0.0d0
         d2lng(i)=0.0d0
      end do

      do i=1, NATOMS

         xx=q(3*i-2)
         yy=q(3*i-1)
         zz=q(3*i)

         dlng(3*i-2)=dlng(3*i-2)-2.0d0*aaxy*xx
         dlng(3*i-1)=dlng(3*i-1)-2.0d0*aaxy*yy
         dlng(3*i)  =dlng(3*i)  -2.0d0*aaz*zz

         d2lng(3*i-2)=d2lng(3*i-2)-2.0d0*aaxy
         d2lng(3*i-1)=d2lng(3*i-1)-2.0d0*aaxy
         d2lng(3*i)  =d2lng(3*i)  -2.0d0*aaz

      end do

c --- loop over all interacting pairs.

      vloc=0.0d0
      tloc=0.0d0

      do n=1, nvpair

         i=ivpair(1, n)
         j=ivpair(2, n)

         dx=-((q(3*j-2))+vpvec(1, n)+(-q(3*i-2)))
         dy=-((q(3*j-1))+vpvec(2, n)+(-q(3*i-1)))
         dz=-((q(3*j))  +vpvec(3, n)+(-q(3*i))  )

         r2=dx*dx+dy*dy+dz*dz

         ibin=int((r2-r2min)*binvrs)+1

         if (ibin.gt.0) then
            dr=(r2-r2min)-bin*dble(ibin-1)
            vloc=vloc+v(1, ibin)+v(2, ibin)*dr
         else
            vloc=vloc+v(1, 1)
         end if
```

```
            br2=bb*bb/r2

            br5=br2*br2*sqrt(br2)

            br52=br5/r2

            dlng(3*i-2)=dlng(3*i-2)+2.5d0*br52*dx
            dlng(3*i-1)=dlng(3*i-1)+2.5d0*br52*dy
            dlng(3*i)  =dlng(3*i)  +2.5d0*br52*dz

            d2lng(3*i-2)=d2lng(3*i-2)+2.5d0*br52*
     *                               (1.0d0-7.0d0*dx**2/r2)
            d2lng(3*i-1)=d2lng(3*i-1)+2.5d0*br52*
     *                               (1.0d0-7.0d0*dy**2/r2)
            d2lng(3*i)  =d2lng(3*i)  +2.5d0*br52*
     *                               (1.0d0-7.0d0*dz**2/r2)

         end do

c --- now sum up the kinetic energy components.

         do i=1, NATOM3
            tloc=tloc+d2lng(i)+dlng(i)**2
         end do

c --- account for mass factor and for double-counting of pairs.

         tloc=-0.5d0*tloc/amass
         vloc=0.5d0*vloc

c --- add in 3body energy

         pot3b = potl3b(q)

c         vloc=vloc+pot3b

         return
         end
c ====================================================================

c        This function calculates the three-body contribution to the
c        potential energy

c ====================================================================

         double precision function potl3b(q)

c --- evaluates the three-body potential energy of the system

         implicit real*8 (a-h, o-z)
         include 'sizes.h'
         include 'qsats.h'

         dimension q(NATOM3)

         potl3b = 0.0d0

c --- loop over all nearest neighbor trimers

         do n=1, nvtrim
            i= ivtrim(1, n)
            j= ivtrim(2, n)
            k= ivtrim(3, n)
            ndiv = ivtrim(4, n)
```

```
          dx1 = vpvectri(1, n)+q(3*j-2)-q(3*i-2)
          dy1 = vpvectri(2, n)+q(3*j-1)-q(3*i-1)
          dz1 = vpvectri(3, n)+q(3*j)-q(3*i)

          dx2 = vpvectri(4, n)+q(3*k-2)-q(3*i-2)
          dy2 = vpvectri(5, n)+q(3*k-1)-q(3*i-1)
          dz2 = vpvectri(6, n)+q(3*k)-q(3*i)

          dx12 = dx2-dx1
          dy12 = dy2-dy1
          dz12 = dz2-dz1

          r1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
          r2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
          r12 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)
c          print *, "side lengths", r1, r2, r12
c ------ get 3B energy and add to correct total
          call He3(r1, r2, r12, E3)
c ------ ndiv accounts for triple counting
          E3 = E3/dble(ndiv)
          potl3b = potl3b+E3
       end do
c9500  format(7(1x, 1pe13.6))
       return
       end

c ----------------------------------------------------------------------

c     msd is a subroutine that calculates the mean squared displacement
c     in all three directions from the snapshots.

c ----------------------------------------------------------------------

        subroutine msd(q, u2x, u2y, u2z, u4x, u4y, u4z)

        implicit real*8 (a-h, o-z)

        include 'sizes.h'

        include 'qsats.h'

        dimension q(NATOM3)
        parameter (bohr=0.529177249d0)

        u2x = 0.0d0
        u2y = 0.0d0
        u2z = 0.0d0

        u4x = 0.0d0
        u4y = 0.0d0
        u4z = 0.0d0

        do l = 1, NATOMS
          u2x = u2x + (q(3*l-2)**2)
          u4x = u4x + (q(3*l-2)**4)

          u2y = u2y + (q(3*l-1)**2)
          u4y = u4y + (q(3*l-1)**4)

          u2z = u2z + (q(3*l)**2)
          u4z = u4z + (q(3*l)**4)
       end do

c --- conversion factor from bohr**2 to angstrom**2
```

```
      b2=bohr*bohr
      u2x = u2x*b2/dble(NATOMS)
      u2y = u2y*b2/dble(NATOMS)
      u2z = u2z*b2/dble(NATOMS)

      u4x = u4x*b2*b2/dble(NATOMS)
      u4y = u4y*b2*b2/dble(NATOMS)
      u4z = u4z*b2*b2/dble(NATOMS)

      return
      end

c ----------------------------------------------------------------------

c     quit is a subroutine used to terminate execution if there is
c     an error.

c     it is needed here because the subroutine that reads the parameters
c     (subroutine input) may call it.

c ----------------------------------------------------------------------

      subroutine quit

      write (6, *) 'termination via subroutine quit'

      stop

      return
      end
```

B.3 VPI Fully-Incorporated 3-body Program (VPI+3B)

The VPI+3B program with fully-incorporated three-body interactions modifies the VPI-2B code to set up interacting trimer lists in the parent subroutine and use the trimers to calculate the change in three-body energy in the child subroutine. This program takes the same input parameters as VPI-2B, however this program is not set up to accept nonequilibrium values of η, γ, and ϵ. The files required to compile and run this program are listed below:

main.f	Main program that determines whether the processor is the parent or a child processor.
cmrg.f	See Sec. A.1.
rsetup-3b-full.f	Replaces rsetup.f in Sec. B.1. This version allows you to skip ahead in the random number generator to allow for multiple simultaneous simulation runs which sample different parts of the random number stream.

parent-3b-full.f	Replaces parent.f in Sec. B.1. This version sets up the interacting trimer list and does not allow for distorted lattices.
input-3b-full.f	Replaces input.f in Sec. B.1. This version does not read in deformation parameters.
vinit.f	See Sec. A.1.
child-3b-full.f	Replaces child.f in Sec. B.1. This version calculates the three-body energy from all nearest neighbor trimers for each accept/reject decision.
he3fci.f	See Sec. A.2.
even.f	See Sec. B.1.
odd.f	See Sec. B.1.
rpsend.f	See Sec. B.1.
tstamp.f	See Sec. A.1.
sizes.h	See Sec. B.2.
qsats.h	See Sec. B.2.

Program files that can be found in the QSATS code and are not reproduced here: **main.f**, **cmrg.f**, **vinit.f**, **even.f**, **odd.f**, **rpsend.f**, **tstamp.f**, **sizes.f**

The snapshots generated from the VPI+3B simulations are then used to calculate the average potential (2+3-body), kinetic, and total energies for each replica, along with the long-range correction to the two-body energy. This is accomplished using eloc-3b-full.f program which currently does not allow for distorted lattices. This program calls on many of the same subroutines as the VPI+3B main program and the VPI(3B) eloc-3b-distortion.f program. The files required to compile and run this program are listed below:

eloc-3b-full.f	Replaces eloc-distortion.f in Sec. B.1. This version adds the three-body energy to the total energy for each replica and does not allow for distorted lattices. It is currently configured to read through 10 sequentially numbered snapshot files of the form "svfile#" where "svfile" is read in the input subroutine.
input-3b-full.f	See **input-3b-full.f** above.
vinit.f	See Sec. A.1.
he3fci.f	See Sec. A.2.
lrc-3d-sub.f	See Sec. B.1.
c6-sub.f	See Sec. B.1.
tstamp.f	See Sec. A.1.
Gauss-Hermite.dat	See Sec. A.1.1.
sizes.h	See Sec. B.2.
qsats.h	Replaces qsats.h in Sec. B.2. This version does not include the deformation parameters and adds an additonal parameter "nskip".

rsetup-3b-full.f

```
c ----------------------------------------------------------------------
c     this subroutine initializes the pseudo random number generators
c     for the replicas.  it also initializes the value of the rscale
c     variable, which is needed to convert integer pseudo random
c     numbers, which are the raw output of the generators, to floating
c     point pseudo random numbers.
c ----------------------------------------------------------------------

      subroutine rsetup

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'

      dimension rseed(6)

      rscale=1.0d0/4294967088.0d0

      write (6, 6000)
6000  format ('INITIALIZING random number seeds'/)

      do i=1, 6
         rseed(i)=12345.0d0
      end do

c --- To impliment simulatneous runs, this is where you call rskip
c     Each replica just needs to have a different starting seed
c     than its counterpart in the other simulation. Skipping ahead by
c     10 rskips or so should be enough.

