EXCHANGE INTERACTIONS IN SOLID ^{3}HE ON A PARALLEL COMPUTER

Thesis by

Sean Macauley Callahan

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena,California

1989

(Submitted August 1st, 1988)

ACKNOWLEDGEMENTS

I would like to express deep gratitude to my advisor, Michael Cross, for the generous support he has given me throughout this work. I am thankful for having been introduced to the fascinating field of condensed matter physics.

I would like to thank Geoffrey Fox, who has acted as my advisor in the field of parallel computing. I have greatly appreciated his advice and his support, which have allowed me to work in the field of parallel computing, and to use the NCUBE parallel computer to provide the computing resources for this work.

Although we have never met personally, I would like to express my great appreciation to David Ceperley, who generously shared a program to calculate the ^{3}He density matrix and provided many useful discussions over the phone.

To my friends and colleagues who have listened to me and who have been listened to by me, and to my relatives who have shown interest and support throughout my studies, I am very grateful.

I would like to thank my parents for helping to instill in me a curiosity about the world and an appreciation of the importance of education, both of which led me to study physics. Without their love I would not be where I am today.

Finally, I would like to thank my wife Teddi, whose advice, love and support have contributed greatly to the success of this work. Sharing my life with her and our three beautiful children, Benjamin, Robert and Jacqueline, is my greatest joy.

ABSTRACT

We have developed a method of calculating the Gruneisen parameter, $\gamma \equiv \partial \ln J / \partial \ln v$ (J - exchange splitting; v - molar volume), for energy splittings that are due to atoms exchanging lattice sites in fermionic systems for which the exchange splittings are small compared with the lattice kinetic and potential energies. The method involves making two separate path integral Monte Carlo calculations, one for a path (sequence of particle positions in time) that doesn't involve atoms exchanging lattice sites, and one for a path in which the atoms do exchange lattice sites. The difference between a quantity calculated for the two systems is related to the Gruneisen parameter.

The central results of this thesis are firstly, that there is no significant variation of the Gruneisen parameters for two, three and four particle exchange (γ_2, γ_3 and γ_4, respectively) for solid ^{3}He, in going from 24 cm^3/mole, which is near melting, to 22 cm^3/mole. This result was extended to 20 cm^3/mole for γ_2. Since no significant variation was observed, the values calculated for different molar volumes were combined to give best estimates for the Gruneisen constants, $\gamma_2 = 15.9 \pm .8$, $\gamma_3 = 16.4 \pm 1.4$ and $\gamma_4 = 13.8 \pm 1.5$. The second result is that the magnitudes of the Gruneisen parameters are very similar for the three types of exchange despite their significantly different geometry. Some very limited runs were done for a system of hard spheres. It is interesting that for the hard sphere diameter chosen, the Gruneisen parameters are of the same order as for solid ^{3}He, $\gamma_2 = 21.7 \pm .8$, $\gamma_3 = 16.8 \pm 2.1$ and $\gamma_4 = 19.3 \pm 1.9$ for the small test systems checked. The final result relates to ^{3}He deposited on a graphite surface. The Gruneisen parameter for three particle exchange for a two-dimensional system with a triangular mesh

is found to be consistent with $\gamma_3 = 50 \pm 5$ for a range of nearest-neighbor, lattice spacings from 3.2 Å to 3.6Å. This is inconsistent with experiment, indicating that three particle exchange confined to two dimensions does not account for the ferromagnetic properties of this system, as was claimed previously.

TABLE OF CONTENTS

CHAPTER 1

Introductory Remarks

Most physical phenomena we observe result from the collective behavior of immense numbers of particles. Often different interactions of roughly equivalent energy scale compete to produce the observed effects. This mixing of different sorts of interactions greatly increases the difficulty encountered in writing down and solving the Hamiltonian. Theoretical descriptions of such systems are correspondingly difficult, and little progress has been made in quantitatively calculating collective phenomena from a microscopic point of view.

Solid ^{3}He provides an extremely rich magnetic system, which exhibits complex collective behavior, but which also has a large separation of energy scales, which allows for a theoretical exploration of some of its properties via quantum Monte Carlo computational techniques. This system attractive not only from a theoretical point of view but also experimentally, since the ordering temperature of 1mK is accessible using nuclear demagnetization cryostats. It is expected that in the next few years significant experimental work will be done in this area.

In the low-density, low-temperature solid there are two quite different magnetic phases in zero magnetic field. Consider the phase diagram, Fig 1, near the solid-liquid phase transition, which for low temperatures is at approximately 30 atmo-

spheres pressure. Below 1mK the solid has a noncubic antiferromagnetic structure and at higher temperatures it is disordered paramagnetic. As the field is increased in the antiferromagnetic region, there is a first-order transition to a high field phase, which may be ferromagnetic (Fig. 2).

Near the 1mK ordering temperature and just above melting pressure, lattice zero-point energies, exchange energies and nuclear, dipole-dipole interaction energies are each separated from the other by several orders of magnitude. The characteristic binding potential of the lattice is of the same order as the zero-point kinetic energy, an indication of the quantum nature of the solid. Order-of-magnitude estimates of these energies are given in Table 1.

Energy Type	Estimate
Lattice Potential	Depth of Helium Potential $\simeq$ 10 K
Zero-Point Kinetic	$\frac{p^2}{2m} \simeq \frac{\hbar^2}{2m\Delta x^2} \simeq$ 10 K $\Delta x \simeq$ Latt. Spacing - Hardcore Diam. $\simeq 2$ Å
Exchange	Hard Core Repulsion? $\rightarrow 10^{-3}$ K
Dipolar Spin-Spin	$\frac{\vec{\mu}\cdot\vec{\mu}}{r^3} \simeq (\frac{e\hbar}{2mc})^2 \frac{1}{L^3} \simeq 10^{-7}$ K L $\simeq$ Lattice Spacing $\simeq$ 4Å

Table 1

Order-of-Magnitude Estimates of Interaction Energies

The excitation of the lattice degrees of freedom is negligible at 1 mK, and the magnetic dipole-dipole interaction energies are small enough to be insignificant with respect to the magnetic ordering properties. Thus, it is the energy splitting of the otherwise degenerate spin levels, resulting from atoms exchanging lattice sites, which is the dominant interaction with respect to magnetic ordering. The order of magnitude of the exchange energies is not estimated in any simple way, a fact that adds to the interest of this system. In fact, it is only by way of experimental observation, such as the 1mK antiferromagnetic ordering temperature, that the

approximate magnitude of the exchange interactions is known.

Since exchanges are very rare on the time scale of a lattice vibration, the original and the exchanged lattice configuration can, to a very good approximation, be treated as a two-state system.[1] The fermionic nature of the atoms, each having spin 1/2, then requires an antisymmetric wavefunction and leads to nuclear spin ordering.

Nearest neighbor exchange in the ground state of the low-density, BCC lattice provides a good example of how exchange interactions can lead to nuclear, magnetic spin ordering. Consider, first, two nearest-neighbor atoms exchanging lattice sites. Since direct magnetic dipole-dipole interaction of the two nuclear spins is not significant, the value of the spin quantum numbers merely act as labels. Their effect is nevertheless important because when considering exchange, the Pauli exclusion principle must be satisfied and the total wave function must be antisymmetric. The lowest-energy state for the system corresponds to a symmetric, nodeless (lowest kinetic energy) spatial wavefunction. The spin wavefunction must therefore be antisymmetric, and hence two-particle exchange is said to favor antiferromagnetism. In the antisymmetric state, the spins of the two exchanging atoms are opposite. Enforcing this condition for all nearest-neighbor pairs in the lattice produces a spin configuration with all atoms in one simple, cubic sub-lattice having spin up and all those in the other having spin down. This is referred to as the normal antiferromagnetic (or NAF) phase. This configuration has the property that all nearest-neighbor pairs in the lattice have opposite spins. This is, of course, a semiclassical description, and quantum spin fluctuations will inevitably complicate the picture, but it is still a reasonable model with which to gain some intuition about the magnetic tendencies of the spin system.

Magnetic ordering in the low-temperature solid is not, in fact, cubic as it would be if only two-particle exchange were significant. The antiferromagnetic phase is thought to consist of planes of up and down spins that are parallel to one face of the cubic lattice. Such a structure is referred to as the uNdN phase,[2] where u and the d refer to the alternate planes of up and down spins, respectively, and N refers to the

number of planes of each type that occur in a repeating pattern. The NAF phase described above has this structure with N=1, but its cubic structure has been ruled out by the observation of a tetragonal symmetry axis, corresponding to a broken symmetry, in nuclear magnetic resonance (NMR) experiments. The u2d2 phase appears to be the best candidate to describe the properties observed so far. This phase clearly requires competition between ferromagnetic and antiferromagnetic interactions. The most likely origin of this is the competition between exchanges of even numbers of particles (which give antiferromagnetic interactions) and exchanges of odd numbers of particles (which give ferromagnetic interactions).[3] The magnetic tendencies of an exchange can be deduced from the fact that n particles exchanging is equivalent to n-1 two-particle exchanges. Hence, exchanges with odd numbers of particles lead to a ground state with a symmetric (ferromagnetic) spin wavefunction. As in the two-particle case exchanges with even numbers of particles favor antiferromagnetism.

This work was motivated by experimental evidence that quantities such as the first two terms in the high-temperature expansion of the specific heat, which depend upon several types of exchange (2-, 3-, 4-particle), all scale with molar volume, v, according to one parameter. Experimentally, quantities with the dimensions of energy vary as v^{γ}, with $\gamma = 18 \pm 2$, from near melting at 24 cm^3/mole to 22 cm^3/mole. The different geometry of two, three and four particle exchange (see Fig 3) would seem to imply that their volume dependence should be very different and hence, that quantities depending on all of them should have a more complicated volume dependence than is experimentally observed.

The primary purpose of this work is to measure the Gruneisen parameters

$$\gamma = \frac{\partial \ln J_P}{\partial \ln v} \tag{1.1}$$

for two, three and four particle exchange, where J_P is the exchange energy for exchange type P using quantum path integral Monte Carlo computational techniques.[4] These types of exchange are considered because, at least qualitatively, they seem

to account for many of the behaviors of this system (see Section 2.3). The simplification provided by the large separation of energy scales in solid ^{3}He allows for the calculation of γ for each type of exchange separately. A semiclassical (WKB) calculation is not applicable because, as was already mentioned, the kinetic and potential energies are of comparable magnitude. The potential is quite broad and actually has a slight maximum at the lattice sites, and so a full solution of the quantum problem is necessary. The particles may, however, be treated as distinguishable, since only a single exchange is involved in each calculation, so that the very hard problems involved in fermionic Monte Carlos are not encountered.

Since this work was started, Ceperley and Jacucci [5] have developed a method of calculating the exchange energies directly by making Monte Carlo moves, with very low acceptance, between a nonexchanging configuration and an exchanging one. This produces values of the exchange energies with errors at the 10% level. They calculate the Gruneisen parameter using the values for the exchange energies at only two points, 20.07 cm^3/mole and 24.12 cm^3/mole, from which they get $\gamma \sim 18$. The method described here is rather different and does not require the direct sampling of the tiny overlap. It therefore provides an independent calculation. In addition, we will be able to calculate γ at specific points between 22 cm^3/mole and 24 cm^3/mole and hence to gain a more detailed picture of how changes in volume affect the different types of exchange. Thus, a direct independent calculation of the Gruneisen constant is a useful complement to the work of Ceperley and Jacucci.

The material covered in this work is essentially as follows. Chapter 2 is a general discussion of the solid helium system, to provide background about some relevant experimental and theoretical issues. In Chapter 3, the theoretical basis for the computational work is laid out, including a path integral formulation of the density matrix, an observable derived from the path integral and related to the Gruneisen parameters for exchange, and a description of the Monte Carlo techniques used. Chapter 4 is a detailed discussion of how the formalism developed in Chapter 3 has been implemented, including a description of some techniques that have been used to reduce statistical errors and to gain an insight into the number of particles that

participate in each type of exchange. This material is, for the most part, common to all of the calculations. Chapter 5 is a description of a calculation of the Gruneisen parameters for exchange in a system of hard spheres. We studied this system originally because it has the simplifying feature that an analytic representation of the two-particle, density matrix may be calculated in a short time approximation, and the hardcore part of the potential for solid ^{3}He is believed to be very important in the suppression of the exchange energies. Chapter 6 covers the three-dimensional computer simulation for solid ^{3}He. The Gruneisen parameter for two, three and four particle exchange will be measured and compared with experiment. A study will also be made of the number of particles that participate in each type of exchange, to address the question of why the Gruneisen parameters for different types of exchange seem to be of similar magnitude. In Chapter 7 a brief look is taken at exchange in ^{3}He deposited as a surface layer on grafoil. It has been proposed that the properties of this system are well accounted for by three-particle exchange in one surface layer.[6] The validity of this assertion is tested by calculating the Gruneisen parameter for three-particle exchange and comparing it with experiment and the theoretical calculation based on the above-mentioned theory. Finally, Chapter 8 contains comments, conclusions and speculations.

References

1. Richard P. Feynman, Robert B. Leighton, and Matthew Sands, *The Feynman Lectures on Physics: Volume III*, Addison-Wesley, Reading, Mass., 1965.
2. D. D. Osheroff, M. C. Cross, and D. S. Fisher, *Phys. Rev. Lett.*, **44**, 792, 1980.
3. D. J. Thouless, *Proc. Phys. Soc. London*, **86**, 893, 1965.
4. M. C. Cross, *Phys. Rev. A (Rapid Communications)*, **34**, 3531, Oct. 1986.
5. D. M. Ceperley and G. Jacucci, *Phys. Rev. Lett.*, **58**, 1648, Apr. 1987.
6. M. Roger and J. M. Delrieu, *Japanese Jour. of Appl. Phys.*, **26**, 267, 1987.

CHAPTER 2

Solid ^{3}He and Exchange

This work concentrates on an antiferromagnetic phase of ^{3}He. To understand how it relates to the other phases, it is useful to look at the P-T phase diagram (Fig. 1). Below about 10K along the melting curve, the liquid borders on a BCC phase of the solid. At approximately 1.0mK (on the melting curve), the solid makes a first-order transition from a disordered paramagnetic phase to an antiferromagnetic phase. As the pressure is raised, from the melting pressure, the transition temperature goes rapidly to zero. Fig. 2 shows an H-T phase diagram, where the T axis lies on the melting curve. As the magnetic field is raised there is a first-order transition from the antiferromagnetic phase to a high-field phase in which the magnetization jumps from a small value to roughly half its saturation value. Above that transition field of about 4.4 kG, the magnetization increases slowly to a projected saturation field at about 180kG. This high-field, pseudoferromagnetic phase is separated from the disordered paramagnetic phase in one of three possible schemes,[1] all of which are consistent with experiment. As shown in Fig. 4b,c,d they are respectively: a first-order transition line turning to a second-order line ending at an upper critical field; a first-order line ending at an upper critical field; or a first-order transition to a critical point. In the final case, there is no clear distinction between the two

phases.

2.1 Experimental Information

Before going into the details of exchange models, it is interesting to outline the model-independent information that has been learned from experiment. There are significant constraints on the nature of the magnetic ordering in the antiferromagnetic phase, which come from NMR experiments, and there are also some very limited neutron-scattering data, which seem to support the conclusions drawn from the analysis of the NMR data. A high-temperature expansion of the free energy yields series for quantities such as the magnetic susceptibility, the specific heat and the pressure. Fitting these experimentally measured quantities allows for the calculation of some of the leading coefficients in the series. As mentioned in the introduction, the scaling behavior of some of these terms with specific volume provides the motivation for this work. The nature of this evidence will be described toward the end of this section. These experimentally measured coefficients can then be related to the parameters of a given exchange Hamiltonian. Finally, some information about spin-wave velocities can be extracted from low-temperature, specific heat measurements.

2.1a NMR

Some of the most elegant work done on solid ^{3}He is the NMR experiment and its analysis.[2,1,3]

Single crystals of the solid were grown directly in the antiferromagnetic state. For many different crystals, NMR signals that were assigned to three differently oriented domains of the ordered phase were observed. The existence of exactly three domains strongly suggests a symmetry axis that must align along one of the three principal axes of the cubic lattice. For a given domain, this axis will be referred to as $\hat{l}$. In Fig. 5, each domain corresponds to a pair of resonance lines, one line above the dashed (Larmor Frequency) line and one below it.

Generally, each domain is expected to have three resonances, corresponding to the three independent directions about which the spins may be rotated. The fact that only two modes are observed over a wide range of magnetic fields and for many different crystal orientations implies that there is a spin-symmetry axis, $\hat{d}$, such that a rotation of all of the spins about it causes no shift in the dipolar interaction energies. The simplest arrangement that satisfies this criterion is for the spins to be either parallel or antiparallel to $\hat{d}$. More complicated structures that satisfy this requirement but that are eliminated by other considerations will not be discussed here. The NMR spectra can be understood in terms of the two anisotropy vectors $\hat{d}$ and $\hat{l}$.[1]

Further information about the relative orientation of $\hat{l}$ and $\hat{d}$ is given by the fact that one of the modes for each domain goes to zero at zero field. For a general spin structure there is a spin realignment for some field strength (γH $\simeq$ .7 in Fig. 6) corresponding to the system switching from primarily minimizing the anisotropic part of the dipole energy,

$$E_D \propto \sum_{i,j} \frac{(\vec{S}_i \cdot \hat{r})(\vec{S}_j \cdot \hat{r})}{r^3}, \tag{2.1}$$

to primarily minimizing the magnetic energy. The absence of this realignment implies that $\hat{d}$ must be perpendicular to both $\hat{l}$ and the magnetic field, since that has the effect of minimizing both energies simultaneously. Viewing this in another way, there is some direction (other than the trivial direction $\hat{d}$) about which $\hat{d}$ can be rotated infinitesimally, such that there is no restoring force from dipole-dipole interactions. This is true only if $\hat{d}$ is perpendicular to $\hat{l}$, in which case there is no restoring force for a rotation of $\hat{d}$ in the plane perpendicular to $\hat{l}$.

One further constraint on the nature of the ordered phase is that since E_D is not zero, the spin structure cannot be cubic. For a cubically symmetric spin structure, the anisotropic part of the dipole energy is zero. One particularly simple spin structure that meets all of these constraints is the uNdN phase. This refers to N planes of up spins followed by N planes of down spins in a repeating pattern,

where $\hat{l}$ defines the normal to the planes. Note that N> 1, since N= 1 is just the cubically symmetric NAF phase discussed in Chapter 1. Mean field analysis of sums of the dipole energies [1] gives some somewhat better fits to the NMR data for the u2d2 phase as compared with the N> 2 phases, but the results are not conclusive.

A scattering peak has been observed in neutron-scattering data,[4] corresponding to the ($\frac{1}{2}$,0,0) wave vector, which disappears above the critical temperature. This is consistent with the u2d2 phase but could possibly be a secondary peak for larger even values of N. Their data consists of 50 excess neutrons scattered during a run of 500 seconds. This experiment is very difficult because of the large cross section for neutron absorption in ^{3}He and the low temperatures that must be maintained. Other directions such as ($\frac{1}{N}$,0,0) for N>2 were not looked at because of the great difficulty of keeping the sample in thermal equilibrium (at the mK level) while continually heating it with a neutron beam, and so the larger values of N weren't ruled out. Because of the limited nature of the data, it can be viewed only as a weak confirmation of the u2d2 phase.

2.1b High-Temperature Series

As well as the excellent qualitative data given by the NMR experiments, some quantitative information can be deduced from the high-temperature data. High, in this case, is relative to the 1mK transition temperature. In order to calculate the high-temperature expansion for the free energy, consider that part of the Hamiltonian involving spin coordinates,

$$\mathrm{H} = \mathrm{H}_I + \mathrm{H}_M, \tag{2.2}$$

where $\mathrm{H}_M = -\gamma\hbar \sum_i \vec{S}_i \cdot \vec{H}$ is the Zeeman splitting ($\vec{H}$ is the magnetic field), and H_I is the interaction Hamiltonian for exchange interactions. The form for H_I is developed in Section 2.2. The free energy is given by

$$F = -T \ln \{tre^{-\beta \mathrm{H}}\}. \tag{2.3}$$

Expanding this expression in powers of $\beta = \frac{1}{T}$ gives[5]

$$F = -\frac{N}{\beta}\Big[\ln 2 + \frac{e_2}{8}\beta^2 - \frac{e_3}{24}\beta^3 ... + \frac{1}{2}\big[\frac{\gamma\hbar}{2}H\beta\big]^2\big[1 + \theta\beta + \frac{\alpha\beta^2}{8} + ...\big] + ...\Big], \quad (2.4)$$

where N is the number of particles in the system and e_2, e_3, θ, and α are the coeficients of the expansion that can be expressed in terms of expectation values of the H_I at high T and zero H. The two highest-order terms, e_2 and θ, are related to the interaction Hamiltonian by

$$e_2 = \frac{4}{N}\langle \mathrm{H}_I{}^2 \rangle \quad (2.5)$$

and

$$\theta = \frac{4}{N}\langle (S^z)^2 \mathrm{H}_I \rangle, \quad (2.6)$$

where the angle brackets imply a trace over the disordered, paramagnetic state that exists at high temperature. In this state the spins have equal probability of being in the up or the down state. Extracting such coefficients for high T measurements gives well-defined averages of the interaction Hamiltonian of interest.

Eqn. (2.4) may be used to derive the expansion for the inverse-susceptibility per unit volume,

$$\chi^{-1} = -\frac{v}{\mu_0 k_B}\big[\frac{\partial^2 F}{\partial H^2}\big]^{-1} = D(T - \theta + \frac{B}{T}) \quad (2.7)$$

with

$$D = \frac{v}{\mu_0 k_B N}\big(\frac{\gamma\hbar}{2}\big)^{-2}, \qquad B = \theta^2 - \frac{\alpha}{8}, \quad (2.8)$$

the specific heat at constant volume,

$$C_V = -T\big[\frac{\partial^2 F}{\partial T^2}\big] = \frac{e_2}{4T^2}, \quad (2.9)$$

and the pressure,

$$P(T,H) = -\frac{\partial F}{\partial v} = \frac{e_2'}{8T} - \frac{e_3'}{24T^2} + \big[\frac{\gamma\hbar H}{2}\big]^2\frac{\beta^2\theta'}{2}, \quad (2.10)$$

where prime denotes the derivative with respect to molar volume. The constants μ_0 and k_B are the permeability of free space and Boltzman's constant, respectively. It should be noted that the data are only good enough to fit e_2, θ and their volume derivatives with any certainty, although in principle there is nothing to prevent fitting the series to high order, and interesting information is still available from the volume dependence of quantities such as e_3 as discussed below.

The most reliably known coefficient is e_2, since for specific volumes near melting, the $1/T^2$ term gives the dominant contribution to the specific heat up to approximately 100-200 mK. Above this temperature range, the contribution from phonons becomes appreciable near melting densities. At higher densities, away from the melting point, phonon contribution becomes significant at lower temperatures. Values of e_2 that have been extracted directly from the specific heat data are relatively accurate near the melting curve, $e_2 \simeq 5.8\ (\mathrm{mK})^2$ at a molar volume of 24.2 cm^3/mole (Fig. 7, open points). Panczyk and Adams [6] measured the temperature dependence of e_2'. Integrating e_2' numerically, they determined e_2 up to an additive constant. At low enough molar volume, the value of e_2 becomes quite small, and so ignoring it causes little error. Verification that the error is indeed small, then comes from consistency with the direct measurements: the solid points in Fig. 7.

It is much more difficult to extract the Curie-Weiss constant, θ, from experiment because at low temperature, where T is much smaller than θ (see Eqn. 2.7), the effect of higher-order terms (in powers of $\beta = \frac{1}{T}$) becomes significant. At higher temperatures large statistical errors come from extracting θ as a small correction to T. Unlike the e_2 data, there is inconsistency between the direct and the integrated values. The direct values come from measurements of χ (Fig. 8, open points), and the integrated values come from θ' extracted from pressure measurements (Fig. 8, solid points). This discrepancy may be due to large systematic errors in the direct measurements or from too small a value for the integration constant. Resolution of this inconsistency probably requires more experimental work. A careful measurement of χ over a wide range of temperature, for example, would probably resolve the issue.

2.1c Volume Dependence of Quantities that Depend upon J

One of the most interesting features of the solid ^{3}He system is that different quantities such as e_2, e_3, θ and T_N (the antiferromagnetic-ordering temperature), which must depend on very different combinations of the exchange energies, can be characterized by one parameter for a range of molar volume from approximately 22 cm^3/mole to 24 cm^3/mole. [7,8,9] The parameter is a Gruneisen parameter. Consider one such quantity, Q, with units of Energy^m. The data show that it would be well represented by the fit $Q = Q_0 v^{m\gamma}$, over the above-mentioned range of molar volume. Stating this another way, the plot of $\ln Q$ versus $\ln v$ would be a straight line with slope $m\gamma$. This requires that over the range of volumes, exchanges with very different geometry vary in nearly the same volume dependence. The values for γ for all quantities measured are consistent with 18 ± 2, according to data available up to this point.

One more precise experiment [10] claims to have found an inconsistency in the scaling of the inverse of the maximum magnetization, M_{max}, over a range of volumes from 21 cm^3/mole to 24 cm^3/mole. They found $\gamma = 16.2 \pm .5$ for M_{max} (Fig. 9), and $\gamma = 18.8 \pm 1.0$ for T_N. The small error bars on γ for M_{max}, however, seem a bit strange for the following reason. Both plots in Fig. 9 have beautiful straight-line fits, but is is a log-linear scale, which would imply an exponential, not a power-law fit. Since it is $\partial \ln Q/\partial \ln v$ that is constant, not $\gamma = v\partial \ln Q/\partial v = \partial \ln Q/\partial \ln v$, the variation in γ over the range of volumes should be $\Delta v/v \simeq 12.5\%$, and so it is hard to see how an error of 3% can be justified!

The fact that these curves seem to fit an exponential is interesting in itself. For smaller ranges of specific volume and data with larger errors, the smaller value of $\Delta v/v$ made an exponential fit indistinguishable from a power-law fit. In this case, however, we may be seeing evidence for different scaling behavior than has been deduced previously from looking at narrower ranges of molar volume.

Although the analysis of the above data by Miura *et al.* seems a bit questionable, they to have some evidence of a breakdown in the scaling of the all quantities

with one Gruneisen constant (assuming that their experiment has no significant systematic errors), which can be seen in the graph of Fig. 10. The fact that all of the points don't lie on a universal curve is evidence for the breakdown of scaling at some level. The logic behind this statement goes as follows. Since there is only one energy scale, J, the magnetization divided by the maximum magnetization must be a function of T/J and therefore of T/T_N, where T_N is the antiferromagnetic transition temperature (which is a function of J). Therefore, $M/M_{sat} = f(T/T_N)$ universally.

Getting back to the central point, it is still a puzzle as to how the different quantities show the same volume dependence, which would be expected to depend on different combinations of the competing exchange rates, and this is the question to be addressed in this work by measuring the Gruneisen parameter for two-, three- and four-particle exchange for molar volumes at 22 cm^3/mole and 24 cm^3/mole, and two-particle exchange at 20 cm^3/mole.

2.1d Low-Temperature Specific Heat

The final piece of experimental data, which may yield some constraints on this system, comes from low-temperature thermodynamic measurements such as H_0 and specific heat, which can be related to average, spin-wave velocities.[1] However, unlike the high-temperature series, which can be expected to yield many expansion coefficients given good enough data, at low temperatures quantum fluctuations make the spin wave velocities very hard to estimate from H_I. For this reason, until better methods of calculation are developed, parameters derived from the low-temperature data should be viewed with skepticism.

2.2 Separation of Energy Scales and the Exchange Hamiltonian

As mentioned in the first chapter, one of the crucial features of this system is the great separation of energy scales. At the highest level is the closed-shell, electronic structure at about 10^5 K. The atomic interaction potential and the lattice,

zero-point kinetic energy are both of order 10 K. This equivalence is a reflection of the quantum nature of the solid. Direct magnetic dipole interactions between nuclear spins are approximately 10^{-7} K, and therefore the spin quantum numbers are essentially constant labels. Exchange interactions are characterized (approximately) by the 1 mK, antiferromagnetic ordering temperature. Given the ratio of 10^{-4} between exchange energies and lattice energies, a description of the system can be made in two parts. First is the zeroth-order, lattice problem neglecting exchange interactions. The Hamiltonian doesn't contain the spins explicitly, and so the solution is that of a Schrodinger equation for N identical particles, where N is the number of particles in the system. Adding the spin labels and taking into account the Pauli principle, antisymmetrization requirement then gives a 2^N-fold degenerate spin system. The second part of the solution is the characterization of an effective exchange Hamiltonian, which leads to a small splitting of the degenerate levels. This effective Hamiltonian can be described, as shown below, in terms of the nuclear spins.

Consider the N! possible permutations of N particles on the lattice. Each permuted configuration of the N particle system can be viewed as a cavity [11,12] in the 3N-dimensional space of configurations. An exchange can then be viewed as a tunneling event through the small tubes connecting the cavities. Since exchange is rare on the time scale of the lattice vibrations, very little error will be made by considering the hops between different cavities (i.e., different types of exchange) separately and neglecting the chance that two hops will happen close enough together in space-time to affect each other.

With this approximation, the energy splitting corresponding to a particular permutation of the particles on the lattice, $P^{(R)}$, can be calculated by considering the two-state system formed by one configuration of the particles and the permuted version of that configuration. If the wavefunction corresponding to the system being confined to one of the cavities is Ψ_0, then the wavefunction for the permuted system

is $P^{(R)}$, and the eigenfunctions are given by

$$\Psi_{\pm} = \frac{1}{\sqrt{2}}(\Psi_0 \pm P^{(R)}\Psi_0). \tag{2.13}$$

Define the energy splitting to be $2J(P)$ so that J_P gives shift from the energy for a non-exchanging system.

Define the exchange operator corresponding to permutation type $P^{(R)}$ as

$$\bar{P}^{(R)} = \frac{1}{2}(P^{(R)} + P^{\dagger(R)}), \tag{2.12}$$

where $P^{\dagger(R)}$ is the inverse permutation. With the two state approximation formalized by the relations

$$P^{(R)^2}\Psi_0 = 0; \qquad P^{\dagger(R)}\Psi_0 = 0; \qquad P^{(R)}P^{\dagger(R)}\Psi_0 = \Psi_0, \tag{2.14}$$

it can be seen that

$$\bar{P}^{(R)}\Psi_{\pm} = \pm\Psi_{\pm}. \tag{2.15}$$

Hence, the form of the effective exchange Hamiltonian (referred to as H_I in Section 2.1) is

$$H_{exchange} = -\sum_P J(P)\bar{P}^{(R)}, \tag{2.11}$$

where the sum is over all permutation types P. The minus sign in front of the exchange Hamiltonian comes from the arguments in Chapter 1: The spatially symmetric nodeless wavefunction has the lowest energy.

The total wavefunction must be antisymmetric under an odd permutation of spin and spatial coordinates together. This allows the exchange operator to be expressed in terms of the spin permutation operator, $P^{(\sigma)}$, since $\bar{P}^{(R)} = (-1)^p P^{(\sigma)}$, where $(-1)^p$ (p is the parity of the exchange, the equivalent number of two-particle exchanges) is positive for exchanges involving even numbers of particles and negative for exchanges involving odd numbers of particles. Rewriting the exchange Hamiltonian in terms of the spin-permutation operator therefore gives

$$H_{exchange} = -\sum_P (-1)^p J(P)P^{(\sigma)}. \tag{2.16}$$

$H_{exchange}$ can be expressed in terms of the nuclear-spin operators by using the fact that any permutation operator can be expressed in terms of products of pairwise permutation operators. To characterize $P^{(\sigma)}$ in terms of spin operators S_i, it is therefore sufficient to write down the two-particle spin permutation operator for particles i and j,

$$P_{ij}^{(\sigma)} = \frac{1}{2} + 2\vec{S}_i \cdot \vec{S}_j. \tag{2.17}$$

2.3 Theoretical Calculations

Before discussing work that includes more than one type of exchange, it is interesting to note that during the 60's it was assumed that only two-particle, nearest-neighbor exchange would be significant. For this case, combining Eqns. (2.16) and (2.17) gives the spin-$\frac{1}{2}$ Heisenberg Hamiltonian,

$$H = 2J_2 \sum_{i,j} \vec{S}_i \cdot \vec{S}_j + const, \tag{2.18}$$

where i and j sum over all nearest-neighbor pairs and J_2 is the two-particle exchange energy. The properties of this Hamiltonian are well studied and if it were the only significant, interaction this system would be interesting primarily as a pedagogical tool. Two-particle exchange was assumed to be the only significant interaction because a WKB expression for J_2,

$$J_2 = J_0 \exp -\Big\{ \int_{path} dx \sqrt{\frac{2m}{\hbar^2}\big[V(x) - E\big]} \Big\}, \tag{2.19}$$

was expected to hold (see Chapter 7). This being the case, the longer path length and larger effective mass for several particles exchanging would exponentially suppress their effect relative to two-particle exchange.

The inconsistency of this model with the physical system is marked. The H-T phase diagram (Fig. 11 inset graph) has significantly different values of the critical temperature and magnetic field, there is no phase with high magnetization, and

finally, the antiferromagnetic phase has cubic spin symmetry, a situation ruled out by the NMR data.

In the face of the data, the idea that steric (hardcore) impedance would have a much different effect on two-particle exchange than some higher exchanges such as four-particle, nearest-neighbor was advanced by Roger, Hetherington and Delrieu.[5] The greater effect of hard-core repulsion on two-particle exchange was believed to diminish its predominance and perhaps to suppress it to the point of insignificance.

The lowered critical values and the additional phase transition observed in experiment relative to a Heisenberg system are positive indications that there is some form of competition between ferromagnetic and antiferromagnetic interactions. The addition of a three ring exchange, for example, might be expected to suppress the antiferromagnetic phase since, as explained in Chapter 1, exchanges involving odd numbers of particles favor ferromagnetic interactions.

The inadequacy of two-particle exchange and the indications of competition in the phase diagrams have led to the consideration of multiple-exchange Hamiltonians. Roger, Delrieu, Hetherington (RDH) [5] have considered several combinations of different exchange types, but the discussion here will focus on one of their more successful exchange Hamiltonians involving four-particle, planar, nearest-neighbor exchange and a three-particle exchange involving two nearest-neighbor separations, and one next nearest-neighbor separation (Fig. 11). Three quantities are known from experiment with enough accuracy be considered to fit the parameters of this model: the leading term in the high-temperature expansion of the specific heat e_2 = 7.0±.3 mK^2, the transition temperature to the antiferromagnetic phase along the melting curve T_c = 1 mK, and the low-temperature, mean spin-wave velocity C = 8.4 cm/s. The Curie-Weiss constant, θ, is not known well enough to provide a very useful constraint on the parameters.

The Hamiltonian for this process is

$$H = -J_t \sum_{ijk} \bar{P}_{ijk} + K_p \sum_{ijkl} \bar{P}_{ijkl}, \tag{2.20}$$

where the sums run over all distinct exchanges of each type, including both forward

and backward permutations, as opposed to the standard convention in which only one direction of permutation is summed over, and the is no factor of 1/2 in the definition of the exchange operators. J_t,K_p corresponds to the three-particle and four-particle exchanges, respectively. The sign convention is such that $J_t > 0$ (ferromagnetic) and $K_p > 0$ (antiferromagnetic). For some calculations, such as the high-temperature expansion, the spin representation [1] is easier to use and can be produced as was done for Eqn. (2.18).

The quantities which RDH finally settled on to fit J_t,K_p were e_2 and C. Evaluating Eqn. (2.5) using the Hamiltonian of Eqn. (2.2) RDH derived the expression

$$e_2 = 2\sqrt{6}(J_t^2 - \frac{7}{8}K_pJ_t + \frac{17}{64}K_p^2), \tag{2.21}$$

and they felt that a calculation of C at low temperatures would be more accurate than the mean-field theory estimate of T_c. It should be noted, however, that the zero-point corrections to C are probably large and that if the Curie-Weiss constant or e_3 were known accurately, they would be preferable candidates for the second parameter. The fit using e_2 and C results in

$$J_t = 0.13\text{mK}, \qquad K_p = 0.385\text{mK}. \tag{2.22}$$

A very appealing feature of this model is that its H-T phase diagram, calculated in mean-field theory, is qualitatively very similar to the experimental data (Fig. 11). There is a first-order transition to the u2d2, antiferromagnetic phase near 1.0 mK. As the field is increased in the antiferromagnetic phase, there is a phase transition across which the magnetization jumps from a small value to a value equal to approximately half the saturation value. RDH have determined that the spin structure of the high-field phase is NAF with the spins canted along the field (CNAF) and that the canting persists even in zero field. This phase is separated from the paramagnetic phase by second-order transition. In addition to the second-order line, there is a first-order line that extends into the CNAF phase, running nearly parallel to the second-order line and ending in a critical point. The first-order line is consistent with the experiments of Osheroff (1982), [13] but it is not clear

from the data if there is a critical point or whether the first-order line continues to enclose the high-field phase.

The most serious disagreement between this model and the data is the value of the critical magnetic field, H_c=15 kG at T=0, which is approximately four times the experimental value. Such a large difference is very unlikely to be accounted for by quantum spin fluctuations, a view supported by the exact solution of a 16 quantum spin system with the Hamiltonian of Eqn. (2.20),[14] in which the critical field was shifted very little compared with the classical value.

In order to improve the agreement between experiment and theory RDH tried including a two-particle, nearest-neighbor exchange. This did have the effect of lowering H_c to about 8 kG, still quite large, but at the expense of adding another antiferromagnetic phase at zero field above the u2d2 phase (Fig. 12). Good specific heat data (Fig. 13) [15] show no enhancement at a second critical temperature as would be expected if there were another phase transition.

The good qualitative agreement obtained by RDH seems to indicate that competition between different types of exchange is the right idea. If the inclusion of more exchange terms is necessary, there needs to be some explanation of why several exchanges are all of the same order for a wide range of specific volume as indicated in the work of Ceperley and Jaccuci.[16] The Gruneisen parameters calculated in this work may provide the first step toward an explanation of this puzzle.

References

1. M. C. Cross and Daniel S. Fisher, *Rev. Mod. Phys.*, **57**, 881, Oct. 1985.
2. D. D. Osheroff, M. C. Cross, and D. S. Fisher, *Phys. Rev. Lett.*, **44**, 792, 1980.
3. E. D. Adams, E. A. Schubert, G. E. Haas, and D. M. Bakalyar, *Phys. Rev. Lett.*, **44**, 789, 1980.
4. A. Benoit, J. Bossy, J. Flouquet, and J. Schweizer, *Jour. de Physique Lettres*, **46**, L925, 1985.
5. M. Roger, J. H. Hetherington, and J. M. Delrieu, *Rev. Mod. Phys.*, **55**, 1, Jan. 1983.
6. M. F. Panczyk and E. D. Adams, *Phys. Rev. A*, **1**, 1356, 1970.
7. D. S. Greywall and P. A. Busch, *Phys. Rev. B*, **36**, 6853, Nov 1987.
8. T. Hata, S. Yamasaki, M. Taneda, T. Kokama, and T. Shigi, *Phys. Rev. Lett.*, **51**, 1573, 1983.
9. K. G. Wildes, M. R. Freeman, J. Saunders, and R. C. Richardson, *Jour. Low Temp. Phys.*, **62**, 67, 1987.
10. Y. Miura, N. Nishida, Y. Takano, H. Fukuyana, S. Ogawa, T. Hata, and T. Shigi, *Phys. Rev. Lett.*, **58**, 381, 1987.
11. D. J. Thouless, *Proc. Phys. Soc. London*, **86**, 893, 1965.
12. M. Roger and J. M. Delrieu, *Japanese Jour. of Appl. Phys.*, **26**, 267, 1987.
13. D. D. Osheroff, *Physica B+C*, **109-110**, 1461, 1982.
14. M. C. Cross and R. N. Bhatt, *Jour. Low Temp. Phys.*, **57**, 573, 1984.
15. D. S. Greywall, *Phys. Rev. B*, **15**, 2604, 1977.
16. D. M. Ceperley and G. Jacucci, *Phys. Rev. Lett.*, **58**, 1648, Apr. 1987.

CHAPTER 3

The Path Integral

The approach taken in this work is to approximate nature by a periodically extended finite system. For the systems considered, the particles are in the quantum mechanical regime (they are very light and sit in a broad potential with a slight maximum at the lattice sites), and so they cannot be evolved using force equations as is done in molecular dynamics. Because of the complexity of the system, the wavefunctions are not known and therefore observable quantities cannot be calculated as expectation values. The approach taken instead is to consider the density matrix for the system propagating from one position eigenstate to another in a time large compared with the time scale for the interactions of interest. This long-time density matrix is expressed as an integral over all paths, between the initial and final configuration, of the short-time (or high-temperature) density matrix, which can be accurately approximated in terms of two-particle density matrices, as explained later in this chapter. The long time density matrix gives the probability of the particular evolution of the system, and when it is expressed as a path integral, the integrand gives the conditional probability distribution of the paths which the system can take between its initial and final positions.

To make the path integral accessible from a computational point of view, it

is discretized (in time) so that the originally continuous path between the initial and final configurations is a list of positions for all of the particles in the system at discrete time intervals. Monte Carlo techniques can then be used to produce a sequence of configurations distributed according to conditional probability distribution of the paths. Any observable (e.g., the energy density) that is not constrained by the system size may be "measured" on these configurations and its average calculated. Assuming that the quantity being measured does not change either when the system size is increased, when there is a decrease in the time interval between the discrete steps in the path or when the total path length is increased, it can be assumed that the calculated average is the correct value of the observable for a macroscopic system at zero temperature. Both of these checks will be done for the systems considered in this work.

In the rest of this chapter the mathematical machinery used for the Monte Carlo simulation is introduced. The density matrix is formulated as a path integral and an expression for the integrand is developed as a pair product form of the density matrix in the short-time limit. Once the path integral is formulated, a suitable Monte Carlo observable is defined, which can be related to the Gruneisen parameters . The general approach for deriving the observable is to introduce a length-scaling parameter, λ, into the path integral. Differentiation with respect to $\ln \lambda$ leads to an expression for the derivative of the path integral with respect to volume. The two-state approximation discussed in Section 2.2 is then used to relate the path integral to the exchange energies and hence to the Gruneisen parameters. Finally, the Monte Carlo method is described along with a method for generating the simultaneous move of several timeslices of one or more particles, instead of the usual one-particle, one-timeslice move.

3.1 From the Density Matrix to the Path Integral

The density matrix is derived from the Schrodinger equation,[1]

$$\frac{\partial \rho(\beta)}{\partial \beta} = -H\rho(\beta), \tag{3.1}$$

where $\hbar\beta$ is imaginary time and the formal solution is

$$\rho(\beta) = e^{-\beta H}. \tag{3.2}$$

The time† may be broken up into n pieces, $\epsilon = \beta/n$, such that

$$\rho(\beta) = \prod_{I=1}^{n} e^{-\epsilon H}. \tag{3.3}$$

To derive the coordinate representation for the density matrix, the relation

$$\rho(x, x'; \beta) = \langle x \,|\, \rho(\beta) \,|\, x' \rangle = \langle x \,|\, \prod_{I=1}^{n} e^{-\epsilon H} \,|\, x' \rangle \tag{3.4}$$

will be used, where x, x' are many-particle position vectors. Inserting a complete set of position eigenstates,

$$1 = \int_{-\infty}^{\infty} |x^I\rangle\langle x^I| \, dx^I, \tag{3.5}$$

between each of the terms in Eqn. (3.4) and taking the expectation value between the state corresponding to all of the particles in their initial positions, $\langle x \,|$, and that for all of the particles in their final positions after a time β, $|\, x' \rangle$,

$$\begin{aligned} &\rho(x, x'; \beta) \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \langle x \,|\, e^{-\epsilon H} \,|\, x^1 \rangle \langle x^1 \,|\, e^{-\epsilon H} \,|\, x^2 \rangle \cdots \langle x^{n-1} \,|\, e^{-\epsilon H} \,|\, x' \rangle dx^1 dx^2 \ldots dx^{n-1} \\ &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \rho(x, x^1; \epsilon) \rho(x^1, x^2; \epsilon) \ldots \rho(x^{n-1}, x'; \epsilon) dx^1 dx^2 \ldots dx^{n-1}. \end{aligned} \tag{3.6}$$

The path is then defined as the sequence of positions at subsequent time intervals,

$$[x] = \{x, x^1, x^2, \ldots, x^{n-1}, x'\}, \tag{3.7}$$

† β will be referred to as the time, even though its units are inverse energy.

and $e^{-\epsilon H}$ is referred to as the time-evolution operator. In general, superscripts will refer to time indices and subscripts will refer to particle number indices. The lack of a subscript implies a many particle vector containing the positions of all of the particles in the system. The path integral is defined as the limiting value of the density matrix as $n \rightarrow \infty$ with $\beta = n\epsilon$ fixed. For future convenience, Eqn. (3.6) can be rewritten as

$$\rho(x, x'; \beta) = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} e^{-S([x];\beta)} d([x]), \tag{3.8}$$

with normalization constants dropped, where

$$d([x]) = \lim_{\substack{n\rightarrow\infty \\ n\epsilon=\beta}} dx^1 dx^2 ... dx^{n-1} \tag{3.9}$$

and

$$S([x];\beta) = -\lim_{\substack{n\rightarrow\infty \\ n\epsilon=\beta}} \sum_{I=0}^{n-1} \ln[\rho(x^I, x^{I+1}; \epsilon)], \tag{3.10}$$

where $x = x^0$ and $x' = x^n$. Finite n versions of these expressions are used for the Monte Carlo computations. The value of n must be large enough so that the quantity being calculated has converged.

3.2 The Many Particle Density Matrix to First Order

Before solving for the density matrix, including the interparticle potential, it is useful to write down the solution for the free-particle density matrix, where the Hamiltonian, $H_0 = p^2/2m$, leads to the Schrodinger equation

$$\frac{\partial \rho_1(x, x'; \beta)}{\partial \beta} = \frac{\hbar^2}{2m} \frac{\partial^2 \rho_1(x, x'; \beta)}{\partial x^2}. \tag{3.11}$$

The solution to this diffusion equation is

$$\rho_1(x, x'; \beta) = \left(\frac{m}{2\pi\hbar^2\beta}\right)^{\frac{3N}{2}} e^{-(\frac{m}{2\hbar^2\beta})(x-x')^2}, \tag{3.12}$$

where the 1 subscript implies the free-particle (or one particle), density matrix and N is the number of particles in the system.

To calculate the density matrix for a system of particles moving in a potential, consider the following perturbation expansion [1] about the free-particle density matrix. Define the difference between the free-particle and the exact density matrix as,

$$\delta\rho(x,x';\epsilon) = \rho(x,x';\epsilon) - \rho_1(x,x';\epsilon), \tag{3.13}$$

which will be small in the short-time limit, $\epsilon \to 0$. Writing the Hamiltonian as $H = H_0 + V$, the formal solution is given by Eqn. (3.2). Since ρ is very close to ρ_1, $\rho/\rho_1 = e^{\epsilon H_0}\rho$ should vary slowly with ϵ. Taking the derivative of this quantity with respect to ϵ,

$$\begin{aligned}\frac{\partial}{\partial\epsilon}\left(e^{\epsilon H_0}\rho\right) &= e^{\epsilon H_0}H_0\rho + e^{\epsilon H_0}\frac{\partial\rho}{\partial\epsilon} \\ &= e^{\epsilon H_0}H_0\rho - e^{\epsilon H_0}\left(H_0+V\right)\rho \\ &= -e^{\epsilon H_0}V\rho.\end{aligned} \tag{3.14}$$

Integrating Eqn. (3.14) and using the boundary condition that $e^{\epsilon H_0}\rho\,|_{\epsilon=0} = 1$,

$$e^{\epsilon' H_0}\rho(\epsilon')\,|_0^\epsilon = -\int_0^\epsilon e^{\epsilon' H_0}V\rho(\epsilon')d\epsilon' \tag{3.15}$$

$$\rho(\epsilon) = e^{-\epsilon H_0} - \int_0^\epsilon e^{-(\epsilon-\epsilon')H_0}V\rho(\epsilon')d\epsilon'. \tag{3.16}$$

Expressing this in the coordinate representation gives

$$\begin{aligned}\langle x\,|\,\rho(\epsilon)\,|\,x'\rangle = \langle x\,|\,\rho_1(\epsilon)\,|\,x'\rangle - \\ \int_0^\epsilon \langle x\,|\,e^{-(\epsilon-\epsilon')H_0}\Big\{\int_{-\infty}^{\infty}|\,x''\rangle\langle x''\,|\,dx''\Big\}V\rho(\epsilon')\,|\,x'\rangle d\epsilon',\end{aligned} \tag{3.17}$$

and since $\langle x'' \,|\, V\rho(\epsilon') \,|\, x' \rangle = V(x'')\langle x'' \,|\, \rho(\epsilon') \,|\, x' \rangle$,

$$\begin{aligned} \rho(x,x';\epsilon) &= \rho_1(x,x';\epsilon) - \int_0^{\epsilon} \int_{-\infty}^{\infty} \rho_1(x,x'';\epsilon-\epsilon')V(x'')\rho(x'',x';\epsilon')dx''d\epsilon' \\ &\simeq \rho_1(x,x';\epsilon) - \int_0^{\epsilon} \int_{-\infty}^{\infty} \rho_1(x,x'';\epsilon-\epsilon')V(x'')\rho_1(x'',x';\epsilon')dx''d\epsilon', \end{aligned} \tag{3.18}$$

where ρ has been replaced by ρ_1 in the final expression in Eqn. (3.18) to produce the perturbation expansion to first-order. The next order correction can be generated by replacing ρ on the right-hand side of the first line in Eqn. (3.18) with the first-order solution.

Since ϵ is very small, the integrand in Eqn. (3.18) is highly peaked around

$$x'' = \frac{x\epsilon + (\epsilon - \epsilon')x'}{\epsilon}, \tag{3.19}$$

as can be seen by finding the minimum of the sum of the exponentials of the two free-particle terms. Performing the x'' integral with $V(x'')$ set equal to its value at the position given in Eqn. (3.19),

$$\delta\rho = -\rho_1(x,x';\epsilon) \int_0^{\epsilon} V\Big(\frac{x\epsilon + (\epsilon-\epsilon')x'}{\epsilon}\Big)d\epsilon'. \tag{3.20}$$

Estimating the integral as a simple sum gives

$$\delta\rho = -\rho_1(x,x';\epsilon)\frac{\epsilon}{2}\big(V(x) + V(x')\big), \tag{3.21}$$

so that

$$\rho(x,x';\epsilon) = \rho_1(x,x';\epsilon)\Big(1 - \frac{\epsilon}{2}\big(V(x) + V(x')\big)\Big). \tag{3.22}$$

Finally, for notational convenience,

$$\rho(x,x';\epsilon) = \rho_1(x,x';\epsilon)e^{-\frac{\epsilon}{2}\left(V(x)+V(x')\right)} \tag{3.23}$$

to first-order in ϵ. Eqn (3.23) is a first order approximation to the many-particle density matrix for a system of particles moving in a potential $V(x)$, which depends upon the separations of all of the particles.

3.3 Formulating the Density Matrix to Include Higher Order Terms

Since it is possible to calculate very accurately the two-particle density matrix, ρ_2, it is useful to express the many-particle density matrix in terms of it and the free-particle density matrix. (Methods for calculating the two-particle density matrix for both a hard-core potential and for the helium potential will be discussed in Chapters 5 and 6, respectively.) Taking Eqn. (3.23) for the special case of a two-particle system, the two-particle density matrix is given by

$$\begin{aligned}\rho_2(x_i, x_j, x_i', x_j'; \epsilon) &= \rho_1(x_i, x_i'; \epsilon)\rho_1(x_j, x_j'; \epsilon)e^{-\frac{\epsilon}{2}\left(V(x_{ij})+V(x_{ij}')\right)} \\ &\equiv \rho_1(x_i, x_i'; \epsilon)\rho_1(x_j, x_j'; \epsilon)\tilde{\rho}_2(x_{ij}, x_{ij}'; \epsilon),\end{aligned} \tag{3.24}$$

where $x_{ij} = x_i - x_j$ and $\tilde{\rho}_2$ is the part of the two-particle density matrix involving interactions between particles. Expressing the many-particle density matrix of Eqn. (3.23) in terms of ρ_1 and $\tilde{\rho}_2$ for a many particle system moving from timeslice I and position x^I to timeslice $I+1$ and position x^{I+1} in a time ϵ gives

$$\rho(x^I, x^{I+1}; \epsilon) = \prod_{i=1}^{N}\left\{\rho_1(x_i^I, x_i^{I+1}; \epsilon)\prod_{j=i+1}^{N}\tilde{\rho}_2(x_{ij}^I, x_{ij}^{I+1}; \epsilon)\right\}, \tag{3.25}$$

where N is the number of particles in the system. This form is used to generate higher order approximations by using better representations for $\tilde{\rho}_2$.[2] This allows the number of timeslices which the path is divided into (n of Eqn. (3.3)) to be smaller for a given accuracy and takes much better account of abrupt potentials such as that of a hard core.

The action as defined in Eqn. (3.10) may then be written

$$S([x];\beta) = \sum_{\substack{I=0 \\ n\epsilon=\beta}}^{n-1}\sum_{i=1}^{N}\left\{\frac{m}{2\hbar^2\epsilon}(x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N}U(x_{ij}^I, x_{ij}^{I+1}; \epsilon)\right\}, \tag{3.26}$$

where

$$U(x_{ij}^I, x_{ij}^{I+1}; \epsilon) \equiv -\ln\left[\tilde{\rho}_2(x_{ij}^I, x_{ij}^{I+1}; \epsilon)\right]. \tag{3.27}$$

U is often referred to as an effective potential because of the similarity to its role in the short-time density matrix. In fact, to first order, $U(x_{ij}, x'_{ij}; \epsilon) = -\epsilon/2\big(V(x_{ij}) + V(x'_{ij})\big)$. A constant term corresponding to the normalization factors in the free-particle density matrices of Eqn. (3.12) has been left out of Eqn. (3.26). This term can be considered as a shift in the total energy, which has no effect on the final results. The path integral, denoted as Z in the language of statistical mechanics instead of as ρ, is then written as

$$Z_P \equiv \rho(x, x'; \beta) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-S([x];\beta)} d([x]). \tag{3.28}$$

The subscript P refers to the exchange path defined by x, $x' = Px$, β. Z_0 will refer to the ground-state path integral for which $P = 1$.

3.4 The Monte Carlo Observable and the Exchange Rate

In this section, the Gruneisen parameters, γ_P, are related to mathematical observables, which are measured on the configurations generated using path integral Monte Carlo techniques. The results from this section are used in the ^{3}He calculations. A different result, derived in an analogous manner, will be used in Chapter 5 for quantum hard spheres. In Section 3.4a, the derivative of the path integral with respect to a lattice scaling parameter is calculated and related to the derivative of the exchange rate with respect to specific volume (volume/particle). In Section 3.4b, the connection will be made between the Monte Carlo observable and γ_P.

3.4a Taking the Derivative of the Path Integral

In order to calculate the volume derivative of Z, begin by explicitly including

a lattice-scaling parameter λ in the expression for the path integral,

$$Z(\lambda x, \lambda x'; \beta) = \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} e^{-S([\lambda x];\beta)} \lambda^{3N(n-1)} d([x]), \tag{3.29}$$

with λ nominally equal to one. Taking the derivative of Eqn. (3.29) then gives

$$\lambda \frac{\partial Z}{\partial \lambda} = 3N(n-1)Z - \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \lambda \frac{\partial S}{\partial \lambda} e^{-S([x];\beta)} \lambda^{3N(n-1)} d([x]). \tag{3.30}$$

With the definition

$$\Phi([\lambda x];\beta) \equiv \lambda \frac{\partial S([\lambda x];\beta)}{\partial \lambda}, \tag{3.31}$$

the derivative of S, given in Eqn. (3.26), with respect to λ evaluated at $\lambda = 1$ gives

$$\Phi([x];\beta) = \sum_{I=0}^{n-1} \sum_{i=1}^{N} \left\{ \frac{m}{\hbar^2 \epsilon} (x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N} (x_{ij}^I \cdot \nabla^I + x_{ij}^{I+1} \cdot \nabla^{I+1}) U(x_{ij}^I, x_{ij}^{I+1}; \epsilon) \right\}. \tag{3.32}$$

Substituting Eqn. (3.31) into Eqn. (3.30) with $\lambda = 1$ and dividing by Z, the result is

$$\frac{\partial \ln Z}{\partial \ln \lambda} = 3N(n-1) - \frac{1}{Z} \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \Phi([x];\beta) e^{-S([x])} d([x]), \tag{3.33}$$

Finally, to relate λ to the specific volume v, note that $v \propto \lambda^3$. Therefore, $\partial \ln \lambda = \frac{1}{3} \partial \ln v$ so that

$$\begin{aligned} \frac{\partial \ln Z}{\partial \ln v} &= N(n-1) - \frac{1}{Z} \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \frac{1}{3} \Phi([x];\beta) e^{-S([x])} d([x]) \\ &= N(n-1) - \frac{1}{3} \langle \Phi([x];\beta) \rangle. \end{aligned} \tag{3.34}$$

The last term in Eqn. (3.34) is the expectation value of $\Phi([x];\beta)$ in the ensemble of configurations produced by the path integral Monte Carlo as explained in Section 3.5 and will be referred to as the Monte Carlo observable.

3.4b Relating the Observable to the Exchange Rate

Let $\bar{P}$ be a hermitian exchange operator as defined in Chapter 2. Consider

$$Z_P = \rho(x, \bar{P}x; \beta) = \langle x \,|\, e^{-\beta H} \,|\, \bar{P}x \rangle. \tag{3.35}$$

The nonexchanging path integral corresponding to $\bar{P} = 1$ will be denoted Z_0. If $\{\Psi_i\}$ is a complete set of energy eigenstates, then

$$Z_P = \sum_i \langle x \,|\, e^{-\beta H} \,|\, \Psi_i \rangle \langle \Psi_i \,|\, \bar{P}x \rangle. \tag{3.36}$$

Now consider the "home based" wavefunction $|\,\phi\rangle$, [3] where all atoms are localized at their lattice sites (no exchange). The permutation operator, acting on $|\phi\rangle$, gives a new permuted state $|\,\bar{P}\phi\rangle$. Given the rarity of exchanges in the lattice, these two configurations form an approximate two-state system, for the purpose of calculating the energy splitting that is due to exchange type P. The eigenfunctions are

$$|\,\Psi_\pm\rangle = \frac{1}{\sqrt{2}}(|\phi\rangle \pm |\,P\phi\rangle), \tag{3.37}$$

with eigenvalues

$$E_\pm = E_0 \mp J_P, \tag{3.38}$$

where E_0 is the energy of the system without exchange and $2J_P = E_- - E_+$, J_P being the exchange energy of exchange type P. Note that in the two-state approximation,

$$\bar{P} \,|\,\Psi_\pm\rangle = \pm \,|\,\Psi_\pm\rangle. \tag{3.39}$$

With the definitions

$$a_\pm^2 = |\,\langle x \,|\, \Psi_\pm \rangle\,|^2, \tag{3.40}$$

and using Eqns. (3.36) and (3.39),

$$\frac{Z_P}{Z_0} = \frac{a_+^2 e^{-\beta E_+} - a_-^2 e^{-\beta E_-}}{a_+^2 e^{-\beta E_+} + a_-^2 e^{-\beta E_-}} = \frac{a_+^2 e^{\beta J_P} - a_-^2 e^{-\beta J_P}}{a_+^2 e^{\beta J_P} + a_-^2 e^{-\beta J_P}}. \tag{3.41}$$

With the further definition

$$\frac{a_+^2}{a_-^2} = e^{2K}, \tag{3.42}$$

Eqn. (3.41) gives

$$\frac{Z_P}{Z_0} = \tanh(\beta J_P + K). \tag{3.43}$$

In the home-based wave function approximation each particle is considered to be confined to its lattice site, and exchange can be viewed to some approximation as coming from the overlap in the wavefunctions of neighboring particles. K is just a measure of the extent of overlap between the wavefunction before and after an exchange. The fact that exchange interactions modify the ground state energy by only 1 part in 10^4 is therefore a good indication that K will be very small. βJ_P can be seen to be much less than one because $J_P \propto 10^{-3}$ K^{-1} empirically, as mentioned above, and $\beta \ll 10^3$ K^{-1} in all of the simulations done for this work.

With $\tanh(\beta J_P + K) \simeq \beta J_P + K$, taking the derivative of the logarithm of Eqn. (3.43) with respect to the logarithm of the specific volume v gives

$$\frac{\partial \ln Z_P}{\partial \ln v} - \frac{\partial \ln Z_0}{\partial \ln v} = \frac{1}{\beta J_P + K}\left(\beta \frac{\partial J_P}{\partial \ln v} - \frac{\partial K}{\partial \ln v}\right). \tag{3.44}$$

Defining minus 1/3 times the left-hand side of Eqn. (3.44) as Θ, it is just minus the difference between the Monte Carlo observable in an exchanging configuration, $\langle \Phi \rangle_P$, and in a non-exchanging configuration, $\langle \Phi \rangle_0$. Θ is therefore related to J_P and K by

$$\Theta \equiv -\frac{1}{3}\left(\langle \Phi \rangle_P - \langle \Phi \rangle_0\right) = \frac{1}{\beta J_P + K}\left(\beta \frac{\partial J_P}{\partial \ln v} - \frac{\partial K}{\partial \ln v}\right). \tag{3.45}$$

With the Gruneisen parameter defined as

$$\gamma_P = \frac{\partial \ln J_P}{\partial \ln v}, \tag{3.46}$$

γ_P is related to J_P and K by

$$\gamma_P = \left(1 + \frac{K}{\beta J_P}\right)\Theta - \frac{1}{\beta J_P}\frac{\partial K}{\partial \ln v}. \tag{3.47}$$

One approach to extract γ_P is by measuring Θ for different values of β and projecting to the $\beta = \infty$ limit, since

$$\gamma_P = \Theta \mid_{\beta=\infty} . \tag{3.48}$$

In order to reduce the large, computational demands of this approach, techniques have been employed to minimize and possibly to eliminate the effect of the terms that include K in Eqn. (3.44). The elimination of those terms would make Eqn. (3.48) valid for finite β. Those terms come from what are essentially end point effects in the path integral.

The path begins with all the particles "frozen" at their lattice sites in the state $|x\rangle$. As the path evolves from that starting point, the particles relax into a pseudogroundstate (the ground state for a system without exchange), which doesn't "know" about the frozen starting point. At some later time the particles participating in the exchange leave the pseudogroundstate and move to their new lattice cells. This exchange period is quite short compared with the total path length, β. Next, the particles settle back into a new exchanged pseudo-ground state and are finally "frozen" into their new lattice sites in the state $|Px\rangle$.

The observable is actually measured over only a minimal range of time containing the exchange, eliminating as much pseudogroundstate as possible from the observable. This serves two purposes. Firstly, the pseudogroundstate in the exchanging configuration will give the same contribution to $\langle\Phi\rangle_P$ as the true ground state in the nonexchanging configuration contributes to $\langle\Phi\rangle_0$, and will therefore cancel numerically when $\langle\Phi\rangle_P - \langle\Phi\rangle_0$ is calculated. Secondly, and more importantly, the regions of the path near the fixed end points contain most of the finite path-length error coming from K, and so avoiding those regions can (to be verified empirically) avoid the effects of K altogether. The only problem occurs when the exchanging region moves too close to one end of the path. In that case, leaving the end points fixed, the rest of the path is shifted so that the exchanging region (or instanton [4]) is moved to the center of the path. How the instanton corresponding to the exchange is measured, and the procedure for shifting it will be described in

detail in Chapter 4. After the shift, the particle positions at the two ends of the path are in nonequilibrium configurations, and so the system is allowed to rethermalize in a Monte Carlo sense. This shifting technique is feasible only because of the empirical fact that the instanton moves very slowly with the number of lattice updates, as compared with the number of updates required to rethermalize the lattice.

3.5 The Monte Carlo Method

This section will present the Monte Carlo methods for calculating

$$\langle \Phi \rangle = \frac{\int \Phi([x]) e^{-S([x])} d([x])}{\int e^{-S([x])} d([x])}. \tag{3.49}$$

The basic idea is to generate a sequence of lattice configurations

$$[x]_1, [x]_2, ..., [x]_N \tag{3.50}$$

distributed according to $e^{-S([x])}$ and to evaluate

$$\langle \Phi \rangle = \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} \Phi([x]_n). \tag{3.51}$$

The Markov chain procedure [5] is used to generate a sequence of configurations beginning with an arbitrary start. After a thermalization period, configurations are produced according to the probability distribution $e^{-S([x])}$.

If R_{ij} is the rate at which lattice configurations $[x]_i$ go to $[x]_j$, then the condition of detailed balance,

$$R_{ij} = R_{ji}, \tag{3.52}$$

is sufficient to guarantee that the sequence of configurations correctly samples the probability distribution. Such a sequence is a Markov chain. If T_{ij} is the probability of picking a move from $[x]_i$ to $[x]_j$ and P_{ij} is the probability of accepting that move, then

$$R_{ij} = T_{ij} P_{ij} e^{-S([x]_i)}, \tag{3.53}$$

where $e^{-S([x]_i)}$ is the probability of being in configuration $[x]_i$. The constraint in Eqn. (3.52) can then be rewritten as

$$\frac{P_{ij}}{P_{ji}} = \frac{T_{ji}}{T_{ij}} e^{-\Delta S_{ji}} \tag{3.54}$$

where $\Delta S_{ji} = S_j - S_i$.

3.5a Standard Metropolis

For the standard Metropolis method (Metropolis *et al.*[6]) method, a trial move is generated, which has the property that the probability of picking state j, given state i, is the same as the probability of picking state i, given state j ($T_{ij} = T_{ji}$). The ratio of the probability of accepting a move $[x]_i \rightarrow [x]_j$ divided by the probability of accepting a move $[x]_j \rightarrow [x]_i$ is then

$$\frac{P_{ij}}{P_{ji}} = e^{-\Delta S_{ji}}. \tag{3.55}$$

If $\Delta S_{ji} < 0$, the move is accepted. Otherwise it is accepted conditionally with probability $e^{-\Delta S_{ji}}$. It can be seen that this prescription satisfies Eqn. (3.55), and hence detailed balance.

In order to generate moves that are not too far from the previous configuration (so that acceptance rates aren't too low), moves of the form

$$[x]_j = [x]_i + \delta[x] \tag{3.56}$$

are typically chosen, where $\delta[x]$ is some "small" increment (often random over the increment size), which has the requisite property that $T_{ij} = T_{ji}$. This type of move has the characteristic that subsequent lattice configurations are highly correlated and hence many updates must be done between measurements of the observable (typically on the order of 50 updates of the whole system).

3.5b Metropolis with Biased Moves

In order to produce less correlated configurations and therefore to move through the space of configurations more rapidly, it is useful to bias the selection of new configurations toward regions where $e^{-S([x])}$ is large. A further benefit of such a scheme is that it is possible to move several consecutive timeslices of some particle simultaneously, effectively decreasing the number of degrees of freedom to be evolved by the Monte Carlo. The method presented here is due to Pollock and Ceperley.[2]

In order to maintain the detailed balance constraint of Eqn. (3.52) with a biased move, the acceptance criterion must be altered to account for the fact that $T_{ij} \neq T_{ji}$, when biasing occurs. For the particular biasing scheme used here, the new position for particle i at timeslice I, $x^I_{i,new}$, does not depend on its old position, x^I_i, but depends only on the particles' positions (all the particles in the system) at an earlier and a later timeslice. This implies that the probability of selecting a new configuration $[x]_j$ given an initial configuration $[x]_i$, depends only on the probability of configuration $[x]_j$ occurring, and hence $T_{ij} = T_j$. The modified acceptance criterion is therefore given by

$$\frac{P_{ij}}{P_{ji}} = \frac{T_i}{T_j} e^{-\Delta S_{ji}} = e^{-\Delta S'_{ji}}, \tag{3.57}$$

where

$$\Delta S'_{ji} = \Delta S_{ji} - \ln\left(\frac{T_i}{T_j}\right). \tag{3.58}$$

The criterion for accepting a move is then the same as that described in Section 3.5a with $\Delta S'_{ji}$ substituted for ΔS_{ji}.

In the remainder of this section, the procedure for generating a biased move will be explained. The form for T_j will be given with a slight change in notation. Since the j subscript refers to a particular set of values of all position coordinates in the path, $T([x])$ will denote the probability distribution for an arbitrary configuration $[x]$.

Consider the density matrix

$$\rho(x, x'; \beta) = \int \rho(x, x''; \tau)\rho(x'', x'; \beta - \tau)dx'', \tag{3.59}$$

where the integrand, if it could be sampled directly, would give the ideal distribution for a trial move position x''. The integrand is too complicated, however, and is therefore approximated by a Gaussian. Rewriting it as

$$\rho(x,x';\beta)\Big(\frac{\rho(x,x'';\tau)\rho(x'',x';\beta-\tau)}{\rho(x,x';\beta)}\Big), \tag{3.60}$$

the term inside the parentheses is the correctly normalized conditional probability of choosing a new point, x'', based on the initial and final points, x and x', respectively. Writing the correction to the free-particle, density matrix as

$$\tilde{\rho}(x,x';\beta) = \frac{\rho(x,x';\beta)}{\rho_1(x,x';\beta)}, \tag{3.61}$$

where as before, ρ_1 is the free-particle density matrix. The correction terms from Eqn. (3.60) may be expanded about $x'' = x, \tau = 0$ giving

$$\begin{aligned}\frac{\tilde{\rho}(x,x'';\tau)\tilde{\rho}(x'',x';\beta-\tau)}{\tilde{\rho}(x,x';\beta)} &\simeq \frac{\tilde{\rho}(x,x;0)\big(\tilde{\rho}(x,x';\beta)+\nabla\tilde{\rho}(x,x';\beta)(x''-x)\big)}{\tilde{\rho}(x,x';\beta)} \\ &\simeq e^{(\nabla \ln \tilde{\rho}(x,x';\beta))(x''-x)}.\end{aligned} \tag{3.62}$$

Expressing the free-particle terms and the correction term as a single Gaussian gives the expression for the conditional probability, $T([x])$, given by

$$T([x]) = \frac{\rho(x,x'';\tau)\rho(x'',x';\beta-\tau)}{\rho(x,x';\beta)} = \Big[\frac{1}{2\pi\sigma^2}\Big]^{\frac{3N}{2}} e^{-\frac{(x''-\bar{x})^2}{2\sigma^2}} + O(\tau) \tag{3.63}$$

with

$$\bar{x} = \frac{(\beta-\tau)x+\tau x'}{\beta} + \frac{\tau(\beta-\tau)}{\beta}s(x,x';\beta), \tag{3.64}$$

$$s(x,x';\beta) = \Lambda\nabla\ln\tilde{\rho}(x,x';\beta), \tag{3.65}$$

$$\sigma^2 = \tau\Lambda\frac{\beta-\tau}{\beta}, \tag{3.66}$$

where $\Lambda = m/\hbar^2$. Eqn (3.63) gives the probability distribution used for calculating T_j/T_i in Eqn. (3.58). A more formal derivation may be found in an appendix of

Pollock and Ceperley.[2] It is interesting to note that the trajectory defined by $\bar{x}$ in Eqn. (3.64) defines the classical path, in the sense that it minimizes the action in a quadratic approximation. The first term in Eqn. (3.64) gives the intermediate positions of the particles at time τ if they were to follow the classical, free-particle trajectory. The second term gives a correction coming from interparticle interactions. When applied to a subset of particles within a larger system, Eqn. (3.64) defines the classical path only for the subset of the system, for the case when the rest of the system fixed.

For the calculations done in this work, τ is set equal to $\beta/2$. Generating a move with several consecutive timeslices between fixed end points is done in stages using a bisection method. First, the center timeslice is generated. Next, the timeslices halfway between the center and the two fixed end points are generated, etc. In this way,

$$1, 3, 7, ..., \sum_{i=0}^{n} 2^i = 2^n - 1 \tag{3.67}$$

timeslices can be threaded in between fixed endpoints.

3.6 Summary

The following is a summary of the important results derived in this chapter. In Sections 3.1-3.3, the path integral representation of the density matrix was developed and is given by

$$Z_P = \rho(x, Px; \beta) = \int_{-\infty}^{\infty} ... \int_{-\infty}^{\infty} e^{-S([x];\beta)} d([x]), \tag{3.68}$$

with the definitions

$$S([x];\beta) = \sum_{\substack{I=0 \\ n\epsilon=\beta}}^{n-1} \sum_{i=1}^{N} \left\{ \frac{m}{2\hbar^2\epsilon}(x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N} U(x_{ij}^I, x_{ij}^{I+1}; \epsilon) \right\} \tag{3.69}$$

and

$$U(x_{ij}^I, x_{ij}^{I+1}; \epsilon) = -\ln\left[\tilde{\rho}_2(x_{ij}^I, x_{ij}^{I+1}; \epsilon)\right]. \tag{3.70}$$

P is an exchange operator. In Section 3.4a, the Monte Carlo observable is derived from the path integral and has the form

$$\left\langle \Phi \right\rangle = \frac{\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \Phi([x];\beta) e^{-S([x])} d([x])}{\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-S([x])} d([x])}, \tag{3.71}$$

where

$$\Phi([x];\beta) = \sum_{I=0}^{n-1} \sum_{i=1}^{N} \left\{ \frac{m}{\hbar^2 \epsilon} (x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N} (x_{ij}^I \cdot \nabla^I + x_{ij}^{I+1} \cdot \nabla^{I+1}) U(x_{ij}^I, x_{ij}^{I+1}; \epsilon) \right\}. \tag{3.72}$$

In Section 3.4b, the Gruneisen parameter is related to the Monte Carlo observable. With end point effects removed, the result is

$$3\gamma_P = -\left\langle \Phi \right\rangle_P + \left\langle \Phi \right\rangle_0. \tag{3.73}$$

In Section 3.5a, the standard Monte Carlo procedure is explained and a final result (not reproduced here), given by Eqns. (3.63-66), is the conditional probability distribution for the intermediate position of the particles in the system, given fixed starting and ending positions. It is derived as a Gaussian approximation to the many-particle density matrix.

References

1. R. P. Feynman, *Statistical Mechanics: A Set of Lectures*, Benjamin/Cummings, Reading, Mass., 1972.
2. E. L. Pollock and D. M. Ceperley, *Phys. Rev. B*, **30**, 2555, Sept. 1984.
3. C. Herring, *Rev. Mod. Phys.*, **34**, 631, 1962.
4. Coleman, *Erice Lectures*, Plenum Press, New York, 1977.
5. K. Binder, *Monte Carlo Methods in Statistical Physics*, Springer-Verlag, New York, NY, 1979.
6. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. M. Teller, and E. Teller, *Jour. Chem. Phys.*, **21**, 1087, 1953.

CHAPTER 4

Implementing the Computation

The purpose of this chapter is to fill the gap between the formalism of Chapter 3 and the C code in Appendix C. All important aspects of the implementation of the Monte Carlo algorithm, for hard spheres and for solid ^{3}He, will be discussed. This includes the standard Metropolis and the biased Metropolis move and accept/reject procedures, how the instanton (region in imaginary time where the exchange takes place) is shifted when it moves too near to one end of the path, and how the particles in the system are divided into shells that are successively farther from the region where the exchange takes place. Also described are the techniques used to deal with the periodic boundary conditions. Not described in this chapter are the algorithms used to calculate the two-particle density matrices. They will be described in Chapters 5 and 6, for the hard spheres and ^{3}He, respectively.

4.1 The Monte Carlo Update Procedure

There are two phases of the Monte Carlo calculations: the update phase when the particle positions are changed, and the measurement phase when the observable is measured. The update procedures will be described in this section and the measurement procedures in the next section.

The update procedure consists of two parts. First, the new particle positions are generated. Second, the new positions are accepted or rejected according to the procedures given in Section 3.5. Two different update schemes are used. One is the standard Metropolis method (Section 3.5a), and the other is a modification to the standard Metropolis method (Section 3.5b), whereby the new particle positions are biased toward regions where the density matrix is large. The first method is simpler and contains many features of the second and so it will be described first.

4.1a Standard Metropolis Updates

For the standard Metropolis update, only one timeslice of one particle is moved at a time. The move size, Δx, is specified as an input parameter to the program. To calculate a new position for particle m at timeslice K, $x^K_{m,new}$, in terms of its old position, $x^K_{m,old}$, the relation

$$x^K_{m,new} = x^K_{m,old} + (\xi - .5)\Delta x \tag{4.1}$$

is used, where ξ is a vector of independent random numbers on [0,1).

After the move has been generated, the first thing to be done is to check whether particle m's hard core, † in its new position, overlaps with the hard core of any other particle in timeslice K. If there is an overlap, then the move is rejected immediately since an overlap would cause the system to have infinite energy. If there are no hard core overlaps, the change in the action S (from Eqn. (3.26)),

$$S([x];\epsilon) = \sum_{\substack{I=0 \\ n\epsilon=\epsilon}}^{n-1} \sum_{i=1}^{N} \left\{ \frac{m}{2\hbar^2\epsilon}(x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N} U(x_{ij}^I, x_{ij}^{I+1};\epsilon) \right\}, \tag{4.2}$$

is calculated. Fortunately, it is not necessary to calculate all of the terms in the sum of Eqn. (4.2) for the old and the new state because many are the same and will cancel identically when the difference is taken. The outer sum over timeslices

† Even the ^{3}He two particle density matrix has a hardcore cutoff.

is different only when the loop variable, I, is equal to $K-1$ or K. Considering the two inner terms separately, the first term contributes only when particle index, i, is equal to m. The second term contributes only when either i or j is equal to m. Rewriting the action, including only the differing terms gives

$$S^{partial} = \sum_{I=K-1}^{K} \left\{ \frac{m}{2\hbar^2\epsilon}(x_m^I - x_m^{I+1})^2 + \sum_{\substack{i=1 \\ i\neq m}}^{N} U(x_{im}^I, x_{im}^{I+1}; \epsilon) \right\} \tag{4.3}$$

With the above definition, the change in the action, ΔS, is calculated,

$$\Delta S = S_{old}^{partial} - S_{new}^{partial}. \tag{4.4}$$

As given by Metropolis, if $\Delta S \leq 0$ then the move is accepted: x_m^K is set equal to $x_{m,new}^K$. If $\Delta S > 0$ the move is accepted with probability $\exp(-\Delta S)$. This is achieved in practice by generating a random number, ξ, on [0,1) and accepting only if $\xi > \exp(-\Delta S)$. If the move is rejected, the value of x_m^K remains at its original value, $x_{m,old}^K$.

One update of the system consists of this procedure applied to each timeslice (excluding the endpoints) of each particle in the system.

4.1b Biased Metropolis Update

The biased Monte Carlo procedure is similar to the standard procedure described above, but the details differ in both move selection and accept/reject. The move consists of a sequence (or thread) of new positions for several consecutive timeslices of one particle (many particle moves are possible, but are not used for this work). If the move is accepted, the entire thread replaces the positions in the stored lattice.

The new particle positions are generated one at a time, starting with the center timeslice between the two fixed endpoints of the new thread (Fig 14). Once the middle timeslice is generated, it is treated as a fixed endpoint for the two segment into which it divides the thread. In an identical procedure, a new middle position

may be generated for each of those two segments, etc. Following this procedure allows for the threading of

$$1, 3, 7, ..., \sum_{i=0}^{n} 2^i \tag{4.5}$$

particles between fixed endpoints. The reason that a bisection procedure is used instead of threading the particles in one at a time from the beginning of the thread to the end, is that instead of needing to know the two-particle density matrix at N values of ϵ where N is the number of particles being threaded in, only $\ln_2 N$ values of ϵ are required for the bisection procedure. Typically, seven consecutive timeslices of one particle are moved simultaneously, leading to acceptance rates from .7 to .9 for the values of ϵ used in this work. The acceptance rate increases with decreasing ϵ, as the Gaussian approximation to the exact density matrix described below becomes more accurate.

The threading procedure is the same for each timeslice in the thread and so the procedure will be described once here. The new position is selected based on a Gaussian distribution, which depends on the positions of all the particles in the system at the two endpoints and the separation of the endpoints. The primary task is to calculate the mean of the Gaussian distribution (from Eqn. (3.64) with $\tau = \epsilon/2$),

$$\bar{x} = \frac{x + x'}{2} - \frac{\epsilon}{4}\Lambda \nabla U(x, x'; \epsilon). \tag{4.6}$$

When only generating the position for one particle as is done for this work, the gradient of the effective potential, U/ϵ, acts only on the coordinates of the particle being moved. Expressing U in terms of the individual particle coordinates gives

$$U(x, x'; \epsilon) = \sum_{\substack{i=1 \\ j=i+1}}^{N} U(r_{ij}, r'_{ij}; \epsilon). \tag{4.7}$$

If particle m is being moved, Eqn. (4.7) can be rewritten as

$$\bar{x} = \frac{x + x'}{2} - \frac{\epsilon}{4}\Lambda \sum_{j \neq m} \nabla_m U(x_{mj}, x'_{mj}; \epsilon). \tag{4.8}$$

Given $\bar{x}$ and the standard deviation, $\sigma^2 = \epsilon\Lambda/4$ from Eqn. (3.66), a new particle position can be generated according to the corresponding Gaussian distribution. This is accomplished by generating a random number, η, according to the distribution $\exp(-x^2/2)$ (see Appendix A), and making the identification that

$$\frac{\eta^2}{2} = \frac{x_{m,new}^2}{2\sigma^2}, \tag{4.9}$$

so that

$$x_{m,new} = \eta\sigma = \eta\sqrt{\frac{\epsilon\Lambda}{4}}. \tag{4.10}$$

As each of the timeslices is threaded in, the probability of the particle's being in its old position and its probability of being in its new position, according to the Gaussian distributions, is saved so that the acceptance criterion can be modified as prescribed in Eqn. (3.58),

$$\Delta S'_{ji} = \Delta S_{ji} - \ln\left(\frac{T_i}{T_j}\right), \tag{4.11}$$

where the identifications $j \equiv new$ and $i \equiv old$ have been made. It is important to note that the Gaussian distributions for the old and new positions will typically be different since they may depend on different intermediate particle positions between the two fixed endpoints. Since T_i is the above-mentioned Gaussian, the last term in Eqn. (4.11) can be written more directly as

$$\ln\left(\frac{T_i}{T_j}\right) = \frac{-(x_{old} - \bar{x}_{old})^2 + (x_{new} - \bar{x}_{new})^2}{2\sigma^2}. \tag{4.12}$$

The probability of the entire thread of particles being in their respective positions is just the product of T_j for the individual particles, and so the the logarithm of the probability of the the old thread's being generated, divided by that for the new thread positions is just the sum of $\ln(T_i/T_j)$ for each timeslice individually. This sum is accumulated while the thread is being generated and is subtracted from ΔS_{ji} when the new thread is tested for acceptance or rejection. Calculating

the change in the action is the same as for the standard method except that several timeslices must be included in the calculation of $S^{partial}$.

When updating the system, the timeslices that are fixed during one update, are in the center of a thread during a subsequent update, thereby assuring that all of the timeslices are updated.

It should be emphasized that whether the Gaussian is an accurate representation of the density matrix is not very important, since the bias is compensated for in the accept/reject procedure. Doing as well as possible does, however, increase the acceptance rate and helps to move through the space of configurations more rapidly.

4.2 Measuring the Observable

In Section 3.3, the Monte Carlo observable was derived by introducing a length-scaling parameter into the path integral and differentiating with respect to it. The observable is then calculated to be

$$\Phi([x];\beta) = \sum_{I=0}^{n-1}\sum_{i=1}^{N}\left\{\frac{m}{\hbar^2\epsilon}(x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N}(x_{ij}^I\cdot\nabla^I + x_{ij}^{I+1}\cdot\nabla^{I+1})U(x_{ij}^I, x_{ij}^{I+1};\epsilon)\right\}. \tag{4.13}$$

A simpler expression for the last term, which involves the effective potential U/ϵ, may be calculated by noting that U can be written as a function of three scalar variables $r_{ij} \equiv \mid x_{ij} \mid$, $r'_{ij} \equiv \mid x'_{ij} \mid$, and θ, the angle between x_{ij} and x'_{ij}, so that

$$U(x_{ij}, x'_{ij};\epsilon) = U(r_{ij}, r'_{ij}, \theta;\epsilon) \tag{4.14}$$

(mathematicians, forgive me). Timeslice indices have been suppressed and the prime is taken to mean the next timeslice. This representation is actually more natural in the sense that the new variables are the ones used when calculating the ^{3}He density matrix.

Using the new variables with $r_{ij} \rightarrow \lambda r_{ij}$ and $r'_{ij} \rightarrow \lambda r'_{ij}$ and differentiating

with respect to λ gives

$$\frac{\partial U}{\partial \lambda} = r_{ij}\frac{\partial U}{\partial r_{ij}} + r'_{ij}\frac{\partial U}{\partial r'_{ij}}, \tag{4.15}$$

and hence the observable can be expressed as

$$\Phi([x];\beta) = \sum_{I=0}^{n-1}\sum_{i=1}^{N}\left\{\frac{m}{\hbar^2\epsilon}(x_i^I - x_i^{I+1})^2 + \sum_{j=i+1}^{N}\left(r_{ij}^I\frac{\partial}{\partial r_{ij}^I} + r_{ij}^{I+1}\frac{\partial}{\partial r_{ij}^{I+1}}\right)U(x_{ij}^I, x_{ij}^{I+1};\epsilon)\right\}. \tag{4.16}$$

The derivatives are calculated numerically from U, for ^{3}He, using finite differences. The effective potential for hard spheres is given by an analytic form and can therefore be differentiated directly.

4.3 Minimizing Statistical and Systematic Errors

Two different techniques have been used to reduce the statistical errors in the Monte Carlo observable. One restricts the number of particles in the system that are updated and measured. The second restricts measurement of the observable to a region of the path around the instanton, which is smaller than the full-path length. The second technique also reduces systematic errors coming from endpoint effects.

4.3a Scaling the System Size

A natural way to minimize statistical error in the observables is to pick as small a system as possible for which an increase in the system size causes no change in the observable measured. For a BCC lattice in three dimensions, the possible sizes are 16,54,128,... for a periodic system ($2N^3$). Unfortunately, the computational requirements scale as N^2.

To avoid this limitation, when a 54 particle periodic system is used, only the particles whose lattice sites fall within some radius of the region where the exchange takes place are updated. The remaining particles sit at their lattice sites and provide a fixed cage of particles for a background potential. With this technique, much

smaller increments in the effective system size are possible. One other interesting piece of information can be extracted from the way the observable scales with the particle number: a measure of the number of particles that play a significant role in the exchange processes.

4.3b Measuring Subsets of the Path and Shifting the Instanton

Ideally, for this calculation the path length would be taken to infinity ($\beta \rightarrow \infty$). The region of the path where the exchange occurs (referred to as the instanton) corresponds to some subset of the path. Away from the instanton, its effects fall off rapidly. In this ideal system, the Monte Carlo observable could be measured for different numbers of timeslices around the instanton, and measurements for equivalent numbers of timeslices would be made for a nonexchanging path. The Gruneisen parameter could then be calculated for the different numbers of timeslices and the result would converge as that size increased.

To simulate this ideal situation as closely as possible, the instanton's position is measured (as explained below) and regions around the instanton which are shorter than the total path length are measured. In order to implement this idea, it is necessary to prevent the instanton from moving too close to either end of the path. Otherwise, it would not be possible to measure a symmetrical region about the instanton. Not allowing the instanton to move too close to the endpoints also has the advantage of eliminating end effects that are absent in the ideal system. The end effects show up as a deformation of the instanton when it moves to close to one endpoint. Analytically it shows up in K of Eqn. (3.42).

In order to apply this idea, it is necessary to measure the position of the instanton. The method described below requires that there be only one instanton (i.e., the exchange only happens once over the path). This requirement is satisfied easily because for accessible path lengths, the probability of more than one exchange is very small. Typical path lengths are of the order of $\beta = 1.0\ \mathrm{K}^{-1}$. Since exchange energies are of the order 1.0 mK, there will typically be 1 exchange per $10^3\ \mathrm{K}^{-1}$ of path length!

Consider first how one might define an instanton for one of the exchanging particles moving from its initial to its final position along a given path. The first step is to project the particle's positions onto the line joining the initial and the final points. On this axis, 0 is defined as the starting point and 1 is located at the final point. Moving from the starting point along the axis away from the finishing point is the negative direction. The instanton is then defined to be located at the average of two positions, calculated by starting at each of the two endpoints and searching toward the other endpoint for the first time that the particle is within some distance of that other endpoint (Fig 15). The distance used was determined empirically to be .2.