      do i = 1, nskip*10
          call rskip(rseed)
      end do

      do i=1, NREPS

         do j=1, 6
            rstatv(j, i)=rseed(j)
         end do

         rstatv(7, i)=-1.0d0
         rstatv(8, i)=0.0d0

         call rskip(rseed)

      end do

      if (idebug.ge.3) then

         write (6, 6001)
6001     format  ('rstatv(1) values:'/)

         do i=1, NREPS
            write (6, 6100) i, rstatv(1, i)
6100        format (i5, 1x, f20.1)
         end do

         write (6, *) ''
      end if

      return
      end
```

parent-3b-full.f

```
c ----------------------------------------------------------------------
c     this is the parent process that runs on node 0.

c     errchk is a subroutine called after every MPI subroutine that
c     checks the MPI error code and reports any errors.

c     This version of the program also sets up the interating trimer
c     list to account for three-body interactions.
c ----------------------------------------------------------------------

      subroutine parent(ierror)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'
      include 'mpif.h'

      dimension istat(MPI_STATUS_SIZE)
      dimension imsg(9), fmsg(7)
      dimension isent(NREPS), ikeep(NATOMS), replic(NATOM7)
      dimension rstate(8)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)

c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================

      ierror=0

c --- read input file.

      call input

      write (6, 6100) ltfile, spfile, svfile
6100  format ('lattice file name  = ', a16/,
     +        'snapshot file name = ', a16/,
     +        'save file name     = ', a16/)

      if (idebug.eq.0) write (6, 6110) idebug, 'NONE'
      if (idebug.eq.1) write (6, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (6, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (6, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (6, 6110) idebug, 'HIGH'

6110  format ('debug level = ', i1,' or ', a8/)

c --- read the potential energy curve.

      call vinit(r2min, bin)

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a16/)

      open (8, file=ltfile, status='old', err=901)

      read (8, *, err=902) nlpts
      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         call quit
      end if

c --- read the edge lengths of the supercell.

      read (8, *, err=903) xlen, ylen, zlen

      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- compute a distance scaling factor.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *, err=904) xtal(i, 1), xtal(i, 2), xtal(i, 3)

         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

c --- this helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
```

```
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)

         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r

      end do

      write (6, 6310) rnnmin
6310  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

      write (6, 6330) xlen, ylen, zlen
6330  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
         ntrim(i)=0
      end do

      nvpair=0
      nvpair1 = 0

      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
```

```
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.

            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

               vpvec(1, nvpair)=dx
               vpvec(2, nvpair)=dy
               vpvec(3, nvpair)=dz

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair

c ------------ keep track of first nearest neighbors to construct
c              interacting trimers.
               if (r/rnnmin.lt.1.05d0) then
                  nvpair1 = nvpair1+1

                  ivpair1(1, nvpair1) = i
                  ivpair1(2, nvpair1) = j

                  vpvec1(1, nvpair1) = dx
                  vpvec1(2, nvpair1) = dy
                  vpvec1(3, nvpair1) = dz

                  npair1(i) = npair1(i)+1

                  ipairs1(npair1(i), i)=nvpair1

               end if

            end if

         end if

      end do
      end do

c --- From the interacting pairs, calculate interacting trimers
      nvtrim=0
      do i = 1, NATOMS
        do j = 1, npair1(i)-1
        do k = j+1, npair1(i)
c--------- keep running count of all trimers
           nvtrim = nvtrim+1

           npairA = ipairs1(j, i) ! gives us our npair ref. number
           npairB = ipairs1(k, i) ! for other 2 atoms in the trimer
c--------- keep record of atoms in trimer (will also use this array to
c          store side how many times trimer is counted)
           ivtrim(1, nvtrim) = i
           ivtrim(2, nvtrim) = ivpair1(2, npairA)
           ivtrim(3, nvtrim) = ivpair1(2, npairB)
```

```
c -------- record vector from central atom to two neighbors

           vpvectri(1, nvtrim) = vpvec1(1, npairA)
           vpvectri(2, nvtrim) = vpvec1(2, npairA)
           vpvectri(3, nvtrim) = vpvec1(3, npairA)

           vpvectri(4, nvtrim) = vpvec1(1, npairB)
           vpvectri(5, nvtrim) = vpvec1(2, npairB)
           vpvectri(6, nvtrim) = vpvec1(3, npairB)
c -------- calculate and store side lenghts and central angle
           dx1 = vpvectri(1, nvtrim)
           dy1 = vpvectri(2, nvtrim)
           dz1 = vpvectri(3, nvtrim)

           dx2 = vpvectri(4, nvtrim)
           dy2 = vpvectri(5, nvtrim)
           dz2 = vpvectri(6, nvtrim)

           dx12 = dx2-dx1
           dy12 = dy2-dy1
           dz12 = dz2-dz1

           side1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
           side2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
           side3 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)


           vpvectri(7, nvtrim) = side1
           vpvectri(8, nvtrim) = side2
           vpvectri(9, nvtrim) = side3

c -------- Now evaluate to see if this trimer will be triple counted

           sidenn = 0
c -------- Keep track of sides greater than 1 NN distances
c          We know sides 1 and 2 will be <= 1NN, only have to
c          test side 3
           ivtrim(4, nvtrim) = 1 ! initially assume it is counted 1x
           if(side3/Rnnmin.gt.1.05) then
              ivtrim(4, nvtrim) = 3
           end if

c -------- Update number of trimers for given central atom
           ntrim(i) = ntrim(i)+1

           itrims(ntrim(i), i) = nvtrim

        end do
        end do
      end do

      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +        'total number of interacting pairs = ', i6)

      write (6, 6403) ntrim(1), nvtrim
6403  format ('atom 1 forms ', i5, ' trimers with interacting
     + neighbors'//, 'total number of interacting trimers = ', i9)
      if (idebug.ge.2) then

         write (6, 6401)
6401     format (/'interaction pair vectors for atom 1 ',
     +            '[NN distances]:'/)

         do i=1, npair(1)
```

```
            ip=ipairs(i, 1)
            d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +        rnnmin
            write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                      vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410        format ('vector # ', i3, ' to atom ', i4, ': ',
     +              3(1x, f9.5), ' length = ', f8.5)
         end do

         write (6, 6402)
6402     format (/'interaction trimer side lengths for atom 1 ',
     +           '[NN distances]:'/)
         do i=1, ntrim(1)
            itri = itrims(i, 1)
            write(6, 6411) itri, ivtrim(2, itri), ivtrim(3, itri),
     +                     vpvectri(7, itri)/rnnmin, vpvectri(8,
     +                     itri)/rnnmin, vpvectri(9, itri)/rnnmin,
     +                     ivtrim(4, itri)
6411        format('trimer # ', i5, ' incuding atoms ', i4, 1x, i4,
     +           ' : side lengths: ', 3(1x, f8.5), ' counted ', i1,
     +           ' times')
         end do
      end if

c --- set the displacement vectors for all replicas to zero.

      write (6, 6500)
6500  format (/'SETTING initial configuration to zero'/)

      do j=1, NREPS
      do i=1, NATOM3
         path(i, j)=0.0
      end do
      end do

c --- initialize random number generator.

      call rsetup

c --- now see if there is an old set of displacement vectors from a
c     previous run.  if not, jump head to line 200.

      open (8, file=svfile, form='unformatted', status='old', err=200)

      write (6, 6510) svfile
6510  format ('READING initial configuration from ', a16/)

      do j=1, NREPS

         read (8) (rstatv(i, j), i=1, 8)
         read (8) (path(i, j), i=1, NATOM3)

      end do

      close (8)

200   if (idebug.ge.3) then

         write (6, 6170)
6170     format ('x(1) and rstatv(1) values for each replica:'/)

         do j=1, NREPS

            write (6, 6180) j, path(1, j), rstatv(1, j)
6180        format (i5, 1x, f15.9, 1x, f20.1)
```

```
         end do

         write (6, *) ''

      end if

c --- this is the output file where snapshots of the replicas will be
c     stored for analysis by another program.

      open (10, file=spfile, form='unformatted')

c --- initialize MPI.