The above procedure gives the instanton for one particle, but can make mistakes in identifying the location of the exchange. Consider, for example, two particle exchange. If the two-particles move far enough in the same direction along the axis joining them, the one moving toward its final destination may have appeared to exchange (Fig 16). The solution to this problem is suggested by the fact that, for such common motions, one particle's projection goes negative while the other's is going positive. Hence, the definition used for the instanton is the average of the individual particle instantons for all of the particles participating in the exchange.

Finally, when the instanton moves within some (empirically determined) distance from either end of the path, it must be shifted back to the center of the path as mentioned above. This is accomplished as follows. Recall the notation used in defining the path,

$$[x] = \{x^0, x^1, x^2, ..., x^{n-1}, x^n\}, \tag{4.17}$$

where x^I is the many-particle position vector for all particles in the system at timeslice I. The endpoints, x^0 and x^n, are fixed, while the rest of the path is allowed to vary. Define the position (in units of the number of timeslices) as T_{inst}, then

$$\Delta T_{inst} \equiv n/2 - T_{inst} \tag{4.18}$$

gives the number of timeslices that the instanton needs to be shifted. If $\Delta T_{inst} < 0$,

then for timeslices 1 to $n - \Delta T_{inst}$, x^I is set equal to $x^{I+\Delta T_{inst}}$. For timeslices $n - \Delta T_{inst} + 1$ to $n - 1$, the particle positions are set equal to the final position, $x^I = x^n$. Fig (17) shows an instanton before and after being shifted. If $\Delta T_{inst} < 0$, then for timeslices 0 to ΔT_{inst}, the particle positions are set equal to the initial position, $x^I = x^0$. For timeslices $\Delta T_{inst} + 1$ to timeslice $n - 1$ x^I is set equal to $x^{I-\Delta T_{inst}}$. After the shift 25 standard Monte Carlo updates are done to rethermalize the lattice.

4.4 Interactions in the Periodic System

The interaction between two particles in the periodic systems considered is taken as the infinite system interaction between one particle and the closest periodic extension (including the particle itself) of the other particle. This can be extended to include further interactions and, in fact, the programs will do this, but that facility has not been used.

This closest separation is found by calculating the closest separation for each dimension, i, separately. Consider that the system size in each dimension is L_i. Then for each dimension, start with

$$\Delta x_i = x_{1,i} - x_{2,i}, \tag{4.19}$$

the difference between the particle positions in the fundamental set of particles for particles 1 and 2 in dimension i. It can be seen that this distance will always be less than half the system size in the given dimension. The prescription below is used for each dimension to calculate the shortest distance.

If $\Delta x_i > L_i/2$ then $\Delta x_i = \Delta x_i - L_i$.

If $\Delta x_i \leq -L_i/2$ then $\Delta x_i = \Delta x_i + L_i$.

4.5 Units and System Parameters

The units used in all of the calculations have energy in units of K and length in units of Å. With these units the quantity Λ is defined by

$$\Lambda \equiv \frac{\hbar^2}{2m} = 16.08 K \text{Å}^2. \tag{4.20}$$

Other units such as time are converted to these units (e.g., time $\rightarrow$ K^{-1} (time/$\hbar$)).

One other issue is important for determining running parameters. Quantities of interest are calculated for certain molar volumes (cm^3/mole), but the program accepts the simple, cubic lattice spacing, D_{cub} ($DCUB$ in the program), as an input parameter. The relationship between the two is calculated by noting that for a cubic piece of the solid with many particles on an edge of side L, there are $(L/D_{cub})^3$ particles in the simple, cubic lattice and therefore $2(L/D_{cub})^3$ particles in the block (A BCC lattice consists of two simple cubic lattices offset from each other by half the simple cubic lattice spacing in each dimension.). The molar volume is then given by

$$v = \frac{L^3}{2(L/D_{cub})^3} N_{Avagadro} = .301 D_{cub}^3. \tag{4.21}$$

CHAPTER 5

Hard Spheres

As a preliminary calculation to explore the idea of calculating the Gruneisen parameter for exchange, we first looked at the problem of hard spheres interacting in three dimensions. The hard-sphere potential between two particles is zero when the particles are separated by greater than a hard core diameter and infinite when their separation is less than or equal to that diameter. Exchange processes for this system should to be similar to those in solid helium because it is the effect of the hard-core part of the potential that is thought to be primarily responsible for suppressing exchange energies far below lattice energies. The hard-sphere system is also a quantum system for which a WKB type calculation of the exchange is not possible. Finally, there is an analytic representation for the two-particle density matrix which, although it is valid in the small time limit, allows a reasonably large mesh in the time direction, as compared with using only the free-particle representation. This makes the system accessible from a computational point of view.

A derivation of the two-particle density matrix, using an image approximation, will be given in Section 5.1. This two-particle density matrix contains an explicit length scale (the hard core-diameter). In Section 5.2, that fact will be used to derive

a Monte Carlo observable with better statistical properties than the more general result from Chapter 3.

5.1 The Two-Particle Density Matrix

The quantity of fundamental importance for this calculation, as for the ^{3}He calculations, is the two-particle density matrix. In the short time limit, an analytic expression for the two-particle density matrix for hard spheres can be derived using an image method that guarantees that the density matrix goes to zero as the hard spheres approach each other.[1]

The Schrodinger equation for the two-particle density matrix is solved by separating the density matrix into the product of center of mass and a relative coordinate terms (see Section 6.2),

$$\rho(x_1, x_2, x_1', x_2'; \epsilon) = \rho_{cm}(R, R'; \epsilon)\rho_{rel}(r, r'; \epsilon) \tag{5.1}$$

where

$$R = \frac{x_1 + x_2}{2}; \qquad M = 2m \tag{5.2}$$

and

$$r = x_1 - x_2; \qquad \mu = \frac{m}{2}. \tag{5.3}$$

M and μ are the effective masses of the center-of-mass system and the relative system, respectively, and m is the particle mass in the original system. The equation for ρ_{cm} is just that of a free particle with mass M,

$$\rho_{cm} \propto e^{-\frac{\hbar^2}{2M\epsilon}(R-R')^2}. \tag{5.4}$$

The short time limit implies that the particles interact only when they are "close" and that the particles will move only a "small amount" in the given time. These distances must be small compared with the important length scale for interparticle interaction: the hard core diameter. When considering the hard core potential as a function of the separation of the surfaces for two spheres that are

very close, the potential is just like that for a point particle in the presence of an infinite hard wall. With a coordinate system for the relative density matrix such that the x-axis is defined by the line joining the centers of the particles at the initial time (Fig 18), define the coordinates u and v by

$$r = (u + D, v), \tag{5.5}$$

where D is the hard-sphere diameter (also the minimum value of $| r |$). With this definition, the problem of finding ρ_{rel} in the short time limit is identical to that of finding the density matrix for a free particle of mass μ with position (u, v) in the presence of a hard wall at $u = 0$. After a time ϵ, the particles should have moved a short enough distance that the curvature of the hard-core potential is not apparent, just as the curvature of the earth isn't apparent while walking on it. The other important factor is that as ϵ is made smaller the interaction falls off much more rapidly, and so the particles must be quite close compared with the particles' radius of curvature before they interact significantly.

The particle, its image and the new position of the particle are depicted in Fig 19 for the system considered as a free-particle in the presense of a hard wall. To insure that the density matrix is zero at $u = 0$, the free-particle density matrix for the image particle hopping from $(-u, v)$ to (u', v') in a time ϵ is subtracted from the free particle density matrix for the particle hopping from (u, v) to (u', v') in a time ϵ,

$$\rho_{rel} \propto e^{-\frac{\mu}{2\hbar^2\epsilon}[(u-u')^2+(v-v')^2]} - e^{-\frac{\mu}{2\hbar^2\epsilon}[(-u-u')^2+(v-v')^2]}. \tag{5.6}$$

Factoring out the first exponential in Eqn. (5.6) and expressing it in terms of the difference coordinates gives

$$\rho_{rel} \propto e^{-\frac{1}{4\Lambda\epsilon}(r-r')^2}\left(1 - e^{-\frac{uu'}{\Lambda\epsilon}}\right), \tag{5.7}$$

where $\Lambda = \hbar^2/m$. Multiplying the first term in Eqn. (5.7) by the expression for ρ_{cm} from Eqn. (5.4) gives the free-particle density matrix for the two particles in the original system. Therefore, expressing Eqn. (5.7) in terms of the particle

positions and dividing by the free-particle density matrix, gives the correction to the free-particle density matrix as

$$\tilde{\rho}_2(x_1, x_2, x_1', x_2'; \epsilon) \equiv \frac{\rho_2}{\rho_1} = 1 - e^{-\frac{1}{\Lambda\epsilon}\left(|x_1 - x_2| - D\right)\left(|x_1' - x_2'| - D\right)}. \tag{5.8}$$

5.2 The Observable

The form for the correction to the two-particle density matrix given in Eqn. (5.8) has the nice feature that it explicitly contains a length scale, D. Below it will be shown how to relate derivatives with respect to D to derivatives with respect to the specific volume. This allows for a more direct calculation of the Gruneisen parameter than is possible for a more general representation of the two-particle density matrix, which cannot be assumed to contain such a length scale explicitly. That case was dealt with in Chapter 3, in which case a length-scaling parameter had to be introduced into the path integral (see Eqn. (3.29)). The observable generated using the explicit length scale has much less statistical noise when averaged over the ensemble of configurations generated by the Monte Carlo procedure. This has the obvious advantage that shorter computer run-times are required to generate results with certain statistical uncertainty.

To relate differentiation with respect to the hard-core diameter, D, to differentiation with respect to the specific volume, v, note that there are actually two length scales in the problem: D, and L the lattice spacing. The exchange rates are expressed as energies and so from dimensional analysis they may be expressed in the form

$$J_P = \frac{\hbar^2}{mL^2} f(\frac{D}{L}). \tag{5.9}$$

If f' is the derivative of f with respect to its argument, the derivative of the logarithm of J_P with respect to L and D are, respectively,

$$\frac{\partial \ln J_P}{\partial L} = -\frac{D}{L^2}\frac{f'}{f} - \frac{2}{L} \tag{5.10}$$

and

$$\frac{\partial \ln J_P}{\partial D} = \frac{1}{L}\frac{f'}{f}. \tag{5.11}$$

The specific volume is proportional to L^3 (L^2 in 2 dimensions), so that $\partial \ln v = 3\partial \ln L$. Using this and eliminating f/f' between Eqns. (5.10) and (5.11) gives the Grunesian parameter in terms of the derivative of J_P with respect to the hard-core diameter,

$$\gamma_P \simeq \frac{\partial \ln J_P}{\partial \ln v} = -\frac{D}{3}\frac{\partial \ln J_P}{\partial D} - 2/3. \tag{5.12}$$

Note that in Eqn. (5.12) if the three is replaced by two the equation is valid in two dimensions. The next problem is to relate the derivative on the right-hand side of Eqn. (5.12) to the path integral in a manner analogous to the derivation of Eqn. (3.48). Taking the derivative of the path integral given in Eqn. (3.28) with respect D gives

$$\frac{\partial \ln Z_P}{\partial D} = \frac{1}{Z_P}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\Phi([x];\beta)e^{-S([x];\beta)}d([x]), \tag{5.13}$$

where using the expression for the action, $S([x];\beta)$, from Eqn. (3.26), Φ is given by

$$\Phi([x];\beta) = -\frac{\partial S}{\partial D} = -\sum_{\substack{I=0\\ n\epsilon=\beta}}^{n-1}\sum_{\substack{i=1\\ j=i+1}}^{NP}\frac{\partial}{\partial D}U(x_{ij}^I, x_{ij}^{I+1};\epsilon). \tag{5.14}$$

Remembering that $U = -\ln\tilde{\rho}_2$, the expression for $\tilde{\rho}_2$ given in Eqn. (5.8), can be used to calculate the derivative of U with respect to D,

$$\frac{\partial}{\partial D}U(x_{ij}^I, x_{ij}^{I+1};\epsilon) = \frac{(x_{ij}^I + x_{ij}^{I+1} - 2D)/\Lambda\epsilon}{1 - e^{(x_{ij}^I - D)(x_{ij}^{I+1} - D))/\Lambda\epsilon}}, \tag{5.15}$$

so that the Monte Carlo observable is given by the expectation value of

$$\Phi([x];\beta) = \sum_{\substack{I=0\\ n\epsilon=\beta}}^{n-1}\sum_{\substack{i=1\\ j=i+1}}^{NP}\frac{(x_{ij}^I + x_{ij}^{I+1} - 2D)/\Lambda\epsilon}{1 - e^{(x_{ij}^I - D)(x_{ij}^{I+1} - D)/\Lambda\epsilon}}. \tag{5.16}$$

An identical argument to that used in Section 3.4b is used to relate the derivative of the logarithm of the exchange rate with respect to the hard-core diameter to the Monte Carlo observables in the large β limit,

$$\frac{\partial \ln J_P}{\partial D} = \langle \Phi \rangle_P - \langle \Phi \rangle_0. \tag{5.17}$$

The final expression for the Grunesian parameter is obtained by combining Eqns. (5.12) and (5.17),

$$\gamma_P = -\frac{D}{3}(\langle \Phi \rangle_P - \langle \Phi \rangle_0) - 2/3. \tag{5.18}$$

5.3 The System

Since this system provides an interesting caricature of the solid helium system with which to try out the ideas outlined in Chapters 3 and 4, a BCC lattice is used at a density of 24 cm^3/mole, to approximate solid ^{3}He near melting as closely as possible. The hard-core diameter has been chosen following the work of Kalos *et al.* [2] to be $.8\sigma$=2.05 Å. They used a hard sphere-potential and fit the hard-core diameter to known ground state energies.

Three-particle threading been used in updating the system for all of the following runs. The runs were done for a 16 particle system. Values of ϵ of .003125 K^{-1} and .00625 K^{-1} were used to verify that the time mesh was close enough to the continuum limit within the statistical uncertainties of the values calculated. At ϵ=.00625 K^{-1}, variation with β was also checked. The results of these tests are shown in Fig 5.1 below. $\gamma_{2,gen}$ is the general expression for the Gruneisen parameter (see Eqn. (4.16)), which must be used in the case of ^{3}He where the effective potential is not expressed analytically, and therefore doesn't have an explicit length scale as does the effective potential for hard spheres. It is reassuring that two very different expressions for the Gruneisen parameter give consistent answers.

$\epsilon(K^{-1})$	$\beta(N_T)\ K^{-1}$	γ_2	$\gamma_{2,gen}$	$N_{T,meas}$
.003125	.4(128)	20.7±1.0	18.0±2.1	81
		20.7±0.7	19.4±1.9	41
.00625	.4(64)	19.8±.8	20.0±1.6	41
	.8(128)	21.7±0.8	18.9±2.7	81
		21.5±0.7	20.6±1.4	41

Table 5.1

Checking the Variation of γ_2 with Respect to ϵ and β.

Exchange Type	γ	v (cm^3/mole)	$N_{T,meas}$
2	21.7±0.8	24	81
	21.5±0.7		41
	23.9±0.7	20	81
	22.9±0.5		41
3	16.8±2.1	24	81
	15.9±0.9		41
4	19.3±1.9	24	81
	15.6±1.3		41

Table 5.2

Gruneisen Constants for Systems with Only the Exchanging Particles Updated.

Note the better error bars for γ_2 as opposed to $\gamma_{2,gen}$, especially for $N_{T,meas}$=81. The column $N_{T,meas}$ gives the range of timeslices over which the observable is measured in the exchanging and the nonexchanging systems (see Section 4.3b). The closest distance which the instanton was allowed to approach the endpoints, before it was shifted back to the center of the path, was 21 for $N_{T,meas}$=41 and 45 for $N_{T,meas}$=81.

The system parameters selected for the runs of the other exchanges were ϵ =

.00625 K^{-1}, and β=.8 K^{-1}. Table 5.2 gives the results of the Gruneisen parameter for two-, three- and four- particle exchange (γ_2, γ_3 and γ_2, respectively) for only the exchanging particles being updated in a system with 16 particles total. One point was run at molar volume, v=20 cm^3/mole and, as can be seen, there is only a 10% variation in the Gruneisen parameter.

It is interesting to note that the magnitudes of the Gruneisen parameters are very similar for the three types of exchange, despite their significantly different geometries. Furthermore, at least for two-particle exchange, the variation with volume is small, although statistically significant.

References

1. J. A. Barker, *Jour. Chem. Phys.*, **70**, 2914, 1979.

2. M. H. Kalos, D. Levesque, and L. Verlet, *Phys. Rev. A*, **9**, 2178, .

CHAPTER 6

Solid ^{3}He

This chapter deals with the calculation of the Gruneisen parameters, γ_P, for two-,three-, and four-particle exchange in solid ^{3}He. First, a description of the ^{3}He two-particle density matrix calculation is given, which includes a description of the way it is stored and retrieved for access by the Monte Carlo program. Readers not interested in the details of this calculation should skip to Section 6.2, which contains a description of the tests and checks that are not part of the Gruneisen parameter calculations, which have been used to verify that the program works properly. In Section 6.3 is a presentation of the data and its analysis, which includes the way in which running parameters were chosen, such as the total path length β, the number of particles to be updated, and the size of the imaginary time mesh ϵ.

6.1 The Two Particle Density Matrix

The calculation described in this section has been implemented in a vectorized Cray program written by David Ceperley, whom I gratefully acknowledge. The algorithm is included in the body of the thesis because it is a crucial ingredient to the

calculation of the Gruneisen parameters and because a comprehensive description is not available elsewhere.

The fundamental element used for all of the ^{3}He calculations is the two-particle density matrix. It is calculated using the Storer matrix squaring method.[1] Before going into the matrix squaring method, a description of how to generate a short-time representation of the two-particle density matrix will be given.

6.1a A Partial Wave Representation

Consider first the Hamiltonian for two particles interacting via the potential $V(|r|)$ separated into center-of-mass and relative parts,

$$H = -\frac{\hbar^2}{2M}\frac{\partial^2}{\partial R^2} - \frac{\hbar^2}{2\mu}\frac{\partial^2}{\partial r^2} + V(|r|) = H_{cm} + H_{rel}, \tag{6.1}$$

where R, r, M, μ are defined as in Eqns. (5.2) and (5.3). Appendix D contains a description of the potential that is used. It is a better approximation to the He-He potential than the standard Lennard-Jones potential. The solution to the Schrodinger equation of Eqn. (3.1) can then be written as

$$\rho(r, R, r', R'; \beta) = \rho_{cm}(R, R'; \beta)\rho_{rel}(r, r'; \beta). \tag{6.2}$$

The center-of-mass density matrix is given by the free-particle solution,

$$\rho_{cm}(R, R'; \beta) = \left(\frac{M}{2\pi\hbar^2\beta}\right)^{\frac{3}{2}} e^{\frac{-M(R-R')^2}{2\hbar^2\beta}} = \left(\frac{1}{\Lambda\pi\beta}\right)^{\frac{3}{2}} e^{\frac{-(R-R')^2}{\Lambda\beta}}, \tag{6.3}$$

where $\Lambda = \hbar^2/m$ (m is the helium atom mass). The piece remaining to be solved is the relative coordinate density matrix,

$$\rho_{rel}(r, r'; \beta) = \langle r | e^{-\beta H_{rel}} | r' \rangle, \tag{6.4}$$

which is the solution to

$$\frac{\partial \rho_{rel}}{\partial \beta} = -H_{rel}\rho_{rel} = +\frac{\hbar^2}{2\mu}\frac{\partial^2 \rho_{rel}}{\partial r^2} - V(|r|)\rho_{rel}. \tag{6.5}$$

The solution to Eqn. (6.5) can be separated into a product of angular and spatial terms. The solutions to the angular part are Legendre polynomials, $P_l(\cos\theta)$, and form the basis of the partial wave expansion, which will be used here. Dropping the rel subscript and explicitly denoting vector quantities with an arrow, the partial wave expansion for the relative density matrix can be written as

$$\rho(\vec{r},\vec{r}';\beta) = \sum_{l=0}^{\infty} \frac{2l+1}{4\pi r r'} \rho_l(r,r';\beta) P_l(\cos\theta), \tag{6.6}$$

where r and r', are the magnitudes of the vectors $\vec{r}$ and $\vec{r}'$ respectively, and θ is the angle between them. Having separated out the angular dependence, the equation for ρ_l is

$$\frac{1}{\Lambda}\frac{\partial^2 \rho_l}{\partial r^2} - \frac{l(l+1)}{\Lambda r^2}\rho_l - V(r)\rho_l = \frac{\partial \rho}{\partial \beta}. \tag{6.7}$$

In the small β (classical) limit, $V(r)$ may be treated as a perturbation Hamiltonian as was done in Section 3.2. The Hamiltonian is $H = H_0 + V$, where

$$H_0 = -\frac{1}{\Lambda}\frac{\partial^2}{\partial r^2} + \frac{l(l+1)}{\Lambda r^2}. \tag{6.8}$$

The solution for ρ_l is then given by

$$\rho_l(r,r';\beta) = \rho_l^{(0)}(r,r';\beta)\rho_l^{(1)}(r,r';\beta) = \rho_l^{(0)}(r,r';\beta)e^{-\frac{\beta}{2}(V(r)+V(r'))}. \tag{6.9}$$

As before, the solution must be found to the Schrodinger equation, which doesn't include the potential. As in Section 3.2 it is just the free-particle density matrix, but in this case the representation is slightly more complicated,

$$\rho_l^{(0)}(r,r';\beta) = (4\Lambda\pi\beta)^{-\frac{3}{2}} 4\pi r r' e^{-\frac{r^2+r'^2}{4\Lambda\beta}} i_l\left(\frac{rr'}{2\Lambda\beta}\right), \tag{6.10}$$

where $i_l(z)$ is a modified spherical Bessel function. It should be noted that if $V(r) = 0$, the summation over l in Eqn. (5.6) produces the free-particle density

matrix for the relative coordinates. Combining Eqns. (5.6,9,10) gives the relative coordinate, two-particle density matrix,

$$\rho(r,r',\theta;\beta) = (4\Lambda\pi\beta)^{-\frac{3}{2}} \sum_{l=0}^{\infty}(2l+1)e^{-\frac{r^2+r'^2}{4\Lambda\beta}} i_l(\frac{rr'}{2\Lambda\beta}) P_l(\cos\theta) e^{-\frac{\beta}{2}(V(r)+V(r'))}, \tag{6.11}$$

in the small β limit.

6.1b Matrix Squaring

Now that a reasonable starting point has been established within the short time limit, the Storer matrix-squaring method [1] will be described. The fundamental relation from which it stems is the exact operator relation

$$e^{-\beta H} = e^{-\frac{\beta}{2}H} e^{-\frac{\beta}{2}H}. \tag{6.12}$$

Expressing it in the coordinate representation for ρ_l gives

$$\rho_l(r,r';\beta) = \int_0^{\infty} \rho_l(r,r'';\frac{\beta}{2})\rho_l(r'',r';\frac{\beta}{2})dr''. \tag{6.13}$$

The form of Eqn. (6.13) even seems to suggest the procedure. Starting with a value of β small enough so that Eqn. (6.11) is sufficiently accurate, generate a complete set of values for ρ_l over some region. The integral of Eqn. (6.13) can be written as a finite sum,

$$[\rho_l(\beta)]_{ij} = \sum_k [\rho_l(\frac{\beta}{2})]_{ik}[\rho_l(\frac{\beta}{2})]_{kj}\Delta r, \tag{6.14}$$

over the discrete set of values for which the density matrix is calculated. Generating the density matrix with twice the value of β is therefore equivalent to squaring a matrix, as can be seen from the form of Eqn. (6.14). This procedure can be applied iteratively to produce successively larger values of β.

Two significant modifications have been made to the basic matrix-squaring

procedure.[2] The first is meant to eliminate the problem of edge effects that come from the finite spatial extent of the matrix. This is essentially the same as a cutoff in the integral and causes some error, especially in the regions near the edge of the matrix. The technique used to avoid this problem consists of setting up the matrix in such a way that it covers a much larger fraction of the space of positions required for the problem at hand, but in a nonlinear way so that the matrix is still a manageable size from a computational point of view. This nonlinear mesh is theoretically arbitrary and could possibly be tailored carefully to the density matrix of interest so that the mesh spacing is very fine where the density matrix is largest and becomes more widely spaced as the density matrix falls off. For this work, a $1/r^2$ mesh was used: The positions for which the density matrix values are stored are given by

$$\frac{1}{r_i^2} = \frac{1}{r_{min}^2}\frac{i}{i_{max}} \tag{6.15}$$

where i runs from 0 to i_{max}, and r_{min} is the hard core cutoff. Since the mesh is nonlinear, generating values of the density matrix between the stored values is trickier than for a uniform mesh. The solution is to interpolate between several gridpoints, by inverting Eqn. (6.15) to find the index corresponding to the given position, keeping the fractional part of the index. The interpolation can then be done using the equally spaced index points. The Lagrange four-point interpolation function found in Abramowitz and Stegun [3] is used to calculate a function $f(i+p)$, where p is the fractional part of the position index. Given that the function is tabulated at equal increments f_j,

$$f(i+p) = A_{-1}(p)f_{i-1} + A_0(p)f_i + A_1(p)f_{i+1} + A_2(p)f_{i+2}. \tag{6.16}$$

Because of the nonlinear mesh, the naive squaring technique will obviously not work and so another method of approximating the integral of Eqn. (6.13) must be found. Since the free particle component of the integrand falls off like $\exp(-x''^2)$ for large x'', it is sensible to use Hermite integration, which is also found in Abramowitz and Stegun. Listed there are the weights, w_i, and the positions, x_i, to evaluate an

integral of the form

$$\int_{-\infty}^{\infty} g(x)dx = \sum_{i=1}^{n} w_i \exp(x_i^2) g(x_i), \tag{6.17}$$

for different values of n. This integral is calculated with $n = 10$, with the zero coordinate of the integral set to $(r + r')/2$, the location where the Gaussian, calculated by combining the two free-particle terms, is maximum.

With these two modifications, the matrix-squaring procedure is applied in a straight-forward way. Values of β used in these calculations vary from .003125 K^{-1} to .1 K^{-1}. Starting with a value of beta of $.1/2^{14}$ K^{-1}, 14 "matrix squarings" are performed and the values are saved between .003125 K^{-1} and .1 K^{-1}. The values stored are actually the values of $U(r_{12}, r'_{12}, \theta) = -\ln \tilde{\rho}_2$ (see Eqn. (3.27)), since the free-particle terms are easy to calculate and U is the quantity that comes into the calculation of the observable. Instead of writing out the matrices for all 32 values of l that are used, an expansion of $U(r_{12}, r'_{12}, \theta; \beta)$ is used to fit the data. The fit is of the form

$$U(r_{12}, r'_{12}, \theta; \beta) = U_0(r) + U_1(r)s + U_2(r)z + U_3(r)sz + U_4(r)s^2 + U_5(r)z^2, \tag{6.18}$$

where

$$r = .5(r_{12} + r'_{12}), \quad s = (\vec{r}_{12} - \vec{r}'_{12})^2 = r_{12}^2 + {r'_{12}}^2 - 2r_{12}r'_{12}cos(\theta), \quad z = (r_{12} - r'_{12})^2. \tag{6.19}$$

$U_0(r)$ is referred to as an endpoint approximation and is calculated from the new values of $\rho_l(r, r'; \beta)$ after a matrix squaring, using

$$\begin{aligned} e^{-U_0(r)} &\equiv \sum_{l=0}^{\infty} \frac{(2l+1)}{2\pi r^2} \rho_l(r, r; \beta) P_l(0) \\ &\simeq \sum_{l=0}^{LMAX-1} \frac{(2l+1)}{2\pi r^2} \rho_l(r, r; \beta) + \sum_{l=LMAX}^{\infty} \frac{(2l+1)}{2\pi r^2} \rho_l^{(0)}(r, r; \beta) e^{-\beta V(r)}. \end{aligned} \tag{6.20}$$

The last term is an approximate correction to account for the higher-order terms that are not computed. It can be calculated using the fact that

$$\sum_{l=0}^{\infty} \frac{(2l+1)}{2\pi r^2} \rho_l^{(0)}(r,r;\beta) = e^{-\frac{1}{4\Lambda\beta}(r-r)^2} = 1, \tag{6.21}$$

and hence the endpoint term is given by

$$U_0(r) = -\ln\left[e^{-\beta V(r)} + \sum_{l=0}^{LMAX-1} \frac{(2l+1)}{2\pi r^2}\left\{\rho(r,r;\beta) - \rho_l^{(0)}(r,r;\beta)\right\}\right] e^{-\beta V(r)}. \tag{6.22}$$

The remaining coefficients are calculated in two ways. First, a 2-variable least-squares fit [4] is done to $\Delta U(r) \equiv U(r) - U_0(r)$ using only U_1 and U_2, leaving the other coefficients set to zero. Second, a 5-variable least-squares fit is done for all 5 coefficients. The value of χ^2 for each of the three fits is calculated. The χ^2 of the 2 parameter fit must be at most .9 times that for the end point fit, or else only the end point fit is used. Similarly, only if the χ^2 for the 5-coefficient fit is at most .9 times that for the 2-coefficient fit are all 5 coefficients used. The nonused coefficients are set to zero.

The limits of integration for the least-squares fit are cut off in both the radial and angular directions. Firstly, the trivial hard core cutoff is respected. This cutoff has been set to 1.55Å in these calculations, well within the Lennard-Jones parameter of 2.6Å. Secondly, looking at Eqn. (6.10), it can be seen that the relative density matrix falls off like $\exp\{-r^2/4\Lambda\beta\}$ for large relative separations. The cutoff in the difference coordinate has been chosen to be $r_{cut}^2 = 18(4\Lambda\beta)$. This same constraint is used to cut off the θ integration, again only for $r > r_{cut}$. Assuming that in a "time" β, particles will travel at most r_{cut}, the largest angular separation possible is given approximately by

$$\theta_{cut} = \sin^{-1}\left(\frac{r_{cut}}{r}\right). \tag{6.23}$$

Two other points need to be made regarding the least-squares fit. If the matrix that must be inverted for the least-squares fit is not invertible, only U_0 is used, and the higher-order coefficients are set to zero.

These coefficients are calculated for values of r corresponding to each of the nonlinear mesh points. Calculating a value for $U(r_{12}, r'_{12}, \theta; \beta)$ is then accomplished using Eqn. (6.18). Each coefficient is calculated by interpolating between the nonlinear mesh points as in Eqn. (6.16). Appendix E gives a listing of the coefficients that were generated using the above described method. The program that produces these values also produces values for $\partial U/\partial\beta$, using a simple finite difference scheme. The use for this quantity, which is essentially an effective potential, is described in Section 6.2c.

6.2 Testing the Program

Three independent tests of the Monte Carlo program have been done in an attempt to assure that it is working properly. The first is a three-dimensional, free-particle calculation, and the second is a harmonic oscillator in one dimension. Both of these were put into the code with very little change in the program structure, and both have analytic answers, which can be compared with the numerically calculated values immediately, to see if everything is working properly. The third test relates the difference in the "expectation value" of the energy in an exchanging lattice and a nonexchanging lattice to the path length β.

6.2a Free Particle

For the following test, a certain path integral involving one free particle in three dimensions was calculated numerically by measuring the expectation value of the Hamiltonian over the path. The value was also calculated analytically, and the results are compared with the calculated value. This quantity is not an expectation value of the energy, but will be referred as to an expectation value for convenience.

The quantity of interest is

$$\langle H \rangle \equiv \frac{\langle x \,|\, H e^{-\beta H} \,|\, x' \rangle}{\langle x \,|\, e^{-\beta H} \,|\, x' \rangle} = -\frac{\partial}{\partial\beta} \ln Z, \tag{6.24}$$

where as usual

$$Z = \langle x \,|\, e^{-\beta H} \,|\, x' \rangle. \tag{6.25}$$

For a free particle,

$$H = \frac{p^2}{2m} = -\frac{\Lambda}{2}\nabla^2, \tag{6.26}$$

and in three dimensions,

$$Z = (2\pi\Lambda\beta)^{-\frac{3}{2}} e^{-\frac{1}{2\Lambda\beta}(x-x')^2}. \tag{6.27}$$

Hence, taking the β derivative of the logarithm of Z gives

$$\langle H \rangle = \frac{3}{2\beta} - \frac{(x-x')^2}{2\Lambda\beta^2}. \tag{6.28}$$

To calculate the observable, Z is written as a path integral as was done in Chapter 3, with the modification that one of the terms contains a p^2,

$$\langle p^2 \rangle = \frac{1}{Z}\int \dots \int \langle x | \dots | x^I \rangle \langle x^I | p^2 e^{-\epsilon H} | x^{I+1} \rangle \langle x^{I+1} | \dots | x' \rangle d[x]. \tag{6.29}$$

Note that p^2 can be inserted in any of the terms, and therefore the observable may be measured over all of the timeslices and the average taken. The remaining task is to evaluate

$$\begin{aligned} \frac{1}{2m}\langle x^I | p^2 e^{-\epsilon H} | x^{I+1} \rangle &= \frac{\Lambda}{2}\nabla_I^2 \langle x^I | e^{-\epsilon H} | x^{I+1} \rangle \\ &= \frac{\Lambda}{2}\nabla_I^2 (2\pi\Lambda\epsilon)^{-\frac{3}{2}} e^{-\frac{1}{2\Lambda\epsilon}(x^{I+1}-x^I)^2} \\ &= \Big(\frac{3}{2\epsilon} - \frac{(x^{I+1}-x^I)^2}{2\Lambda\epsilon^2}\Big)\langle x^I | e^{-\epsilon H} | x^{I+1} \rangle. \end{aligned} \tag{6.30}$$

So finally,

$$\langle H \rangle = \frac{3}{2\epsilon} - \Big\langle \frac{(x^{I+1}-x^I)^2}{2\Lambda\epsilon^2} \Big\rangle, \tag{6.31}$$

where the angle brackets on the right-hand side denote an average over timeslices and configurations.

The above procedure for determining the observable is equivalent to the formal procedure of setting $\beta = n\epsilon$ and considering n fixed so that $\partial\beta = n\partial\epsilon$. The

observable can then be determined by taking the ϵ derivative of the Z, written as a path integral, directly. This technique will be used in the next two sections.

This test was run for the case $x = x'$, and therefore $\langle H \rangle = 3/2\beta$. The path length used was $\beta = .8$ and the results listed in Table 6.1 show that after 2000 updates, the calculated value is consistent with the theoretical value.

Analytical	Numerical
1.88 K	1.81 ± .23 K

Table 6.1
Comparison of analytic and numerically calculated results
for a free particle in three dimensions.

This test result verifies that the biased Monte Carlo threading technique and the accept/reject algorithms work properly.

6.2b Harmonic Oscillator in One Dimension

The free particle test did not check the part of the biasing that involves the interaction potential, and so the following test was selected as one of the few cases where an analytic result can be computed for comparison with the numerically calculated value.

The differential equation for the density matrix of a harmonic oscillator in one dimension is given by

$$-\frac{\partial \rho}{\partial \beta} = \frac{\Lambda}{2}\frac{\partial^2 \rho}{\partial x^2} + \frac{1}{2\Lambda\beta_\omega^2}x^2\rho, \tag{6.32}$$

where $\beta_\omega^2 = 1/(\hbar\omega)^2$ (ω is the characteristic frequency of the hamonic oscillator). The solution to this differential equation,[5] with the initial condition

$$\rho(x, x'; 0) = \delta(x - x'), \tag{6.33}$$

is

$$\rho(x,x';\beta) =$$

$$(2\pi\Lambda\beta_\omega \sinh(\frac{\beta}{\beta_\omega}))^{-\frac{1}{2}} \exp\Big\{-\frac{1}{2\Lambda\beta_\omega \sinh(\frac{\beta}{\beta_\omega})}[(x+x')^2 \cosh(\frac{\beta}{\beta_\omega}) - 2xx']\Big\}. \tag{6.34}$$

Simplifying to the case $x = x'$, and using Eqn. (6.24) gives

$$\langle H \rangle = \frac{\coth(\frac{\beta}{\beta_\omega})}{2\beta_\omega}. \tag{6.35}$$

A small ϵ form was used for $U(x,x';\epsilon)$ (see Eqn. (3.23)),

$$U(x,x';\epsilon) = \frac{\epsilon}{2}\big(V(x) + V(x')\big) = \frac{\epsilon}{4\Lambda\beta_\omega^2}(x^2 + x'^2). \tag{6.36}$$

As explained near the end of Section 6.1b, an ϵ derivative will be used to calculate the observable here. For the purposes of this derivative it is important to include the normalization terms, which have usually been omitted, in the path integral, so that

$$Z = N \int \dots \int e^{-S([x];\beta)} d[x], \tag{6.37}$$

with

$$N = \Big(\frac{1}{2\pi\Lambda\epsilon}\Big)^{\frac{DNn}{2}}, \tag{6.38}$$

where D is the spatial dimension of the system, N is the number of particles and n is the number of timeslices (for this case $N = D = 1$, and n=128.). Using action, S, given in Eqn. (3.26), the observable is

$$\langle H \rangle = \frac{1}{2\epsilon} - \Big\langle \frac{(x^{I+1} - x^I)^2}{2\Lambda\epsilon^2} \Big\rangle + \Big\langle \frac{1}{4\Lambda\beta_\omega^2}\big[(x^{I+1})^2 + (x^I)^2\big] \Big\rangle. \tag{6.39}$$

Again, the timeslice and configuration averages are implied by the angle brackets on the right-hand side. Identify the first two terms in the observable as the kinetic energy and the last term as the potential energy.

When β is of the same order as β_ω, the observable is essentially equal to the free-particle result, $\langle H \rangle = 1/2\beta$. Since it is the effect of the potential that needs to be tested, the values of β and β_ω were chosen so that $\beta \simeq 10\beta_\omega$, for which case the potential makes a significant contribution to the observable. The parameters used were $\beta = .8\text{K}^{-1}$, $\beta_\omega = .125\text{K}^{-1}$. The results shown below in Table 6.2 again show consistency between the analytical and the numerical results.

Analytical	Numerical KE + PE
4.0 K	(80-77.8±.2) + 1.71±.02 K 3.9±.2 K

Table 6.2

Comparison of Analytic and Numerically Calculated Results for a One-Dimensional Harmonic Oscillator.

6.2c Energy Check

Ceperley[2] has suggested that the following check, which can be done for the lattices from which the Gruneisen parameters are calculated. Starting with Eqn. (3.43),

$$\frac{Z_p}{Z_0} = \tanh(\beta J_P + K) \simeq \beta J_P + K, \tag{6.40}$$

the derivative of the logarithm of this expression gives

$$\frac{\partial \ln Z_P}{\partial \beta} - \frac{\partial \ln Z_0}{\partial \beta} = \frac{1}{\beta + K/J_P}. \tag{6.41}$$

The expressions on the left-hand side of Eqn. (6.41) are just the negative of the expectation values of H in the exchanging and the nonexchanging configurations as shown in the last section. The corresponding observable is

$$\begin{aligned} \langle H \rangle &= \left\{ \frac{3N}{2\epsilon} - \left\langle \frac{(x^{I+1} - x^I)^2}{2\Lambda\epsilon^2} \right\rangle \right\} + \left\langle \frac{\partial U}{\partial \epsilon} \right\rangle \\ &\equiv E + V, \end{aligned} \tag{6.42}$$

where again the average over timeslices and configurations is implied by the angle brackets on the right hand-side. The term in braces is the kinetic contribution to the energy (defined to be E) and the last term is the potential contribution (defined to be V). The ϵ derivatives of U are produced by the same program that produces U itself, as mentioned at the end of Section 6.1.

Finally, the relation to be used to check the lattices is

$$\begin{aligned} \frac{1}{\beta} &= -\langle H \rangle_P + \langle H \rangle_0 \\ &\equiv \Delta E - \Delta V. \end{aligned} \tag{6.43}$$

6.3 Determination of ϵ and β

The approach taken in the determination of the path mesh ϵ, and the total path length β, was to run on the smallest system size so that a systematic check could be done across both parameters. The Gruneisen parameter for two-particle exchange, γ_2 (γ_3 and γ_4 will refer to the Gruneisen parameter for three- and four-particle exchange), was measured for a system consisting of two particles being updated in a system of 54 particles (see Section 4.3a). Variation of γ_2 with β was also checked for 8 particles being updated in a system of 54 particles, 8/54 using the notation M/S to denote M particles being updated in a system of S particles, because the constraint of having the nearest neighbors of the exchanging particles fixed could change the shape and width of the instanton.

Three-particle threading (see Section 4.1b) has been used for the data in Table 6.3, but seven-particle threading has been used in the rest of the calculations. The acceptance rate for three-particle threading is approximately .9, and for seven-particle threading it is approximately .7.

The 2/54 system has been run for values of the imaginary time step $\epsilon = .003125$, .00625, .0125 and .025. Consistency is also checked with respect to total path length β. The unfamiliar parameters listed in the table below are $N_{T,meas}$ and N_{shift}. $N_{T,meas}$ is the range of timeslices over which the Monte Carlo observable

was measured (see Section 4.3b). N_{shift} gives the closest distance from the ends of the path (in units of timeslices) which the instantons were allowed to move before being shifted back to the center of the path (see Section 4.3b).

$\epsilon(K^{-1})$	$\beta(N_T)\ K^{-1}$	γ_2	$N_{T,meas}$	N_{shift}
.003125	.2(64)	20.2±1.0	41	21
	.4(128)	24.2±1.1	81	45
		23.0±.8	41	
.00625	.4(64)	23.6±1.1	41	21
	.8(128)	24.2±1.1	81	45
		22.8±1.2	41	
.0125	.8(64)	19.9±.8	41	21
.025	1.6(64)	14.1±1.3	41	21

Table 6.3

Checking the Variation of γ_2 with Respect to ϵ for 2/54.

Although β is not constant over the range of ϵ, the values of γ_2 compare at the same value of β pairwise for the first three values of β, and for $\epsilon = .025\ K^{-1}$, the β is larger than necessary (which will cause no error). Fig. 20 shows a graph of γ_2 for the data of Table 6.3, from which it can be seen that at $\epsilon = .00625\ K^{-1}$, γ_2 has converged to within the error bars. A fit line has been omitted intentionally from the graph of this data, because it is likely that at least some of the variation in γ_2 is from errors in $U = -\ln \tilde{\rho}_2$.

The next question to be answered is whether at $\epsilon = .00625\ K^{-1}$, the total number of timeslices, N_T, should be 64 or 128. From the data in Table 6.3 it would seem that N_T=64 would be adequate, but when the number of particles being updated is increased to 8, the variation of γ_2 with N_T indicates (as shown in Table 6.4) that N_T=128 is a better choice.

$\beta(N_T)$ K^{-1}	γ_2	$N_{T,meas}$	N_{shift}
.4(64)	14.3±1.3	41	21
.8(128)	17.8±1.5	81	45
	17.7±1.1	41	
1.6(256)	16.3±3.1	121	70
	15.1±2.3	81	
	14.7±1.5	41	

Table 6.4
Two particle exchange. Variation with β for 8/54.

The fact that for N_T=64, the value of γ_2 is too low indicates that the instanton's shape is being affected when it moves within 21 timeslices of the ends of the path. This can be seen to be the case, since for $\beta = .8$, a measurement over 41 timeslices gives an answer consistent with the rest of the data. It is interesting that even though a range of 41 timeslices is enough to measure the instanton, a larger range must feel its effect (i.e., the values of γ_2 agree for $\beta = .8$ for $N_{T,meas}$= 41 and 81, but the $\beta = .4$ value disagrees).

After determining the run parameters to be $\epsilon = .00625$ K^{-1} and $\beta = .8$, the main results of this work were generated.

6.4 Results

With the exception of one data point, which were run on a Cray XMP, all of the following Gruneisen parameter data was run on a 500 node NCUBE parallel computer. Typically, 1/2 to 3/4 of the computer was dedicated to this computation from 10pm to 9am, and 1/4 to 1/2 during the rest of the day for a period of three weeks. 128 nodes of the NCUBE is roughly equivalent to one head of a Cray XMP for this (nonvectorized) code, and so the total runtime is equivalent to approximately 900 hours on a Cray XMP. The program is 80 percent efficient, running on 64 nodes of the NCUBE for a 54 particle system with N_T=128 and seven particle threading. Typically, 2-6 separate jobs were run on different portions of the

nodes simultaneously, because the efficiency decreases rapidly when the number of nodes is increased.