      MPI_R=MPI_DOUBLE_PRECISION

      call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)

      call errchk(0, ierr, 100000)

      write (6, 6600) ntasks-1
6600  format ('number of child processes = ', i3/)

      if (ntasks-1.gt.size(iwork)) then

         write (6, 6610)
6610     format ('too many child processes; expand the iwork array.'/
     +           'also note that write statements for HIGH '
     +           'debugging level may fail on some systems.')

         call quit

      end if

c --- this array just counts how evenly the workload was spread among
c     the child processes.

      do i=1, ntasks-1
         iwork(i)=0
      end do

c --- broadcast integer constants to all child processes.

      imsg(1)=NATOMS
      imsg(2)=NATOM3
      imsg(3)=NATOM6
      imsg(4)=NATOM7
      imsg(5)=NREPS
      imsg(6)=NIP
      imsg(7)=NPAIRS
      imsg(8)=NVBINS
      imsg(9)=idebug

      do itask=1, ntasks-1

         call MPI_SEND(imsg,
     +                 9,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0101,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100101)
      end do

      if (idebug.gt.0) open (9, file='debug.log')

      if (idebug.eq.1) write (9, 6110) idebug, 'MINIMAL'
      if (idebug.eq.2) write (9, 6110) idebug, 'LOW'
      if (idebug.eq.3) write (9, 6110) idebug, 'MEDIUM'
      if (idebug.eq.4) write (9, 6110) idebug, 'HIGH'

c --- broadcast floating-point constants to all child processes.

      fmsg(1)=tau
      fmsg(2)=bin
      fmsg(3)=r2min
      fmsg(4)=amass
      fmsg(5)=aaxy
      fmsg(6)=aaz
      fmsg(7)=bb

      do itask=1, ntasks-1

         call MPI_SEND(fmsg,
     +                 8,
     +                 MPI_R,
     +                 itask,
     +                 0102,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100102)

      end do

c --- broadcast the interacting-pair vectors to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(vpvec,
     +                 3*NPAIRS,
     +                 MPI_R,
     +                 itask,
     +                 0103,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100103)

      end do

c --- broadcast the list of atom id numbers for the interacting pairs
c     to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ivpair,
     +                 2*NPAIRS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0104,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100104)

      end do
```

```
c --- broadcast the size of each stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1

         call MPI_SEND(npair,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0105,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100105)

      end do

c --- broadcast the list of interacting pair id numbers that define the
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(ipairs,
     +                 NIP*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0106,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100106)

      end do

c --- broadcast the potential energy curve V(R) to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(v,
     +                 2*NVBINS,
     +                 MPI_R,
     +                 itask,
     +                 0107,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100107)

      end do

c --- Now begin broadcasting interacting trimer information

c --- broadcast the interacting trimer vectors (with side lengths and
c     angles) to all child processes

      do itask=1, ntasks-1

         call MPI_SEND(vpvectri,
     +                 9*NTRIMS,
     +                 MPI_R,
     +                 itask,
     +                 0108,
     +                 MPI_COMM_WORLD,
     +                 ierr)
```

```
        call errchk(0, ierr, 100108)

       end do
c --- broadcast the list of atom id numbers for all interacting trimers
c     to all child processes

      do itask=1, ntasks-1

         call MPI_SEND(ivtrim,
     +                 4*NTRIMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0109,
     +                 MPI_COMM_WORLD,
     +                 ierr)

        call errchk(0, ierr, 100109)

      end do
c --- broadcast the size of each trimer stencil to all child processes.  all
c     stencils should be the same size, but we treat this as a variable.

      do itask=1, ntasks-1

         call MPI_SEND(ntrim,
     +                 NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0110,
     +                 MPI_COMM_WORLD,
     +                 ierr)

        call errchk(0, ierr, 100110)

      end do
c --- broadcast the list of interacting trimer id numbers that define the
c     stencils to all child processes.

      do itask=1, ntasks-1

         call MPI_SEND(itrims,
     +                 NIT*NATOMS,
     +                 MPI_INTEGER,
     +                 itask,
     +                 0111,
     +                 MPI_COMM_WORLD,
     +                 ierr)

         call errchk(0, ierr, 100111)

      end do

      if (idebug.gt.0) write (9, *) 'end parent PART ONE'
      if (idebug.gt.0) write (9, *) ''
c =====================================================================
c     PART TWO: PERFORMING THE SIMULATION
c =====================================================================

c --- initialization of various progress counters.

c --- this is how many iterations we have done.

      loop=0
```

```
c --- these tell us about the acceptance ratio for the atom moves.

      ztacc=0.0d0
      ztrej=0.0d0

      ztacc0=0.0d0
      ztrej0=0.0d0

300   loop=loop+1
c --- these counters make sure that we don't lose a replica somewhere in
c     the ether. we use them to count how many replicas have been sent and
c     received.

      nsent=0
      nrcvd=0

c --- this is a list of flags that are zero for replicas that haven't yet
c     been sent to a child for processing, positive for replicas that have
c     been sent, and negative for replicas that have been processed and
c     returned to the parent.

c     isent(n) is set to the (positive) task id of the receiving child
c     process when a replica is sent.  this is basically leaving a trail
c     of crumbs so that we can track down the replicas and ask the children
c     to return them to us.

      do nrep=1, NREPS
         isent(nrep)=0
      end do

c --- first do all odd replicas.

      call oddrep(loop, nsent, nrcvd, MPI_R)

c --- then do all even replicas.

      call evnrep(loop, nsent, nrcvd, MPI_R)

c --- check for lost replicas.

      if (nsent.ne.NREPS.or.nrcvd.ne.NREPS) then
         write (6, *) 'replicas have been lost!'
         write (6, *) 'nsent = ', nsent
         write (6, *) 'nrcvd = ', nrcvd
         ierror=1
      end if

c --- take a snapshot every so often.

      if (mod(loop, nprint).eq.0) then

         zacc=ztacc-ztacc0
         zrej=ztrej-ztrej0

         ztacc0=ztacc
         ztrej0=ztrej

         if (idebug.gt.0) then
            write (9, 9400) zacc, zrej, 100.0d0*zacc/(zacc+zrej)
9400        format ('accepted = ', f11.0, 1x,
     +              'rejected = ', f11.0, 3x,
     +              '% accepted = ', f6.2)
            call flush(9)
         end if
```

```
c ------ we only actually take snapshots of every 11th replica.

         do k=1, NREPS, 11
            write (10) (path(i, k), i=1, NATOM3)
         end do

      end if

c --- do the next loop if needed.

      if (loop.lt.nloop) goto 300

      open (12, file = 'nacc-atoms.dat')
      do i = 1, NREPS
         do j = 1, NATOMS
            acct =dble(zaccv(i, j))
            rejt =dble(zrejv(i, j))
            write (12, *) i, j, 100.0d0*acct/(rejt+acct)
         end do
      end do
      close(12)

c --- otherwise save a checkpoint file.

      write (6, 6810) svfile
6810  format ('SAVING final configuration to ', a16/)

      open (8, file=svfile, form='unformatted')

      do k=1, NREPS
         write (8) (rstatv(i, k), i=1, 8)
         write (8) (path(i, k), i=1, NATOM3)
      end do

      if (idebug.ge.3) then

         write (6, 6170)

         do k=1, NREPS
            write (6, 6180) k, path(1, k), rstatv(1, k)
         end do

         write (6, *) ''

      end if

      close (8)

      close (10)

      if (idebug.gt.0) then
         write (9, *) ''
         write (9, *) 'QSATS is done!'
         write (9, *) ''
      end if

c --- show how much work every child did.

      if (idebug.gt.0) then
         do i=1, ntasks-1
            write (9, 9100) i, iwork(i)
9100        format ('task ', i3, ' received ', i9, ' replicas')
         end do
      end if
```

```
c --- tell the children we're all done.

      do itask=1, ntasks-1

         imsg(1)=0

         call MPI_SEND(imsg,
     +                1,
     +                MPI_INTEGER,
     +                itask,
     +                0204,
     +                MPI_COMM_WORLD,
     +                ierr)

         call errchk(0, ierr, 100204)

      end do

      write (6, 6900) ztacc
6900  format ('total number of accepted moves = ', f20.1)

      write (6, 6901) ztrej
6901  format ('total number of rejected moves = ', f20.1/)

      if (idebug.gt.0) write (9, *) ''
      if (idebug.gt.0) write (9, *) 'end parent PART TWO'

      return

901   write (6, *) 'error opening lattice file'
      goto 999
902   write (6, *) 'error reading number of atoms from lattice file'
      goto 999
903   write (6, *) 'error reading (unscaled) supercell edge lengths'
      goto 999
904   write (6, *) 'error reading atom number ', i
      goto 999

999   call quit

      return
      end
```

input-3b-full.f

```
c ----------------------------------------------------------------------
c     this inputs the names of various I/O files and also reads in the
c     parameters for the simulation

c     This version does not account for distorted lattices.
c ----------------------------------------------------------------------

      subroutine input

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'

      character*8 inword

c --- read in filenames.
c     spfile = snapshot file
c     svfile = checkpoint "save" file in the main VPI program
c            = tabulated output datafile in the eloc file
c     ltfile = lattice file

      read (5, 5000, err=922) spfile
5000  format (a30)
      read (5, 5000, err=923) svfile
      read (5, 5000, err=924) ltfile

c --- set debug level.

      read (5, 5001, err=931) inword
5001  format (a8)

      if (inword.eq.'NONE') then
         idebug=0
      else if (inword.eq.'MINIMAL') then
         idebug=1
      else if (inword.eq.'LOW') then
         idebug=2
      else if (inword.eq.'MEDIUM') then
         idebug=3
      else if (inword.eq.'HIGH') then
         idebug=4
      else
         write (6, *) 'invalid debug level'
      end if

c --- read in the simulation parameters.
c     tau = imaginary time step in a.u.
c     den = number density in atoms per cubic bohr
c     amass = atomic mass of the He-4 atoms
c     aaxy = trial wavefunction a_xy parameter
c     aaz = trial wavefunction a_z parameter
c     bb = trial wavefuncton b parameter
c     nloop = total MCCs
c     nprint = snapshot interval
c     nskip = number of calls to rskip in rsetup.f
c           this allows for sequential runs

      read (5, *, err=901) tau
      read (5, *, err=902) den
      read (5, *, err=903) amass
      read (5, *, err=904) aaxy
      read (5, *, err=904) aaz
      read (5, *, err=905) bb
      read (5, *, err=906) nloop
      read (5, *, err=907) nprint
      read (5, *, err=908) nskip

      write (6, 6000) NATOMS, NREPS
6000  format ('REPEATING input parameters'//,
     +        'atom count    = ', i6/,
     +        'replica count = ', i6/)

      write (6, 6001) tau, den, amass, aaxy, aaz, bb, dzscale
6001  format ('tau                = ', f14.7, ' au time'/,
     +        'density            = ', f14.7, ' atoms per cubic bohr'/,
     +        'atomic mass        = ', f14.7, ' electron masses'/,
     +        'alpha-xy parameter = ', f14.7, ' bohr**(-2)'/,
     +        'alpha-z parameter  = ', f14.7, ' bohr**(-2)'/,
     +        'B parameter        = ', f14.7, ' bohr'/)

      write (6, 6002) nloop, nprint
6002  format ('number of simulation steps = ', i8/,
     +        'snapshot interval          = ', i8/)

      return

901   write (6, *) 'error reading time step value'
      goto 999
902   write (6, *) 'error reading density value'
      goto 999
903   write (6, *) 'error reading atomic mass value'
      goto 999
904   write (6, *) 'error reading aa value'
      goto 999
905   write (6, *) 'error reading bb value'
      goto 999
906   write (6, *) 'error reading nloop value'
      goto 999
907   write (6, *) 'error reading nprint value'
      goto 999
908   write (6, *) 'error reading nskip value'
      goto 999
909   write (6, *) 'error reading nskip value'
      goto 999
921   write (6, *) 'error reading RNG file name'
      goto 999
922   write (6, *) 'error reading snapshot file name'
      goto 999
923   write (6, *) 'error reading save file name'
      goto 999
924   write (6, *) 'error reading lattice file name'
      goto 999
931   write (6, *) 'error reading debug level'
      goto 999
932   write (6, *) 'error reading RNG initialization mode'
      goto 999
999   call quit

      return
      end
```

child-3b-full.f

```
c ----------------------------------------------------------------------
c     this is the child process that runs on all nodes except node 0
c     (which is running the parent process).

c     This version of code accounts for three-body interactions in the
c     accept/reject decision
c ----------------------------------------------------------------------

      subroutine child(MPI_R)

      implicit double precision (a-h, o-z)

      include 'mpif.h'

      include 'sizes.h'

      common /rancm1/ rscale

      dimension replic(NATOM6), npair(NATOMS), rv(NATOM3)
      dimension ntrim(NATOMS)

      dimension istat(MPI_STATUS_SIZE)

      dimension vpvec(3, NPAIRS)
      dimension ivpair(2, NPAIRS)
      dimension ipairs(NIP, NATOMS)
      dimension vpvectri(9, NTRIMS)
      dimension ivtrim(4, NTRIMS)
      dimension itrims(NIT, NATOMS)

      dimension xx(NATOMS), yy(NATOMS), zz(NATOMS)
      dimension xtri(NIT, 2), ytri(NIT,2), ztri(NIT, 2)

      dimension r2old(NATOMS), r2new(NATOMS), v1(NATOMS), v2(NATOMS)

      dimension v(2, NVBINS)

      dimension imsg(9), fmsg(7)
      dimension naccv(NREPS, NATOMS), nrejv(NREPS, NATOMS)
      dimension rstate(8)

      parameter (half=0.5d0)
      parameter (two=2.0d0)
      parameter (one=1.0d0)

c ======================================================================
c     PART ONE: INITIALIZATION
c ======================================================================

c --- numerical factor for random number generator.

      rscale=1.0d0/4294967088.0d0

c --- determine which process this is and store it in myid.

      call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)

c --- receive all of the information that is broadcast by the parent
c     process.

c --- first receive some integer constants.  these are primarily used to
c     check that the arrays are properly dimensioned.
```

```
      call MPI_RECV(imsg,
     +              9,
     +              MPI_INTEGER,
     +              0,
     +              0101,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200101)

      istop=0

      if (imsg(1).ne.NATOMS) then
         write (6, *) 'size mismatch 1: ', imsg(1)
         istop=1
      end if

      if (imsg(2).ne.NATOM3) then
         write (6, *) 'size mismatch 2: ', imsg(2)
         istop=1
      end if

      if (imsg(3).ne.NATOM6) then
         write (6, *) 'size mismatch 3: ', imsg(3)
         istop=1
      end if

      if (imsg(4).ne.NATOM7) then
         write (6, *) 'size mismatch 4: ', imsg(4)
         istop=1
      end if

      if (imsg(5).ne.NREPS) then
         write (6, *) 'size mismatch 5: ', imsg(5)
         istop=1
      end if

      if (imsg(6).ne.NIP) then
         write (6, *) 'size mismatch 6: ', imsg(6)
         istop=1
      end if

      if (imsg(7).ne.NPAIRS) then
         write (6, *) 'size mismatch 7: ', imsg(7)
         istop=1
      end if

      if (imsg(8).ne.NVBINS) then
         write (6, *) 'size mismatch 8: ', imsg(8)
         istop=1
      end if

      if (istop.eq.1) call quit

      idebug=imsg(9)

c --- debugging output.

      if (idebug.eq.4) write (30+myid, *) 'idebug = ', idebug

c --- next receive some floating-point constants.

      call MPI_RECV(fmsg,
     +              7,
     +              MPI_R,
     +              0,
     +              0102,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200102)

      tau=fmsg(1)
      bin=fmsg(2)
      r2min=fmsg(3)
      amass=fmsg(4)
      aaxy=fmsg(5)
      aaz=fmsg(6)
      bb=fmsg(7)

      if (idebug.eq.4) then
         write (30+myid, *) 'tau = ', tau
         write (30+myid, *) 'bin = ', bin
         write (30+myid, *) 'r2min = ', r2min
         write (30+myid, *) 'amass = ', amass
         write (30+myid, *) 'aaxy = ', aaxy
         write (30+myid, *) 'aaz = ', aaz
         write (30+myid, *) 'bb = ', bb
      end if

c --- compute the inverse of the potential energy V(R) bin width, to
c     avoid unnecessary divisions.

      binvrs=one/bin

c --- compute gaussian scaling parameters.

      gscale=sqrt(half*tau/amass)
      gscal2=sqrt(tau/amass)

c --- next receive the vectors that connect pairs of atoms in a stencil.

      call MPI_RECV(vpvec,
     +              3*NPAIRS,
     +              MPI_R,
     +              0,
     +              0103,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200103)

c --- next receive the list of pairs of atoms.

      call MPI_RECV(ivpair,
     +              2*NPAIRS,
     +              MPI_INTEGER,
     +              0,
     +              0104,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200104)

c --- next receive the number of atoms that belong to each atom's stencil.
c     this should really be the same for every atom for a regular crystal
```

```
c     lattice, but we treat it as a variable.

      call MPI_RECV(npair,
     +              NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0105,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200105)

c --- next receive the pairs that constitute each atom's stencil.

      call MPI_RECV(ipairs,
     +              NIP*NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0106,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200106)

c --- next receive the potential energy curve V(R) for interpolation.

      call MPI_RECV(v,
     +              2*NVBINS,
     +              MPI_R,
     +              0,
     +              0107,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200107)

c --- next receive the vectors of atoms in each trimer, along with
c     side lengths and central angles

      call MPI_RECV(vpvectri,
     +              9*NTRIMS,
     +              MPI_R,
     +              0,
     +              0108,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200108)
c --- next receive the list of atom id numbers for all interacting
c     trimers

      call MPI_RECV(ivtrim,
     +              4*NTRIMS,
     +              MPI_INTEGER,
     +              0,
     +              0109,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200109)
```

```
c --- next receive the size of each trimer stencil

      call MPI_RECV(ntrim,
     +              NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0110,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200110)
c --- next receive the list of interacting trimer id numbers that define
c     the stencils

      call MPI_RECV(itrims,
     +              NIT*NATOMS,
     +              MPI_INTEGER,
     +              0,
     +              0111,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200111)

      if (idebug.eq.4) then
         write (30+myid, *) 'child moving to PART TWO'
         call flush(30+myid)
      end if

c =====================================================================
c     PART TWO: PERFORMING THE SIMULATION
c =====================================================================

100   idrep=0

      nacc=0
      nrej=0

c --- send request for data (message type 1201) to parent.  the first
c     time through, or if we are waiting for all children to sync up,
c     there are no results to send back to the parent, so we indicate
c     this by setting idrep=0 just above, and then sending this to
c     the parent in imsg(1).