The largest amount of data was generated for molar volume, v=24 cm^3/mole, which is near the solid/liquid phase transition, where exchange energies should be largest. At this volume the variation of γ with system size was checked most closely, and the results there (along with the very large statistical uncertainty for the larger systems) determined what system sizes were chosen to run at v=22 cm^3/mole. The lack of variation in the Gruneisen parameters between 24 cm^3/mole and 22 cm^3/mole prompted us to check one exchange at 20 cm^3/mole, but as seen in Fig. 21 it shows no significant deviation from the rest of the data.

The Gruneisen parameter data are tabulated below in Tables 6.5,6,7 for each type of exchange, separately. The graphical presentation of the data in Figs. 21,22,23 includes only the $N_{T,meas}$=81 data because it should be the most accurate, including larger deviations of the instanton width than for the $N_{T,meas}$=41 measurements. Fig. 25 shows all of the data for $N_{T,meas}$=81, including the smallest system data, and Fig. 24 shows the same data minus the small system points.

$N_{updated}$	γ_2	v (cm^3/mole)	$N_{T,meas}$
2	24.3±1.1	24	81
	22.8±1.2		41
8	17.8±1.5	24	81
	17.7±1.1		41
	14.9±1.8	22	81
	10.5±2.2		41
	15.3±1.6	20	81
	16.3±1.3		41
14	15.5±2.3	24	81
	15.9±1.5		41
	17.3±3.3	22	81
	14.3±2.4		41
24	13.0±3.1	24	81
	13.8±2.1		41
54	20.2±7.7	24	81
	11.1±5.5		41

Table 6.5

Two-particle exchange: $\epsilon = .00625$, β=.8.

$N_{updated}$	γ_3	v (cm^3/mole)	$N_{T,meas}$
3	19.3±1.7	24	81
	13.9±1.8		41
10	16.2±2.8	24	81
	14.8±2.0		41
	16.1±3.1	22	81
	18.0±1.8		41
16	14.4±2.9	24	81
	14.3±2.1		41
22	16.6±3.1	22	81
	16.8±2.2		41
26	20.3±4.8	24	81
	16.3±5.1		41

Table 6.6

Three-particle exchange: $\epsilon = .00625$, β=.8.

$N_{updated}$	γ_4	v (cm^3/mole)	$N_{T,meas}$
4	31.1±2.4	24	81
	20.7±1.4		41
14	13.8±2.9	24	81
	14.3±2.9		41
	12.0±2.2	22	81
	13.4±2.2		41
24	13.3±3.1	24	81
	10.8±2.5		41
	18.0±4.5	22	81
	13.5±3.3		41

Table 6.7

Four-particle exchange: $\epsilon = .00625$, β=.8.

The data for the smallest system sizes have been omitted from the graphs that present the data for each type of exchange. The reason, as alluded to above, is that the smallest systems for each type of exchange differ qualitatively from all of the bigger systems, in that the group of exchanging particles interacts directly with the fixed cage of particles. In all larger systems, the particles that interact with the fixed cage are nominally doing the same thing for the exchanging and the nonexchanging configurations. There will be second-order differences becausse of the exchange, but that is why scaling with system size is still checked. For a large enough number of particles being updated, the particles interacting with the fixed cage will have an identical environment for both the exchanging and nonexchanging configurations. The data for two- and three- particle exchange seem to indicate that the second system size (8 and 10 particles updated, respectively) are already very close to that limit (see Figs. 22 and 23). The values for four-particle exchange (Fig. 24) are consistent with the same conclusion, but it is not so clear because of the lack of good large-system-size data. What is clear is that over the range of molar volume from 22 cm^2/mole to 24 cm^3/mole, as seen experimentally, the different types of exchange all vary with nearly the same Gruneisen parameters.

Two final issues will be addressed here before the presentation of our primary results. The first is the estimation of error bars. A very simple procedure was used. The data were grouped into sets of 50 values, and the average was computed for each set. These averages were treated as statistically independent values, for which the average and the standard deviation of the mean was calculated. This procedure is consistent with the more subjective procedure of calculating the autocorrelation function,

$$C_i = \frac{N}{N-i} \frac{\sum_{n=1}^{N-i} (x_n - \bar{x})(x_{n+i} - \bar{x})}{\sum_{n=1}^{N} (x_n - \bar{x})^2} \tag{6.44}$$

(where the prefactor has been set to one since $N \gg i$), and picking the point, i_d, where the data are decorrelated (which is the subjective part). Typically, to make

this comparison, the data were considered decorrelated when the autocorrelation function had fallen from its maximum value of 1 to .1. When i_d is determined, the standard deviation of the mean for the whole data set is multiplied by $\sqrt{i_d}$ to correct for the fact that only $1/i_d$ of the measurements are independent.

The second issue is the "energy" check of Section 6.2c. Table 6.8 shows the results of applying the check which, from the discussion in Section 6.2, is expected to be 1/.8=1.25 K. The difference between the two observables in Eqn. (6.43) has been rewritten for the purpose of display in Eqn. (6.45) below as the difference between the kinetic and the potential parts of the observable separately, so

$$\frac{1}{\beta} = \Delta E - \Delta V. \tag{6.45}$$

Exchange Type	ΔE (K)	ΔV (K)	$\Delta E - \Delta V$ (K)
2	17.4±1.6	12.7±0.4	4.6±1.7
	11.5±2.3	7.8±0.4	3.6±2.3
	6.1±2.6	5.8±1.0	.3±2.8
	13.3±3.0	5.7±1.0	7.6±3.2
	9.5±11.0	8.9±4.0	.5±12.0
3	20.3±3.0	13.5±1.0	6.8±3.2
	12.5±3.9	7.6±0.8	5.0±4.1
	10.6±4.3	6.4±1.0	4.1±4.5

Table 6.8

Energy Check for Two- and Three-Particle Exchange.

The fact that the results don't match the expected value is actually not surprising in retrospect, because of the shifting procedure used to remove end effects, and the related fact that measurements are performed over only fractions of the path. It might be possible to define an effective β, but it is not clear *a priori* if it

should be $\beta = \infty$ or the value of β corresponding to the range of timeslices over which the observable is measured, or some other value.

Given the apparent constancy of the Gruneisen parameter, the final quoted values for γ_2, γ_3 and γ_4 are calculated as weighted averages over the data for all the system sizes and densities shown in Figs. 21, 22, 23 (this excludes values for the smallest systems). The values given in Table 6.9 below are calculated using the numbers in Tables 6.5,6,7 for $N_{T,meas}$=81. The weighted average $\bar{x}$ is given in terms of the individual measurements x_i and their errors σ_i by

$$\bar{x} = \Big[\sum_{i=1}^{N} \frac{x_i}{\sigma_i} \pm \sqrt{N}\Big] \Big/ \sum_{i=1}^{N} \frac{1}{\sigma_i}. \tag{6.46}$$

The error for this natural definition of $\bar{x}$ was calculated from

$$\sigma_{\bar{x}}^2 = \sum_{i=1}^{N} \Big(\frac{\partial \bar{x}}{\partial x_i}\Big)^2 \sigma_{x_i}^2. \tag{6.47}$$

γ_2	γ_3	γ_4
15.9±0.8	16.4±1.4	13.8±1.5

Table 6.9

Gruneisen Constants for Two-, Three- and Four-Particle Exchange.

References

1. A. D. Klemm and R. G. Storer, *Austr. Jour. Phys.*, **26**, 43, 1073.
2. David Ceperley, private communication.
3. Milton Abramowitz and Irene A. Stegun, *Handbook of Mathematical Functions*, Dover , New York, NY, 1972.
4. William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes*, Cambridge Univ. Press, Cambridge, England, 1988.
5. R. P. Feynman, *Statistical Mechanics: A Set of Lectures*, Benjamin/Cummings, Reading, Mass., 1972.

CHAPTER 7

Exchange in Two-Dimensional ^{3}He: A Test

The following chapter is a check to see whether a theoretical calculation involving exchange in two dimensions in a high-density approximation, and extended to lower density, has properly calculated the Gruneisen parameter for three-particle exchange in a two-dimension system with a triangular lattice. This result has been used as evidence that planar three-particle exchange is the mechanism by which ferromagnetic ordering takes place in a physical system involving ^{3}He deposited on a surface of exfoliated graphite (grafoil).[1,2]

The original interest in the two-dimensional ^{3}He system came from the experimental observation of surface-induced ferromagnetism in confined liquid ^{3}He.[3] In a later experiment by Franco *et al.* on grafoil,[4] coverages from 1 to 5 monolayers were studied. They observed that at 2.2 layers, ferromagnetism appears suddenly and reaches a maximum at 2.5 layers. This is thought to imply that layer two solidifies partially at a coverage of 2.2 layers and completes solidification at 2.5 layers.

These results indicate that there is some sort of ferromagnetic interaction involving the solid phase in the second layer. One possibility is three-particle exchange within the second layer (three-particle exchange is favored in a two-dimensional, triangular geometry (Fig 27)). Seemingly, at least as likely, is exchange between the

partially filled third layer and the solid second layer. Roger [5] has made a high-density calculation of the three-particle exchange, for which the particles are constrained to move in two dimensions. He used a two-dimensional, triangular geometry and a WKB type expression to calculate the exchange energy in a high-density approximation, which he optimistically[6] claims should be valid up to a lattice spacing of 3.2 Å. He compares these calculations with an extrapolation of exchange rates measured by Richards using NMR[7] for the first monolayer with nearest-neighbor spacings from 3.25Å to 3.6 Å. These values were extrapolated to a lattice spacing of 3.85Å corresponding to a valid density for the second monolayer, where experiment and theory remarkably agree (Fig 26).

In the next section, the WKB calculation of Roger will be briefly outlined and related to the discussion of Chapter 3. The results of that calculation and a calculation made using the methods developed in this work are compared in Section 7.2.

7.1 The Theoretical Calculation of Roger

The following is a brief outline of Roger's calculation,[5] which involves a WKB expression for the exchange energy, which should be valid at high densities where the potential energy predominates with respect to the kinetic energy. Writing the path integral as in Chapter 3,

$$Z_P = \langle x \,|\, e^{-\beta H} \,|\, x' \rangle = C \int d[x] e^{-S/\hbar}, \tag{7.1}$$

where the action is taken in the continuum limit ($\epsilon = \tau/\hbar \to 0$),

$$S = \int_0^T d\tau \Big[\frac{m}{2} \Big(\frac{dx}{dt} \Big)^2 + V\big(x(t)\big) \Big]. \tag{7.2}$$

An explicit factor of $1/\hbar$ has been factored out, so $T = \hbar\beta$ has units of time. Consider an expansion in energy eigenstates as in Eqn. (3.36), where again only

the two-state system $\Psi_\pm$ with eigenvalues $E_\pm = E_0 \mp J_P$ are considered. In this case, the difference between the overlap of the position eigenstates with the odd and even wavefunctions is neglected,

$$\langle \Psi_+ | x \rangle = \langle \Psi_+ | x' \rangle \simeq \langle \Psi_- | x \rangle = -\langle \Psi_- | x' \rangle \equiv \phi_0, \tag{7.3}$$

(Remember that $x' = Px$.) Then

$$\begin{aligned} Z_P &\simeq \phi_0^2 (e^{-E_+ T/\hbar} - e^{-E_- T/\hbar}) \\ &= 2\phi_0^2 e^{-E_0 T/\hbar} \sinh(J_P T/\hbar). \end{aligned} \tag{7.4}$$

With the conditions

$$\frac{\hbar}{E_0} \ll T \ll \frac{\hbar}{J_P}, \tag{7.5}$$

Eqn. (7.4) gives

$$J_P = \frac{\hbar C}{2T\phi_0^2} e^{-E_0 T/\hbar} \int d[x] e^{-S/\hbar} \tag{7.6}$$

For the semiclassical limit (valid for small $\hbar^2/m$), the integral of Eqn. (7.4) is dominated by the classical path, defined by the relation $\delta S = 0$. Minimizing the action defines the classical (highest probability) path and leads to the usual WKB form,

$$J_P \sim \exp\Big\{ -\frac{1}{\hbar} \int_{x'}^{x} dx [2m(V - E_0)]^{\frac{1}{2}} \Big\}. \tag{7.7}$$

The potential along the exchange path is estimated by a one dimensional sinusoidal potential, which is essentially the first fourier component of the potential along the exchange path,

$$V(x) - E_0 = \frac{1}{2} V_M \Big[1 - \cos\Big(\frac{\pi x}{L} \Big) \Big], \tag{7.8}$$

where L is the half length of the exchanging path, and V_M is the value of the potential for the classical path at the mid-point (and presumably the maximum).

The prefactor in Eqn. (7.6) is estimated by calculating the one dimensional wavefunction and using an expression for the exchange energy in terms of a flux integral over the plane, Σ, orthogonal to the exchange path,[8]

$$J_P = \frac{E_- - E_+}{2} = \frac{\hbar^2}{2m} \frac{\int_\Sigma (\Psi_- \nabla \Psi_+ - \Psi_+ \nabla \Psi_-) dS}{\int |\Psi_+ \Psi_-| dv}. \tag{7.9}$$

The potential used corresponds to the repulsive part of the Lennard-Jones potential,

$$V = \sum_{i<j} 4\epsilon |\frac{\sigma}{r_{ij}}|^{12}, \tag{7.10}$$

where $\sigma = 2.65$Å is slightly larger than the standard value. The exponent of Eqn. (7.7) is estimated by minimizing the classical action for the exchange path. For the two-dimensional, triangular lattice with the potential of Eqn. (7.10) this leads to the following expression for the three particle exchange energy J_3,

$$J_3 \simeq 73\left(\frac{\sigma}{a}\right)^{9.5} \exp\left\{-51.7\left(\frac{\sigma}{a}\right)^5\right\}\text{K}, \tag{7.11}$$

where a is the lattice spacing.

Eqn. (7.11) is not an accurate estimate of J_3 because the prefactors have been calculated very roughly, but the Gruneisen parameter for the exchange should be given accurately within the validity of the high-density approximation. It is given by

$$\gamma_3 = \frac{\partial \ln J_3}{\partial \ln v} = \frac{a}{2} \frac{\partial \ln J_3}{\partial a} = 129.25\left(\frac{\sigma}{a}\right)^5 - 4.75. \tag{7.18}$$

7.2 Results

The calculation of the Gruneisen parameter for three-particle exchange was made in the same way as for those calculated in Chapter 6 (using the same program), with the dimension set to be 2 and with a two-dimensional, triangular lattice. The full ^{3}He potential described in Appendix D is used. The system was 18 particles

($3^2 2$) of which 6 were typically updated. For one value of the lattice spacing, a, all 18 particles were updated to verify that 6 particles was an adequate number. The results, along with the predictions of Roger, and the values from experiment are presented below in Table 7.1. The fit line drawn through the experimental data (Fig 26) is, in fact, an exponential, not a power law fit, but the range of a is small enough that the variation of the Gruneisen parameter is not significant compared with the variation of the data, especially at the larger values of a. The slope of the experimental line is $\partial \ln J/\partial a \simeq 14.0$ Å^{-1}. The Gruneisen parameters were calculated using

$$\gamma = \frac{\partial \ln J}{\partial \ln v} = \frac{a}{2}\frac{\partial \ln J}{\partial \ln a}. \tag{7.19}$$

The Monte Carlo calculations were done only accurately enough to show the incorrect nature of Roger's calculation.

a (Å)/γ	Roger	Monte Carlo	Experiment
3.2	42	50±5 (18 parts. 49±10)	23±2
3.4	32	55±4	24±2
3.6	23	49±4	25±2

Table 7.1

Comparison of Gruneisen Parameters.

7.3 Conclusions

Two interesting conclusions can be drawn from these results. First is that the high-density approximation of Roger is not very good at low densities, although as the density increases (a decreases), his value gets closer to the Monte Carlo result. Secondly, it seems that the physically intuitive idea, that the three particle

exchanges aren't well constrained to lie in a plane, is probably correct. This also leaves open the possibility that interlayer exchange is important.

Finally, this Monte Carlo calculation could be modified to check the effect of out-of-plane exchange by working in three dimensions with the addition of a substrate potential varying in the direction normal to the plane. The effect of interlayer exchange could also be measured, although exchange with the liquid layer would necessarily involve some sort of average over different starting points for the particles whose path begins in the liquid.

References

1. Osheroff, preprint, Oct. 1987
2. M. Roger and J. M. Delrieu, *Japanese Jour. of Appl. Phys.*, **26**, 267, 1987.
3. A. I. Ahonen *et al.*, *Jour. Phys. C*, **9**, 1665, 1976.
4. H. Franco, R. E. Rapp, and H. Godfrin, *Phys. Rev. Lett.*, **57**, 1161, 1986.
5. Roger, *Phys. Rev. B*, **30**, 6432, Dec. 1984.
6. M. C. Cross and Daniel S. Fisher, *Rev. Mod. Phys.*, **57**, 881, Oct. 1985.
7. M. Richards, *Phase Trans. in Surface Films*, Plenum, NY, , 1980.
8. R. A. Guyer and A. K. Mahan, *Phys. Rev. A*, **7**, 1105, 1973.

CHAPTER 8

Conclusion

The central results of this work are summarized in Table 6.9: $\gamma_2 = 15.9 \pm .8$, $\gamma_3 = 16.4 \pm 1.4$ and $\gamma_4 = 13.8 \pm 1.5$. These results are systematically low as compared with the most recent values of Ceperley and Jacucci,[1] which are $\gamma_2 = 19.0 \pm .4$, $\gamma_3 = 19.9 \pm .4$ and $\gamma_4 = 17.7 \pm .4$. They ran on systems consisting of 54 and 128 particles, and so it is possible that the difference is a finite-size effect. The small systems give a greater contribution to our weighted averages because of their smaller error bars. The average values for γ_2 and γ_3 for the largest system sizes are both near 20, but the bars are too large to make any reasonable conclusion from that fact.

Experimentally measured Gruneisen parameters are consistent with $\gamma = 18 \pm 2$, which is consistent with both sets of calculated values, although Ceperley and Jacucci's are admittedly a bit closer. It is also important to point out that the physical observables are combinations of all possible types of exchange and don't say anything about the individual Gruneisen parameters. It is possible, for example, that some higher exchange is having an effect.

To improve our results at the larger system sizes could resolve the issue of consistency, but would require significantly more computing time. Estimates of the

amount of time needed will be given in Section 8.2, but first a brief discussion of Ceperley and Jacucci's clever method for calculating the exchange energies will be given.

8.1 The Method of Ceperley and Jacucci

Starting with Eqn. (3.42),

$$\frac{Z_P}{Z_0} = \tanh(\beta J_P + K), \tag{8.1}$$

the idea[2] is to consider Monte Carlo moves that take configurations between the exchanging and the nonexchanging systems. This is done by clipping out a segment of timeslices somewhere in the path of the particles, which participate in the exchange (for either the exchanging or nonexchanging configuration). The different ends of the paths that are left "dangling" are then reidentified, and using the techniques described in Section 4.1b (with a more accurate Gaussian representation for the density matrix), a new sequence of particle positions is threaded between the dangling ends, which have been reidentified as belonging to the same particle. In this way, a path with an exchanging geometry can be mapped into a path with a nonexchanging geometry, and a path with a nonexchanging geometry can be mapped into a path with an exchanging geometry.

With a Monte Carlo move that takes the system between configurations, the ratio of the density matrix for the exchanging system divided by the density matrix for the nonexchanging configuration is just the ratio of the number of Monte Carlo steps the system spends in the exchanging system divided by the number of steps it spends in the nonexchanging system. In fact, the moves need not be made from one system to the other. It is sufficient to run a system of each type and measure the rates at which each one hops to the other. If f_P is the fraction of the Monte Carlo hops from the exchanging configuration to the nonexchanging configuration, which are successful, and f_0 is the successful fraction going in the other direction,

$$\frac{Z_P}{Z_0} = \frac{f_0}{f_P}. \tag{8.3}$$

One limitation of this method is that as the exchange energies decrease, the overlap between the Gaussian probability distributions, used to map between the two systems, goes to zero. This reflects the fact that for smaller exchange rates, more particles participate in the exchange and hence reconnecting only the exchanging particles becomes a poorer way to move from a nonexchanging to an exchanging configuration. On the other hand, the methods developed in this work will do better in the high-density regions where the Gruneisen parameters are larger (and hence the relative error is less). Larger systems may have to be used, but the observable grows only linearly with computing time. Hence, where their method is infeasible, our method will still work.

8.2 Estimation of Computer Runtimes for Larger Systems

To estimate the computer run time required to get values of γ with error bars on the order of one for 54 and 128 particle systems, consider the data for the Monte Carlo observable measured in the nonexchanging state for several system sizes as listed in Table 8.1.

System Size	Monte Carlo Observable
2	590.0±1.6
8	2247.0±4.6
14	3852.0±6.8
54	12990.0±17.2

Table 8.1

The Monte Carlo Observable for the Nonexchanging State.

The error bars can be seen to scale approximately with system size (actually, there is a slight decrease proportionally). The problem is that the Gruneisen parameters, which are calculated as the difference of two such numbers, are essentially fixed, while the errors increase with system size. To get an error of approximately 3

in the Monte Carlo observable (hence 1 in γ; see Eqn. (3.45)) requires a reduction of $17.2/3 \simeq 6$ in the error. Since the error scales with the total number of measurements, N, as $1/\sqrt{N}$, this reduction in error would require an increase in the runtime by a factor of 36. The points quoted in Table 8.1 are all for 750 measurements. The 54 particle system (for the exchanging and nonexchanging systems) required approximately 48 hours of runtime on 250 nodes of the NCUBE parallel computer, which is roughly 96 hours of nonvectorized Cray XMP cpu time. This implies that 1700 hours would be required, running on half of the NCUBE (50 days). Doubling that time would give the values for both three and four particle exchange, since the nonexchanging configuration would not have to be run again. Running on a 128 particle system and keeping the target error bars fixed would increase the computing requirements in two ways. Scaling the statistical error with system size, a factor of $(128/54)^2 = 5.8$ would be required to maintain the error bars calculated for the 54 particle system. Secondly, a factor of $128^2/54^2 = 5.6$ comes from the increase in the time required to calculate the interactions between all the particles in the system during update and measurement phases of the calculation. The total factor is therefore a factor of 32. Given these large estimates, more computing power is required to make calculations involving these larger systems feasible.

One way to achieve a speedup of approximately 3-4 is to vectorize the code for the Cray. The next generation of parallel computers, which will be available in the next couple of years, will have nodes that run at approximately 10 Mflops (million floating point instructions per second), as compared with a maximum Cray speed for the program of approximately 40 Mflops if it is vectorized. 64 nodes of such a parallel computer would yield on the order of 500 Mflops, with no vectorization required. This is a factor of 10 faster that the whole NCUBE running at 100 % efficiency.

8.3 The Future

As mentioned above, the program can be vectorized. This would be achieved by performing a vector move of all timeslices in the path at once. One idea that

would improve the vectorization and the efficiency of the parallel algorithm is to allow for the simultaneous threading of different segments within one thread. As an example, consider seven-particle threading. The new position for the middle timeslice must be generated first, but once that is complete, the generation of the two three-particle subthreads can proceed simultaneously.

Although our method doesn't scale well to large systems, it is still possible to extract interesting physics when smaller systems are studied with present computing speeds. For example, even at 3.2 Å, nearest-neighbor separations in the two-dimensional system studied in Chapter 7, no finite-system dependence was detected (within admittedly large error bars) in going from updating only 6 particles in an 18 particle system to updating all 18 particles. This nearest-neighbor spacing in the 3 dimensional system would correspond to a molar volume of 15.2 cm^3/mole, quite a high density. As mentioned at the end of Chapter 7, including a substrate potential and removing the two-dimensional constraint for the system discussed there would provide an interesting comparison with experiment.

The observable used to measure the Gruneisen parameter for the hard-sphere model has much smaller error bars than for the ^{3}He observable, and the run time is faster for this system because of a much simple expression for the two-particle density matrix. It is interesting to note that for the small systems consisting of the exchanging group of particles being updated in a system of 16 particles, the Gruneisen parameters (see Table 5.2) are of the same magnitude as for ^{3}He. The hard-sphere model might be useful as a complement to simple analytical models to provide rough comparisons of how γ varies as compared with predictions of the analytical models. Another interesting, although computationally more intensive, possibility is to compare carefully the results of large (54 particle) hard-sphere systems with those for solid ^{3}He to try to extract which features of the ^{3}He system are from hard-core effects and which features are a more complicated function of the interaction potential.

References

1. David Ceperley, private communication.

2. D. M. Ceperley and G. Jacucci, *Phys. Rev. Lett.*, **58**, 1648, Apr. 1987.

Appendix A

Parallel Programming Issues

The opportunity to participate in the emerging field of parallel computing has been interesting and exciting. The programming model used to port the program listed in Appendix C to a parallel machine has been developed in the last couple of years at Caltech, and I have been lucky to have participated in its development. This appendix will first describe this programming model, which is closely related to the Cubix I/O system.[1] Next, a description of a subset of the Cros III communication system[1] will precede a discussion of the algorithm used to implement the parallel Monte Carlo program.

One note before continuing is in order. The Cubix I/O system and the Cros III communication system work well only for programs written in a synchronous fashion. Thus whenever one node (processor) of the parallel computer writes to another node, that other node must read the message before the two nodes continue. This means that each processor must be given a roughly equal amount of work, or many nodes will sit idle waiting for the node with more work to catch up. Hence, it is said that the programming model described below is useful for regular problems.

A.1 A Powerful Parallel Programming Model for Regular Problems

Before discussing the Cubix programming model, a schematic description of typical parallel computing hardware will be given, followed by a brief discussion of a historical programming model for parallel computing (still held by many).

Most parallel computers consist of a host computer, with an attached array of computers that provide the parallel computing facility. The host is a multi-user computer with an advanced operating system and attached peripherals such as diskdrives, tapedrives, network interfaces, etc. Programs to be run on the array are downloaded from the host, and output from the array typically must go through the host. (There are parallel disk systems on some machines such as the NCUBE parallel computer which are attached directly to the array.) The individual nodes of the array are typically much simpler than the host, often having no memory protection and less memory than a typical multi-user computer.

The historical model presented here for comparison is often referred to as the 80/20 picture, where 80% of the code making up a parallel program resides on the sequential host computer, and 20% resides on the array of parallel computers. The program for the host contains most of the code, including I/O, user interface, etc. The second part that runs on the array, contains only the computationally intensive parts of the program and the communication necessary for reading data into the cube and writing it out. The program flow starts in the host program. It runs until it reaches a section involving a large amount of computation. At that point a request is sent to the array (including data and whatever else is needed) for the large computation to be done. The host program then reads the answer from the array when the computation is complete.

On the face of it, this sounds like quite a reasonable approach to take, but it has inherent difficulties that hobble the efficient development of new programs, and furthermore, give no hope at all that such a program could ever be simply ported, either to or from a sequential computer. The largest problem for new program development is that host-array communications, which are quite similar from program to program, must be developed anew each time a program is written. Porting working code is even worse, since the first thing that must be done is

to rip the program into two pieces and insert communication calls into the two pieces. Any attempt to debug the array program involves the further introduction of communication in both the host and the node programs.

The key to the Cubix programming model is that only one program is written, and it runs on the array. Cubix is made up of a universal program that runs on the host, and a library of I/O subroutines, which may be called from the array program. The I/O subroutines make the standard C (or Fortran) I/O facilities, which are actually on the host, appear to be available directly on the array. This natural way of standardizing host node communication has the enormous advantage that a program that runs on a sequential computer can be compiled and run on one node of the array with no modification. The process of parallelization then consists of allowing all of the nodes to run the program with the same parameters in sections of the code that can't be parallelized. When a part of the program that can be parallelized is reached, each node does its bit of the work (this is usually achieved by modifying the limits in loops which are summed over in the calculation), after which the results are combined and the program continues in a sequential manner.

There is one caveat to be made here. If the hardware making up the nodes is not completely reliable, occasionally some node might calculate a different answer in a sequential section of the program, where all of the nodes are relied upon to give the same answer. This is mentioned here because the NCUBE parallel computer, used as the primary computer for this work, has this feature, and extra communication had to be added to compensate. With reasonably designed hardware, this caveat would not be an issue.

Caveat aside, when a program is to be ported to a parallel machine, the Cubix programming model decreases the time to develop, port and modify parallel programs by at least a factor of 20 based on practical experience gained in the Caltech Concurrent Computation Project. It also allows programs of much greater complexity to be developed than would otherwise be possible.

Before discussing the specific issues involved in the parallelization of the program used to calculate the Gruneisen parameters for exchange, the communication

routines used (a subset of Cros III) [1] will be described. The first routines described are used to provide transparently a mapping of the array connection topology onto a grid suitable for the parallel decomposition of the problem being solved. They are

gridinit(griddim,num),

gridcoord(proc,coord),

gridchan(proc,dir,sign).

Gridinit initializes a *griddim* dimensional mesh. *Num* is an array passed to *gridinit* that has *griddim* elements, each giving the size of one dimension of the mesh. All nodes call *gridinit* at the beginning of the program. *Gridcoord* returns the mesh coordinates in the *griddim* dimensional array *coord* for node number *proc*. *Gridchan* returns the channel mask which node *proc* must use to communicate with the node connected to it in the dimension *dir*. There are two processors connected in each dimension, so if *sign*=-1, it is the minus *dir* direction, and if *sign*=1, it is the plus *dir* direction.

Once the communication mesh (which is periodic in each of the dimensions) is set up, the following communication routines may be used. One of the most useful routines (and primarily used in the program listed in Appendix C) is

cshift(inbuf,inmask,insz,outbuf,outmask,outsz).

It reads *insz* bytes from channel *inmask* into the buffer pointed to by *inbuf*, and writes *outsz* bytes to channel *outmask* from the buffer pointed to by *outbuf*.

The other routines used for communication between individual nodes are *cread* and *cwrite*.

cread(buf,inmask,0,nbytes)

reads *nybtes* from channel *inmask* into the buffer pointed to by *buf*. The third argument allows for a simultaneous read, but this will not be described here. The function complementary to *cread* is

cwrite(buf,outmask,nbytes),

which writes *nbytes* from the buffer pointed to by *buf*, to the channel *outmask*.

The final two commands are used to deal with more global communication than

the previously described commands. The first is

$$broadcast(buf,src,mask,nbytes),$$

which causes *nybtes* bytes under the pointer *buf* on node *src* to be placed under the pointer *buf* on all the other nodes, when *mask* is set equal to the total number of nodes minus one. Different values of *mask* can be used to broadcast to different subsets of the cube, but that won't be discussed here. Finally, the command

$$combine(buf,func,sz,nitems)$$

is used to combine the *nitems* different items (each of size *sz*, stored under the pointer *buf* on each node) according to the function *func*. The combined result is placed under the pointer *buf* on each node. For example, consider the case where *buf* is the pointer to a floating point number, so that $sz = sizeof(float)$ and $nitems = 1$. If *func* adds two floating point numbers, each node will end up with the sum of the floating point numbers stored under *buf* from all the processors.

A.2 Implementing a Portable Parallel Program

The program which calculates the Gruneisen parameters was ported from a sequential computer. Currently, with the change of one compiler switch, it can run on either a parallel or a sequential computer, including a 500 node Ncube parallel computer, an Elxsi 6400, and a Cray XMP. The program won't vectorize on the Cray as currently written, but it appears to be possible to do so while keeping the ability to run on the NCUBE.

There are two sorts of parallelism that can be exploited in the Monte Carlo algorithm described in Chapters 4 and 5. A two-dimensional communication mesh is used to take advantage of them (Fig 28). The first sort of parallelism comes from the fact that timeslices not adjacent to each other can be updated simultaneously. This fact is used to divide the path among as many nodes as possible. This is the most efficient sort of parallelism because during the update of one thread, no communication in the time direction is necessary. Consider the case where n_p particles are being threaded. Each node in the time directions has $m(n_p + 1) + 1$ timeslices of the path. Typically, m=1, but it doesn't have to be. Each set of

n_p+1 timeslices corresponds to the first, fixed timeslice and n_p timeslices which are threaded in. The position of the last fixed timeslice for each thread is identical to the first, fixed timeslice of the next thread. If the next thread is on the next node in the time direction, two copies of that fixed timeslice are kept, one in each node. To make a connection with the program in Appendix C, the first and last timeslice in each node are stored in $ts\_range[0]$ and $ts\_range[1]$, respectively. As mentioned above, $ts\_range[1]$ in one node is equal to $ts\_range[0]$ in the next node in the time direction. The typical number of timeslices used for the calculations of Chapter 6 is 128+1. Therefore, for the value of $n_p = 7$, at most 16 nodes can be used in the time direction. This parallelism can be seen in Appendix C in the function *update_thr*().

After each update made as described above, each node updates a slightly different portion of the path using a new set of threads, beginning with the one that has fixed the initial fixed timeslice at $ts\_range[0] + (n_p/2 + 1)$, and ending with the one that has its final fixed timeslice at $ts\_range[1] + (n_p/2 + 1)$ (The very last thread described in this way goes off the end of the path and so it isn't updated.). This guarantees that the fixed end points for the update described above are in the middle of the new threads and therefore get updated. Before this shifted updated can occur, the timeslices from $ts\_range[1]$ to $ts\_range[0] + (n_p/2 + 1)$ in each processor must be read from the processor with a time mesh index one greater than its own. Similarly, after this update, the modified data are shifted back so that the first type of update can be performed with a consistent path.

The second sort of parallelism that may be extracted involves interparticle interactions within each timeslice. More than one particle per timeslice may not be updated simultaneously because of the long-range nature of the interaction potential, but when the pairwise interactions between one particle and all of the other particles in the system are calculated, the work may be divided between several processors, and the results combined at the end (see the function $S()$ in the file mov.c in Appendix C). This is achieved by using the spatial coordinate of the processors in the two-dimensional, communication mesh ($(pcoord[0], pcoord[1])$ = (time,space)). Consider the case where there are N_s nodes in the spatial direction, numbered from

0 to $N_s - 1$. Node n_s will calculate the interactions with particles n_s, $n_s + N_s$, $n_s + 2N_s$, etc. In this way all pairwise interactions are calculated. Both forms of parallelism may be seen in the subroutine *obs*() in Appendix C.

References

1. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.

Appendix B

Random Numbers

B.1 Linear Congruential

A linear congruential pseudorandom number generator is used for the programs written for this work and has been setup to work when the programs are compiled for either a parallel computer [1] or for a sequential computer.

The basic algorithm is defined by

$$R_{n+1} = (aR_n + b) \mathrm{mod} m, \qquad (B.1)$$

where $\{R_n\}$ defines a sequence of pseudorandom numbers. The values for a,b and m have been taken from the ANSI standard publication .[2]

$$a = 1103515245$$

$$b = 12345$$

$$m = 2^{32}$$

Floating point numbers on [0,1) are derived from the integer numbers R_n by dividing by the modulus m,

$$f_n = \frac{R_n}{m}. \qquad (B.2)$$

The way this algorithm is used on a parallel computer is by noting that a relation can be written down that gives the $n+k$th random number in terms of the nth,

$$R_{n+k} = A_k R_n + B_k, \qquad (B.3)$$

with

$$A_k = (a^k) \mathrm{mod} m \qquad (B.4)$$

and †

$$B_k = (b \sum_{i=0}^{k-1}) \mathrm{mod} m. \qquad (B.5)$$

The forms for A_k and B_k are easily derived by repeatedly substituting the expression for R_j, in terms of R_{j-1}, into the recursion relation of Eqn. (B.1).

B.2 The Polar Method of Producing Normal Random Numbers

This algorithm comes from Knuth,[2] who says, "The polar method is quite slow, but it has essentially perfect accuracy ..." It uses a pseudorandom number generator such as the one described in Section B.1 to generate two random numbers that are scaled to the range [-1,1). Call these numbers r_1 and r_2. Defining

$$s = r_1^2 + r_2^2, \qquad (B.6)$$

it checks to see whether $S < 1$ and, if not, generates two more numbers. This procedure is repeated until the two random numbers satisfy the requisite condition. Then defining an intermediate value

$$w = \sqrt{(-2\ln(s)/s)}, \qquad (B.7)$$

two independent normal random numbers distibuted according to

$$f(x) = e^{-x^2/2} \qquad (B.8)$$

are given by

$$n_1 = r_1 w; \quad n_2 = r_2 w. \qquad (B.9)$$

† The expression for B in the reference is in error, but Eqn. (B.5) gives the correct expression.

References

1. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood Cliffs, NJ, 1988.
2. K. E. Knuth, *The Art of Computer Programming vol. 2 Seminumerical Algorithms*, Addison Wesley, Reading, Mass., 1973.

Appendix C

Source Code

The following is a listing of the important sections of the source code used to produce the results for the Gruneisen paramters for ^{3}He. For the sake of brevity, I/O, debugging, and the user interface code have not been reproduced here. Before the code is a listing of the subroutine names in the order in which they appear, with a brief explaination of the function of their function is.

```
setup.c:setup() /* Setup System Parameters */
setup.c:pot_init() /* Initialize Density Matrix */
setup.c:int samevec(x1,x2)
setup.c:int is_in(x1,rad,pn,ts,iarr,sw)

par.c:par_setup() /* Setup parallel program parameters */

layer.c:layer_setup(ex_type) /* Setup shell structure to update subsystems */

obs.c:struct obsret obs(nlyr,tsmid) /* Calculate observable */
obs.c:float poten(nlyr,tsmid) /* Calculate Potential energy using V(r) */
obs.c:float eff_poten(nlyr,tsmid) /* Calculate V_eff(r) = dU/d(eps) */

update.c:update_thr(num,nlyr) /* Do threading update */
update.c:update(num,nlyr) /* Do standard Metropolis update */

mov.c:movthr(pn1,ts)  /* update one thread */
mov.c:movpar(pn1,ts)  /* update one timeslice */
mov.c:float S(xindex,pn1,ts1,ts2) /* Calculate the Changed Part of the Action */
mov.c:int overflow(pn1,ts,parswitch,test) /* Check for hard core overlap */

sbias.c:sbias(pn1,ts1,ts2,bind,choice) /* Calculate bias from V_eff */
```

```
/* loop() calculates the interaction of one particle with the rest of the
system (see comment above program listing) */
loop.c:float loop(funcp,pass_pn1,pass_ts1,pass_ts2,pass_bind,xx1,xx1p,lpfl)
loop.c:float S_func() /* Low level calculations */
loop.c:float effpoten_func()
loop.c:float poten_func()
loop.c:float obs_func()
loop.c:float sbias_func()
loop.c:cvec0() /* Calculate r12 r12p */
loop.c:cvec() /* Calculate r12 r12p and costh */
loop.c:int min0(a,b)
loop.c:int max0(a,b)
loop.c:float amin1(a,b)

inst.c:int check_inst(extype,retwidth) /* return instanton position and width */
inst.c:plot_inst(extype)  /* Print out a listing of instanton values */
inst.c:shift(tsinst,extype) /* Shift the instanton */

init.c:xinit(exnum)  /* Initialize a lattice */
init.c:ginit() /* initialize the ground state.

prand.c:unsigned int seedgt()
prand.c:pranset(seed)
prand.c:float pranf()
prand.c:float normal()
prand.c:backup_seed()

u.c:float u0(bind) /* Returns End Point Approximation to U */
u.c:float u(bind) /* Returns U */
u.c:float uprime(bind) /* Returns V_eff = dU/d(eps) */
u.c:getd() /* Load two-particle density matrix parameters */
u.c:interp() /* Interpolate between nonlinear mesh points */

/*** parms.h ***                                    Sean Callahan */

#define DIM 3       /* The dimension of the space */
#define NPSD 3      /* NPSD = the number of particles on one edge of the
                        simple cubic lattice
                        */

#define NP 54       /* NP = 2*NPSD**3  is the total number of particles*/

#define NT 128      /* NT is the total number of timeslices including the
                        first and last fixed ones
                        */

#define DCUB 4.304  /* this is the spacing(in Angstroms) of the simple
                        cubic lattice. Nearest-neighbor spacing is
                        sqrt(3*(DCUB/2)**2).
                        Molar specific volume is related to DCUB by
                        V_M = .301 * DCUB^3.  V_M(4.304) = 24.0 cm^3/mole

               V_M   24    23.5  23        22.5  22    21.5  21    20.5  20
               DCUB 4.304 4.274 4.243 4.212 4.181 4.149 4.117 4.084 4.050
                        */

#define CUTOFF 5.0  /* This cutoff is for calculation of the bias.
```

```
                        Only hard-core interactions are significant */

#define RMIN 1.55   /* hard core cutoff */
#define RMIN2 2.4336 /* (RMIN+DELTA)^2 */

#define LAM 16.0802 /* This is h_bar squared over m (mass of HE 3) in
                       units where distance is measured in units of
                       Angstroms and energy is measured in 1K(k_Boltzman).
                       */

#define NBETA 6          /* the number of values of beta for which the two
                       particle density matrix has been tabulated
                       */

#define BIND_OFS 1  /* Gives the offset in the U() array of DBETA */
#define DBETA .00625 /* the spacing between timeslices of the lattice
                        */

#define NDIM 70          /* the number of mesh points for the two particle
                       density matrix.
                       */

#define DELTA .001  /* delta for calculating gradients */

#define NINDEX 2    /* 0 - 1 particle threading; 1 - 3 thr; 2 - 7 thr; */

/*
edge_sz  - the size of one edge of the unit volume ( = NPSD*DCUB )
*/

extern float edge_sz[3],hedge_sz[3];
extern float x[NT][NP][3],xthr[NT][3];
extern int nmov[5];
extern float cx;
extern int ip;
extern int try,acc1,acc2,sphov;
extern float r12,r12p,costh;

/* Parallel variables */
extern int pcoord[2],tmc[2],tm[2],sm[2];
extern int gdim[2];
extern int cell_sz[2],nt_sz,pos_sz[2],szpnd,ex_sz;
extern int ts_range[2];

extern int upts;
extern int downts;

#define NLAYER 4
extern int layer[NP],layer_ind[NLAYER+2],crossref[NP];
extern int layer_ofs[NP][3];
extern float rlayer[4][NLAYER - 1];
extern int xcon[4][6][2],ts_size;

float loop(),S_func(),sbias_func(),obs_func();

struct obsret {
     float KE;
     float PE;
     float KE_old;
```

```
} ;

/*** setup.c ***                                    Sean Callahan */
#include <math.h>
#include "parms.h"

/* Note that exchange type 4 is only correct here in two dimensions.  For the
   three-dimensional case the correct values are set in setup()
*/
int xcon[4][6][2] = {
     0,1 , 1,0 , -1,-1 , -1,-1 , -1,-1 ,-1,-1 ,
     0,2 , 2,0 , -1,-1 , -1,-1 , -1,-1 ,-1,-1 ,
     0,2 , 1,0 , 2,1 , -1,-1 , -1,-1 ,-1,-1 ,
     0,2 , 2,3 , 3,1 , 1,0 , -1,-1 ,-1,-1
};
float edge_sz[3],hedge_sz[3];
int reset_tslim = 21;

setup()
{
int n,i,a,b,c,d,np;

ip = 2;
cx = RMIN*RMIN * (NDIM-1);
#if DIM==3
for (n=0;n<DIM;n++) {
     edge_sz[n] = (float)NPSD*DCUB; /* Edge size of periodic region */
     hedge_sz[n] = edge_sz[n]/2.;
}
#endif
#if DIM==2
edge_sz[0] = (float)NPSD*DCUB; /* Edge size of periodic region in x dir*/
hedge_sz[0] = edge_sz[0]/2.;
edge_sz[1] = (float)NPSD*DCUB*sqrt(3.); /* Edge size in y dir*/
hedge_sz[1] = edge_sz[1]/2.;
edge_sz[2] = 0.; hedge_sz[2] = 0.;
#endif

upts = NT - reset_tslim;
downts = reset_tslim;

#if (DIM==2)
np = 2*NPSD*NPSD;
#endif
#if (DIM==3)
np = 2*NPSD*NPSD*NPSD;
#endif
if (np!=NP) {
     printf("ERROR - Constant NP is inconsistent with constant NPSD\n");
     printf("np = %d,npsd=%d\n",np,NPSD);
     abort(30);
}

nmov[0] = 1;  /* Initialize threading information */
for (i=1;i<5;i++) nmov[i] = nmov[i-1] + (1<<i);

#if (DIM==3)
/* intitialize 4 particle nearest-neighbor exchange particle values */
```

```
a = 1;
b = 2*(NPSD*NPSD + 1);
c = b + 1;
d = b + 2*NPSD;

xcon[3][0][0] = a;
xcon[3][0][1] = b;

xcon[3][1][0] = b;
xcon[3][1][1] = c;

xcon[3][2][0] = c;
xcon[3][2][1] = d;

xcon[3][3][0] = d;
xcon[3][3][1] = a;
#endif

pot_init();
par_setup();
}

pot_init()
{
     getd();   /* not bin_getd() for the Cray */
}

int samevec(x1,x2)
float x1[3],x2[3];
{
     int n;
     for (n=0;n<DIM;n++) if (x1[n] != x2[n]) return(0);
     return(1);
}


int is_in(x1,rad,pn,ts,iarr,sw)
/* if sw==0
   returns 0 => particle (pn,ts) not within rad of x1[].
             1 => one encarnation of particle (pn,ts) is within rad. It's
                  integer offset is in iarr[0][0-2].
             2 => more than one encarnation ... range of offsets stored in
                  iarr[0][n] to iarr[1][n].
   else
     returns 0 => if not in
               1 => if in with iarr giving offset of closest reflection.
*/
float x1[3],rad;
int pn,ts,iarr[2][3];
{
     float compdist2,compdelta,dist2,rad2,delta[3];
     int n,flag;

     dist2 = 0.;
     rad2 = rad*rad;
     for (n=0;n<DIM;n++) {
          delta[n] = x[ts][pn][n] - x1[n];
          iarr[0][n] = 0;
          while (delta[n]<0. && delta[n]<=(-hedge_sz[n])) {
```

```
                delta[n] += edge_sz[n];
                iarr[0][n]++;
            }
            while (delta[n]>0. && delta[n]> hedge_sz[n]) {
                delta[n] -= edge_sz[n];
                iarr[0][n]--;
            }
            dist2 += delta[n]*delta[n];
        }

        if (dist2 > rad2) return(0);
        if (sw==1) return(1);

        flag = 0;
        for (n=0;n<DIM;n++) {
            compdist2 = dist2 - delta[n]*delta[n];
            compdelta = delta[n];
            iarr[1][n] = iarr[0][n];
            while(1) {
                compdelta += edge_sz[n];
                if ((compdist2+compdelta*compdelta) <= rad2) {
                    flag = 1;
                    iarr[1][n]++;
                } else {
                    break;
                }
            }

            compdelta = delta[n];
            while(1) {
                compdelta -= edge_sz[n];
                if ((compdist2+compdelta*compdelta) <= rad2) {
                    flag = 1;
                    iarr[0][n]--;
                } else {
                    break;
                }
            }
        }
        if (flag) return(2);
        else return(1);
}

/*** par.c ***                                    Sean Callahan */

#ifdef PAR
#include <cros.h>
struct cubenv cubinfo;
#endif

#include <stdio.h>
#include "parms.h"
#include <math.h>

int doc;
int pcoord[2] = {0,0};
int ts_range[2] = {0,(NT-1)};
int gdim[2] = {1,1};
```

```
int tmc[2],tm[2],sm[2],nt_sz,pos_sz[2],szpnd,ex_sz;

/* Set up the parameters to run on a parallel machine using the Cros 3
   communication system and the Cubix I/O system.
*/
par_setup() {
#ifdef PAR
     int nsz;
     if (gridinit(2,gdim)==-1) {
          printf("gridinit ERROR\n");
          abort(70);
     }
     cparam(&cubinfo);
     doc = cubinfo.doc;
     /* Set up the two-dimensional grid. */
     if (gridcoord(cubinfo.procnum,pcoord)==-1) {
          printf("gridcoord ERROR\n");
          abort(71);
     }

     /* Get the communication masks for communicating with neighboring nodes */
     tmc[0] = tm[0] = gridmask(cubinfo.procnum,0,-1);
     tmc[1] = tm[1] = gridmask(cubinfo.procnum,0,1);
     sm[0] = gridmask(cubinfo.procnum,1,-1);
     sm[1] = gridmask(cubinfo.procnum,1,1);

     if (pcoord[0]==0) tm[0] = 0;  /* not periodic in time */
     if (pcoord[0]==(gdim[0]-1)) tm[1] = 0;

     nsz = nmov[NINDEX]/2 + 1;
/* The sizes of chunks to communicate must always be that for a 3 vector since
   the data in memory in three vector format (even if DIM!=3)
*/

/* Set up communication sizes
*/
     nt_sz = 3*NP*sizeof(float);
     pos_sz[0] = 3*nsz*NP*sizeof(float);
     pos_sz[1] = 3*(nsz+1)*NP*sizeof(float);
     szpnd = (NT-1)/gdim[0];
     ex_sz = szpnd*3*NP*sizeof(float);

     if ((NT-1)%gdim[0] != 0) {
          printf("ERROR: NT doesn't divide evenly among the nodes\n");
          printf("NT = %d; gdim = %d %d\n",gdim[0],gdim[1]);
          abort(51);
     }
     /* Make sure the threading works
     */
     if ( szpnd%(nmov[NINDEX]+1) != 0) {
          printf("ERROR: # of prtcls/node != int*(nmov[NINDEX]+1)\n");
          abort(50);
     }
     /* Setup the range of timeslices in this processor */
     ts_range[0] = pcoord[0]*szpnd;
     ts_range[1] = ts_range[0] + szpnd;
#endif
}
```

```
/*** layer.c ***                                    Sean Callahan */

#include <stdio.h>
#include <math.h>
#include "parms.h"

int layer[NP],layer_ind[NLAYER+2],crossref[NP];
int layer_ofs[NP][3];
float rlayer[4][NLAYER - 1] = {
     4.,5.,6.,
     4.,5.,6.,
     4.5,5.5,6.,
     4.,5.,6.
};

/* Set up shells of particles which are within distances in the array
rlayer[][] of the exchanging region
*/
layer_setup(ex_type)
int ex_type;
{
     int pn,iarr[2][3],i,count,n;
     float xav[3];

     /* crossref[index] gives the particle number given the layer[pn] number for
        particle pn (i.e. - crossref[layer[pn]] = pn) */
     for (pn=0;pn<NP;pn++) crossref[pn] = 255;

     for (n=0;n<DIM;n++) xav[n] = 0.;
     count = 0;

     /* First do layer 1 of the exchanging set of particles
     */
     while(xcon[ex_type][count][0] != -1) {
          pn = xcon[ex_type][count][0];
          for (n=0;n<DIM;n++) xav[n] += x[0][pn][n];
          layer[count] = pn;
          layer_ofs[count][0] = layer_ofs[count][1] = layer_ofs[count][2] = 0;
          crossref[pn] = count;
          count++;
     }
     layer_ind[0] = 0;
     layer_ind[1] = count;

     for (n=0;n<DIM;n++) xav[n] /= (float)count;

     /* Search for all the particles within each layer radius, and build the
        layer[] array.  layer_ind is the index for each layer which gives
        the starting point for each shell of particles in the layer[] array.
     */
     for (i=0;i<(NLAYER-1);i++) {

          for (pn=0;pn<NP;pn++) {

               if (crossref[pn] != 255) continue;
               if (is_in(xav,rlayer[ex_type][i],pn,0,iarr,1)) {
                    layer[count] = pn;
                    layer_ofs[count][0] = iarr[0][0];
                    layer_ofs[count][1] = iarr[0][1];
```

```
                    layer_ofs[count][2] = iarr[0][2];
                    crossref[pn] = count;
                    count++;
               }

          }
          layer_ind[i+2] = count;

     }


     /* The last layer gets the rest of the particles
     */
     for (pn=0;pn<NP;pn++) {
          if (crossref[pn] != 255) continue;
          is_in(xav,100.,pn,0,iarr,1);
          layer[count] = pn;
          crossref[pn] = count;
          layer_ofs[count][0] = iarr[0][0];
          layer_ofs[count][1] = iarr[0][1];
          layer_ofs[count][2] = iarr[0][2];
          count++;
     }

     if (count != NP) {
          printf("ERR:count = %d should be = %d\n",count,NP-1);
          abort(120);
     }

     layer_ind[NLAYER+1] = NP;

}

/*** obs.c ***                                        Sean Callahan */
  Measure observable

  obs = 1/(LAM*DBETA)*(x-xprime)^2 + Sum over all pairs (j>i) of
          { r_ij dU_ij/dr_ij + r_ijprime dU_ij/dr_ijprime }
*/

#include <stdio.h>
#include <math.h>
#include "parms.h"

#ifdef FAST
#define cshift fshift
#endif

int ts_size=30; /* This give the default distance around the instanton
                       to measure
                   */

/* nlyr gives the layer number for the shell of particles to measure and
   tsmid is the position of the instanton
*/
struct obsret obs(nlyr,tsmid)
int nlyr,tsmid;
{
```

```
        float cnst,x1[3],x1p[3];
        int ts,pn1,n,bind,index,count,lowts,hights;
        struct obsret retobs,com[2];
        int sw;
        float in,out;

        out = 3.1415;

        lowts = tsmid - ts_size;
        hights = tsmid + ts_size;
        cnst = 1/(DBETA*LAM);
        com[0].KE=com[0].KE_old=com[0].PE=0.;
        com[1].KE=com[1].KE_old=com[1].PE=0.;
        retobs.KE_old = 0.;
        retobs.KE = 0.;
        retobs.PE = 0.;
        bind = 0; /* use smallest timeslice */

        for (ts=ts_range[0];ts<ts_range[1];ts++) {
             if (ts<lowts || ts > hights) continue;
             if (ts==0) continue;
             for (count=0;count<layer_ind[nlyr];count++) {
                  pn1 = layer[count];
                  for (n=0;n<DIM;n++) { /*set x1 and add in free-particle part*/
                       x1[n] = x[ts][pn1][n];
                       x1p[n] = x[ts+1][pn1][n];

        /* The Kinetic term is calculated using a split-point approximation
             (see Creutz prob. 3 Chpt. 1) instead of the standard divergent
             finite difference method */
                       if (pcoord[1]==0) retobs.KE_old +=
                            cnst*(x1[n]-x1p[n])*(x1[n]-x[ts-1][pn1][n]);
        /* and with the standard method */
                       if (pcoord[1]==0) retobs.KE +=
                            cnst*(x1[n]-x1p[n])*(x1[n]-x1p[n]);
                  }
                  retobs.PE += loop(obs_func,count,ts,ts+1,bind,x1,x1p,nlyr);
             }
        }

    #ifdef PAR /* Communicate results */
        com[0] = retobs;
        sw = 1;
        for (index=0;index<(gdim[1]-1);index++) {
             cshift(&com[sw],sm[0],sizeof(struct obsret),
                    &com[(sw+1)%2],sm[1],sizeof(struct obsret));
             retobs.KE_old += com[sw].KE_old;
             retobs.KE += com[sw].KE;
             retobs.PE += com[sw].PE;
             sw ^= 1;
        }
        com[0] = retobs;
        sw = 1;
        for (index=0;index<(gdim[0]-1);index++) {
             cshift(&com[sw],tmc[1],sizeof(struct obsret),
                    &com[(sw+1)%2],tmc[0],sizeof(struct obsret));
             retobs.KE_old += com[sw].KE_old;
             retobs.KE += com[sw].KE;
             retobs.PE += com[sw].PE;
```

```
          sw ^= 1;
          in = 0.;
          cshift(&in,tmc[0],sizeof(float),&out,tmc[1],sizeof(float));
          if (in!=out) abort(915);
     }
#endif

     return(retobs);
}

float poten(nlyr,tsmid)
int nlyr,tsmid;
{
     float cnst,x1[3],x1p[3],poten_func();
     int ts,pn1,n,bind,index,count,lowts,hights;
     float retobs,com[2];
     int sw;
     float in,out;

     out = 3.1415;

     lowts = tsmid - ts_size;
     hights = tsmid + ts_size;
     cnst = 1/(DBETA*LAM);
     com[0]=0.; com[1]=0.;
     retobs = 0.;
     bind = 0; /* use smallest timeslice */

     for (ts=ts_range[0];ts<ts_range[1];ts++) {
          if (ts<lowts || ts > hights) continue;
          for (count=0;count<layer_ind[nlyr];count++) {
               pn1 = layer[count];
               for (n=0;n<DIM;n++) { /*set x1 and add in free-particle part*/
                    x1[n] = x[ts][pn1][n];
                    x1p[n] = x[ts+1][pn1][n];
               }
               retobs += loop(poten_func,count,ts,ts+1,bind,x1,x1p,nlyr);
          }
     }

#ifdef PAR
     com[0] = retobs;
     sw = 1;
     for (index=0;index<(gdim[1]-1);index++) {
          cshift(&com[sw],sm[0],sizeof(float),
                 &com[(sw+1)%2],sm[1],sizeof(float));
          retobs += com[sw];
          sw ^= 1;
     }
     com[0] = retobs;
     sw = 1;
     for (index=0;index<(gdim[0]-1);index++) {
          cshift(&com[sw],tmc[1],sizeof(float),
                 &com[(sw+1)%2],tmc[0],sizeof(float));
          retobs += com[sw];
          sw ^= 1;
          in = 0.;
          cshift(&in,tmc[0],sizeof(float),&out,tmc[1],sizeof(float));
          if (in!=out) abort(915);
```

```
    }
#endif

    return(retobs);
}

float eff_poten(nlyr,tsmid)
int nlyr,tsmid;
{
    float cnst,x1[3],x1p[3],effpoten_func();
    int ts,pn1,n,bind,index,count,lowts,hights;
    float retobs,com[2];
    int sw;
    float in,out;

    out = 3.1415;

    lowts = tsmid - ts_size;
    hights = tsmid + ts_size;
    cnst = 1/(DBETA*LAM);
    com[0]=0.; com[1]=0.;
    retobs = 0.;
    bind = 0; /* use smallest timeslice */

    for (ts=ts_range[0];ts<ts_range[1];ts++) {
        if (ts<lowts || ts > hights) continue;
        for (count=0;count<layer_ind[nlyr];count++) {
            pn1 = layer[count];
            for (n=0;n<DIM;n++) { /*set x1 and add in free-particle part*/
                x1[n] = x[ts][pn1][n];
                x1p[n] = x[ts+1][pn1][n];
            }
            retobs += loop(effpoten_func,count,ts,ts+1,bind,x1,x1p,nlyr);
        }
    }

#ifdef PAR
    com[0] = retobs;
    sw = 1;
    for (index=0;index<(gdim[1]-1);index++) {
        cshift(&com[sw],sm[0],sizeof(float),
               &com[(sw+1)%2],sm[1],sizeof(float));
        retobs += com[sw];
        sw ^= 1;
    }
    com[0] = retobs;
    sw = 1;
    for (index=0;index<(gdim[0]-1);index++) {
        cshift(&com[sw],tmc[1],sizeof(float),
               &com[(sw+1)%2],tmc[0],sizeof(float));
        retobs += com[sw];
        sw ^= 1;
        in = 0.;
        cshift(&in,tmc[0],sizeof(float),&out,tmc[1],sizeof(float));
        if (in!=out) abort(915);
    }
#endif

    return(retobs);
```

```
}

/*** update.c ***                                        Sean Callahan */

#include "parms.h"
#include <math.h>
#include <stdio.h>

int counter = 0;    /* Particles outside nlyr get updated once in 10 updates*/
extern int updateall;

/* Do num biased updates for particles within layer nlyr (note: updateall
was set equal to 0 for all the runs performed */
update_thr(num,nlyr)
int num,nlyr;
{
     int hofs,count,ts,ofs,j,ulim1,llim2,ulim2,uplim,min0();

     hofs = nmov[NINDEX]/2 + 1;
     ofs = nmov[NINDEX]+1;

     ulim1 = layer_ind[nlyr];
     if (nlyr == (NLAYER+1))  {
          ulim2 = llim2 = NP;
     } else {
          llim2 = ulim1;
          ulim2 = NP;
     }
     for (j=0;j<num;j++) {

          for (count=0;count<ulim1;count++) {
               for (ts=ts_range[0];(ts+ofs)<=ts_range[1];ts+=ofs) {
                    movthr((int)layer[count],ts);
               }
          }
          if (counter==9 && updateall) {
               for (count=llim2;count<ulim2;count++) {
                    for (ts=ts_range[0];(ts+ofs)<=ts_range[1];ts+=ofs) {
                         movthr((int)layer[count],ts);
                    }
               }
          }

#ifdef PAR
          if (tm[0]!=0 || tm[1]!=0) {
               cshift(&x[ts_range[1]+1][0][0],tm[1],pos_sz[0],
                       &x[ts_range[0]+1][0][0],tm[0],pos_sz[0]);
          }
#endif

          for (count=0;count<ulim1;count++) {
               uplim = min0((NT-1),(ts_range[1]+hofs));
               for (ts=ts_range[0]+hofs;(ts+ofs)<=uplim;ts+=ofs) {
                    movthr((int)layer[count],ts);
               }
          }
          if (counter==9 && updateall) {
               for (count=llim2;count<ulim2;count++) {
```

```
                    uplim = min0((NT-1),(ts_range[1]+hofs));
                    for (ts=ts_range[0]+hofs;(ts+ofs)<=uplim;ts+=ofs) {
                         movthr((int)layer[count],ts);
                    }
               }
          }

          counter = (counter + 1)%10;

#ifdef PAR
          if (tm[0]!=0 || tm[1]!=0) {
               cshift(&x[ts_range[0]][0][0],tm[0],pos_sz[1],
                      &x[ts_range[1]][0][0],tm[1],pos_sz[1]);
          }
#endif

     }
}

update(num,nlyr)
int num,nlyr;
{
     int count,ts,j,k,ulim1,llim2,lowlim;

     ulim1 = layer_ind[nlyr];
     if (nlyr == (NLAYER+1)) llim2 = NP;
     else llim2 = ulim1;

     if (pcoord[0]==0) lowlim = ts_range[0] + 1;
     else lowlim = ts_range[0];

     for (j=0;j<num;j++) {

          for (k=0;k<5;k++) {
               for (count=0;count<ulim1;count++) {
                    for (ts=lowlim;ts<ts_range[1];ts++) {
                         movpar((int)layer[count],ts);
                    }
               }
          }
          if (updateall) {
               for (count=llim2;count<NP;count++) {
                    for (ts=lowlim;ts<ts_range[1];ts++) {
                         movpar((int)layer[count],ts);
                    }
               }
          }

#ifdef PAR
          if (tm[0]!=0 || tm[1]!=0) {
               cshift(&x[ts_range[1]][0][0],tm[1],nt_sz,
                      &x[ts_range[0]][0][0],tm[0],nt_sz);
          }
#endif

     }
}
```

```
/*** mov.c ***                                        Sean Callahan */

#include <stdio.h>
#include <math.h>
#include "parms.h"

extern float movsz;
extern int try_ov;

float rets[3]; /* This is the place where bias vector is returned */

/* Make biased move */
movthr(pn1,ts)
int pn1,ts;
{
     int sep,ibeta,i,j,ts1,ts2,im,n,lim,ofs,bind,overflow(),ovflag,index;
     float sig,rbar_old[3],rbar_new[3],normal(),beta,dStr,dS,S(),pranf();
     float rets_old[3],rets_new[3];

     ofs = nmov[NINDEX] + 1;
     dStr = 0.;

     /* Set up the endpoints of the new thread */
     for (n=0;n<DIM;n++) {
          xthr[ts][n] = x[ts][pn1][n];
          xthr[ts+ofs][n] = x[ts+ofs][pn1][n];
     }

/* Generate the thread, storing it in the array xthr[][]. */

     lim = 1;
     for (i=NINDEX;i>=0;i--) {

          sep = nmov[i]/2 + 1;
          ts1 = ts;
          im = ts1 + sep;
          ts2 = im + sep;
          for (j=0;j<lim;j++) {    /* Each iteration of this loop threads in
                                      one particle. */
               ibeta = ts2 - ts1;
               beta = DBETA*ibeta;
               sig = sqrt(beta*LAM*.25);
               bind = i + 1;  /* used as an index into density matrix array */

               /* Calculate the interaction part of the bias for the old and the
                  new thread.
               */
               sbias(pn1,ts1,ts2,bind,1);
               for (n=0;n<DIM;n++) rets_new[n] = rets[n];
               sbias(pn1,ts1,ts2,bind,0);
               for (n=0;n<DIM;n++) rets_old[n] = rets[n];

               for (n=0;n<DIM;n++) {
                    rbar_new[n] = .5*(xthr[ts1][n]+xthr[ts2][n])
                                  + .25*beta*rets_new[n] ;
                    rbar_old[n] = .5*(x[ts1][pn1][n]+x[ts2][pn1][n])
                                  + .25*beta*rets_old[n] ;
                    xthr[im][n] = sig*normal() + rbar_new[n];
                    dStr += ( pow((x[im][pn1][n] - rbar_old[n]),2.) -
```

```
                        pow((xthr[im][n] - rbar_new[n]),2.) )/(2.*sig*sig);
                }

#ifdef PAR /* Communicate the result if this is a parallel computer.
                This fixed a hardware bug which very infrequently caused the
                results to be different for different nodes of the NCUBE
                hypercube
              */
     for (index=0;index<(gdim[1]-1);index++) {
          if (pcoord[1]==index)
               cwrite(&xthr[im][n],sm[1],DIM*sizeof(float));
          if (pcoord[1]==(index+1))
               cread(&xthr[im][n],sm[0],0,DIM*sizeof(float));
     }
#endif

                /* Check to see if the new position overlaps hard-cores with any
                   other particles in the system
                */
                ovflag = overflow(pn1,im,0,0);

                if ( ovflag ) {
                     sphov++;
                     return;
                }

                ts1 = ts1 + ibeta;
                ts2 = ts2 + ibeta;
                im = im + ibeta;

          }
          lim = 2*lim;
     }

     /* Now accept or reject the move. */

     ts2 = ts + ofs;
     /* dS = Sj - Si + log(T_i/T_j) */

     dS = S(1,pn1,ts,ts2) - S(0,pn1,ts,ts2) + dStr;

#ifdef PAR
     for (index=0;index<(gdim[1]-1);index++) {
          if (pcoord[1]==index) cwrite(&dS,sm[1],sizeof(float));
          if (pcoord[1]==(index+1)) cread(&dS,sm[0],0,sizeof(float));
     }
#endif

     try++;

     if (dS < 0.) {
          acc1++;
          for (i=(ts+1);i<=(ts2-1);i++) {
               for (n=0;n<DIM;n++) x[i][pn1][n] = xthr[i][n];
          }
     } else if (exp(-dS) > pranf()) {
          acc2++;
          for (i=(ts+1);i<=(ts2-1);i++) {
               for (n=0;n<DIM;n++) x[i][pn1][n] = xthr[i][n];
```

```
          }
     }

}

movpar(pn1,ts)
int pn1,ts;
{
     int n,overflow(),ovflag;
     float dS,S(),pranf();

     for (n=0;n<DIM;n++) {
          xthr[ts-1][n] = x[ts-1][pn1][n];
          xthr[ts][n] = x[ts][pn1][n] + 2*(pranf()-.5)*movsz;
          xthr[ts+1][n] = x[ts+1][pn1][n];
     }

     ovflag = overflow(pn1,ts,try_ov,0);
     if ( ovflag ) {
          sphov++;
          return;
     }

     /* Now accept or reject the move. */

     dS = S(1,pn1,ts-1,ts+1) - S(0,pn1,ts-1,ts+1);

     try++;
     if (dS <= 0.) {
          acc1++;
          for (n=0;n<DIM;n++) x[ts][pn1][n] = xthr[ts][n];
     } else if (exp(-dS) > pranf()) {
          acc2++;
          for (n=0;n<DIM;n++) x[ts][pn1][n] = xthr[ts][n];
     }

}

/* Calculate the part of the action which is different for the old and the
   new thread.
*/
float S(xindex,pn1,ts1,ts2)
/* xindex = 0 -> use x(i,np,nt) everywhere; = 1 -> use xthr for the thread */
int xindex,pn1,ts1,ts2;
{
     int ts,n,bind,index;
     float x1[3],x1p[3],cnst,retS;
     int sw;
     float com[2];

     retS = 0.;
     bind = 0;
     cnst = 1/(2*LAM*DBETA);

     for (ts=ts1;ts<ts2;ts++) {
          if (xindex==1) {
               for (n=0;n<DIM;n++) {
                    x1[n] = xthr[ts][n];
                    x1p[n] = xthr[ts+1][n];
```

```
            }
        } else {
            for (n=0;n<DIM;n++) {
                x1[n] = x[ts][pn1][n];
                x1p[n] = x[ts+1][pn1][n];
            }
        }

        /* add in free-particle part for each particle being moved */

        if (pcoord[1]==0) /* On a parallel machine only add it in once! */
            for (n=0;n<DIM;n++) retS += cnst*(x1[n]-x1p[n])*(x1[n]-x1p[n]);

        retS += loop(S_func,pn1,ts,ts+1,bind,x1,x1p,0);
    }

#ifdef PAR      /* Combine the results */
    com[1] = 0.;
    com[0] = retS;
    sw = 1;
    for (index=0;index<(gdim[1]-1);index++) {
        cshift(&com[sw],sm[0],sizeof(float),
               &com[(sw+1)%2],sm[1],sizeof(float));
        retS += com[sw];
        sw ^= 1;
    }
#endif

    return(retS);
}

int overflow(pn1,ts,parswitch,test)
int pn1,ts,parswitch,test; /* if parswitch==1 spatial parallelization is done */
{
    int index,n,retov,pn2;
    float amin1(),x1[3],dist,delta;

    int start,inc;
    int sw,com[2];

    retov = 0;

    if (parswitch) {
        start = pcoord[1];
        inc = gdim[1];
    } else {
        start = 0;
        inc = 1;
    }

    if (test) {
        for (n=0;n<DIM;n++) x1[n] = x[ts][pn1][n];
    } else {
        for (n=0;n<DIM;n++) x1[n] = xthr[ts][n];
    }

    for (pn2=start;pn2<NP;pn2+=inc) {
        if (pn1==pn2) continue;
        dist = 0.;
```

```
          for (n=0;n<DIM;n++) {
               delta = x[ts][pn2][n] - x1[n];
               while ( delta<0. && delta<=(-hedge_sz[n])) delta += edge_sz[n];
               while ( delta>0. && delta>   hedge_sz[n] ) delta -= edge_sz[n];
               dist += delta*delta;
               if (dist > RMIN2) goto check_next;
          }

          /* dist <= RMIN2 return indication of hard core overlap */
          retov = 1;
          break;

check_next:;

     }

#ifdef PAR
     com[1]=0;
     com[0] = retov;
     sw = 1;
     for (index=0;index<(gdim[1]-1);index++) {
          cshift(&com[sw],sm[0],sizeof(int),
                 &com[(sw+1)%2],sm[1],sizeof(int));
           retov |= com[sw];
          sw ^= 1;
     }
#endif

     return(retov);

}

/*** sbias.c ***                                          Sean Callahan */

#include "parms.h"
#include <math.h>

extern float rets[3];

#ifdef FAST
#define cshift fshift
#endif

/* Calculate the interaction part of the bias
*/
sbias(pn1,ts1,ts2,bind,choice)
/* if choice is 0 use x[ts][pn][] otherwise use xthr[ts][] */
int pn1,ts1,ts2,bind,choice;
{
     int n,index;
     float x1[3],x1p[3],retval;

     int sw,mes_sz;
     float com[2][3];

     rets[0]=0.;rets[1]=0.;rets[2]=0.;

     if (choice) {
```

```
          for (n=0;n<DIM;n++) {
               x1[n] = xthr[ts1][n];
               x1p[n] = xthr[ts2][n];
          }
     } else {
          for (n=0;n<DIM;n++) {
               x1[n] = x[ts1][pn1][n];
               x1p[n] = x[ts2][pn1][n];
          }
     }

     retval = loop(sbias_func,pn1,ts1,ts2,bind,x1,x1p,0);
     for (n=0;n<DIM;n++) rets[n] *= -LAM;

#ifdef PAR
     for (n=0;n<DIM;n++) {
          com[0][n] = rets[n];
          com[1][n] = 0.;
     }
     mes_sz = DIM*sizeof(float);
     sw = 1;
     for (index=0;index<(gdim[1]-1);index++) {
          cshift(&com[sw][0],sm[0],mes_sz,
                 &com[(sw+1)%2][0],sm[1],mes_sz);
          for (n=0;n<DIM;n++) rets[n] += com[sw][n];
          sw ^= 1;;
     }
#endif

}

/*** loop.c ***                                          Sean Callahan */

#include <stdio.h>
#include <math.h>
#include "parms.h"

extern float rets[3];
extern int mode;

float *x1,*x1p;
float x2[3],x2p[3],xofs[3];
int pn1,pn2,bind,ts1,ts2;

/* This is the function where the interactions between one particle
   (pass_pn1 if lpfl==0 or layer[pass_pn1] if lpfl!=0) and the other
   particles in the system.  If lpfl==0, the interactions of the
   particle are calculated with all other particles in the system.
   If lpfl!=0, the interactions are calculated only with those particls
   that have a layer index less than its own.  The arguments have been
   made to be externals for speed, since funp() will be called many times.
   If mode==1, only the closest periodic extension is included in the
   calculation of the interactions, if mode==0, several periodic extensions
   are included.
*/
float loop(funcp,pass_pn1,pass_ts1,pass_ts2,pass_bind,xx1,xx1p,lpfl)
float (*funcp)();
int pass_pn1,pass_ts1,pass_ts2,pass_bind;
```

```
float xx1[3],xx1p[3];
int lpfl; /* if lpfl==1 don't include cases where pn1>=pn2 */
{
     int n,i,j,k,flag,jj[3],cnt1,cnt2,lm[2][3],cn[3];
     float retval,delta;

     retval = 0.;
     x1 = xx1; x1p = xx1p;
     bind = pass_bind; pn1 = pass_pn1;
     ts1  = pass_ts1 ; ts2 = pass_ts2;

     if (lpfl) {
          cnt1 = pn1;
          pn1 = layer[pn1];
     }

     for (pn2=pcoord[1];pn2<NP;pn2+=gdim[1]) {
          if (pn1==pn2) continue;
          cnt2 = crossref[pn2];
          if (lpfl) {
               if (cnt1>=cnt2) continue;     /* get the counting right */
          }
          /* Calculate the closest periodic extension (or a range if mode==0)
          */
          for (n=0;n<DIM;n++) {
               x2[n] = x[ts1][pn2][n] ;
               x2p[n] = x[ts2][pn2][n] ;
               delta = x2[n] - x1[n];
               cn[n] = 0;
               if ( delta<0. && delta<=(-hedge_sz[n]) ) {
                    delta += edge_sz[n];
                    cn[n]++;
               } else if ( delta>0. && delta>hedge_sz[n] ) {
                    delta -= edge_sz[n];
                    cn[n]--;
               }
               if (mode==0) {
                    if (delta>0) {
                         lm[0][n] = cn[n] - 1;
                         lm[1][n] = cn[n];
                    } else {
                         lm[0][n] = cn[n];
                         lm[1][n] = cn[n] + 1;
                    }
               }
          }
          if (mode==1) {
               xofs[0] = ((float)cn[0])*edge_sz[0];
               xofs[1] = ((float)cn[1])*edge_sz[1];
               xofs[2] = ((float)cn[2])*edge_sz[2];
               cvec();
               retval += funcp();
          } else {
               for (i=lm[0][0];i<=lm[1][0];i++) {
                    xofs[0] = ((float)i)*edge_sz[0];
                    for (j=lm[0][1];j<=lm[1][1];j++) {
                         xofs[1] = ((float)j)*edge_sz[1];
#if (DIM==3)
                         for (k=lm[0][2];k<=lm[1][2];k++) {
```

```
                                xofs[2] = ((float)k)*edge_sz[2];
                                /* exclude extention at far corner */
                                if ((i-cn[0]) && (j-cn[1]) && (k-cn[2])) continue;

                                cvec();
                                retval += funcp();

                        }
#endif
#if (DIM==2)
                                cvec();
                                retval += funcp();
#endif
                    }
                }
        }

        }
        return(retval);
}

float S_func()
{
        float u();
        return(u(bind));
}

float effpoten_func()
{
        float uprime();
        return(uprime(bind));
}

float poten_func()
{
        float pot1();
        return(pot1(r12));
}

#define HIST_SZ 200
#define SINHIST_SZ 20
#define SINMIN 0.
#define SINMAX 1.
#define TMIN 0.
#define TMAX 3.
#define RRMIN 1.
#define RMAX 4.
extern int binstate;
extern int rhist[HIST_SZ],shist[HIST_SZ];
extern float mhist[HIST_SZ],mvhist[HIST_SZ];
extern int *rshist;
extern float *rsmhist,*rsmvhist;

float obs_func()
{
        float u(),retobs,u1,u2,sine,s2,r;

        retobs = 0.;
```

```
        u1 = u(bind);

        r12 += DELTA;
        u2 = u(bind);
        r12 -= DELTA;
        retobs += r12*(u2-u1)/DELTA;

        r12p += DELTA;
        u2 = u(bind);
        r12p -= DELTA;
        retobs += r12p*(u2-u1)/DELTA;

        if (binstate) {
             s2 = 1. - costh*costh;
             if (s2 < .000001) sine = 0.;
             else sine = sqrt(s2);
             r = .5*(r12 + r12p);
             fbin(mhist,HIST_SZ,RRMIN,RMAX,r,fabs(retobs));
             fbin(mvhist,HIST_SZ,RRMIN,RMAX,r,retobs);
             bin(shist,HIST_SZ,SINMIN,SINMAX,sine);
             bin(rhist,HIST_SZ,RRMIN,RMAX,r);
             bin2(rshist,HIST_SZ,SINHIST_SZ,RRMIN,RMAX,SINMIN,SINMAX,r,sine);
             fbin2(rsmhist,HIST_SZ,SINHIST_SZ,RRMIN,RMAX,SINMIN,SINMAX,
                       r,sine,fabs(retobs));
             fbin2(rsmvhist,HIST_SZ,SINHIST_SZ,RRMIN,RMAX,SINMIN,SINMAX,
                       r,sine,retobs);
        }

        return(retobs);
}

float sbias_func()
{
        int n;
        float u1,u2,u0(),amin1();

        if (amin1(r12,r12p) > CUTOFF) return(0.);
        u1 = u0(bind);

        for (n=0;n<DIM;n++) {
             x1[n] += DELTA;
             cvec0();
             u2 = u0(bind);
             x1[n] -= DELTA;
             rets[n] += (u2 - u1)/DELTA;
        }
        return(0.);
}

/* cvec0 returns r12 and r12p only */
cvec0()
{
        float vr12[3],vr12p[3];
        int n;

        r12 = 0.; r12p = 0.;
        for (n=0;n<DIM;n++) {
             vr12[n] = x1[n] - (x2[n] + xofs[n]);
             vr12p[n] = x1p[n] - (x2p[n] + xofs[n]);
```

```
          r12 = r12 + vr12[n]*vr12[n];
          r12p = r12p + vr12p[n]*vr12p[n];
     }
     r12 = sqrt(r12);
     r12p = sqrt(r12p);
     if  (r12<RMIN || r12p<RMIN) abort(821);
}

/* Given the vectors x1[],x2[],x1p[],x2p[], cvec returns r12,r12p and costh
*/
cvec()
{
     float dot,vr12[3],vr12p[3];
     int n;

     dot = 0.;
     r12 = 0.;
     r12p = 0.;

     for (n=0;n<DIM;n++) {
          vr12[n] = x1[n] - (x2[n] + xofs[n]);
          vr12p[n] = x1p[n] - (x2p[n] + xofs[n]);
          dot += vr12[n]*vr12p[n];
          r12 += vr12[n]*vr12[n];
          r12p += vr12p[n]*vr12p[n];
     }

     r12 = sqrt(r12);
     r12p = sqrt(r12p);
     if  (r12<RMIN || r12p<RMIN) abort(821);
     costh = dot/(r12*r12p);
}

int min0(a,b)
int a,b;
{
     if (a<b) return(a);
     else return(b);
}

int max0(a,b)
int a,b;
{
     if (a>b) return(a);
     else return(b);
}

float amin1(a,b)
float a,b;
{
     if (a<b) return(a);
     else return(b);
}
```

```
/*** inst.c ***                                        Sean Callahan */

#include <stdio.h>
#include <math.h>
#include "parms.h"

#define MULTFACT 0.2
extern int updateall;
extern int groundstate;

int check_inst(extype,retwidth)
int extype,*retwidth;
{
     int n,ts,pn,ts1,ts2,in_ts,width,count,check_inst();
     float mod,u[3],value[NT];

     ex_latt();     /* made sure all nodes have the correct lattice */
     if (groundstate) {
          *retwidth = 0;
          return(NT/2);
     }

     count = 0;
     for (ts=0;ts<NT;ts++) value[ts] = 0.;

     /* For all exchanging particles */
     while(xcon[extype][count][0] != -1) {
          pn = xcon[extype][count][0];
          mod = 0.;
          for (n=0;n<DIM;n++) { /* Calculate unit vector pointing for initial
                                        to final position */
               u[n] = x[NT-1][pn][n] - x[0][pn][n];
               mod += u[n]*u[n];
          }
          if (mod==0.) return(NT/2);
          /* dividing by the square of the norm normalizes all particle
             paths for the exchanges to unit length
          */
          for (n=0;n<3;n++) u[n] = u[n]/mod;

          for (ts=0;ts<NT;ts++) { /* Project path onto exchange axis
                         averaging over all exchanging particles */
               for (n=0;n<3;n++)
                    value[ts] += u[n]*(x[ts][pn][n] - x[0][pn][n]);
          }
          count++;
     }

     for (ts=0;ts<NT;ts++) value[ts] /= count;

     /* Measure instanton position */

     for (ts=0;ts<NT;ts++) {
          if (value[ts] >= (1.-MULTFACT)) {
               in_ts = ts;
               break;
          }
     }
     ts1 = in_ts;
```

```
    for (ts=(NT-1);ts>=0;ts--) {
        if (value[ts] <= MULTFACT) {
            width = in_ts - ts;
            ts2 = ts;
            in_ts = (in_ts + ts)/2;
            break;
        }
    }

    *retwidth = width;
    fprintf(stderr,"center = %d; width = %d ts1->ts2 = %d->%d\n",
            in_ts,width,ts1,ts2);
    fflush(stderr);
    if (width < 0) printf("ERROR: negative instanton\n");

    if (in_ts>upts || in_ts<downts) { /*Shift lattice if necessary */
        shift(in_ts,extype);
        return(check_inst(extype,retwidth));
    }
    return(in_ts);
}

plot_inst(extype)
int extype;
{
    int n,ts,pn,count;
    float mod,u[3],value[NT];
    FILE *fp;
    char name[50],choice[10];

    ex_latt();     /* make sure all nodes have the correct lattice */

    printf("Enter \"s\" for single particle inst or \"t\" for the average\n");
    scanf("%s",choice);

    switch(choice[0]) {

    case('s') :

        printf("Enter particle number to print inst for\n");
        scanf("%d",&pn);

        mod = 0.;
        for (n=0;n<DIM;n++) {
            u[n] = x[NT-1][pn][n] - x[0][pn][n];
            mod += u[n]*u[n];
        }

        if (mod==0.) {
            printf("Ground state instanton projected in x-dir\n");
            u[0] = 1; u[1] = 0.; u[2] = 0.;
            mod = 1.;
        }

        for (n=0;n<DIM;n++) u[n] = u[n]/mod;

        for (ts=0;ts<NT;ts++) {
            value[ts] = 0.;
```

```
            for (n=0;n<DIM;n++)
                value[ts] += u[n]*(x[ts][pn][n] - x[0][pn][n]);
        }

        break;

    case('t') :

        count = 0;
        for (ts=0;ts<NT;ts++) value[ts] = 0.;

        while(xcon[extype][count][0] != -1) {
            pn = xcon[extype][count][0];
            mod = 0.;
            for (n=0;n<DIM;n++) {
                u[n] = x[NT-1][pn][n] - x[0][pn][n];
                mod += u[n]*u[n];
            }

            if (mod==0.) {
                printf("Ground state instanton projected in x-dir\n");
                u[0] = 1; u[1] = 0.; u[2] = 0.;
                mod = 1.;
            }

            /* dividing by the square of the norm normalizes all particle
               paths for the exchanges to unit length
            */
            for (n=0;n<DIM;n++) u[n] = u[n]/mod;

            for (ts=0;ts<NT;ts++) {
                for (n=0;n<DIM;n++)
                    value[ts]+= u[n]*(x[ts][pn][n]-x[0][pn][n]);
            }
            count++;
        }
        for (ts=0;ts<NT;ts++) value[ts] /= count;

        break;

    default:

        return;

    }
    do {
        printf("Enter file name to output values to\n");
        scanf("%s",name);
    } while ((fp=fopen(name,"w"))==(FILE *)0);

    for (ts=0;ts<NT;ts++) fprintf(fp,"%f\n",value[ts]);
    fclose(fp);

}

shift(tsinst,extype)
int tsinst,extype;
{
    int pn,ts,dts,n;
```

```
        printf("tsinst=%d\n",tsinst);
        if (tsinst>(NT/2)) {
             dts = tsinst - (NT/2);
             printf("SHIFTING instanton to center from above, dts=%d\n",dts);
             for (pn=0;pn<NP;pn++) {
                  for (ts=1;ts<NT-dts;ts++) {
                       for (n=0;n<DIM;n++) {
                            x[ts][pn][n] = x[ts+dts][pn][n];
                       }
                  }
                  for (ts=(NT-dts);ts<(NT-1);ts++) {
                       for (n=0;n<DIM;n++) {
                            x[ts][pn][n] = x[NT-1][pn][n];
                       }
                  }
             }
        } else {
             dts = (NT/2) - tsinst;
             printf("SHIFTING instanton to center from below, dts=%d\n",dts);
             for (pn=0;pn<NP;pn++) {
                  for (ts=NT-2;ts>dts;ts--) {
                       for (n=0;n<DIM;n++) {
                            x[ts][pn][n] = x[ts-dts][pn][n];
                       }
                  }
                  for (ts=(dts);ts>0;ts--) {
                       for (n=0;n<DIM;n++) {
                            x[ts][pn][n] = x[0][pn][n];
                       }
                  }
             }
        }

        if (updateall) {
             printf("Doing 5 updates after shift\n");
             fflush(stdout);
             update_thr(5,5);
        } else {
             printf("Doing 5 updates after shift of exchanging set\n");
             fflush(stdout);
             update_thr(5,extype);
        }

}

/*****************************************************************/
/*                                                               */
/*      This software is copyrighted by Caltech, 1985            */
/*      Author: John Salmon, Steve Otto                          */
/*        Modified by Sean Callahan for this application.
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <math.h>
#include "parms.h"
```

```
/*
        Some routines to do parallel random number generation.  The idea is
        to take a standard linear congruential algorithm, and have every
        processor compute the Pth iterate of that algorithm (where P is the
        number of processors.)  If the seeds are set up properly, the processors
        leapfrog over one another, and it is just as good as having used the
        basic algorithm on a sequential machine.
*/

#define MULT    1103515245
#define ADD     12345
#define MASK    ( 0xffffffff )
static double _twoto32 = 4294967296.;
extern FILE *seed_fp;
extern doc;

unsigned int AAA,BBB,randx;
unsigned int tmprandx;

unsigned int seedgt()
{
     return(randx);
}

pranset(seed)
/*
Unlike the sequential algorithm, you MUST call pranset, even if you want
a seed of 1.  Otherwise, all processors will produce the same sequence.
*/
unsigned int seed;
{
     int nlayers,tcoord;

     nlayers = gdim[0];
     AAA = 1;
     BBB = 0;
     for(tcoord=0; tcoord<nlayers; tcoord++){
          AAA = (MULT * AAA) & MASK;
          BBB = (MULT * BBB + ADD) & MASK;
          if (tcoord == pcoord[0]) randx = (AAA*seed + BBB) & MASK;
     }
}

float pranf()
/* Return a random float in [0, 1.0) */
{
     float retvalue;

     retvalue=(float)randx / _twoto32 ;
     randx = (AAA*randx + BBB)& MASK;
     return( retvalue );
}

static float norm_sv;
static int flag = 1;
float normal()
/*
This is the Polar method for normal distributions, as described on or near
page 104 of Knuth, Seminumerical Algorithms.  To quote Knuth, "The polar
```

```
method is quite slow, but it has essentially perfect accuracy, and it is very
easy to write a program for the polar method..."  'nuf said.  Algorithm due
to Box, Muller and Marsaglia.
*/
{
     float v1, v2; /* uniformly distributed on [-1, 1) */
     float s;      /* radius squared of a point pulled from a uniform circle */
     float  foo;   /* A useful intermediate value. */
     float pranf();

     if (flag) {
          do{
               v1 = 2.0 * pranf()- 1.0;
               v2 = 2.0 * pranf()- 1.0;
               s = v1*v1 + v2*v2;
          } while(s >= 1.0);
          foo = sqrt( -2.0 * log(s)/s);
          norm_sv = v2*foo;
          flag = 0;
          return(v1*foo);
     }
     flag = 1;
     return(norm_sv);
}

backup_seed()
{
     unsigned int mask;
     tmprandx = seedgt();
     /* I am making the assumption that pcoord[] = (0,0) is node 0 */
#ifdef PAR
     mask = (1<<doc) -1;
     broadcast(&tmprandx,0,mask,sizeof(int));
#endif
     rewind(seed_fp);
     fprintf(seed_fp,"%u\n",tmprandx);
     fflush(seed_fp);
     if (ferror(seed_fp)) abort(5566);
}

/*** p.h ***                                    Sean Callahan */
/* parameters defined for the two-particle density matrix program */


#define XDIM    70
#define LDIM    32
#define NTRM    9
#define XDIM2 2485  /* = XDIM*(XDIM+1)/2 */
#define NDMS    8
#define NMT     3        /* NMT is the maximum order for polynomial fitting */
#define LEVELS 14   /* the number of levels down the matrix squaring goes */

/*#define RMIN  1.550 */
/*#define CUTR  5.5 /* zero potential is at CUTR ( in angstroms) */
#define CUTR    12.5     /* zero potential is at CUTR ( in angstroms) */
#define TMID  10.0  /* average temp of highest beta */

#define HBS2M 8.0420/10.22
```

```
/* this uses effective mass */
#define EHBS2M 2*8.0420/10.22
#define IP       2

/*** u.c ***                                          Sean Callahan */

#include <math.h>
#include <stdio.h>
#include "parms.h"

int nkt[3] = {1,2,5};
float fit[2][NBETA][NDIM][6],rv[NDIM],chi[NBETA][NDIM][3];
int nfit[NDIM][NBETA];

/* these are arguments for interp() */
float r,a1,a2,a3,a4;
int ix;

/* Return only the endpoint approximation */
float u0(bind)
int bind;
{
float ui[4];
int i,ii;
float retu0;
bind += BIND_OFS;

if (bind > (NBETA-1)) {
     printf("ERROR in u0(): r12,r12p,bind = %f %f %d\n",r12,r12p,bind);
     exit(0);
}

r = .5*(r12 + r12p);
interp();

for (i=0;i<4;i++) {
     ii = ix + i - 1;
     ui[i] = fit[0][bind][ii][0];
}
retu0 = a1*ui[0] + a2*ui[1] + a3*ui[2] + a4*ui[3];
return(retu0);
}


/* Return -ln(rho_2 tilde) */
float u(bind)
int bind;
{
float t[5],ui[4];
int i,ii,j;
float retu;
bind += BIND_OFS;

r = .5*(r12 + r12p);
interp();

t[0] = r12*r12 + r12p*r12p - 2*r12*r12p*costh;
t[1] = (r12 - r12p)*(r12 - r12p);
```

```
t[2] = t[0]*t[1];
t[3] = t[0]*t[0];
t[4] = t[1]*t[1];

for (i=0;i<4;i++) {
     ii = ix + i - 1;
     ui[i] = fit[0][bind][ii][0];
     if (nfit[ii][bind] != 0) {
          for (j=0;j<nkt[nfit[ii][bind]];j++) {
               ui[i] += t[j]*fit[0][bind][ii][j+1];
          }
     }
}

retu = a1*ui[0] + a2*ui[1] + a3*ui[2] + a4*ui[3];
return(retu) ;
}

/* This is the beta derivative of U     (the effective potential) */
float uprime(bind)
int bind;
{
float t[5],ui[4];
int i,ii,j;
float retu;
bind += BIND_OFS;

r = .5*(r12 + r12p);
interp();

t[0] = r12*r12 + r12p*r12p - 2*r12*r12p*costh;
t[1] = (r12 - r12p)*(r12 - r12p);
t[2] = t[0]*t[1];
t[3] = t[0]*t[0];
t[4] = t[1]*t[1];

for (i=0;i<4;i++) {
     ii = ix + i - 1;
     ui[i] = fit[1][bind][ii][0];
     if (nfit[ii][bind] != 0) {
          for (j=0;j<nkt[nfit[ii][bind]];j++) {
               ui[i] += t[j]*fit[1][bind][ii][j+1];
          }
     }
}

retu = a1*ui[0] + a2*ui[1] + a3*ui[2] + a4*ui[3];
return(retu) ;
}



/* Read in the density matrix fit parameters
*/
getd()
{
float tmpu[5][3][2],beta,a,b,c,d,*flptr;
int nf,jj,k,kk,nmt,n,ir,ib,in;
FILE *fp;
```

```
nmt = 3;

if ((fp = fopen("dmhe3","r"))==(FILE *)0) {
     printf("ERROR: Can't find the file dmhe3\n");
     exit(248);
}

for (ib=0;ib<NBETA;ib++) {
     fscanf(fp,"%d %f %f %f %f %f",&n,&beta,&a,&b,&c,&d);
     for (ir=1;ir<NDIM;ir++) {
          fscanf(fp,"%f",&rv[ir]);
          flptr = &chi[ib][ir][0];
          for (jj=0;jj<nmt;jj++) fscanf(fp,"%f",flptr++);
          flptr = &chi[ib][ir][0];
          for (in=0;in<nmt;in++) {
               k = nkt[in];
               for (kk=0;kk<k;kk++) fscanf(fp,"%f",&tmpu[kk][in][0]);
               for (kk=0;kk<k;kk++) fscanf(fp,"%f",&tmpu[kk][in][1]);
          }

          nf = 0;
          if (chi[ib][ir][1] < chi[ib][ir][0]) nf = 1;
          if (chi[ib][ir][2] < chi[ib][ir][1]) nf = 2;

          nfit[ir][ib] = nf;

          fit[0][ib][ir][0] = tmpu[0][0][0];
          fit[1][ib][ir][0] = tmpu[0][0][1];

          if (nf != 0) {
               for (jj=0;jj<nkt[nf];jj++) {
                    fit[0][ib][ir][jj+1] = tmpu[jj][nf][0];
                    fit[1][ib][ir][jj+1] = tmpu[jj][nf][1];
               }
          }
     }
}

fclose(fp);
}

/* Return interpolation information given the sum coordinate
*/
interp()
{
float xx,p,pm1,pm2,pp1,amin1();

xx = cx/(r*r);
xx = amin1(xx,(float)(NDIM));

/*  xx should be in range 0 to (NDIM-1) */

ix = xx;
ix = max0(ix,1);
ix = min0(ix,NDIM-3);
p = xx-ix;
pm1 = p-1.;
pm2 = p-2.;
```

```
pp1 = p+1.;

#define si .16666666666666666

a1 = -si*p*pm1*pm2;
a2 = .5*pm1*pp1*pm2;
a3 = -.5*p*pp1*pm2;
a4 = si*p*pp1*pm1;
}
```