200   imsg(1)=idrep

      imsg(2)=nacc
      imsg(3)=nrej

      call MPI_SEND(imsg,
     +              3,
     +              MPI_INTEGER,
     +              0,
     +              1201,
     +              MPI_COMM_WORLD,
     +              ierr)

      call errchk(myid, ierr, 201201)

c --- on the other hand, if there are results to send back, then we
c     do so here.

      if (idrep.gt.0) then
```

```
c ------ first we send a message of type 1202 that contains the atoms'
c        new positions.

            call MPI_SEND(replic,
     +                    NATOM3,
     +                    MPI_R,
     +                    0,
     +                    1202,
     +                    MPI_COMM_WORLD,
     +                    ierr)

            call errchk(myid, ierr, 201202)

c ------ then we send a message of type 1203 that contains the updated
c        random number generator state vector.

            call MPI_SEND(rstate,
     +                    8,
     +                    MPI_DOUBLE_PRECISION,
     +                    0,
     +                    1203,
     +                    MPI_COMM_WORLD,
     +                    ierr)

            call errchk(myid, ierr, 201203)

         end if

c --- wait for acknowledgement (message type 0204) from parent.  the
c     parent also uses this to signal the child that more input will
c     be sent.

c     if imsg(1) is positive, it is a replica number that represents the
c     next replica that this child should process.

c     if imsg(1) is negative, then this child needs to wait for the
c     other children to sync up, and so the child goes back to the top
c     of PART TWO.

c     if imsg(1) is zero, there is no more work to be done.

         call MPI_RECV(imsg,
     +                 1,
     +                 MPI_INTEGER,
     +                 0,
     +                 0204,
     +                 MPI_COMM_WORLD,
     +                 istat,
     +                 ierr)

         call errchk(myid, ierr, 200204)

c --- loop back and wait for more input if instructed by parent.

         if (imsg(1).lt.0) goto 100

c --- terminate if the simulation is complete.

         if (imsg(1).eq.0) then
            if (idebug.eq.4) write (30+myid, *) 'child is done!'
            return
         end if

c --- if there is a new replica to process, then receive data from
```

```
c     the parent.

c --- we need to save the replica number that we are about to work on.

      idrep=imsg(1)

c --- first receive the loop number, in a message of type 0207.

      call MPI_RECV(loop,
     +              1,
     +              MPI_INTEGER,
     +              0,
     +              0207,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200207)

c --- next receive the old atomic coordinates and the means of the
c     neighboring replicas' coordinates, in a message of type 0205.

      call MPI_RECV(replic,
     +              NATOM6,
     +              MPI_R,
     +              0,
     +              0205,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200205)

c --- next receive the random number generator state vector, in
c     a message of type 0206.

      call MPI_RECV(rstate,
     +              8,
     +              MPI_DOUBLE_PRECISION,
     +              0,
     +              0206,
     +              MPI_COMM_WORLD,
     +              istat,
     +              ierr)

      call errchk(myid, ierr, 200206)

c --- generate provisional new atomic positions by adding gaussian
c     displacements.

c --- first choose the appropriate gaussian scaling factor.

      if (idrep.eq.1.or.idrep.eq.NREPS) then
         gsc=gscal2
      else
         gsc=gscale
      end if

c --- then add the gaussian displacements.

      do nn=1, NATOM3
         call gstep(rstate, gg, rscale)
         replic(NATOM3+nn)=replic(NATOM3+nn)+gg*gsc
      end do
```

```
c --- attempt to move each atom in turn.

      nacc=0
      nrej=0

      do nn=1, NATOMS

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'moving atom 1'
               call flush(30+myid)
            end if
         end if

c ------ set up the coordinates of the atoms that are in this atom's
c        stencil.

         do i=1, npair(nn)

            ip=ipairs(i, nn)
            j=ivpair(2, ip)

            xx(i)=replic(3*j-2)+vpvec(1, ip)
            yy(i)=replic(3*j-1)+vpvec(2, ip)
            zz(i)=replic(3*j-0)+vpvec(3, ip)

         end do

         do i = 1, ntrim(nn)

            itri = itrims(i, nn)
            j = ivtrim(2, itri)
            k = ivtrim(3, itri)

            xtri(i,1) = replic(3*j-2)+vpvectri(1, itri)
            ytri(i,1) = replic(3*j-1)+vpvectri(2, itri)
            ztri(i,1) = replic(3*j)+vpvectri(3, itri)

            xtri(i, 2) = replic(3*k-2)+vpvectri(4, itri)
            ytri(i, 2) = replic(3*k-1)+vpvectri(5, itri)
            ztri(i, 2) = replic(3*k)+vpvectri(6, itri)

         end do

c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'after do loop, xx(1) = ', xx(1)
               call flush(30+myid)
            end if
         end if

c ------ get the old and new coordinates of the atom that we're about
c        to try to move.

         xold=replic(3*nn-2)
         yold=replic(3*nn-1)
         zold=replic(3*nn-0)

         xnew=replic(3*nn-2+NATOM3)
         ynew=replic(3*nn-1+NATOM3)
         znew=replic(3*nn-0+NATOM3)
```

```
c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'xold, xnew = ', xold, xnew
               call flush(30+myid)
            end if
         end if

c ------ compute the old and new distances between this atom and all
c        of the atoms in the stencil.

c ------ the do loops are split up to promote vectorization, although
c        i'm not sure this is necessary.

         do i=1, npair(nn)
            r2old(i)=(xx(i)-xold)**2
         end do

         do i=1, npair(nn)
            r2old(i)=r2old(i)+(yy(i)-yold)**2
         end do

         do i=1, npair(nn)
            r2old(i)=r2old(i)+(zz(i)-zold)**2
         end do

         do i=1, npair(nn)
            r2new(i)=(xx(i)-xnew)**2
         end do

         do i=1, npair(nn)
            r2new(i)=r2new(i)+(yy(i)-ynew)**2
         end do

         do i=1, npair(nn)
            r2new(i)=r2new(i)+(zz(i)-znew)**2
         end do

c ------ compute the change in potential energy.

         do i=1, npair(nn)

c --------- use linear interpolation.

            ibin1=int((r2old(i)-r2min)*binvrs)+1
            ibin2=int((r2new(i)-r2min)*binvrs)+1

            if (ibin1.gt.0) then
               dr1=(r2old(i)-r2min)-bin*dble(ibin1-1)
               v1(i)=v(1, ibin1)+v(2, ibin1)*dr1
            else
               v1(i)=v(1, 1)
            end if

            if (ibin2.gt.0) then
               dr2=(r2new(i)-r2min)-bin*dble(ibin2-1)
               v2(i)=v(1, ibin2)+v(2, ibin2)*dr2
            else
               v2(i)=v(1, 1)
            end if

         end do
         dv=0.0

         do i=1, npair(nn)
            dv=dv+v1(i)-v2(i)
         end do

c ------ Calculate the three body energy. Looping through all
c        trimers formed by atom of interest and two of its 56 neighbors.

         v3old = 0.0d0
         v3new = 0.0d0

         do i= 1, ntrim(nn)
            itri = itrims(i, nn)
            x01old = xtri(i, 1)-xold
            y01old = ytri(i, 1)-yold
            z01old = ztri(i, 1)-zold

            side1old = sqrt(x01old**2+y01old**2+z01old**2)

            x02old = xtri(i, 2)-xold
            y02old = ytri(i, 2)-yold
            z02old = ztri(i, 2)-zold

            side2old = sqrt(x02old**2+y02old**2+z02old**2)

            x12old = x02old-x01old
            y12old = y02old-y01old
            z12old = z02old-z01old

            side3old = sqrt(x12old**2+y12old**2+z12old**2)

            x01new = xtri(i, 1)-xnew
            y01new = ytri(i, 1)-ynew
            z01new = ztri(i, 1)-znew

            side1new = sqrt(x01new**2+y01new**2+z01new**2)

            x02new = xtri(i, 2)-xnew
            y02new = ytri(i, 2)-ynew
            z02new = ztri(i, 2)-znew

            side2new = sqrt(x02new**2+y02new**2+z02new**2)

            x12new = x02new-x01new
            y12new = y02new-y01new
            z12new = z02new-z01new

            side3new = sqrt(x12new**2+y12new**2+z12new**2)

            call He3(side1old, side2old, side3old, E3old)
            call He3(side1new, side2new, side3new, E3new)

c --------- Account for all cases which result in triple counting
c           All situations depends on average positions, not
c           instantaneous snapshots

             ndiv = ivtrim(4, itri)

             E3old = E3old/dble(ndiv)
             E3new = E3new/dble(ndiv)

             v3old = v3old+E3old
             v3new = v3new+E3new
           end do
           dv3 = v3old-v3new
           dv = dv+dv3
c ------ debugging output.

         if (nn.eq.1) then
            if (idebug.eq.4) then
               write (30+myid, *) 'dv = ', dv
               call flush(30+myid)
            end if
         end if

         dv=dv*tau

c ------ deal with trial function for first and last replicas.

         if (idrep.eq.1.or.idrep.eq.NREPS) then

            dpsi=0.0

            do i=1, npair(nn)
               dpsi=dpsi+
     +              (1.0d0/sqrt(r2old(i)))**5-
     -              (1.0d0/sqrt(r2new(i)))**5
            end do

            soldxy=xold**2+yold**2
            snewxy=xnew**2+ynew**2

            dpsi=0.5d0*bb**5*dpsi+aaxy*(soldxy-snewxy)+
     +           aaz*(zold**2-znew**2)

c --------- debugging output.

            if (nn.eq.1) then
               if (idebug.eq.4) then
                  write (30+myid, *) 'evaluating trial function'
                  write (30+myid, *) 'dpsi = ', dpsi
                  call flush(30+myid)
               end if
            end if

c --------- also remember to scale the change in potential energy by
c           one-half for the end replicas.

            dv=half*dv+dpsi

         end if

c ------ choose whether to accept the new position.

         call rstep(rstate, zran, rscale)

         if (dv.ge.0.0) then

c --------- accept this move.

            replic(3*nn-2)=xnew
            replic(3*nn-1)=ynew
            replic(3*nn-0)=znew

            nacc=nacc+1

         else if (zran.lt.exp(dv)) then

c --------- accept this move.
```

```
            replic(3*nn-2)=xnew
            replic(3*nn-1)=ynew
            replic(3*nn-0)=znew

            nacc=nacc+1

         else

c --------- reject this move.

            nrej=nrej+1

         end if

c --- end of loop over atoms.

      end do

c --- go back to send these results back to the parent.

      goto 200

      end
```

qsats.h

```
c --- parameters and counters for the VPI simulation.

      common /monte/  zaccv(NREPS, NATOMS), zrejv(NREPS, NATOMS),
     +                naccv(NATOMS), nrejv(NATOMS), zm,
     +                tau, den, scale, amass, ztacc, ztrej,
     +                nloop, nprint, nacc, nrej, irrst, idebug,
     +                nskip

c --- trial wave function parameters.

      common /psitri/ aaxy, aaz, dzscale, bb

c --- random number generator variables.

      double precision zm1, zm2, rm1, rm2, rscale, rstatv

      common /moduli/ zm1, zm2, rm1, rm2

      common /rancm1/ rscale

      common /rancm2/ rstatv(8, NREPS)

c --- potential energy lookup table.

      common /potcom/ v(2, NVBINS)

c --- VPI replicas and atomic masses.

      common /vpi/    path(NATOM3, NREPS),
     +                pathnu(NATOM3, NREPS),
     +                zmass(NATOM3)

c --- filenames.

      character*30 spfile, svfile, ltfile
      common /files/  spfile, svfile, ltfile

c --- description of the crystal lattice.

      common /crystl/ xtal(NATOMS, 3)

      common /box/    xlen, ylen, zlen, dxmax, dymax, dzmax

c --- arrays dealing with interacting pairs and trimers.

      common /vpairs/ vpvec(3, NPAIRS), vpvec1(3, NPAIRS),
     +                vpvectri(9, NTRIMS),
     +                ivpair(2, NPAIRS), ivpair1(2, NPAIRS),
     +                ivtrim(4, NTRIMS),
     +                ipairs(NIP, NATOMS), ipairs1(NIP, NATOMS),
     +                itrims(NIT, NATOMS),
     +                npair(NATOMS), npair1(NATOMS),
     +                ntrim(NATOMS),
     +                nvpair, nvpair1, nvtrim

c --- counters to monitor load balancing.

      common /parcom/ iwork(255)
```

eloc-3b-full.f

```
c ----------------------------------------------------------------------

c     this computes the total energy and the expectation value of the
c     potential energy from the snapshots recorded by QSATS.
c     The three-body energy is calculated here considering on those
c     trimers formed by first nearest neighbors.
c     The total energy is
c     reported including three body interactions which have been fully
c     incorporated into the wavefunction optimization.
c     This program is set up to read snapshot files from 10 subsequent
c     VPI jobs with 100 snapshots each.
c ----------------------------------------------------------------------

      program eloc3b10files

      implicit double precision (a-h, o-z)
      real*8 lrctot
      include 'sizes.h'
      include 'qsats.h'

      parameter (bohr=0.529177249d0)

c --- this common block is used to enable interpolation in the potential
c     energy lookup table in the subroutine local below.

      common /bincom/ bin, binvrs, r2min

c --- set up arrays needed for average energies and mean
c     squared displacement calculations

      dimension q(NATOM3), vtavg(NREPS), vtavg2(NREPS),
     +          etavg(NREPS), etavg2(NREPS), v3avg(NREPS),
     +          v3avg2(NREPS), u2xavg(NREPS),
     +          u2yavg(NREPS), u2zavg(NREPS), u4xavg(NREPS),
     +          u4yavg(NREPS), u4zavg(NREPS)

c --- This is necessary to alternate between files
```

```
c     Assumes sequential numbering of files
      character(len=2) :: nfile_string
      parameter (half=0.5d0)
      parameter (one=1.0d0)

c --- initialization.

      call tstamp

      write (6, 6001) NREPS, NATOMS, NATOM3, NATOM6, NATOM7,
     +                NVBINS, RATIO, NIP, NPAIRS
6001  format ('compile-time parameters:'//,
     +        'NREPS  = ', i6/,
     +        'NATOMS = ', i6/,
     +        'NATOM3 = ', i6/,
     +        'NATOM6 = ', i6/,
     +        'NATOM7 = ', i6/,
     +        'NVBINS = ', i6/,
     +        'RATIO  = ', f6.4/,
     +        'NIP    = ', i6/,
     +        'NPAIRS = ', i6/)

      call input
      call vinit(r2min, bin)
      binvrs=one/bin

c --- read crystal lattice points.

      write (6, 6200) ltfile
6200  format ('READING crystal lattice from ', a16/)

      open (8, file=ltfile, status='old')

      read (8, *) nlpts
      if (nlpts.ne.NATOMS) then
         write (6, *) 'ERROR: number of atoms in lattice file = ', nlpts
         write (6, *) 'number of atoms in source code = ', NATOMS
         stop
      end if

c --- read the edge lengths of the supercell.

      read (8, *) xlen, ylen, zlen

c --- compute a distance scaling factor.
      den0=dble(NATOMS)/(xlen*ylen*zlen)

c --- scale is a distance scaling factor, computed from the atomic
c     number density specified by the user.

      scale=exp(dlog(den/den0)/3.0d0)

      write (6, 6300) scale
6300  format ('supercell scaling factor computed from density = ',
     +        f12.8/)

      xlen=xlen/scale
      ylen=ylen/scale
      zlen=zlen/scale

      write (6, 6310) xlen, ylen, zlen
6310  format ('supercell edge lengths [bohr]         = ', 3f10.5/)

      dxmax=half*xlen
      dymax=half*ylen
      dzmax=half*zlen

      do i=1, NATOMS

         read (8, *) xtal(i, 1), xtal(i, 2), xtal(i, 3)
         xtal(i, 1)=xtal(i, 1)/scale
         xtal(i, 2)=xtal(i, 2)/scale
         xtal(i, 3)=xtal(i, 3)/scale

      end do

      close (8)

      write (6, 6320) xtal(NATOMS, 1), xtal(NATOMS, 2),
     +                xtal(NATOMS, 3)
6320  format ('final lattice point [bohr]            = ', 3f10.5/)

c --- this variable helps us remember the nearest-neighbor distance.

      rnnmin=-1.0d0

      do j=2, NATOMS

         dx=xtal(j, 1)-xtal(1, 1)
         dy=xtal(j, 2)-xtal(1, 2)
         dz=xtal(j, 3)-xtal(1, 3)

c ------ this sequence of if-then-else statements enforces the
c        minimum image convention.

         if (dx.gt.dxmax) then
            dx=dx-xlen
         else if (dx.lt.-dxmax) then
            dx=dx+xlen
         end if

         if (dy.gt.dymax) then
            dy=dy-ylen
         else if (dy.lt.-dymax) then
            dy=dy+ylen
         end if

         if (dz.gt.dzmax) then
            dz=dz-zlen
         else if (dz.lt.-dzmax) then
            dz=dz+zlen
         end if

         r=sqrt(dx*dx+dy*dy+dz*dz)
         if (r.lt.rnnmin.or.rnnmin.le.0.0d0) rnnmin=r
      end do

      write (6, 6330) rnnmin
6330  format ('nearest neighbor (NN) distance [bohr] = ', f10.5/)

      write (6, 6340) xlen/rnnmin, ylen/rnnmin, zlen/rnnmin
6340  format ('supercell edge lengths [NN distances] = ', 3f10.5/)

c --- compute interacting pairs.

      do i=1, NATOMS
         npair(i)=0
         npair1(i)=0
      end do
      nvpair=0
      nvpair1=0
      do i=1, NATOMS
      do j=1, NATOMS

         if (j.ne.i) then

            dx=xtal(j, 1)-xtal(i, 1)
            dy=xtal(j, 2)-xtal(i, 2)
            dz=xtal(j, 3)-xtal(i, 3)

c --------- this sequence of if-then-else statements enforces the
c           minimum image convention.

            if (dx.gt.dxmax) then
               dx=dx-xlen
            else if (dx.lt.-dxmax) then
               dx=dx+xlen
            end if

            if (dy.gt.dymax) then
               dy=dy-ylen
            else if (dy.lt.-dymax) then
               dy=dy+ylen
            end if

            if (dz.gt.dzmax) then
               dz=dz-zlen
            else if (dz.lt.-dzmax) then
               dz=dz+zlen
            end if

            r2=dx*dx+dy*dy+dz*dz

            r=sqrt(r2)

c --------- interacting pairs are those for which r is less than a
c           certain cutoff amount.

            if (r/rnnmin.lt.RATIO) then

               nvpair=nvpair+1

               ivpair(1, nvpair)=i
               ivpair(2, nvpair)=j

               vpvec(1, nvpair)=dx
               vpvec(2, nvpair)=dy
               vpvec(3, nvpair)=dz

               npair(i)=npair(i)+1

               ipairs(npair(i), i)=nvpair
c ------------ for three-body calculations, keep track of only first
c              nearest neighbors.

               if (r/rnnmin.lt.1.05) then

                 nvpair1=nvpair1+1

c -------------- store information about this pair (i->j) in arrays.