Appendix D

The He-He Interaction Potential

Two different representations are used for the He-He potential. The first is the potential of Aziz *et al.*,[1] which is used for interatomic spacings greater than 1.828 Å. The Ceperley-Partridge form [2] is used for separations less than that distance.

D.1 The Aziz Form

The form used by Aziz *et al.* is

$$V(r) = \epsilon V^*(x); \qquad x = \frac{r}{r_m} \tag{D.1}$$

$$V^*(x) = Ae^{-\alpha x} - \left\{\frac{C_6}{x^6} + \frac{C_8}{x^8} + \frac{C_{10}}{x^{10}}\right\} F(x) \tag{D.2}$$

$$\begin{aligned} F(x) &= \exp\left\{-\left(\frac{D}{x} - 1\right)^2\right\} \qquad x \leq D \\ &= 1 \qquad x > D. \end{aligned} \tag{D.3}$$

This form was used to fit the data for intermediate temperature virial coefficient and thermal conductivity data, and high temperature viscosity data. The fit

parameters found are listed below.

$$D = 1.241314$$

$$r_m = 2.9673\text{Å}$$

$$\epsilon = 10.8\text{K}$$

$$A = 544850.4$$

$$\alpha = 13.353384$$

$$C_6 = 1.3732412$$

$$C_8 = .4253785$$

$$C_{10} = .1781$$

In the words of Aziz *et al.*, "In spite of a few remaining inconsistencies, when all the different macroscopic properities are considered, the potential produces the best representation of the helium interaction potential available at this time." See Fig 29 for a plot of this potential.

D.2 The Ceperley-Partridge Form

The Ceperley-Partridge form, valid for r<1.828 Å, is given by

$$V(r) = \exp(-\beta\frac{r}{r_b}) \sum_{k=-1}^{4} a_k \Big(\frac{r}{r_b}\Big)^k. \qquad (D.4)$$

The parameters have been fit to shock wave experiments at high temperature and

pressure. The fit parameters are

$$\beta = 3.32316$$

$$r_b = .52910$$

$$a_{-1} = 1.263030 \times 10^6 \mathrm{K}$$

$$a_0 = 1.399549 \times 10^6 \mathrm{K}$$

$$a_1 = -8.389601 \times 10^5 \mathrm{K}$$

$$a_2 = 7.426020 \times 10^6 \mathrm{K}$$

$$a_3 = -2.006420 \times 10^6 \mathrm{K}$$

$$a_4 = 8.570426 \times 10^5 \mathrm{K}.$$

This potential gives a much better account of the data in the region where r<1.8 Å, where the Aziz potential is too repulsive.

References

1. R. A. Aziz, V. P. S. Nain, J. s. Carley, W. L. Taylor, and G. T. McConville, *Jour. Chem. Phys.*, **9**, 4330, May 1979.

2. D. M. Ceperley and H. Partridge, *Jour. Chem. Phys.*, **84**, 820, Jan. 1986.

Appendix E

Density Matrix Coefficients

The following is a listing of the fit parameters from which $U(r_{12}, r'_{12}, \theta; \beta)$ is calculated (see Eqn. (6.18)). The value of β is listed above each set of data, and the columns listed from left to right are, the sum coordinate ($r = .5(r_{12} + r'_{12})$), the nonlinear array index, and U_0 through U_5, respectively. It should be noted that the shape of U is not very smooth for values of r that are less than approximately 1.7 Å, especially for large angular separations. This is not significant, however, for this application because the penetration to that level makes effectively zero contribution to the calculations which have been done for this work. This has been verified by histogramming events as a funcion of the radial sum coordinate ($.5 * (r_{12} + r'_{12})$).

```
beta = 0.003125

12.88   1 -6.5679e-06          0          0          0          0          0
 9.10   2 -5.7091e-05          0          0          0          0          0
 7.43   3 -.00019754           0          0          0          0          0
 6.44   4 -.00047821           0          0          0          0          0
 5.76   5 -.00095334           0          0          0          0          0
 5.26   6  -.0016809           0          0          0          0          0
 4.87   7   -.002721           0          0          0          0          0
 4.55   8  -.0041318           0          0          0          0          0
 4.29   9  -.0059629           0          0          0          0          0
 4.07  10   -.008247           0          0          0          0          0
 3.88  11    -.01099           0          0          0          0          0
```

3.72	12	-.014152	0	0	0	0	0
3.57	13	-.017629	0	0	0	0	0
3.44	14	-.021242	0	0	0	0	0
3.32	15	-.024751	-.00050119	-.0028881	0	0	0
3.22	16	-.027853	-.0008696	-.0033843	.0010039	2.0257e-05	-.010386
3.12	17	-.030179	-.00072066	-.0096852	.0014389	6.7423e-05	-.016309
3.03	18	-.031291	-.00041559	-.019282	.0017965	.00024027	-.024221
2.95	19	-.030692	.0001433	-.033146	.0021519	.00052702	-.034373
2.88	20	-.02783	.0010811	-.052344	.0026408	.00089228	-.046919
2.81	21	-.022106	.002548	-.078007	.0034557	.0012803	-.061847
2.75	22	-.012881	.0047622	-.11135	.0050745	.0015325	-.079107
2.68	23	.00050702	.0077327	-.15316	.0069819	.0018675	-.09791
2.63	24	.018738	.01158	-.20428	.0091523	.002315	-.11753
2.58	25	.042492	.01644	-.26535	.011676	.0028674	-.13723
2.53	26	.072441	.02244	-.33676	.014618	.0035229	-.15624
2.48	27	.10923	.029679	-.41862	.017949	.004311	-.17368
2.43	28	.15349	.038222	-.5106	.021589	.005285	-.18879
2.39	29	.20577	.0483	-.6126	.026093	.0062217	-.20111
2.35	30	.26659	.059864	-.72352	.030973	.0073118	-.21011
2.31	31	.33643	.072951	-.84249	.036135	.0085779	-.21536
2.28	32	.41565	.087608	-.9682	.041517	.010007	-.21702
2.24	33	.5046	.104	-1.1005	.047014	.011489	-.21308
2.21	34	.60352	.12181	-1.2361	.052527	.013249	-.20733
2.18	35	.71259	.14118	-1.3747	.057925	.015157	-.19825
2.15	36	.83194	.16205	-1.515	.06281	.017306	-.1858
2.12	37	.96123	.18371	-1.6548	.064063	.020808	-.16866
2.09	38	1.1012	.20755	-1.7957	.06775	.023272	-.14972
2.06	39	1.2514	.23284	-1.9357	.071117	.025746	-.12749
2.04	40	1.4117	.25937	-2.0742	.073505	.02844	-.10135
2.01	41	1.582	.28726	-2.2107	.075204	.031136	-.070791
1.99	42	1.762	.31645	-2.3449	.075783	.033879	-.034673
1.96	43	1.9516	.34735	-2.4796	.075146	.036214	.017796
1.94	44	2.1504	.3798	-2.6151	.072674	.038254	.092177
1.92	45	2.3579	.40958	-2.7183	.074767	.042506	.074165
1.90	46	2.5739	.44275	-2.8384	.073913	.044762	.13228
1.88	47	2.798	.47634	-2.9544	.069858	.047631	.20569
1.86	48	3.0297	.51083	-3.0676	.063637	.050384	.30132
1.84	49	3.2684	.54772	-3.1958	.04893	.052146	.52218
1.82	50	3.5135	.58032	-3.2637	.048563	.05713	.46044
1.80	51	3.7647	.61701	-3.3628	.033997	.059567	.62876
1.79	52	4.0216	.65553	-3.4744	.0089705	.060974	.97747
1.77	53	4.2837	.69058	-3.5294	.0066294	.064232	.95793
1.75	54	4.5505	.72621	-3.612	-.01277	.066743	1.2549
1.74	55	4.8221	.76531	-3.7375	-.063664	.066693	2.2035
1.72	56	5.0983	.79922	-3.7529	-.051532	.069787	1.9906
1.71	57	5.3791	.83857	-3.8665	-.14	.070373	3.4736
1.69	58	5.6639	.87466	-3.877	-.15196	.071809	3.5111
1.68	59	5.9523	.91421	-3.9722	-.30458	.071479	6.0818
1.66	60	6.244	.94912	-3.9026	-.30787	.072649	5.4883
1.65	61	6.5379	.98613	-3.8873	-.58386	.07403	8.9537
1.64	62	6.8344	1.0214	-3.713	-.62263	.072414	7.8
1.62	63	7.1327	1.0581	-3.4969	-.66478	.068395	4.7148
1.61	64	7.4308	1.0937	-3.4668	-1.2614	.06328	16.353
1.60	65	7.7275	1.123	-3.3984	-1.4207	.056995	19.299
1.58	66	8.024	1.1536	-3.6277	-1.6702	.046984	57.49
1.57	67	8.3161	1.1822	-3.8547	-1.5121	.032918	147.05
1.56	68	8.6051	1.219	-4.1766	0	0	0
1.55	69	8.8871	1.2266	-3.0673	0	0	0

beta = 0.006250

12.88	1	-1.3149e-05	0	0	0	0	0
9.10	2	-.00011442	0	0	0	0	0
7.43	3	-.00039589	0	0	0	0	0
6.44	4	-.00095876	0	0	0	0	0
5.76	5	-.001913	0	0	0	0	0
5.26	6	-.0033775	0	0	0	0	0
4.87	7	-.0054762	0	0	0	0	0
4.55	8	-.008329	-.00029252	.001871	0	0	0
4.29	9	-.012036	-.0004503	.0028576	0	0	0
4.07	10	-.016656	-.00062486	.0038073	0	0	0
3.88	11	-.022182	-.00078905	.0042525	0	0	0
3.72	12	-.028495	-.0013979	.0063946	1.6049e-05	.00014232	-.0024797
3.57	13	-.035333	-.0017432	.0064124	.00017838	.00017692	-.0053369
3.44	14	-.042278	-.0020893	.0047533	.0003956	.00025706	-.0099592
3.32	15	-.048765	-.0023174	.00013081	.00073589	.00037244	-.016703
3.22	16	-.054084	-.0022079	-.0092909	.0012739	.00048882	-.025527
3.12	17	-.05738	-.0015643	-.025486	.0020745	.00059556	-.036029
3.03	18	-.05767	-.00018459	-.05039	.0031753	.00069562	-.047503
2.95	19	-.054272	.0019942	-.085242	.0045512	.00086097	-.059243
2.88	20	-.045268	.0051616	-.13097	.0064483	.0010604	-.070672
2.81	21	-.029834	.0093414	-.18767	.0087175	.0013995	-.081054
2.75	22	-.0068	.014686	-.25506	.011442	.0018733	-.08992
2.68	23	.024949	.021293	-.33225	.014609	.0025014	-.096999
2.63	24	.066432	.029247	-.41807	.018087	.0033175	-.10199
2.58	25	.11854	.038572	-.5107	.021829	.0043361	-.10523
2.53	26	.18203	.049348	-.60852	.025833	.0055222	-.10699
2.48	27	.2575	.061595	-.70989	.029997	.0068705	-.10753
2.43	28	.34537	.075523	-.81458	.034318	.0082875	-.10593
2.39	29	.44595	.090741	-.91918	.038535	.0099156	-.10436
2.35	30	.55939	.10739	-1.0236	.04263	.011667	-.1021
2.31	31	.68572	.12548	-1.1271	.046567	.013502	-.099236
2.28	32	.82486	.14499	-1.2291	.050433	.015359	-.095929
2.24	33	.97665	.16648	-1.3336	.054	.017023	-.086635
2.21	34	1.1408	.18817	-1.4284	.057844	.019039	-.086362
2.18	35	1.3172	.21116	-1.522	.061115	.021088	-.083686
2.15	36	1.5053	.23526	-1.6133	.064075	.023149	-.08001
2.12	37	1.7048	.26024	-1.7013	.066613	.025309	-.076413
2.09	38	1.9153	.28605	-1.786	.06865	.027546	-.072884
2.06	39	2.1365	.31282	-1.8678	.070818	.029663	-.06983
2.04	40	2.3679	.34034	-1.9464	.072631	.031791	-.066997
2.01	41	2.6091	.36861	-2.0222	.074192	.033873	-.064264
1.99	42	2.8595	.39792	-2.0963	.076784	.035557	-.062005
1.96	43	3.1189	.42776	-2.1715	.076883	.037511	-.045487
1.94	44	3.3867	.45818	-2.2482	.074478	.039605	-.011431
1.92	45	3.6626	.48696	-2.2981	.079965	.042183	-.059803
1.90	46	3.946	.51731	-2.3604	.079722	.044573	-.058283
1.88	47	4.2365	.54755	-2.4187	.078215	.04738	-.059259
1.86	48	4.5335	.57763	-2.4723	.072493	.050738	-.061228
1.84	49	4.8365	.61014	-2.5455	.061221	.052909	.042705
1.82	50	5.1449	.6406	-2.5783	.06907	.05554	-.080549
1.80	51	5.4581	.67252	-2.629	.064362	.057996	-.086013
1.79	52	5.7759	.70534	-2.6928	.048623	.06035	.037024
1.77	53	6.0979	.73678	-2.7246	.054296	.062971	-.11638
1.75	54	6.4238	.77558	-2.803	.061627	.060065	-.014942
1.74	55	6.7535	.81381	-2.909	.040028	.058293	.554
1.72	56	7.0867	.84513	-2.931	.076373	.060677	.1465
1.71	57	7.4231	.87916	-3.0078	.018914	.062778	.85315

1.69	58	7.7626	.91099	-3.044	.064549	.065065	.48059
1.68	59	8.1048	.94551	-3.1193	-.046064	.067109	1.809
1.66	60	8.4496	.97737	-3.1527	.040469	.06931	.92113
1.65	61	8.7965	1.0128	-3.2028	-.23099	.071299	3.5341
1.64	62	9.1452	1.0444	-3.2401	-.15295	.073667	2.6407
1.62	63	9.4953	1.0698	-3.2172	-.097414	.083066	-.028252
1.61	64	9.8462	1.1058	-3.1544	-.92628	.084668	6.6711
1.60	65	10.197	1.1383	-3.2093	-1.097	.086276	8.7728
1.58	66	10.547	1.1705	-3.409	-1.267	.087756	23.476
1.57	67	10.896	1.2025	-4.0223	-1.4199	.089166	115.9
1.56	68	11.242	1.2345	-6.6708	-1.3492	.090366	1176.3
1.55	69	11.583	1.2731	-2.9451	-20.47	.08762	1077

beta = 0.012500

12.88	1	-2.6186e-05	0	0	0	0	0
9.10	2	-.00022581	0	0	0	0	0
7.43	3	-.00077922	0	0	0	0	0
6.44	4	-.0018967	0	0	0	0	0
5.76	5	-.0038144	-.00012304	.00058965	0	0	0
5.26	6	-.0067887	-.00023147	.0012631	0	0	0
4.87	7	-.011084	-.00045789	.0020703	3.1062e-07	2.8843e-05	.00017247
4.55	8	-.016949	-.00064722	.0041022	0	0	0
4.29	9	-.024575	-.0010758	.0066264	-8.8252e-06	4.4967e-05	-.00047652
4.07	10	-.034473	-.0014467	.0096944	1.5494e-06	4.2611e-05	-.0018469
3.88	11	-.045718	-.0019816	.012648	7.7368e-05	6.1585e-05	-.0046038
3.72	12	-.058236	-.0025574	.013635	.00028143	9.7036e-05	-.0089791
3.57	13	-.071232	-.0028937	.0095712	.00066276	.00013705	-.014598
3.44	14	-.083538	-.0027217	-.0025335	.0013245	.00017449	-.020874
3.32	15	-.093647	-.0018877	-.024816	.0023078	.00024542	-.027088
3.22	16	-.099791	-.00026051	-.058204	.0036648	.00038412	-.032658
3.12	17	-.10004	.0022506	-.10229	.0054031	.00063023	-.037241
3.03	18	-.092463	.0056826	-.1554	.0074592	.001028	-.040818
2.95	19	-.075216	.01029	-.21644	.0097985	.0015597	-.043081
2.88	20	-.046676	.015966	-.28197	.012336	.0022541	-.044821
2.81	21	-.0055486	.023108	-.35187	.015063	.0030367	-.045406
2.75	22	.049184	.031402	-.42226	.017806	.0039621	-.045995
2.68	23	.11819	.041078	-.49292	.02052	.0049689	-.046218
2.63	24	.20188	.052271	-.56421	.023336	.006006	-.045633
2.58	25	.30037	.064609	-.63281	.025981	.0071155	-.045466
2.53	26	.41363	.078189	-.6996	.028517	.0082558	-.045133
2.48	27	.54145	.092959	-.76426	.03103	.0093941	-.04479
2.43	28	.6835	.10867	-.82667	.033724	.010571	-.04468
2.39	29	.83941	.12523	-.8857	.035992	.011806	-.04476
2.35	30	1.0087	.14257	-.94192	.038127	.01308	-.045088
2.31	31	1.1909	.16055	-.9952	.040042	.014419	-.045705
2.28	32	1.3855	.17917	-1.0458	.041865	.015779	-.046712
2.24	33	1.5919	.19891	-1.0987	.04299	.01708	-.042495
2.21	34	1.8096	.21823	-1.1411	.045349	.018507	-.048547
2.18	35	2.0382	.23844	-1.1842	.047194	.019871	-.05174
2.15	36	2.277	.25957	-1.2271	.049334	.021052	-.054217
2.12	37	2.5257	.28145	-1.2694	.052452	.022044	-.057715
2.09	38	2.7839	.30269	-1.3076	.053929	.023516	-.061089
2.06	39	3.051	.32425	-1.3443	.055291	.02501	-.064886
2.04	40	3.3266	.34611	-1.3796	.056503	.026527	-.069113
2.01	41	3.6103	.36816	-1.4136	.057794	.028087	-.074089
1.99	42	3.9017	.39028	-1.4462	.059165	.029734	-.080364
1.96	43	4.2004	.41267	-1.4788	.057225	.031565	-.077531
1.94	44	4.506	.43596	-1.5134	.053514	.033173	-.0648

1.92	45	4.818	.45826	-1.5366	.059927	.034806	-.10493
1.90	46	5.1361	.48147	-1.5667	.060536	.036397	-.11336
1.88	47	5.4599	.50482	-1.5964	.060991	.038004	-.12175
1.86	48	5.789	.5283	-1.626	.061232	.039624	-.1296
1.84	49	6.1227	.55303	-1.6579	.04914	.041069	-.090052
1.82	50	6.4606	.57569	-1.6832	.061901	.042855	-.158
1.80	51	6.8023	.60518	-1.7372	.046626	.04257	-.052358
1.79	52	7.1473	.66049	-1.9661	.091294	.03004	.46898
1.77	53	7.4956	.6933	-2.1207	.16554	.026638	.84656
1.75	54	7.847	.72042	-2.2002	.17942	.026848	1.1089
1.74	55	8.2012	.74934	-2.256	.144	.026785	1.4336
1.72	56	8.558	.77551	-2.4638	.2298	.027021	2.6782
1.71	57	8.9172	.80498	-2.5287	.18174	.026763	3.2268
1.69	58	9.2787	.83158	-2.8648	.31116	.026871	6.4958
1.68	59	9.6422	.86231	-2.9133	.1961	.026322	7.3251
1.66	60	10.007	.88853	-3.5072	.44825	.026401	16.038
1.65	61	10.374	.99019	-3.3476	0	0	0
1.64	62	10.742	1.0187	-3.8892	0	0	0
1.62	63	11.111	1.0475	-4.7095	0	0	0
1.61	64	11.48	1.0671	-4.8303	0	0	0
1.60	65	11.849	1.0934	-6.1849	0	0	0
1.58	66	12.217	1.1196	-8.5961	0	0	0
1.57	67	12.583	1.1452	-13.482	0	0	0
1.56	68	12.946	1.1703	-25.7	0	0	0
1.55	69	13.305	1.1854	-28.818	0	0	0

beta = 0.025000

12.88	1	-4.7942e-05	0	0	0	0	0
9.10	2	-.00041099	0	0	0	0	0
7.43	3	-.0014828	-6.3467e-05	.00018356	0	0	0
6.44	4	-.0037336	-.00013289	.00060433	0	0	0
5.76	5	-.0081141	-.00013121	.00099572	-4.6281e-06	-2.1453e-06	9.5559e-05
5.26	6	-.014502	-.00031418	.0026216	-1.3915e-05	-2.7883e-06	.00012425
4.87	7	-.023769	-.00072965	.0059925	-2.4391e-05	3.9198e-06	-.00017079
4.55	8	-.036421	-.001457	.011394	-2.3569e-05	2.2527e-05	-.0012031
4.29	9	-.052697	-.0022578	.016752	4.4952e-05	3.3622e-05	-.0032518
4.07	10	-.072299	-.002904	.018102	.00025897	3.4519e-05	-.0060522
3.88	11	-.094144	-.0031501	.011263	.00071013	3.6813e-05	-.0090674
3.72	12	-.11627	-.0029646	-.0057114	.0014505	8.4014e-05	-.011839
3.57	13	-.13596	-.0023434	-.032261	.0024795	.00022087	-.014183
3.44	14	-.15015	-.0010417	-.067349	.0037543	.00046103	-.015874
3.32	15	-.15584	.00081316	-.10685	.005168	.00083498	-.017357
3.22	16	-.1504	.0038628	-.15135	.0067479	.0012799	-.018135
3.12	17	-.13174	.0077733	-.1955	.0083004	.0018257	-.019065
3.03	18	-.098346	.013013	-.24017	.0097684	.0024222	-.019674
2.95	19	-.049407	.019586	-.28588	.011472	.0030403	-.01998
2.88	20	.015385	.027073	-.32863	.012817	.0037276	-.020318
2.81	21	.096044	.035513	-.37074	.014461	.004429	-.020722
2.75	22	.19243	.044721	-.40951	.015692	.0051814	-.021123
2.68	23	.30428	.054789	-.44623	.016864	.0059438	-.021588
2.63	24	.43119	.065495	-.48136	.018415	.0067221	-.022562
2.58	25	.57261	.076985	-.51363	.019499	.0075186	-.023465
2.53	26	.72788	.08918	-.54413	.020694	.008296	-.024742
2.48	27	.8963	.10135	-.57175	.021597	.0091937	-.026272
2.43	28	1.0772	.11358	-.59845	.022883	.010152	-.028134
2.39	29	1.2701	.12605	-.62199	.023608	.011159	-.030296
2.35	30	1.4744	.13855	-.64343	.02417	.012235	-.032875
2.31	31	1.6896	.15136	-.66364	.024762	.013313	-.035929

2.28	32	1.9155	.16482	-.68369	.02564	.014324	-.039488
2.24	33	2.1514	.17901	-.70429	.024257	.015374	-.037754
2.21	34	2.3971	.19162	-.71967	.026176	.016626	-.045489
2.18	35	2.652	.20457	-.73371	.027081	.017928	-.052348
2.15	36	2.9158	.2181	-.74618	.027061	.019222	-.059276
2.12	37	3.1879	.2317	-.75684	.026874	.020567	-.067942
2.09	38	3.468	.24837	-.77065	.023596	.021479	-.070877
2.06	39	3.7556	.26827	-.79827	.021051	.021751	-.063433
2.04	40	4.0505	.29321	-.85214	.031895	.02048	-.059727
2.01	41	4.3521	.31266	-.88559	.040011	.020466	-.067065
1.99	42	4.6603	.32816	-.90297	.040748	.021593	-.071989
1.96	43	4.9747	.34521	-.91687	.035497	.022559	-.071004
1.94	44	5.295	.36218	-.92585	.027272	.023594	-.071862
1.92	45	5.6208	.37461	-.95585	.04236	.025185	-.086009
1.90	46	5.952	.39017	-.9756	.04285	.026432	-.086042
1.88	47	6.288	.40581	-.99587	.043333	.027692	-.084748
1.86	48	6.6287	.42149	-1.016	.043724	.028974	-.083291
1.84	49	6.9735	.44208	-.99594	.0056707	.029671	-.10098
1.82	50	7.3219	.45287	-1.0523	.044193	.031628	-.09182
1.80	51	7.6736	.47096	-1.0818	.035942	.0326	-.028687
1.79	52	8.0282	.51913	-1.3964	.028768	.026955	.89456
1.77	53	8.3856	.56267	-1.9513	.15231	.021052	2.6894
1.75	54	8.7456	.58337	-2.0979	.17046	.021188	3.4109
1.74	55	9.1081	.60729	-2.013	.070814	.021117	3.1492
1.72	56	9.4729	.62542	-2.6616	.23618	.021352	7.6456
1.71	57	9.8397	.76069	-2.7842	0	0	0
1.69	58	10.208	.78322	-3.2413	0	0	0
1.68	59	10.579	.80109	-3.5637	0	0	0
1.66	60	10.951	.8246	-4.2764	0	0	0
1.65	61	11.324	.84042	-4.815	0	0	0
1.64	62	11.698	.86333	-6.1123	0	0	0
1.62	63	12.072	.88639	-8.0046	0	0	0
1.61	64	12.447	.89899	-9.3451	0	0	0
1.60	65	12.821	.91989	-13.395	0	0	0
1.58	66	13.195	.94173	-20.694	0	0	0
1.57	67	13.566	.96557	-35.493	0	0	0
1.56	68	13.934	.98664	-72.617	0	0	0
1.55	69	14.297	.96922	-82.509	0	0	0

beta = 0.050000

12.88	1	-5.0323e-05	0	0	0	0	0
9.10	2	-.00068031	-4.9916e-05	.00011163	0	0	0
7.43	3	-.0034151	-3.1621e-05	.00022178	-7.3393e-07	-4.0563e-07	1.7084e-05
6.44	4	-.0085038	-.00011375	.00096242	-4.0185e-06	-1.4991e-06	6.1763e-05
5.76	5	-.017497	-.00046913	.0037545	0	0	0
5.26	6	-.031843	-.0011443	.0097545	-1.887e-05	7.1251e-06	-.00055022
4.87	7	-.052711	-.0019794	.015584	3.3485e-05	6.5208e-06	-.001579
4.55	8	-.080308	-.002588	.015545	.00024226	7.2762e-07	-.002897
4.29	9	-.11299	-.003193	.0071043	.0006656	3.2016e-05	-.0041942
4.07	10	-.14709	-.003593	-.010134	.0012768	.00013101	-.0052237
3.88	11	-.17818	-.0041635	-.03066	.0019645	.00033702	-.0063066
3.72	12	-.20195	-.0038455	-.057601	.0028418	.00058982	-.0069344
3.57	13	-.21415	-.0030509	-.082593	.0035757	.00092455	-.0077172
3.44	14	-.21177	-.00085119	-.11127	.0045185	.0012834	-.0081837
3.32	15	-.19384	.0019638	-.13613	.0050712	.0017211	-.0086508
3.22	16	-.16026	.0053958	-.16226	.0059562	.0021856	-.0091291
3.12	17	-.11087	.0092361	-.18412	.0064127	.0027026	-.0095702
3.03	18	-.045363	.013766	-.20464	.0068638	.0032319	-.01005

2.95	19	.036485	.018573	-.22466	.0077255	.0037981	-.010947
2.88	20	.13442	.024278	-.24096	.0080783	.0043847	-.011824
2.81	21	.24761	.029905	-.25733	.0088377	.0050399	-.013005
2.75	22	.37491	.035958	-.26863	.0089374	.0057407	-.014426
2.68	23	.51521	.041962	-.27788	.0089933	.0064944	-.016273
2.63	24	.66775	.047433	-.28922	.0096168	.0073213	-.017968
2.58	25	.83212	.053009	-.29386	.0095452	.0081898	-.020843
2.53	26	1.0081	.058354	-.2964	.0092975	.009125	-.02428
2.48	27	1.1955	.063891	-.29812	.0090167	.010079	-.028224
2.43	28	1.394	.069179	-.30531	.0095042	.011079	-.030526
2.39	29	1.6034	.075304	-.30445	.0092612	.01205	-.036113
2.35	30	1.8231	.081747	-.30235	.0089506	.013038	-.042805
2.31	31	2.0529	.088427	-.29835	.0085384	.01405	-.05108
2.28	32	2.2922	.095309	-.29153	.0078819	.015095	-.061546
2.24	33	2.5406	.10652	-.2823	.00091563	.015892	-.061629
2.21	34	2.7977	.11407	-.28544	.0023689	.016891	-.072004
2.18	35	3.0632	.1217	-.28427	.0038986	.017901	-.086498
2.15	36	3.3365	.13126	-.27777	.0056994	.018674	-.11
2.12	37	3.6174	.13786	-.25612	.0042545	.019961	-.1379
2.09	38	3.9054	.14442	-.23179	.00254	.021293	-.17137
2.06	39	4.2002	.15094	-.20498	.00050866	.022672	-.21096
2.04	40	4.5015	.15744	-.17579	-.0018595	.024093	-.25753
2.01	41	4.809	.16393	-.14426	-.0046002	.025551	-.31204
1.99	42	5.1224	.17041	-.11033	-.0077083	.027046	-.3757
1.96	43	5.4415	.18082	-.034455	-.026499	.028259	-.47519
1.94	44	5.766	.18904	.052196	-.04365	.029707	-.61324
1.92	45	6.0957	.18987	.082678	-.023475	.031724	-.82046
1.90	46	6.4304	.19626	.1446	-.029054	.033364	-.99719
1.88	47	6.7697	.20271	.21159	-.03548	.035036	-1.2038
1.86	48	7.1133	.20918	.28256	-.042617	.036741	-1.4397
1.84	49	7.4609	.22288	.54951	-.11197	.03788	-2.0685
1.82	50	7.8119	.22214	.60932	-.068711	.040245	-2.7517
1.80	51	8.166	.22866	.7347	-.080551	.042052	-3.3422
1.79	52	8.5229	.2428	1.1178	-.18061	.043205	-4.7172
1.77	53	8.8826	.24188	1.2851	-.1246	.045739	-6.5732
1.75	54	9.2448	.24867	1.4832	-.14469	.047627	-7.9173
1.74	55	9.6094	.26641	2.4032	-.38905	.04859	-12.738
1.72	56	9.9761	.26201	2.4183	-.22169	.051529	-16.082
1.71	57	10.345	.28095	3.7246	-.58359	.052413	-25.594
1.69	58	10.716	.27555	3.8418	-.34435	.055567	-33.506
1.68	59	11.088	.29724	6.0718	-.99172	.056266	-56.116
1.66	60	11.462	.28919	6.041	-.54297	.059753	-71.961
1.65	61	11.836	.3155	10.277	-1.8602	.06013	-131.4
1.64	62	12.212	.32094	13.206	-2.2321	.062281	-226.96
1.62	63	12.589	.30947	14.271	-1.2898	.066345	-341.07
1.61	64	12.965	.34567	25.153	-5.3623	.065739	-626.75
1.60	65	13.341	.3556	36.908	-8.1275	.067521	-1334.5
1.58	66	13.716	.36614	57.211	-13.131	.069191	-3183.4
1.57	67	14.089	.37717	97.31	-23.151	.070682	-9253.3
1.56	68	14.458	.38687	211.95	-45.561	.07196	-47731
1.55	69	14.823	.51336	111.9	-100.96	.054777	14791

beta = 0.100000

12.88	1	6.0597e-05	0	0	0	0	0
9.10	2	-.0020387	-1.2603e-05	5.8034e-05	-2.3884e-07	-1.0654e-07	6.935e-06
7.43	3	-.0074836	-7.4141e-05	.00069757	-2.264e-06	-1.0629e-06	3.4772e-05
6.44	4	-.019454	-.00043026	.0040402	-1.2655e-05	-1.7635e-06	-3.8056e-05
5.76	5	-.041484	-.0013854	.011399	2.0663e-05	3.7258e-07	-.00055357

5.26	6	-.075798	-.0022082	.01155	.00013166	6.1143e-06	-.0010529
4.87	7	-.11962	-.0036762	.0057953	.00043589	5.9983e-05	-.0016924
4.55	8	-.16732	-.0050726	-.0076485	.00080947	.00016982	-.0020987
4.29	9	-.21284	-.0071469	-.018331	.0011331	.00034864	-.0027165
4.07	10	-.24656	-.0078006	-.035668	.0016796	.00054296	-.0031078
3.88	11	-.26504	-.0076775	-.047948	.0018585	.00079694	-.0035042
3.72	12	-.26936	-.0071388	-.064335	.0023935	.0010705	-.0038887
3.57	13	-.2603	-.0065818	-.074626	.002538	.0013859	-.0042776
3.44	14	-.23659	-.0057719	-.088499	.0031461	.0017054	-.0047836
3.32	15	-.19547	-.0040756	-.096257	.0032918	.0020461	-.0053435
3.22	16	-.13551	-.0017393	-.10819	.003895	.0023967	-.0059544
3.12	17	-.058132	.0013982	-.11299	.0039417	.0027762	-.0067846
3.03	18	.033718	.0049478	-.1175	.0040058	.0031749	-.0077546
2.95	19	.13762	.0078309	-.12391	.0044651	.0036271	-.0088217
2.88	20	.25262	.01102	-.12379	.0044439	.0040969	-.010552
2.81	21	.37884	.013568	-.13028	.004947	.0046108	-.011527
2.75	22	.51669	.016396	-.12762	.0048786	.0051362	-.013925
2.68	23	.66637	.01941	-.12678	.004832	.0056717	-.016218
2.63	24	.82779	.02204	-.13495	.0053003	.0062484	-.016524
2.58	25	1.0007	.02542	-.13427	.0051393	.0068148	-.018747
2.53	26	1.1847	.029125	-.1345	.0049276	.0073915	-.020792
2.48	27	1.3796	.033058	-.13394	.00466	.0079834	-.023172
2.43	28	1.585	.036329	-.13742	.0050164	.0086401	-.02459
2.39	29	1.8006	.040407	-.12949	.0046387	.0092816	-.0301
2.35	30	2.0262	.04453	-.11752	.0042331	.009948	-.038208
2.31	31	2.2615	.048697	-.10188	.0037918	.01064	-.049208
2.28	32	2.5059	.052922	-.083266	.0032879	.011357	-.063149
2.24	33	2.7591	.057727	-.056523	.00043879	.012098	-.075455
2.21	34	3.0208	.061358	-.051116	.0019847	.012894	-.08845
2.18	35	3.2905	.065367	-.031957	.0020056	.01371	-.10867
2.15	36	3.5676	.069766	-.0042283	.00091291	.014535	-.13563
2.12	37	3.852	.074167	.025929	-.00041522	.015387	-.16783
2.09	38	4.1431	.078549	.058484	-.0019965	.016268	-.20599
2.06	39	4.4405	.082899	.093195	-.003842	.017176	-.25067
2.04	40	4.7441	.087212	.12975	-.0059623	.018111	-.30238
2.01	41	5.0535	.09149	.16779	-.0083676	.019072	-.36155
1.99	42	5.3684	.095737	.20701	-.011069	.020058	-.42859
1.96	43	5.6887	.10066	.27655	-.018287	.021023	-.53764
1.94	44	6.0141	.10546	.36283	-.027575	.022022	-.68516
1.92	45	6.3443	.10846	.4157	-.02486	.023132	-.8806
1.90	46	6.6793	.11258	.47581	-.029557	.02421	-1.0477
1.88	47	7.0186	.11673	.5379	-.034765	.025306	-1.2372
1.86	48	7.3621	.12093	.601	-.040523	.026417	-1.4471
1.84	49	7.7094	.12653	.79338	-.065037	.027453	-1.9868
1.82	50	8.06	.12934	.90841	-.062036	.028689	-2.6496
1.80	51	8.4135	.13357	1.015	-.071385	.029857	-3.1558
1.79	52	8.7698	.13904	1.2545	-.10215	.030952	-4.173
1.77	53	9.1287	.14204	1.5148	-.10714	.032246	-6.004
1.75	54	9.4902	.14641	1.6802	-.12277	.033465	-7.1192
1.74	55	9.854	.1529	2.2514	-.20296	.034548	-10.51
1.72	56	10.22	.15498	2.5137	-.18519	.035976	-14.101
1.71	57	10.588	.16162	3.262	-.29525	.037081	-20.248
1.69	58	10.958	.16366	3.7713	-.2833	.038569	-28.747
1.68	59	11.33	.17101	5.0056	-.47941	.039659	-42.698
1.66	60	11.703	.17251	5.6927	-.44283	.041241	-60.356
1.65	61	12.077	.18122	8.0543	-.85623	.042268	-97.164
1.64	62	12.453	.18434	10.246	-1.0033	.043742	-164.29
1.62	63	12.829	.18565	12.87	-1.0381	.045439	-278.62
1.61	64	13.205	.19691	18.67	-2.2461	.046263	-455.27

1.60	65	13.581	.20117	27.082	-3.2048	.047695	-971.45
1.58	66	13.956	.20513	42.272	-4.8249	.049173	-2413.3
1.57	67	14.328	.20841	75.722	-7.8525	.050721	-7990.9
1.56	68	14.698	.20833	202.94	-14.104	.052542	-58124
1.55	69	15.063	.2503	196.59	-36.982	.049319	-45961

FIGURE CAPTIONS

1. Pressure/Temperature phase diagram for Solid ^{3}He from Guyer.[1]

2. Phase diagram (H-T) showing the low field (antiferromagnetic) phase from Cross and Fisher.[2]

3. The geometry of two-, three- and four-particle exchange in B.C.C. ^{3}He from Cross.[3]

4. a) shows the H-T phase diagram for solid ^{3}He. b),c),d) show the different possible phase transition structures that are consistent with experiment. They are, respectively, a first order transition line turning to a second-order line ending at an upper critical field, a first-order line ending at an upper critical field, or a first-order transition to a critical point. In the final case, there is no clear distinction between the two phases.