                 ivpair1(1, nvpair1)=i
                 ivpair1(2, nvpair1)=j
```

```
                vpvec1(1, nvpair1)=dx
                vpvec1(2, nvpair1)=dy
                vpvec1(3, nvpair1)=dz

                npair1(i)=npair1(i)+1
                ipairs1(npair1(i), i)=nvpair1
             end if
          end if
        end if
      end do
      end do

      write (6, 6400) npair(1), nvpair
6400  format ('atom 1 interacts with ', i3, ' other atoms'//,
     +        'total number of interacting pairs = ', i6/)


c --- To save time later, we are going to calculate all of the first
c     nearest neighbor trimers now, along with angles and vectors
      nvtrim=0

      do i = 1, NATOMS
        do j = 1, npair1(i)-1
        do k = j+1, npair1(i)
c--------- keep running count of all trimers
           nvtrim = nvtrim+1
c -------- record vector from central atom to two neighbors

           npairA = ipairs1(j, i) !tells us the nvpair reference number
           npairB = ipairs1(k, i) ! for each of the atoms in the trimer

c--------- keep record of atoms in trimer
           ivtrim(1, nvtrim) = i !central atom
           ivtrim(2, nvtrim) = ivpair1(2, npairA)
           ivtrim(3, nvtrim) = ivpair1(2, npairB)

           vpvectri(1, nvtrim) = vpvec1(1, npairA)
           vpvectri(2, nvtrim) = vpvec1(2, npairA)
           vpvectri(3, nvtrim) = vpvec1(3, npairA)

           vpvectri(4, nvtrim) = vpvec1(1, npairB)
           vpvectri(5, nvtrim) = vpvec1(2, npairB)
           vpvectri(6, nvtrim) = vpvec1(3, npairB)
c -------- calculate and store side lenghts and central angle
           dx1 = vpvectri(1, nvtrim)
           dy1 = vpvectri(2, nvtrim)
           dz1 = vpvectri(3, nvtrim)

           dx2 = vpvectri(4, nvtrim)
           dy2 = vpvectri(5, nvtrim)
           dz2 = vpvectri(6, nvtrim)

           dx12 = dx2-dx1
           dy12 = dy2-dy1
           dz12 = dz2-dz1

           side1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
           side2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
           side3 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)

           vpvectri(7, nvtrim) = side1
           vpvectri(8, nvtrim) = side2
           vpvectri(9, nvtrim) = side3
c -------- We know sides 1 and 2 = Rnn. If side 3 is lt or
c          equal to Rnn, trimer will be triple counted
```

```
           ivtrim(4, nvtrim) = 1

           s3test = side3/rnnmin

           if (s3test.lt.1.05) then !if side is within 1.05Rnn, trimer
                                    ! will be triple counted
              ivtrim(4, nvtrim) = 3
           end if

c -------- Update number of trimers for given central atom
           ntrim(i) = ntrim(i)+1

           itrims(ntrim(i), i) = nvtrim
        end do
        end do
      end do

      write (6, 6403) ntrim(1), nvtrim
6403  format ('atom 1 forms ', i3, 'trimers with interacting
     + neighbors'//, 'total number of interacting trimers = ', i9)

      do i=1, npair(1)
         ip=ipairs(i, 1)
         d=sqrt(vpvec(1, ip)**2+vpvec(2, ip)**2+vpvec(3, ip)**2)/
     +     rnnmin
         write (6, 6410) ip, ivpair(2, ip), vpvec(1, ip)/rnnmin,
     +                   vpvec(2, ip)/rnnmin, vpvec(3, ip)/rnnmin, d
6410     format ('vector # ', i3, ' to atom ', i4, ': ',
     +           3(1x, f9.5), ' length = ', f8.5)
      end do

         write (6, 6402)
6402     format (/'interaction trimer side lengths for atom 1 ',
     +           '[NN distances]:'/)
         do i=1, ntrim(1)
            itri = itrims(i, 1)
            write(6, 6411) itri, ivtrim(2, itri), ivtrim(3, itri),
     +                     vpvectri(7, itri)/rnnmin, vpvectri(8,
     +                     itri)/rnnmin, vpvectri(9, itri)/rnnmin,
     +                     ivtrim(4, itri)
6411        format('trimer # ', i3, 'incuding atoms ', i4, 1x, i4, ':',
     +           'side lengths: ', 3(1x, f8.5), ' counted: ',
     +            i1, 1x, 'times')
         end do

c --- initialization.

      loop=0
      nfile = 1
      do k=1, NREPS
         vtavg(k)=0.0d0
         etavg(k)=0.0d0
         vtavg2(k)=0.0d0
         etavg2(k)=0.0d0
         u2xavg(k)=0.0d0
         u2yavg(k)=0.0d0
         u2zavg(k)=0.0d0
         u4xavg(k)=0.0d0
         u4yavg(k)=0.0d0
         u4zavg(k)=0.0d0
      end do
       nl = LENGTH(spfile)

c ---- the snapshot files are assumed to follow the format
c      'spfile##' where spfile is read in input.f
```

```
299    write(nfile_string, '(I2)') nfile
       if(nfile.lt.10) nfile_string = nfile_string(2:2)
       open (10, file=spfile(1:nl)//nfile_string, form='unformatted')
       print *, spfile(1:nl)//nfile_string
c --- this loops reads the snapshots saved by QSATS.

300   loop=loop+1

      do k=1, NREPS, 11

         read (10, end=600) (path(i, k), i=1, NATOM3)

c ------ compute the local energy and the potential energy.

         do i=1, NATOM3
            q(i)=path(i, k)
         end do
         call local(q, tloc, vloc, pot3b)
c ------ convert to kelvin per atom.

         tloc=tloc/(3.1668513d-6*dble(NATOMS))
         vloc=vloc/(3.1668513d-6*dble(NATOMS))
         pot3b=pot3b/(3.1668513d-6*dble(NATOMS))

c ------ accumulate the results.
c ------ note, vloc does not include pot3b
         v3avg(k)=v3avg(k)+pot3b
         v3avg2(k)=v3avg2(k)+(pot3b)**2

         vtavg(k)=vtavg(k)+vloc
         vtavg2(k)=vtavg2(k)+(vloc)**2
         etavg(k)=etavg(k)+tloc+vloc+pot3b
         etavg2(k)=etavg2(k)+(tloc+vloc+pot3b)**2

c ------ compute <u^2> in all three directions
         call msd(q, u2x, u2y, u2z, u4x, u4y, u4z)

         u2xavg(k) = u2xavg(k)+u2x
         u2yavg(k) = u2yavg(k)+u2y
         u2zavg(k) = u2zavg(k)+u2z

         u4xavg(k) = u4xavg(k)+u4x
         u4yavg(k) = u4yavg(k)+u4y
         u4zavg(k) = u4zavg(k)+u4z

350      continue

      end do

      goto 300

c --- account for overshooting.

600   loop = loop-1
      nfile = nfile+1
      close(10)
      print *, "file closed"
      if(nfile.le.10) goto 299

      write (6, 6600) loop
6600  format ('number of snapshots = ', i6/)

c --- compute the averages and standard deviations.
      b2 = bohr*bohr
      open (4, file='eloc-3b.dat')
      do k=1, NREPS, 11

      v3avg(k)=v3avg(k)/dble(loop)
      v3avg2(k)=v3avg2(k)/dble(loop)
      vtavg(k)=vtavg(k)/dble(loop)
      vtavg2(k)=vtavg2(k)/dble(loop)
      etavg(k)=etavg(k)/dble(loop)
      etavg2(k)=etavg2(k)/dble(loop)

      v3sd=sqrt(v3avg2(k)-v3avg(k)**2)
      vsd=sqrt(vtavg2(k)-vtavg(k)**2)
      esd=sqrt(etavg2(k)-etavg(k)**2)

      u2xavg(k)=u2xavg(k)/dble(loop)
      u2yavg(k)=u2yavg(k)/dble(loop)
      u2zavg(k)=u2zavg(k)/dble(loop)

      u4xavg(k)=u4xavg(k)/dble(loop)
      u4yavg(k)=u4yavg(k)/dble(loop)
      u4zavg(k)=u4zavg(k)/dble(loop)

      u2xsd=sqrt(u4xavg(k)-u2xavg(k)**2)
      u2ysd=sqrt(u4yavg(k)-u2yavg(k)**2)
      u2zsd=sqrt(u4zavg(k)-u2zavg(k)**2)

c --- calculate apparent gaussian parameters for each replica

      axprm = b2*1.0d0/(4.0d0*u2xavg(k))
      ayprm = b2*1.0d0/(4.0d0*u2yavg(k))
      azprm = b2*1.0d0/(4.0d0*u2zavg(k))

      axsd = b2*sqrt((1.0d0/(16.0d0*u2xavg(k)**2))*u2xsd**2)
      aysd = b2*sqrt((1.0d0/(16.0d0*u2yavg(k)**2))*u2ysd**2)
      azsd = b2*sqrt((1.0d0/(16.0d0*u2zavg(k)**2))*u2zsd**2)

      call lrc(axprm, ayprm, azprm, rnnmin, lrctot)

      write (6, 6610) k, 'V2AVG = ', vtavg(k)
6610  format ('replica ', i3, 1x, a9, f10.5, ' Kelvin')
6620  format ('replica ', i3, 1x, a9, 1pe13.6, ' Angstrom**2')

      write (6, 6610) k, 'V2 SD = ', vsd
      write (6, 6610) k, 'V3AVG = ', v3avg(k)
      write (6, 6610) k, 'V3 SD = ', v3sd
      write (6, 6610) k, 'E2AVG = ', etavg(k)
      write (6, 6610) k, 'E2 SD = ', esd
      write (6, 6620) k, 'u2x  =   ', u2xavg(k)
      write (6, 6620) k, 'u2x sd = ', u2xsd
      write (6, 6620) k, 'u2y  =   ', u2yavg(k)
      write (6, 6620) k, 'u2y sd = ', u2ysd
      write (6, 6620) k, 'u2z  =   ', u2zavg(k)
      write (6, 6620) k, 'u2z sd = ', u2zsd

      write (4, 6630) k, vtavg(k), vsd, v3avg(k), v3sd,
     +                etavg(k), esd, u2xavg(k), u2xsd,
     +                u2yavg(k), u2ysd, u2zavg(k), u2zsd,
     +                axprm, axsd, ayprm, aysd, azprm, azsd, lrctot
6630  format(1x, i3, 6(1x, 1pe13.6), 6(1x, 1pe13.6), 6(1x, 1pe13.6)
     +       1x, 1pe13.6)
      end do

      flush (4)
      stop
      end

c ----------------------------------------------------------------------

c     this subroutine computes the local energy and potential energy
c     of a configuration.

c ----------------------------------------------------------------------

      subroutine local(q, tloc, vloc, pot3b)

      implicit double precision (a-h, o-z)

      include 'sizes.h'
      include 'qsats.h'

      common /bincom/ bin, binvrs, r2min

c --- alpha is the exponential parameter in psi:

c     psi = N * exp(-alpha*(r-r0)**2) * Jastrow

c --- bb is the exponential parameter in Jastrow:

c     ln Jastrow(ij) = -0.5 * (bb/rij)**5

      dimension q(NATOM3), dlng(NATOM3), d2lng(NATOM3)

      do i=1, NATOM3
         dlng(i)=0.0d0
         d2lng(i)=0.0d0
      end do

      do i=1, NATOMS

         xx=q(3*i-2)
         yy=q(3*i-1)
         zz=q(3*i)


         dlng(3*i-2)=dlng(3*i-2)-2.0d0*aaxy*xx
         dlng(3*i-1)=dlng(3*i-1)-2.0d0*aaxy*yy
         dlng(3*i)  =dlng(3*i)  -2.0d0*aaz*zz

         d2lng(3*i-2)=d2lng(3*i-2)-2.0d0*aaxy
         d2lng(3*i-1)=d2lng(3*i-1)-2.0d0*aaxy
         d2lng(3*i)  =d2lng(3*i)  -2.0d0*aaz

      end do

c --- loop over all interacting pairs.

      vloc=0.0d0
      tloc=0.0d0

      do n=1, nvpair

         i=ivpair(1, n)
         j=ivpair(2, n)

         dx=-((q(3*j-2))+vpvec(1, n)+(-q(3*i-2)))
         dy=-((q(3*j-1))+vpvec(2, n)+(-q(3*i-1)))
         dz=-((q(3*j))  +vpvec(3, n)+(-q(3*i))  )

         r2=dx*dx+dy*dy+dz*dz
         ibin=int((r2-r2min)*binvrs)+1

         if (ibin.gt.0) then
            dr=(r2-r2min)-bin*dble(ibin-1)
            vloc=vloc+v(1, ibin)+v(2, ibin)*dr
         else
            vloc=vloc+v(1, 1)
         end if

         br2=bb*bb/r2

         br5=br2*br2*sqrt(br2)

         br52=br5/r2

         dlng(3*i-2)=dlng(3*i-2)+2.5d0*br52*dx
         dlng(3*i-1)=dlng(3*i-1)+2.5d0*br52*dy
         dlng(3*i)  =dlng(3*i)  +2.5d0*br52*dz

         d2lng(3*i-2)=d2lng(3*i-2)+2.5d0*br52*
     *                            (1.0d0-7.0d0*dx**2/r2)
         d2lng(3*i-1)=d2lng(3*i-1)+2.5d0*br52*
     *                            (1.0d0-7.0d0*dy**2/r2)
         d2lng(3*i)  =d2lng(3*i)  +2.5d0*br52*
     *                            (1.0d0-7.0d0*dz**2/r2)

      end do

c --- now sum up the kinetic energy components.

      do i=1, NATOM3
         tloc=tloc+d2lng(i)+dlng(i)**2
      end do

c --- account for mass factor and for double-counting of pairs.

      tloc=-0.5d0*tloc/amass
      vloc=0.5d0*vloc

c --- add in 3body energy

      pot3b = potl3b(q)

      return
      end
c ====================================================================

c     This function calculates the three-body contribution to the
c     potential energy

c ====================================================================

      double precision function potl3b(q)

c --- evaluates the three-body potential energy of the system

      implicit real*8 (a-h, o-z)
      include 'sizes.h'
      include 'qsats.h'

      dimension q(NATOM3)

      potl3b = 0.0d0
```

```
c --- loop over all nearest neighbor trimers

      do n=1, nvtrim
         i= ivtrim(1, n)
         j= ivtrim(2, n)
         k= ivtrim(3, n)
         ndiv = ivtrim(4, n)

         dx1 = vpvectri(1, n)+q(3*j-2)-q(3*i-2)
         dy1 = vpvectri(2, n)+q(3*j-1)-q(3*i-1)
         dz1 = vpvectri(3, n)+q(3*j)-q(3*i)

         dx2 = vpvectri(4, n)+q(3*k-2)-q(3*i-2)
         dy2 = vpvectri(5, n)+q(3*k-1)-q(3*i-1)
         dz2 = vpvectri(6, n)+q(3*k)-q(3*i)

         dx12 = dx2-dx1
         dy12 = dy2-dy1
         dz12 = dz2-dz1

         r1 = sqrt(dx1*dx1+dy1*dy1+dz1*dz1)
         r2 = sqrt(dx2*dx2+dy2*dy2+dz2*dz2)
         r12 = sqrt(dx12*dx12+dy12*dy12+dz12*dz12)

c ------ get 3B energy and add to total
c ------ ndiv accounts for triple counting
         call He3(r1, r2, r12, E3)
         E3 = E3/dble(ndiv)
         potl3b = potl3b+E3
      end do
      return
      end
c ----------------------------------------------------------------------

c     msd is a subroutine that calculates the mean squared displacement
c     in all three directions from the snapshots.

c ----------------------------------------------------------------------

       subroutine msd(q, u2x, u2y, u2z, u4x, u4y, u4z)

       implicit real*8 (a-h, o-z)

       include 'sizes.h'
       include 'qsats.h'

       dimension q(NATOM3)
       parameter (bohr=0.529177249d0)

       u2x = 0.0d0
       u2y = 0.0d0
       u2z = 0.0d0

       u4x = 0.0d0
       u4y = 0.0d0
       u4z = 0.0d0

       do l = 1, NATOMS
          u2x = u2x + (q(3*l-2)**2)
          u4x = u4x + (q(3*l-2)**4)

          u2y = u2y + (q(3*l-1)**2)
          u4y = u4y + (q(3*l-1)**4)

          u2z = u2z + (q(3*l)**2)
          u4z = u4z + (q(3*l)**4)
      end do

c --- conversion factor from bohr**2 to angstrom**2

      b2=bohr*bohr
      u2x = u2x*b2/dble(NATOMS)
      u2y = u2y*b2/dble(NATOMS)
      u2z = u2z*b2/dble(NATOMS)

      u4x = u4x*b2*b2/dble(NATOMS)
      u4y = u4y*b2*b2/dble(NATOMS)
      u4z = u4z*b2*b2/dble(NATOMS)

      return
      end


c ----------------------------------------------------------------------
c     This function calculates the length of a string without trailing
c     blanks. This is necessary to use variable input snapshot files.
c ----------------------------------------------------------------------

      INTEGER FUNCTION LENGTH(string)
      CHARACTER*(*) STRING
      DO 15, I = LEN(STRING), 1, -1
        IF(STRING(I:I) .NE. ' ') GO TO 20
15    CONTINUE
20    LENGTH = I
      END
c ----------------------------------------------------------------------

c     quit is a subroutine used to terminate execution if there is
c     an error.

c     it is needed here because the subroutine that reads the parameters
c     (subroutine input) may call it.

c ----------------------------------------------------------------------

      subroutine quit

      write (6, *) 'termination via subroutine quit'

      stop

      return
      end
```

Vita

Ashleigh Locke Barnes (born Ashleigh Rebecca Locke) was born to her loving parents Lisa and James in 1990. She attended Flower Mound High School in Flower Mound, Texas, and graduated as Salutatorian in 2008. After high school, she studied at Baylor University in Waco, Texas, and performed research in medicinal organic chemistry under Dr. Kevin Pinney. In 2012, she graduated Summa Cum Laude from Baylor with her B.S. in Chemistry. She married Austin Robert Barnes in the summer of 2012 before starting the Ph.D. program in physical chemistry at the University of Tennessee, with a concentration in theoretical chemistry. Ashleigh plans to continue her education through postdoctoral research, with the ultimate goal of teaching chemistry at the college level where she hopes to give undergraduate students early exposure to the theoretical sciences.