5. NMR spectrum from Osheroff *et al.*[4] for a solid crystal of ^{3}He containing three different domains of the antiferromagnetic phase.

6. Typical NMR spectrum for a uniaxial spin structure, from Cross and Fisher,[2] with a nonuniaxial dipole energy (The dipole energy is not described by one axis $\hat{l}$.) Evidence of the spin realignment is evident at $\gamma H = .7$.

7. The leading coefficient of the high-temperature expansion of the specific heat ($C_v \sim e_2/T^2$) as a function of molar volume, from Cross and Fisher.[2] The open points are from direct measurements, and the solid points are derived from pressure measurements.

8. The Curie-Weiss temperature θ as a function of molar volume, from Cross and Fisher.[2] The open points are from direct measurements, and the solid points are derived from pressure measurements.

9. The antiferromagnetic, ordering temperature, T_N, and the inverse of the maximum magnetization, $1/M_{max}$, plotted as a function of molar volume, from Miura *et al.*[5]

10. The Magnetization divided by the maximum magnetization, M/M_{max}, as a

function of the inverse of the temperature divided by the antiferromagnetic ordering temperature, T_N/T, from Miura *et al.*[5]

11. Mean-field phase diagram for the RDH model with J_t=.13 mK and K_p=.385 mK from Roger *et al.*[6] The circles and triangles are experimental data. The inset graph shows the NAF phase diagram for comparison with the diagram of RDH, and experiment.

12. Predicted phase diagram for the multiple exchange model (Eqn. 2.16) including two-, three- and four-particle exchange based on a mean-field calculation, from Cross.[3]

13. Plot of the specific heat through the high field to paramagnetic phase transition at two molar volumes and two magnetic fields, from Cross.[3]

14. Sequence of the generation of new positions for seven-particle threading.

15. Procedure for defining two timeslice positions whose average gives the instanton's position.

16. Common motion, which can be mistaken for an instanton when looking at the projection of only one particle's position onto the axis joining its initial and final positions.

17. An example of an instanton for two-particle exchange, and the effect of shifting it 16 timeslices toward the beginning of the path.

18. The coordinates x and y are defined by the difference vector between two particles at a given time: x lies along the difference vector and y is perpendicular to it. The coordinates u and v are used to map this configuration onto the system of a free particle in the presence of a hard wall (u=x-D,v=y).

19. Pictured are the particle at (u,v) and its image at (-u,v). The particle at (u,v) propagates to (u',v').

20. Two-particle exchange for a 2/54 system plotted as a function of the imaginary time increment, ϵ.

21. The Gruneisen constant plotted as a function of inverse system size for two-particle exchange.

22. The Gruneisen constant plotted as a function of inverse system size for three-

particle exchange.

23. The Gruneisen constant plotted as a function of inverse system size for four-particle exchange.

24. All of the Gruneisen constant data at molar volumes of 22 cm^3/mole and 24 cm^3/mole, except for the smallest system sizes, as a function of inverse system size.

25. All of the Gruneisen constant data at molar volumes of 22 cm^3/mole and 24 cm^3/mole as a function of inverse system size.

26. Three-particle exchange energy plotted as a function of nearest-neighbor lattice spacing in the triangular lattice from Roger.[7]

27. A two-dimensional triangular lattice.

28. Communication mesh used for the parallel version of the program that calculates the Gruneisen parameters.

29. The Aziz potential of Appendix D.

References

1. R. A. Guyer and A. K. Mahan, *Phys. Rev. A*, **7**, 1105, 1973.
2. M. C. Cross and Daniel S. Fisher, *Rev. Mod. Phys.*, **57**, 881, Oct. 1985.
3. Michael Cross, *Japanese Jour. of Appl. Phys.*, **26 Suppl(26-3)**, 1855, 1987.
4. D. D. Osheroff, M. C. Cross, and D. S. Fisher, *Phys. Rev. Lett.*, **44**, 792, 1980.
5. Y. Miura, N. Nishida, Y. Takano, H. Fukuyana, S. Ogawa, T. Hata, and T. Shigi, *Phys. Rev. Lett.*, **58**, 381, 1987.
6. M. Roger, J. H. Hetherington, and J. M. Delrieu, *Rev. Mod. Phys.*, **55**, 1, Jan. 1983.
7. Roger, *Phys. Rev. B*, **30**, 6432, Dec. 1984.

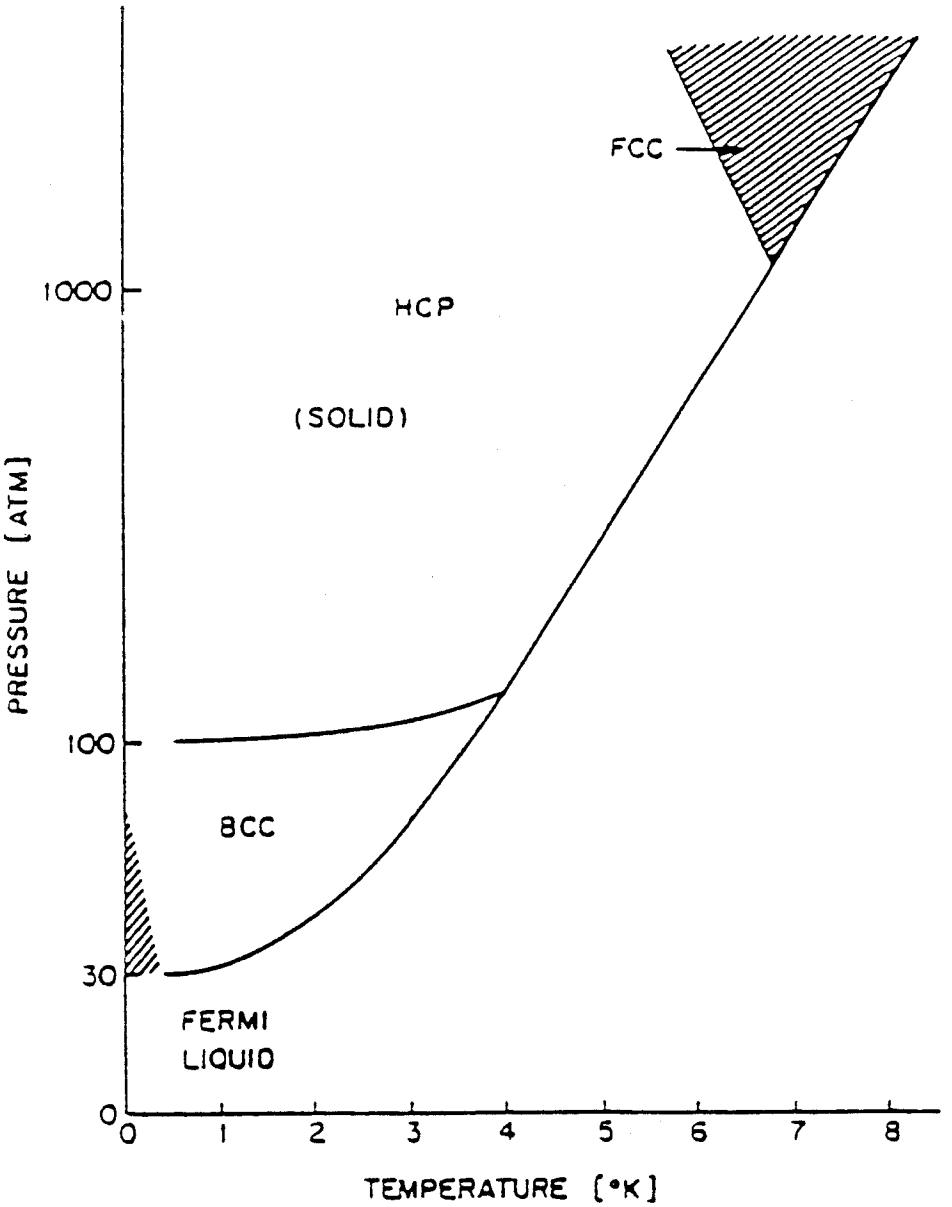

FCC
1000
HCP
(SOLID)
PRESSURE [ATM]
100
BCC
30
FERMI
LIQUID
0
0
1
2
3
4
5
6
7
8
TEMPERATURE [°K]

Figure 1

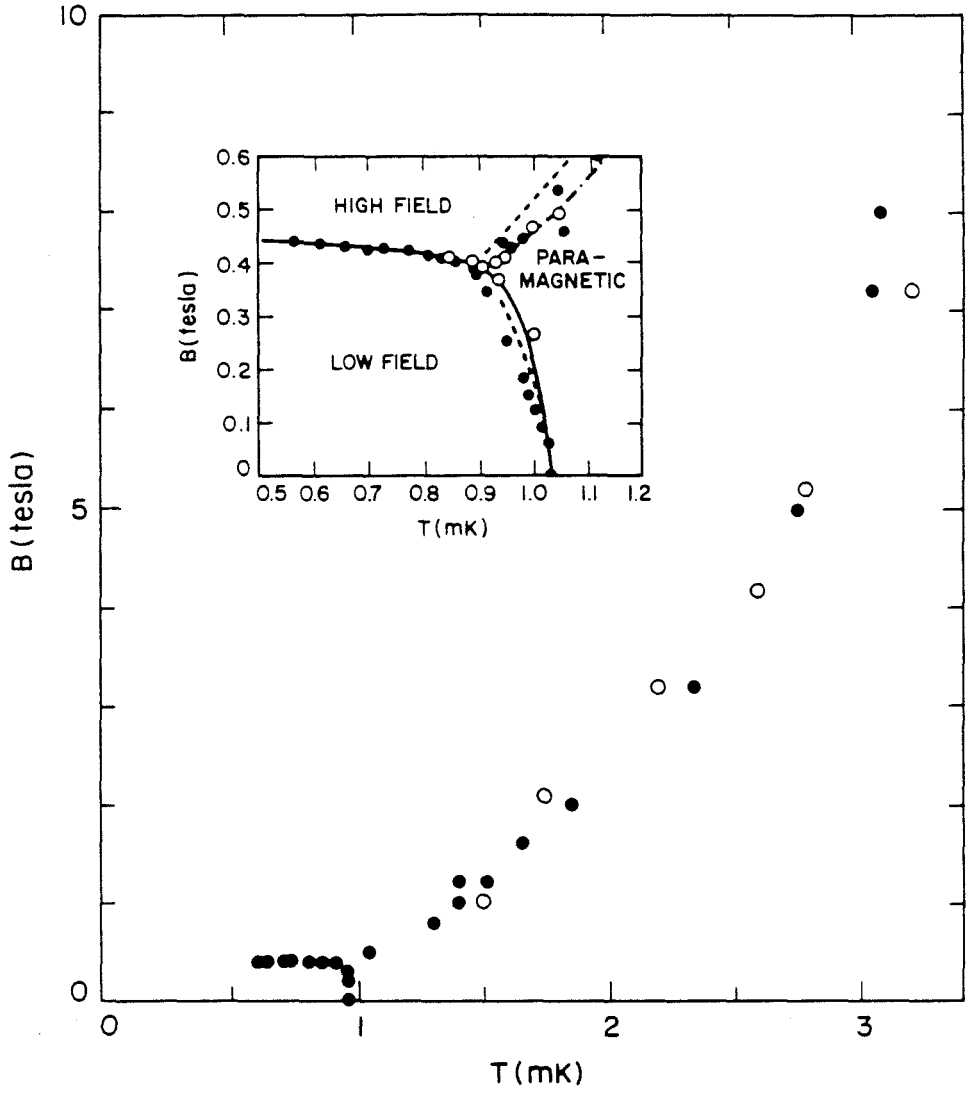

10
5
0
B (tesla)
0
1
2
3
T (mK)
0.6
0.5
0.4
0.3
0.2
0.1
0
B (tesla)
HIGH FIELD
PARA-
MAGNETIC
LOW FIELD
0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2
T (mK)

Figure 2

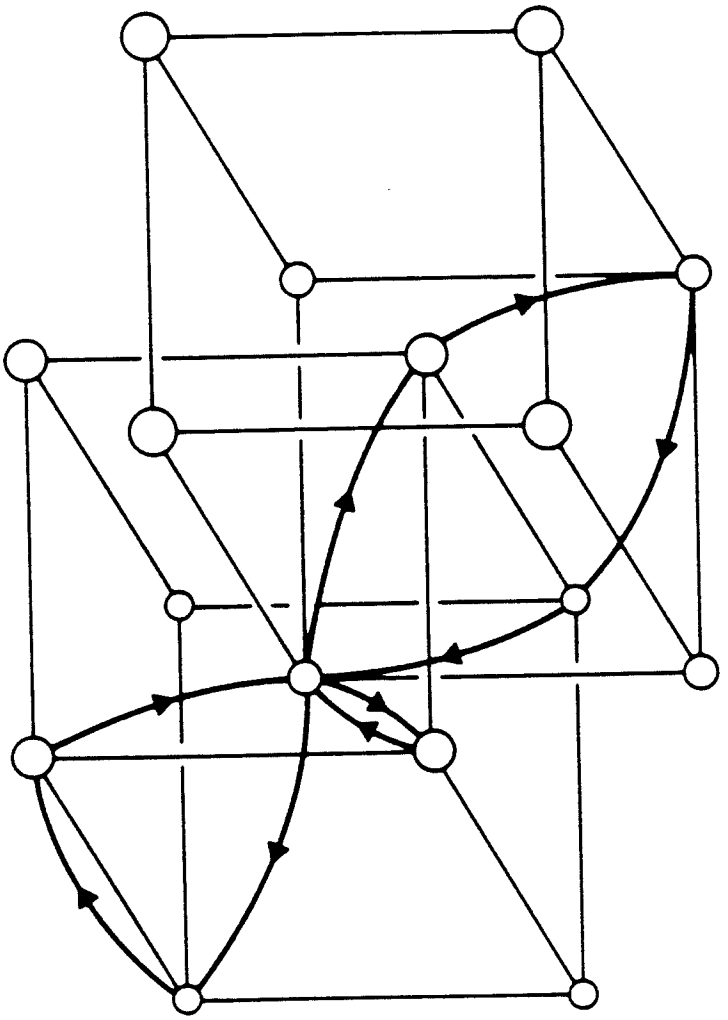


Figure 3

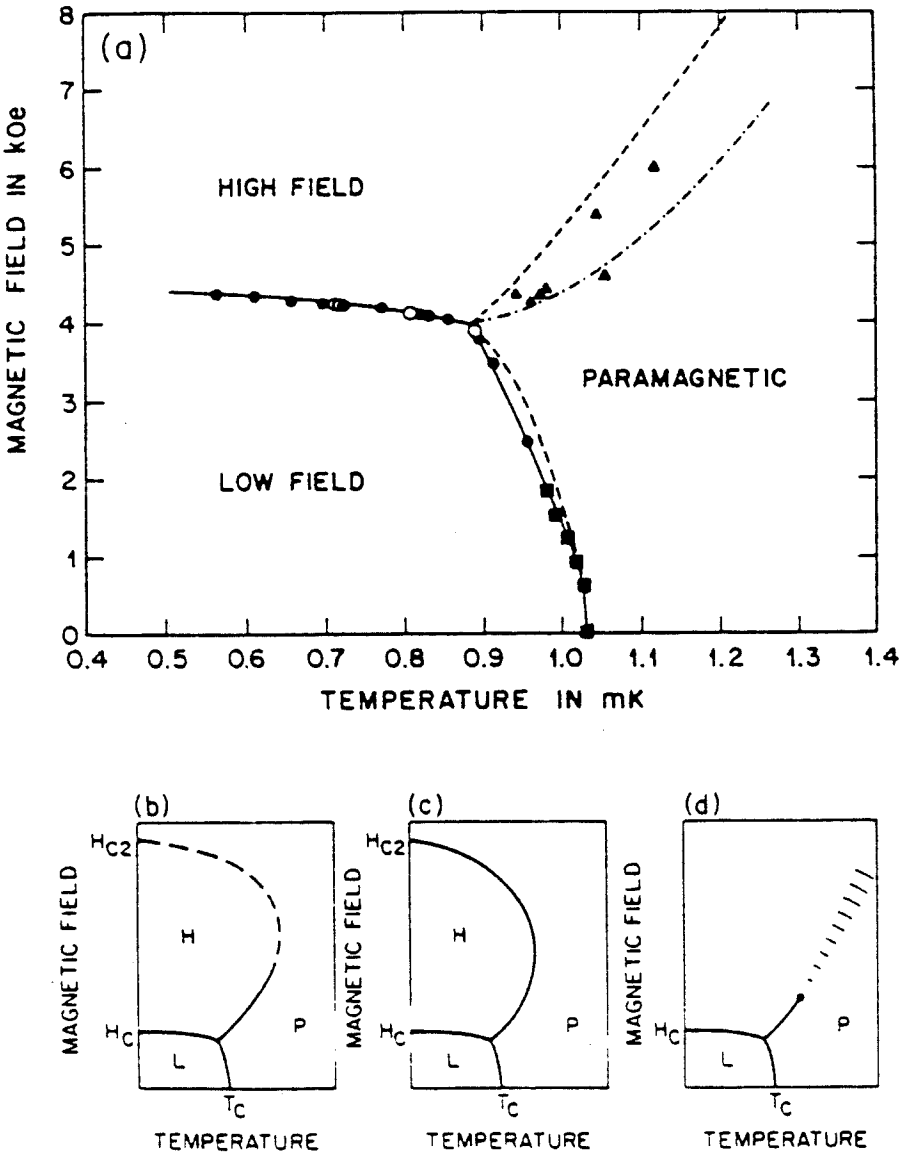

(a)
HIGH FIELD
PARAMAGNETIC
LOW FIELD
MAGNETIC FIELD IN kOe
TEMPERATURE IN mK
0
1
2
3
4
5
6
7
8
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
1.3
1.4
(b)
(c)
(d)
H_{c2}
H_c
T_c
H
P
L
MAGNETIC FIELD
TEMPERATURE

Figure 4

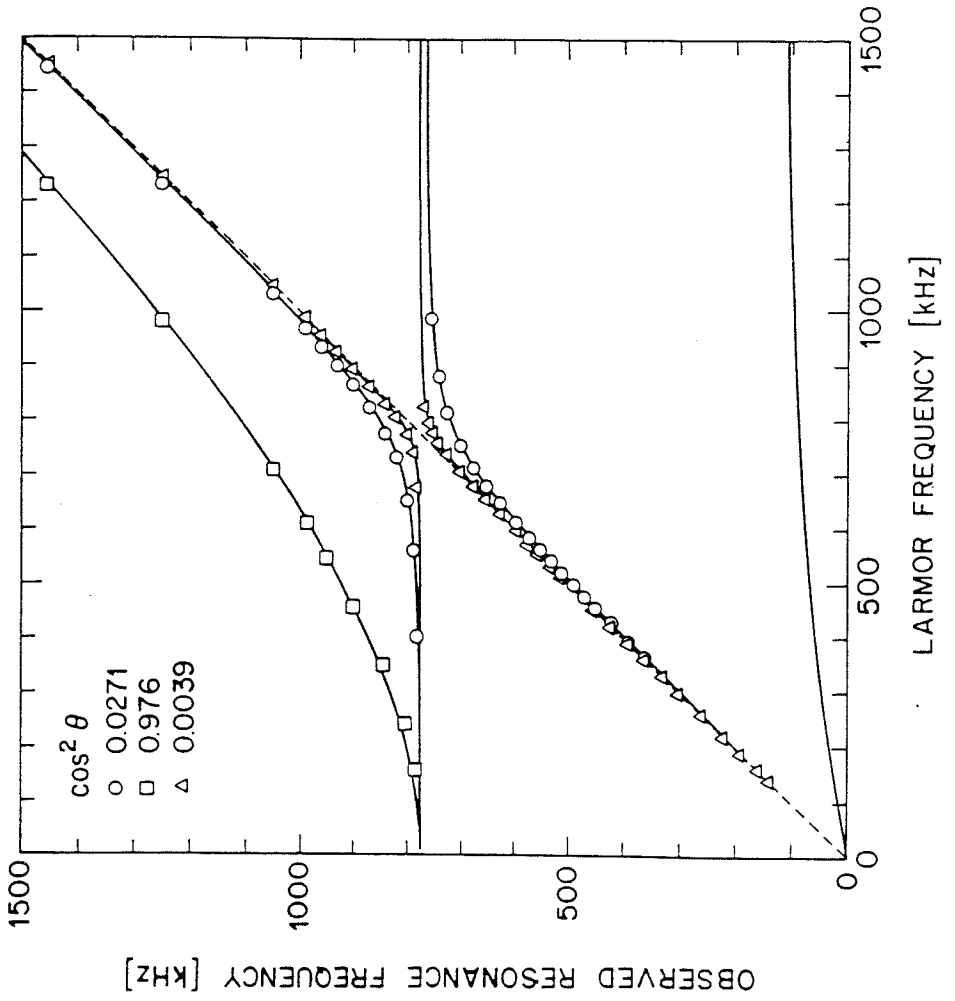

OBSERVED RESONANCE FREQUENCY [kHz]
LARMOR FREQUENCY [kHz]
cos² θ
○ 0.0271
□ 0.976
△ 0.0039
0
500
1000
1500

Figure 5

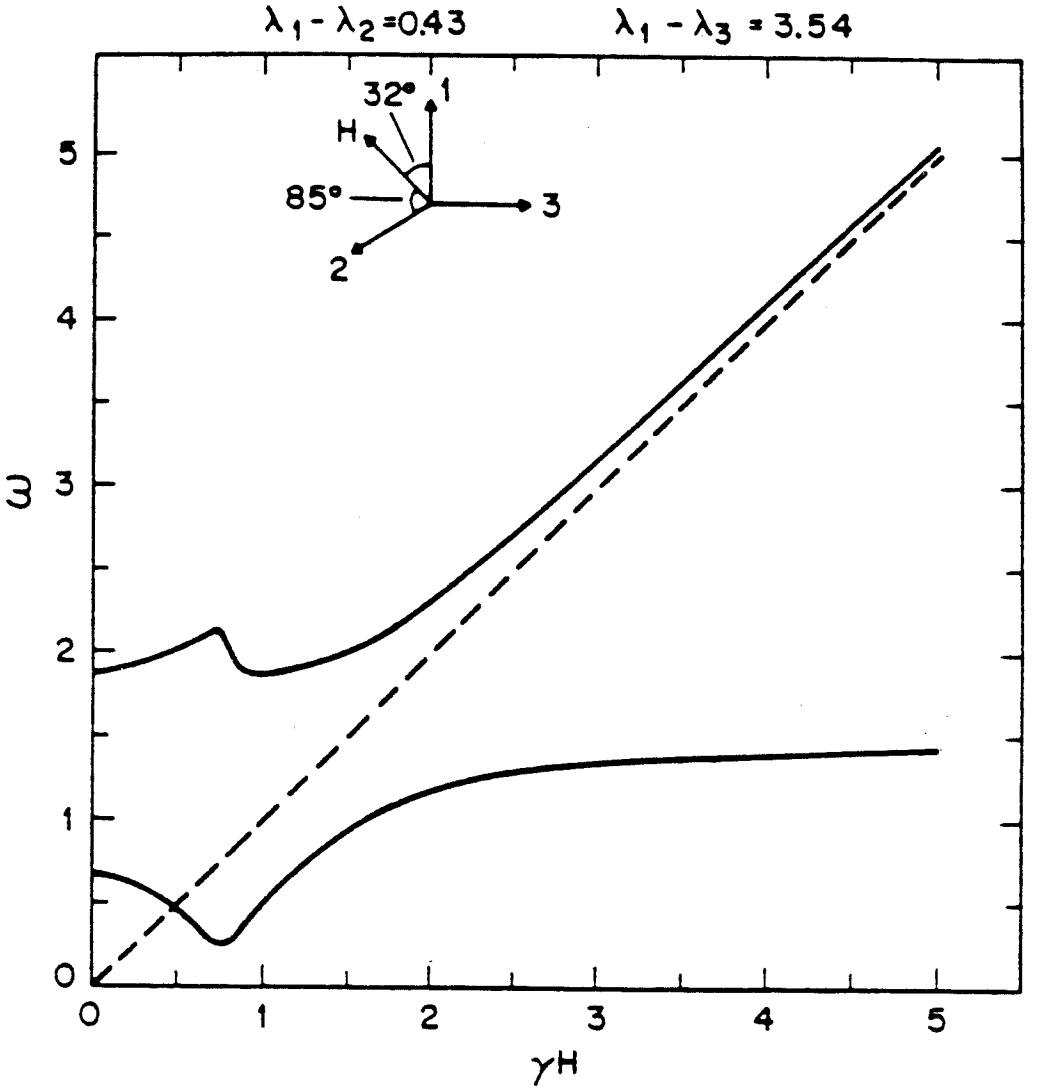


Figure 6

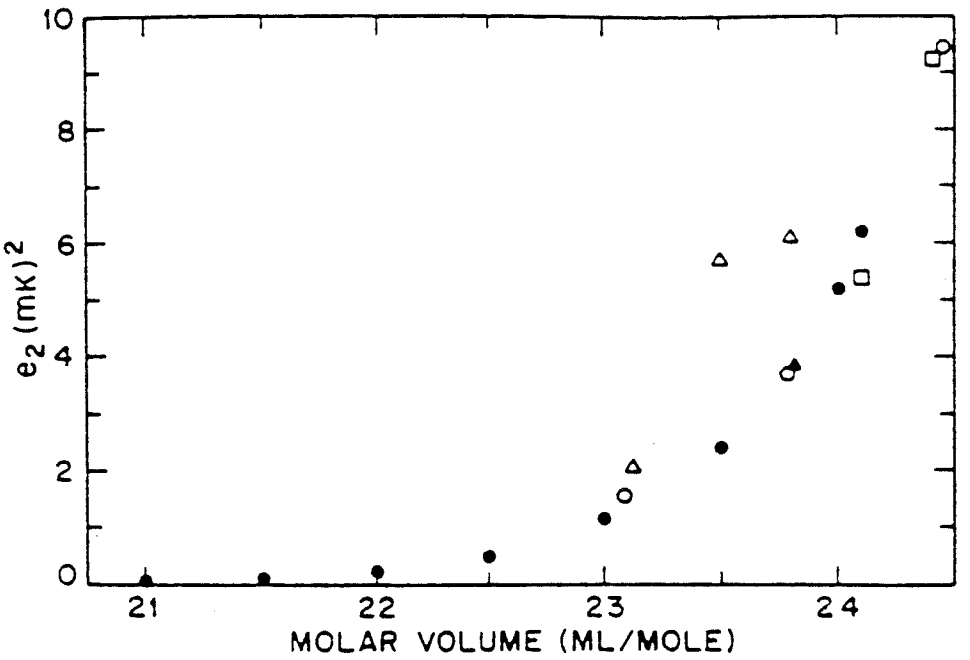


Figure 7

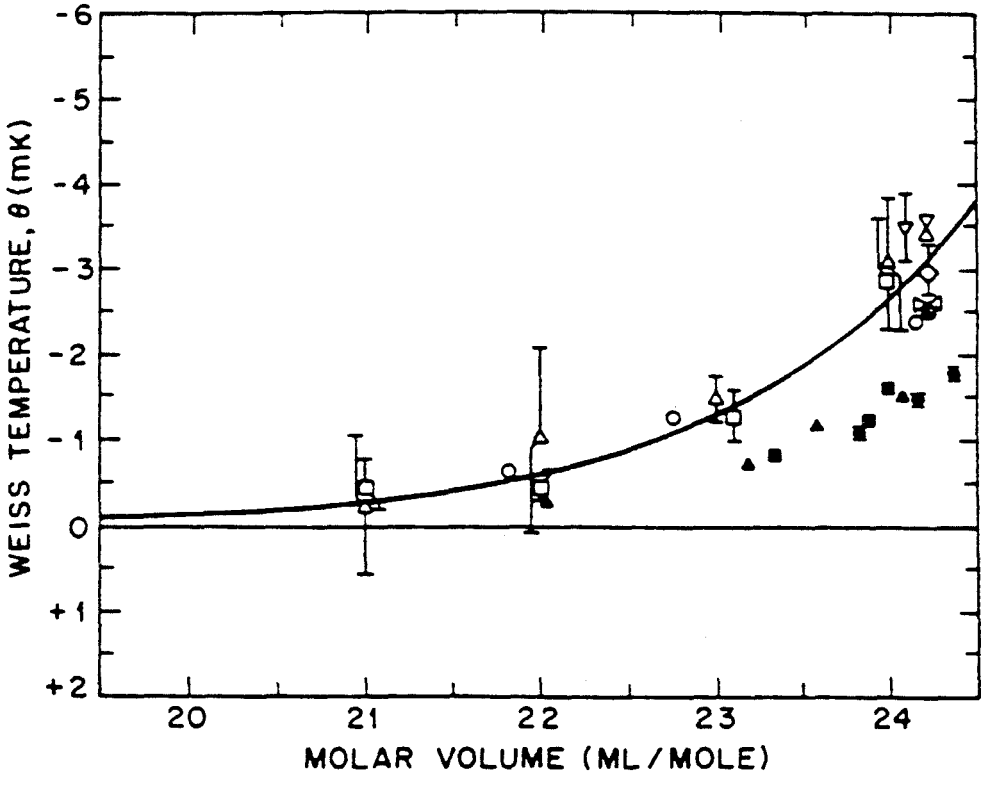

-6
-5
-4
-3
-2
-1
0
+1
+2
WEISS TEMPERATURE, θ (mK)
20
21
22
23
24
MOLAR VOLUME (ML/MOLE)

Figure 8

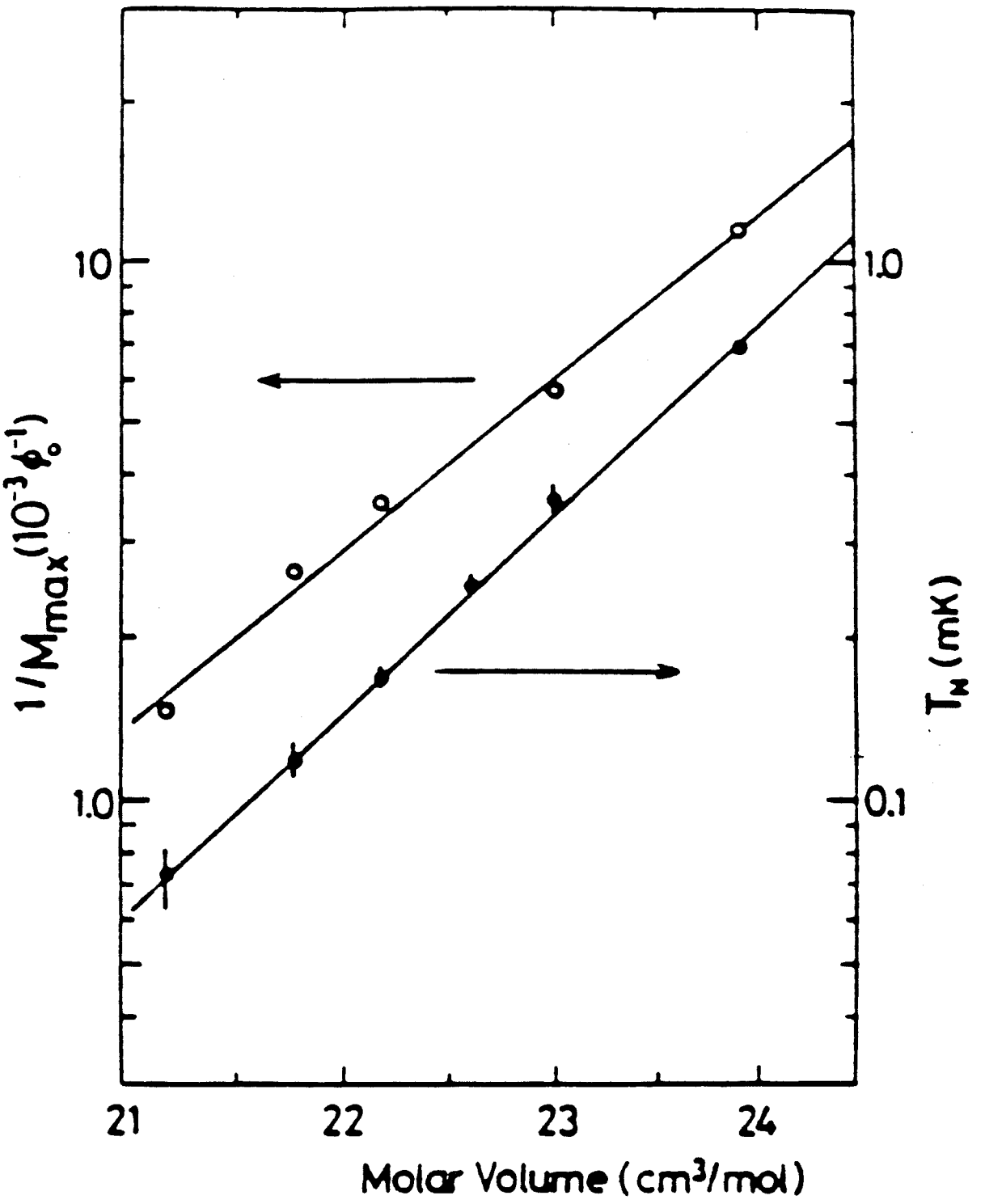

10
1.0
1.0
0.1
$1/M_{max}(10^{-3}\phi_o^{-1})$
T_M (mK)
21
22
23
24
Molar Volume (cm^3/mol)

Figure 9

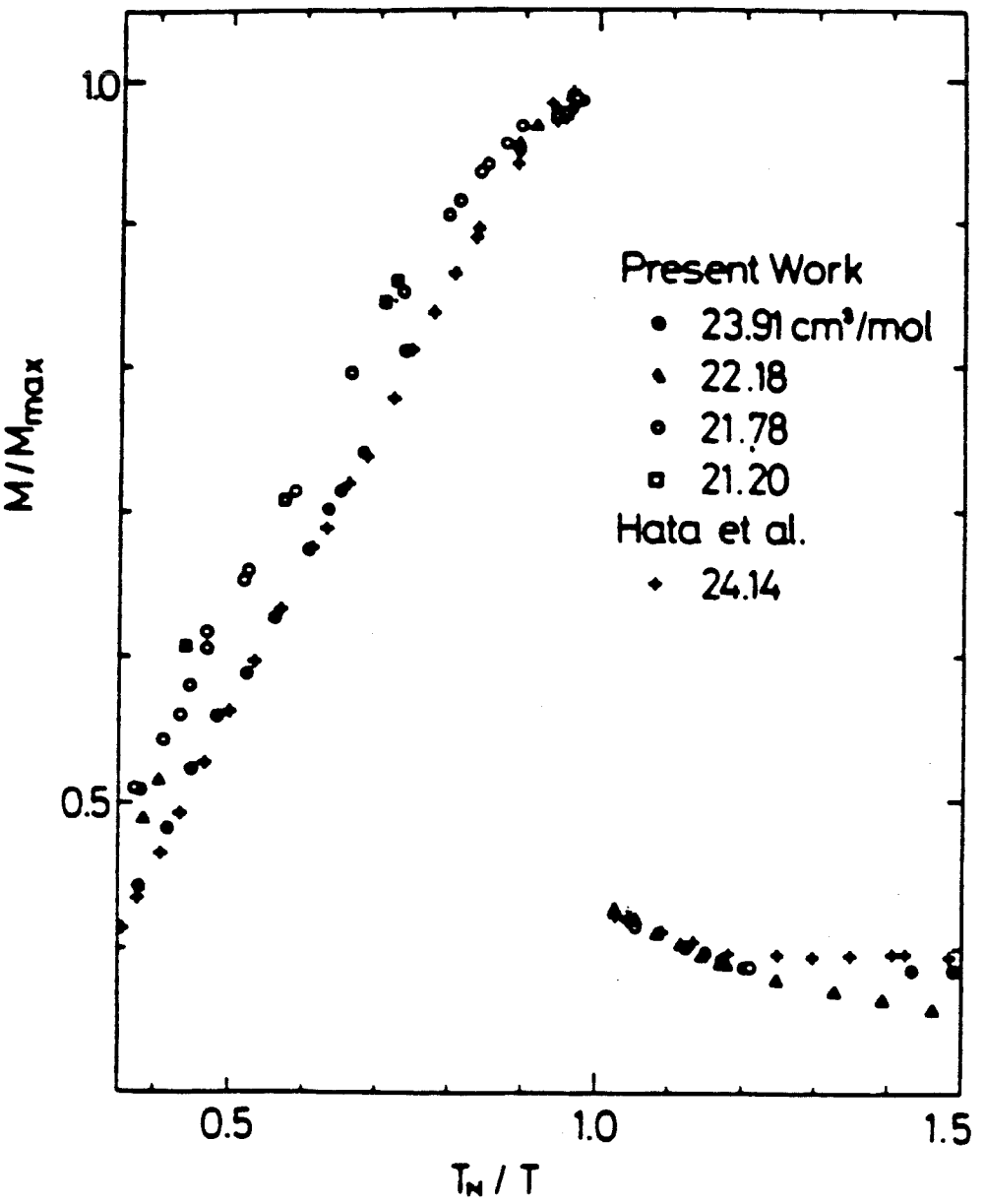

1.0
0.5
M / M_max
0.5
1.0
1.5
T_N / T
Present Work
23.91 cm³/mol
22.18
21.78
21.20
Hata et al.
24.14

Figure 10

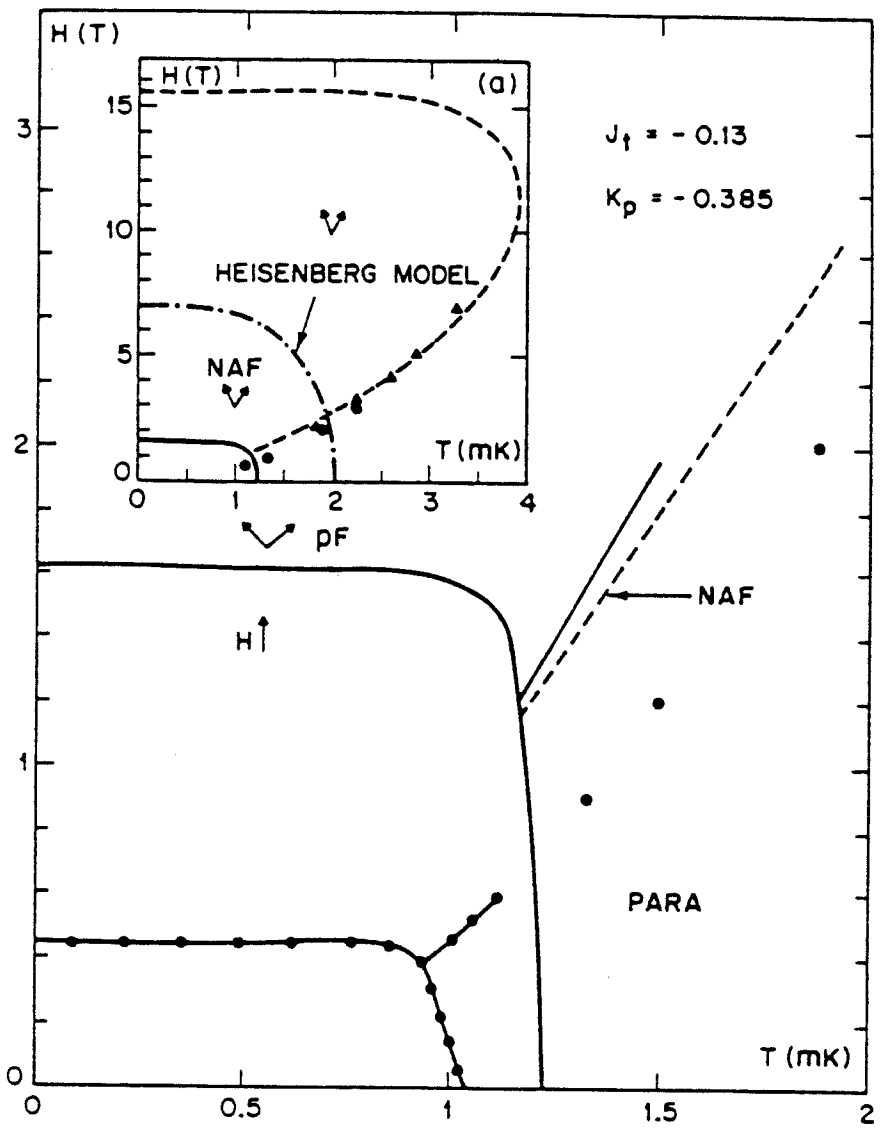

H (T)
H(T)
(a)
15
10
5
0
HEISENBERG MODEL
NAF
T (mK)
0
1
2
3
4
pF
J_t = - 0.13
K_p = - 0.385
NAF
H
PARA
T (mK)
3
2
1
0
0
0.5
1
1.5
2

Figure 11

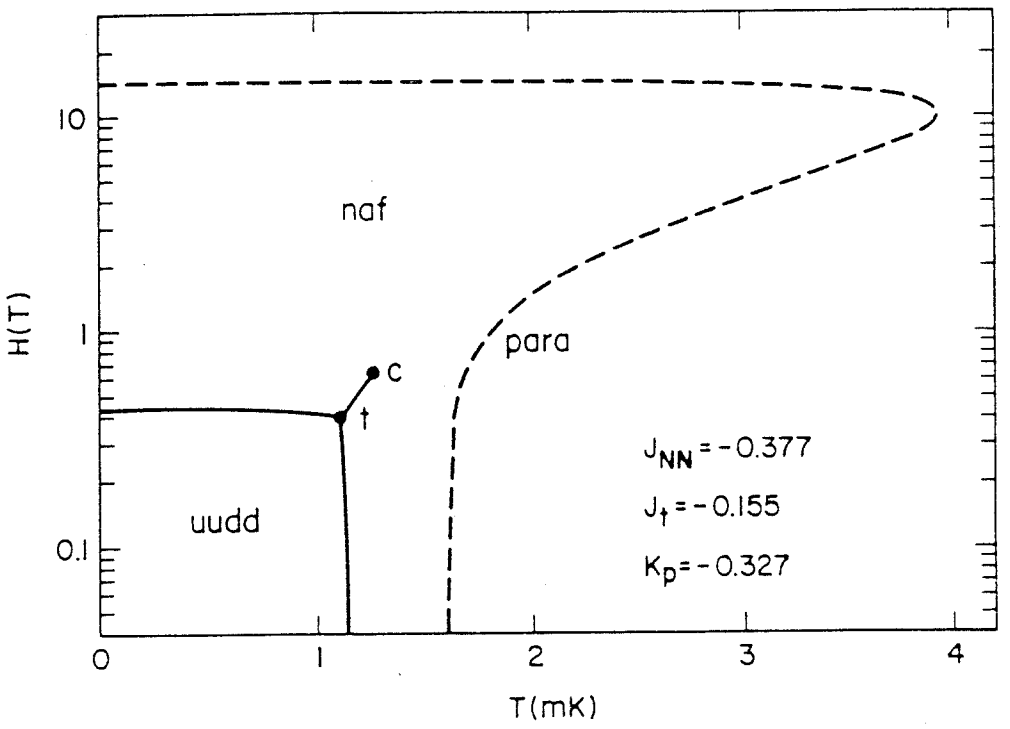


Figure 12

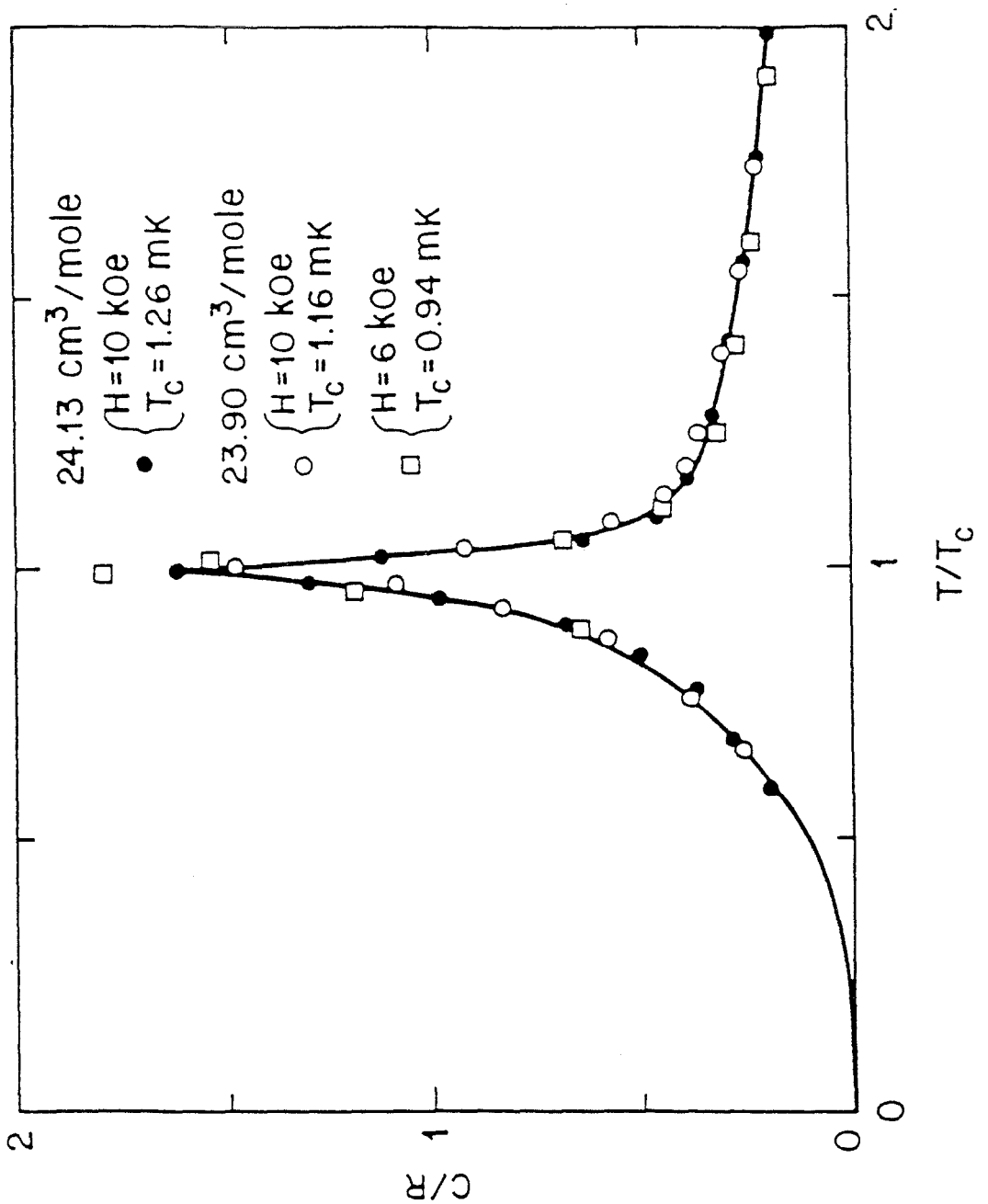

24.13 cm³/mole
H=10 kOe
Tc=1.26 mK
23.90 cm³/mole
H=10 kOe
Tc=1.16 mK
H=6 kOe
Tc=0.94 mK
C/R
T/Tc
0
1
2

Figure 13

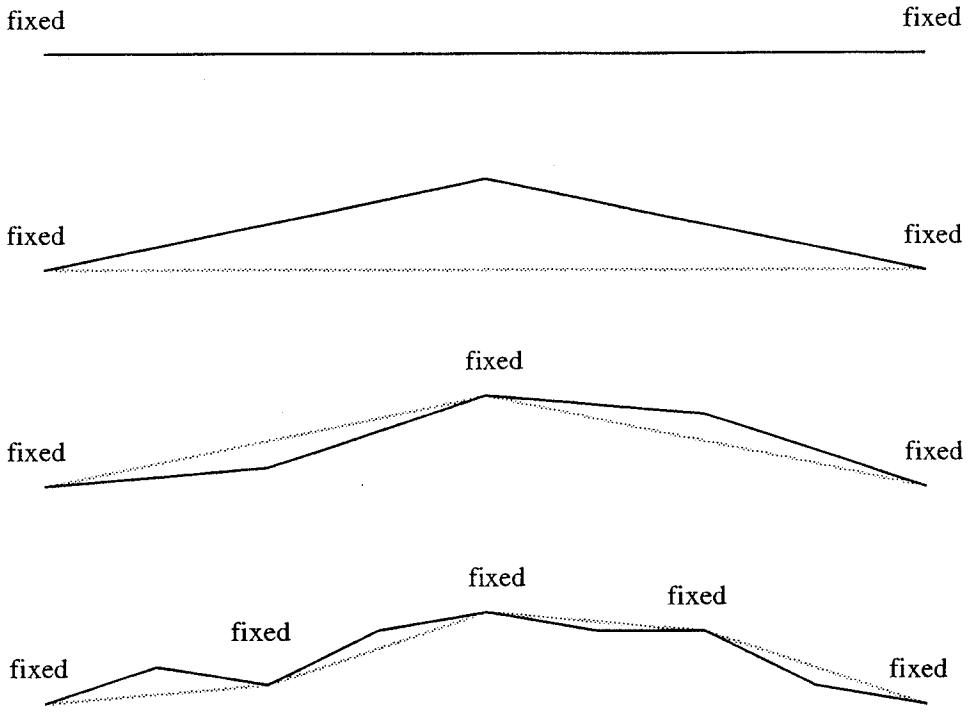

fixed
fixed
fixed
fixed
fixed
fixed
fixed
fixed
fixed
fixed
fixed

Figure 14

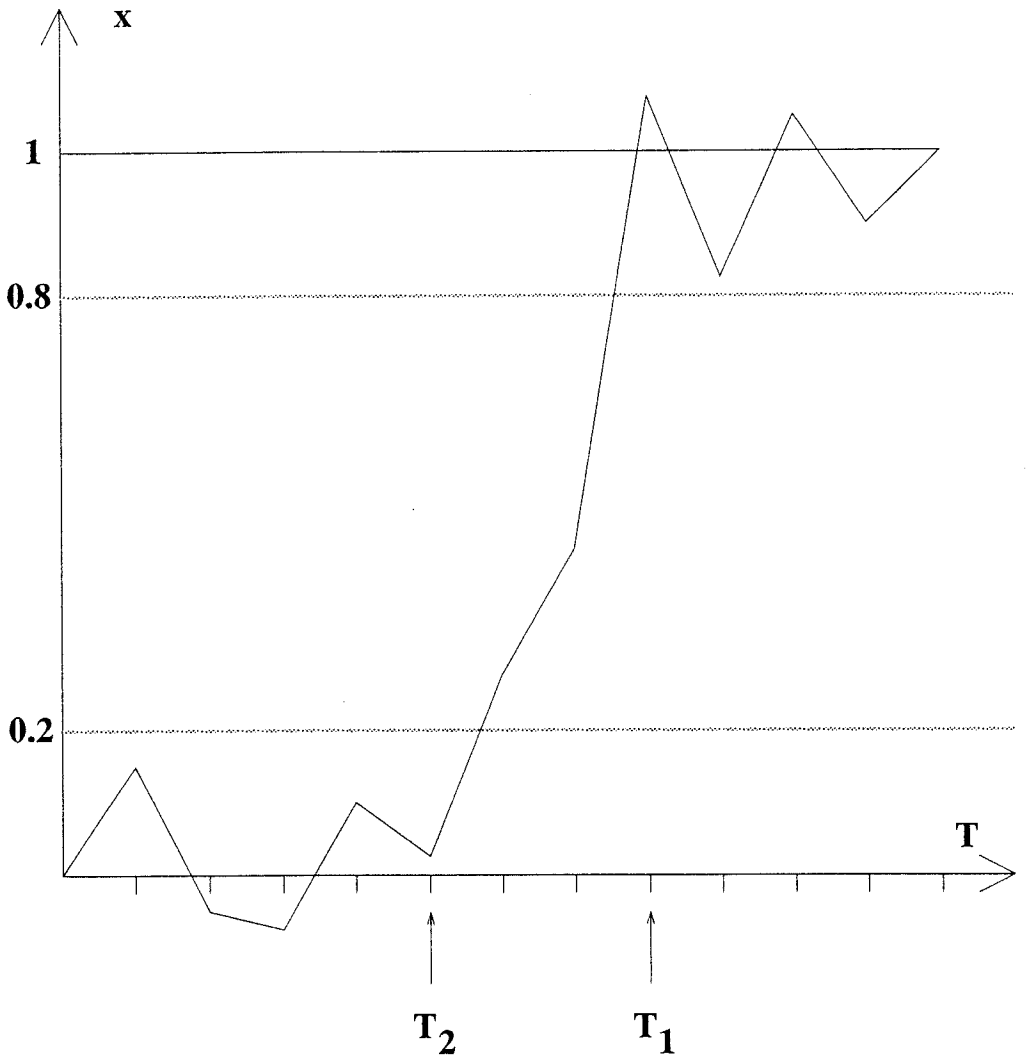


Figure 15

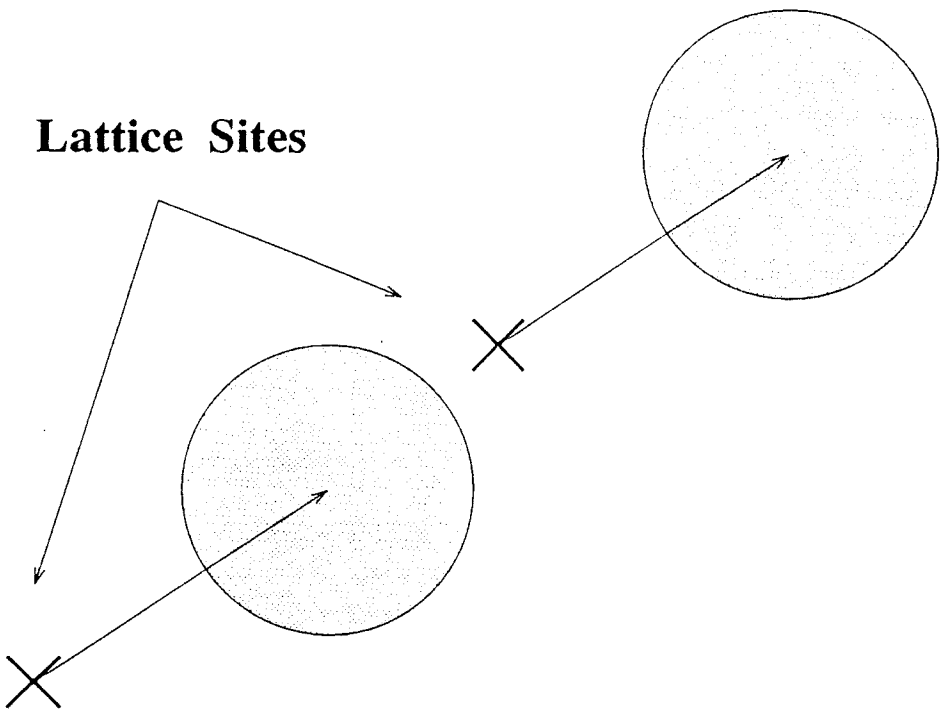


Figure 16

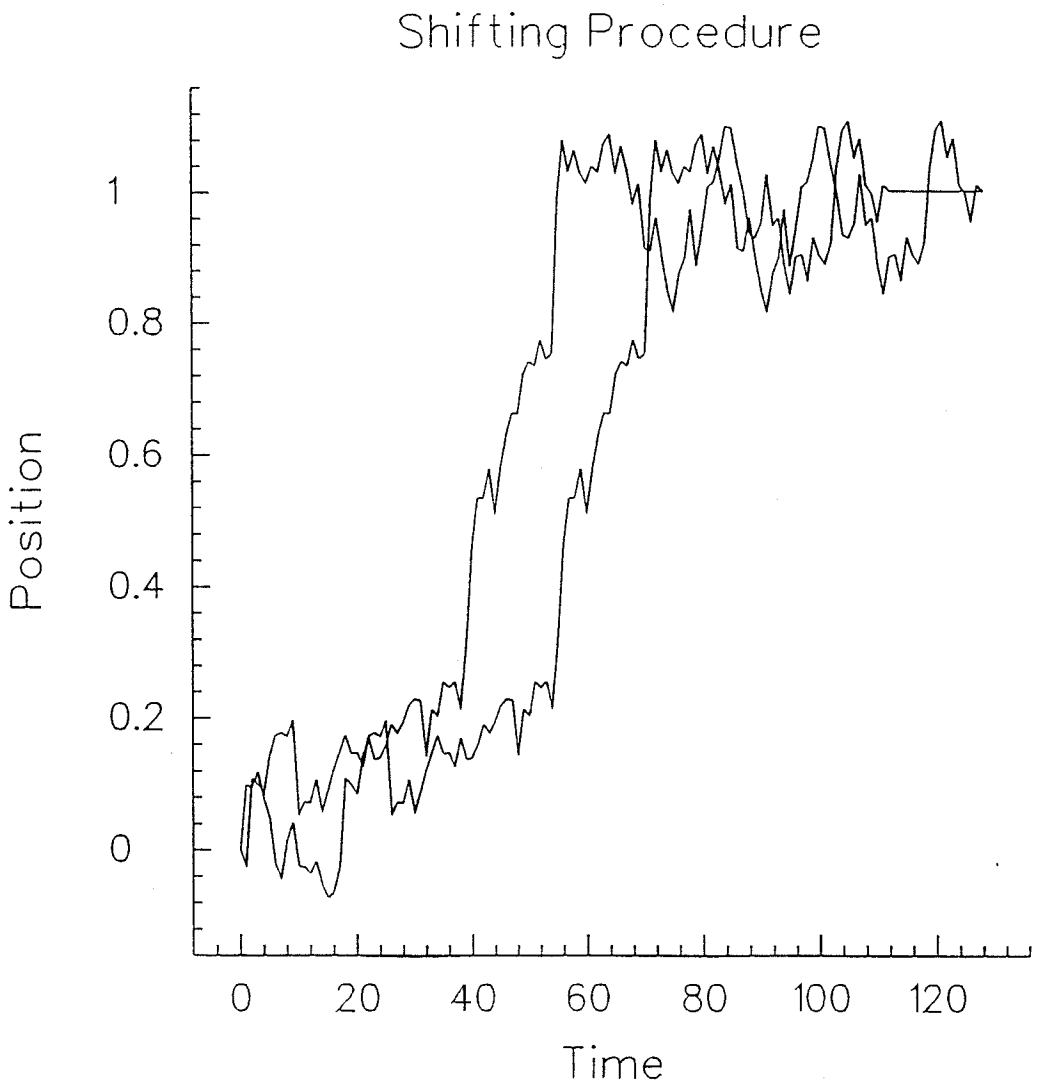
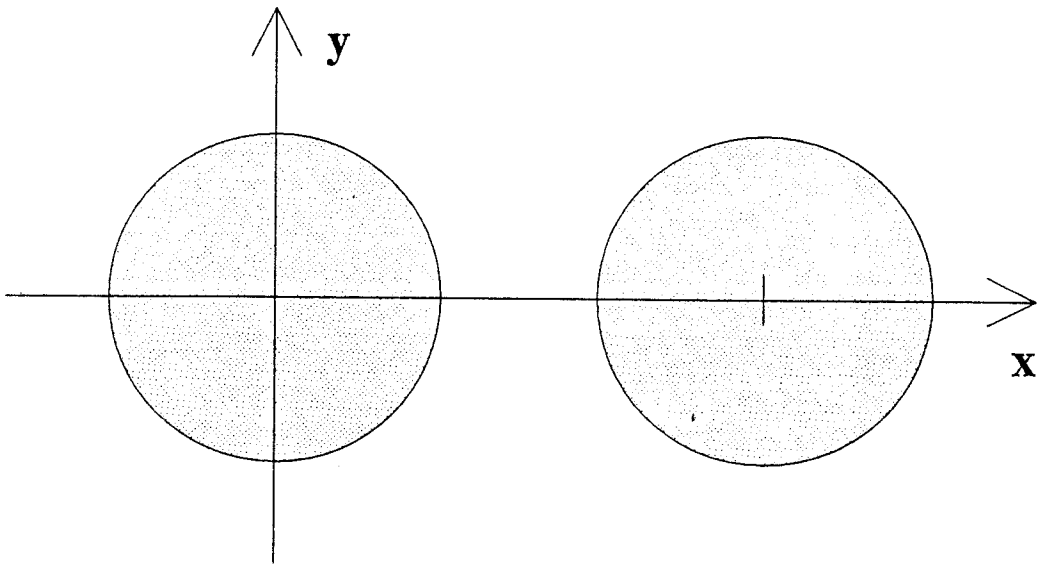

Shifting Procedure
Position
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
Time

Figure 17



$$\mathbf{r = (u + D, v)}$$

D = diameter

Figure 18

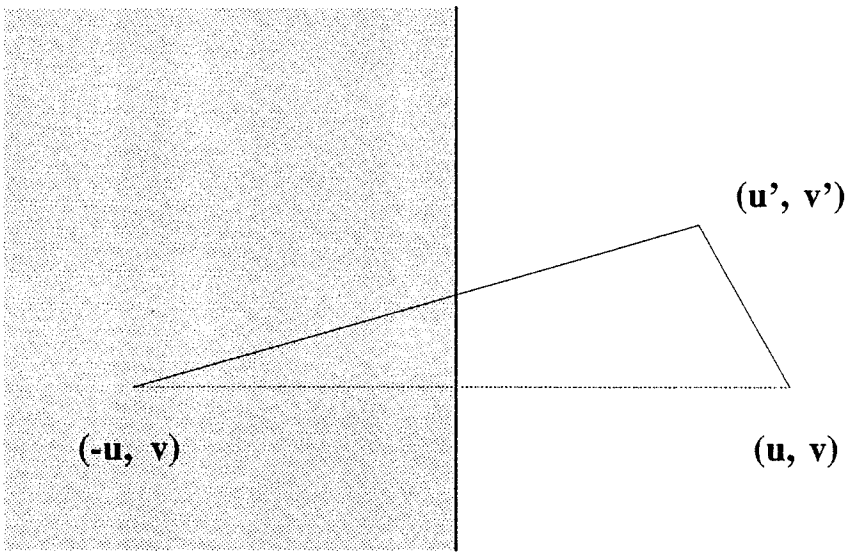


Figure 19

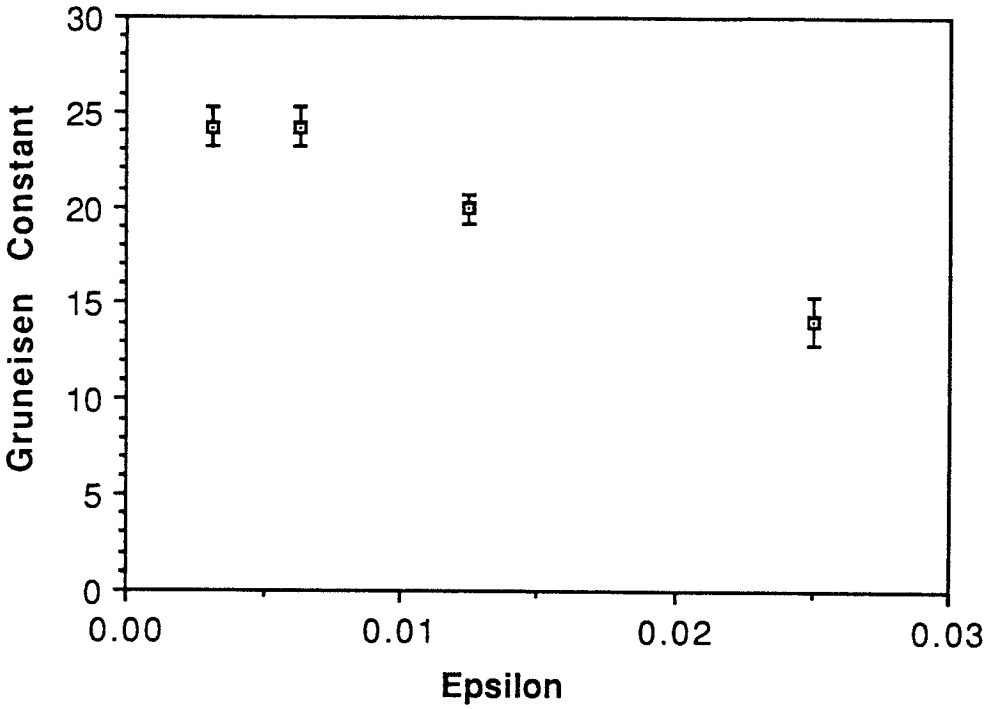


Figure 20

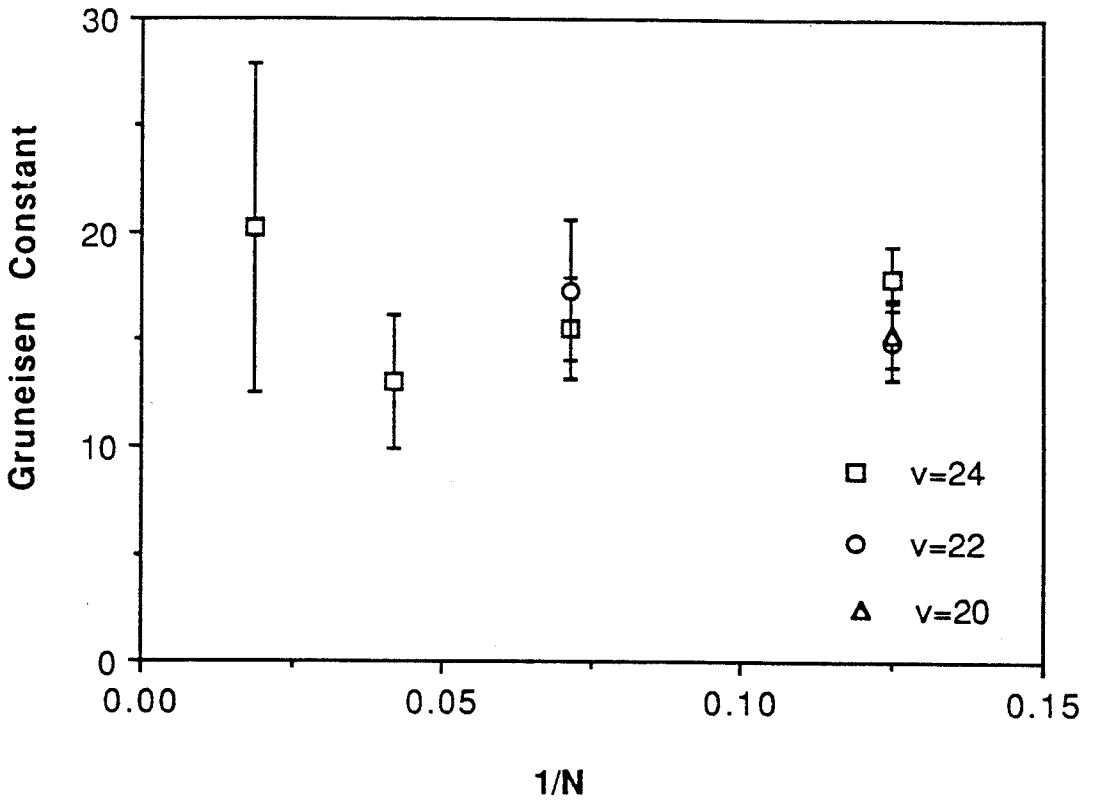


Figure 21

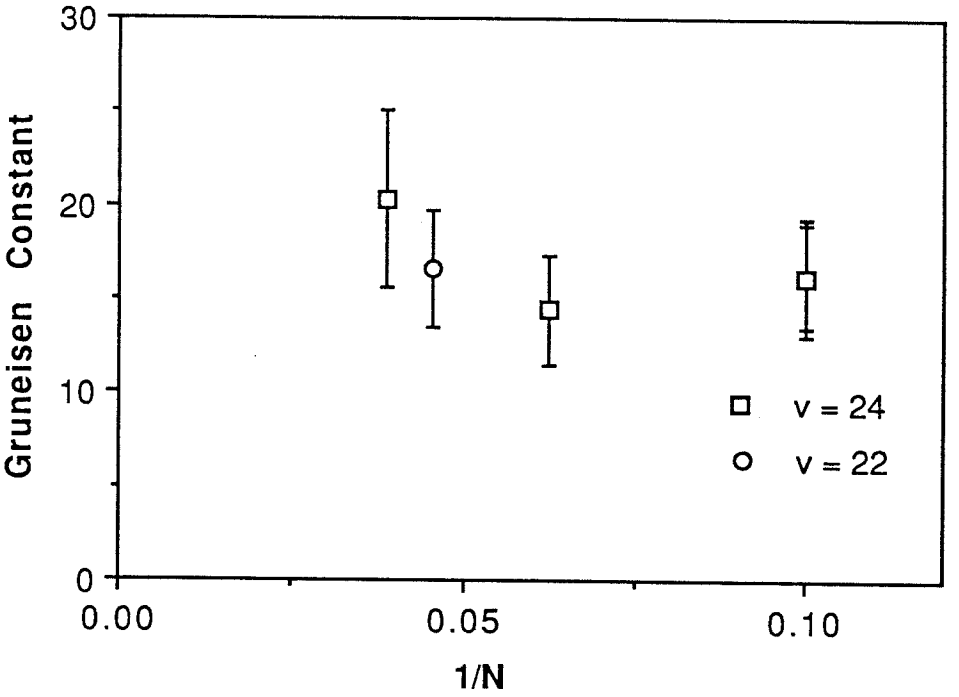


Figure 22

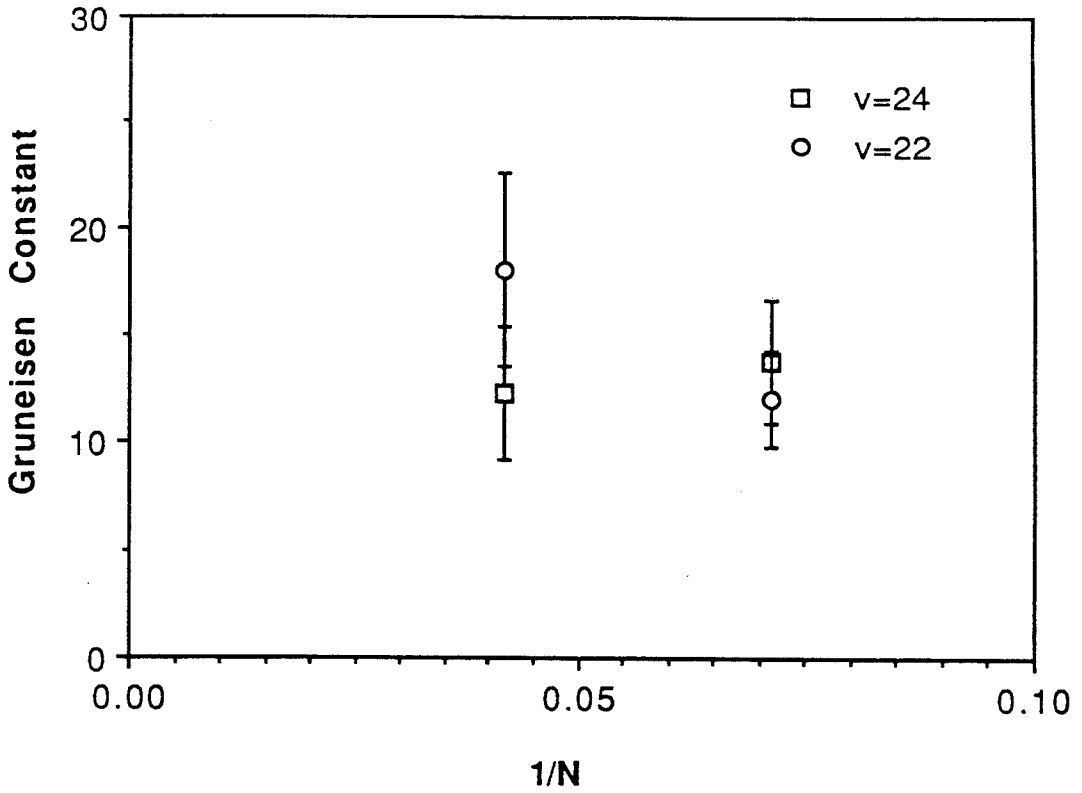


Figure 23

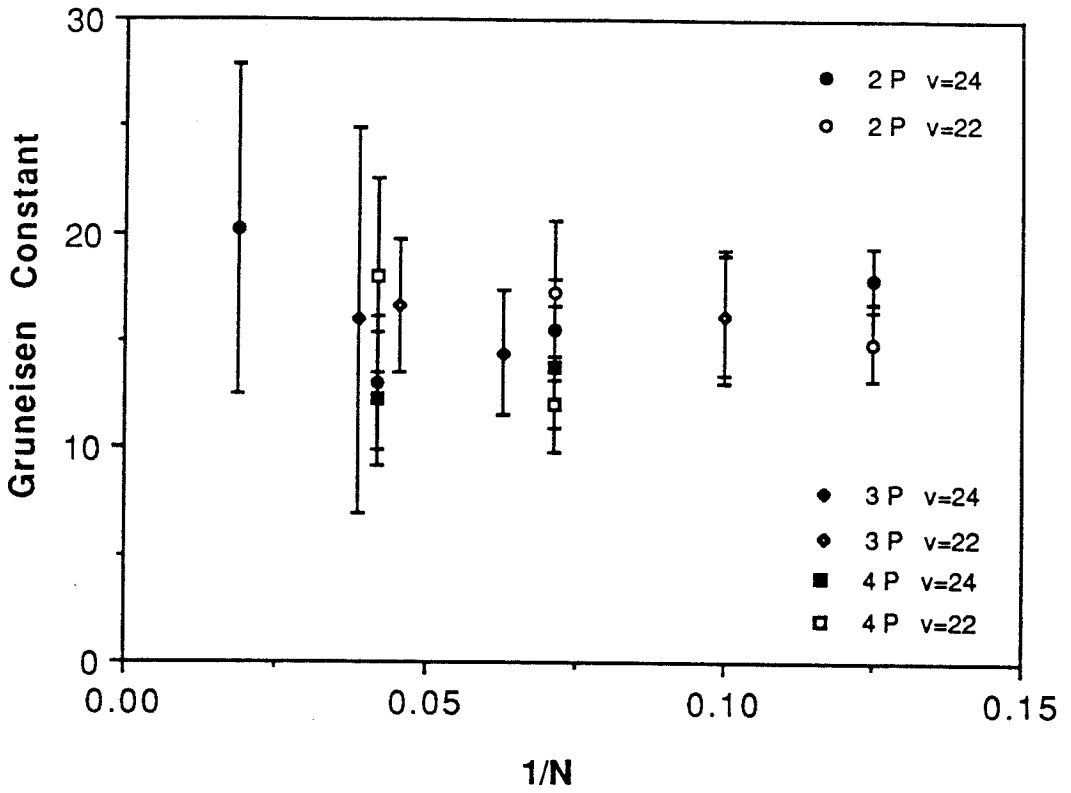

30
20
10
0
0.00
0.05
0.10
0.15
Gruneisen Constant
1/N
2 P v=24
2 P v=22
3 P v=24
3 P v=22
4 P v=24
4 P v=22

Figure 24

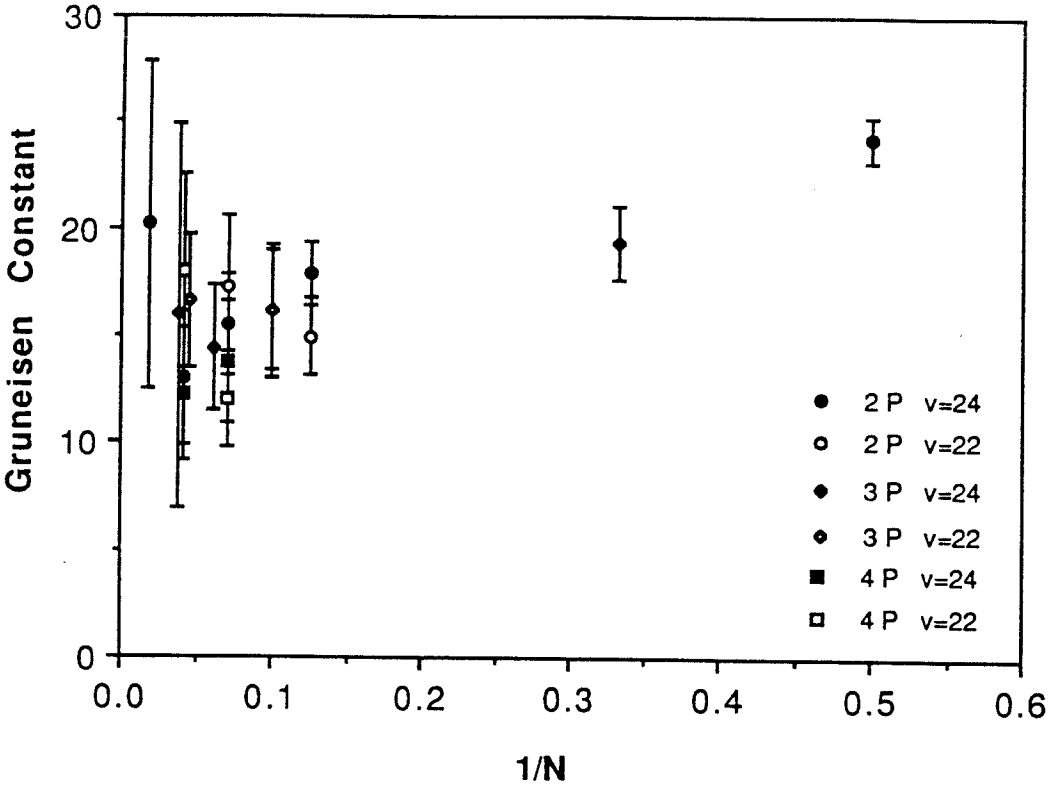

Gruneisen Constant
30
20
10
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
1/N
2 P v=24
2 P v=22
3 P v=24
3 P v=22
4 P v=24
4 P v=22

Figure 25

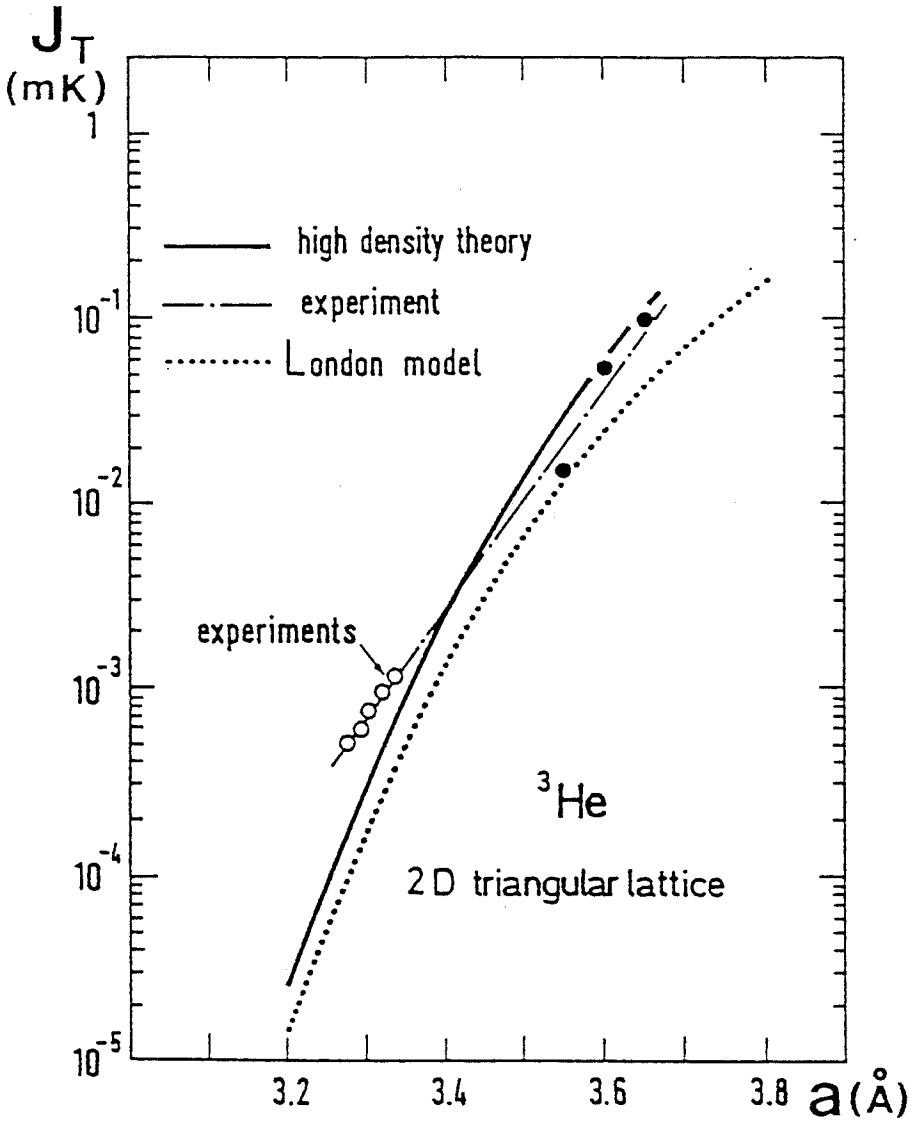

J_T (mK)
1
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
high density theory
experiment
London model
experiments
^{3}He
2D triangular lattice
3.2
3.4
3.6
3.8
a(Å)

Figure 26

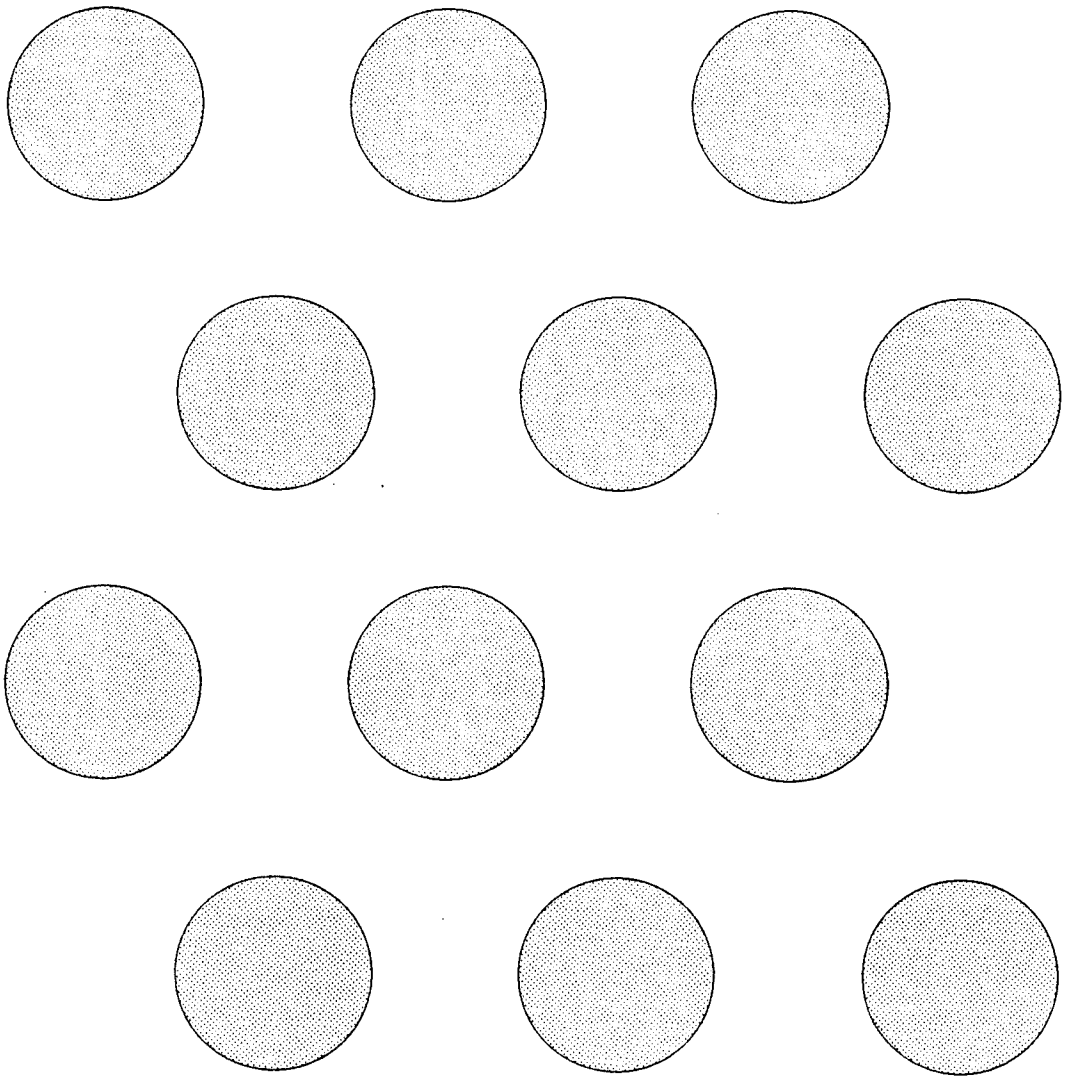
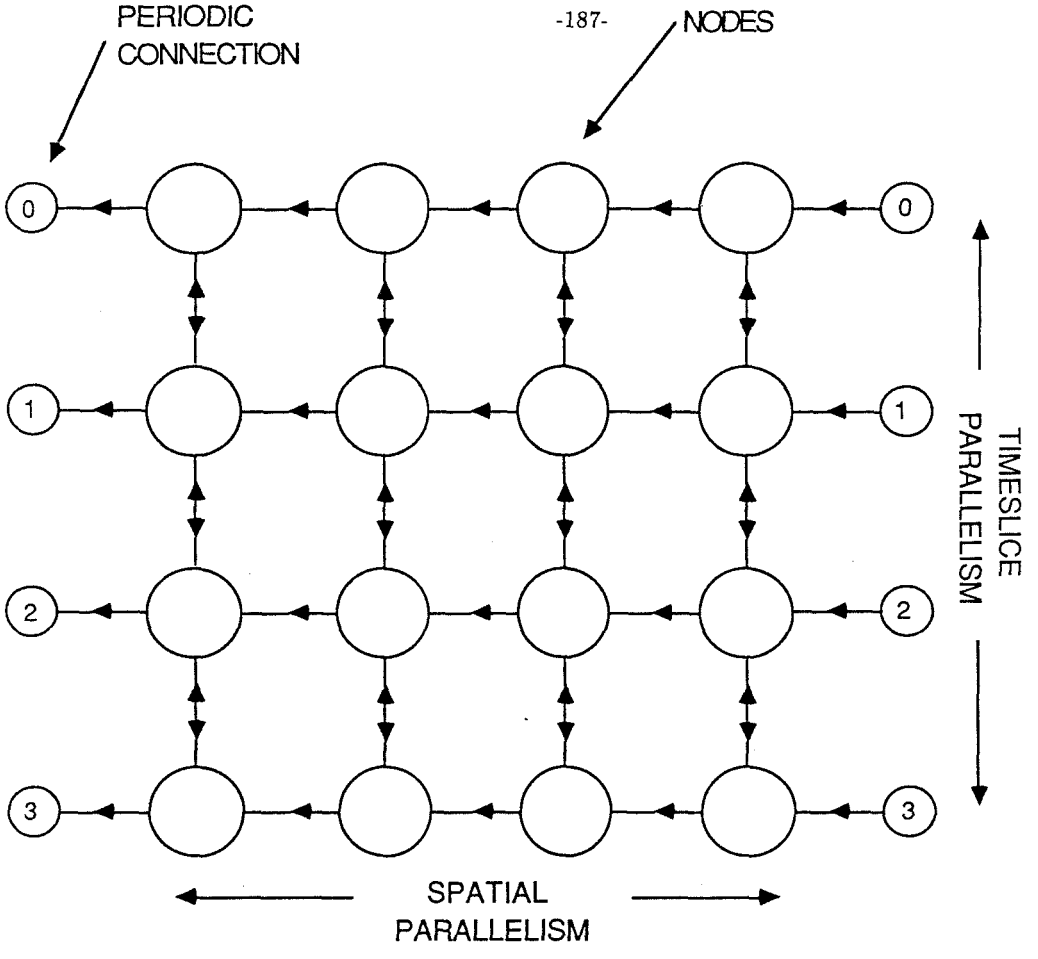


Figure 27



- ARROWS INDICATE COMMUNICATIONS DIRECTIONS.
- HORIZONTAL COMMUNICATION TAKES PLACES AFTER A SPATIAL CALCULATION INVOLVING A CERTAIN TIMESLICE.
- VERTICLE COMMUNICATIN TAKES PLACE AFTER ALL TIMESLICE CALCULATIONS INVOLVING HORIZONTAL GROUPS ARE DONE. **Figure 28**

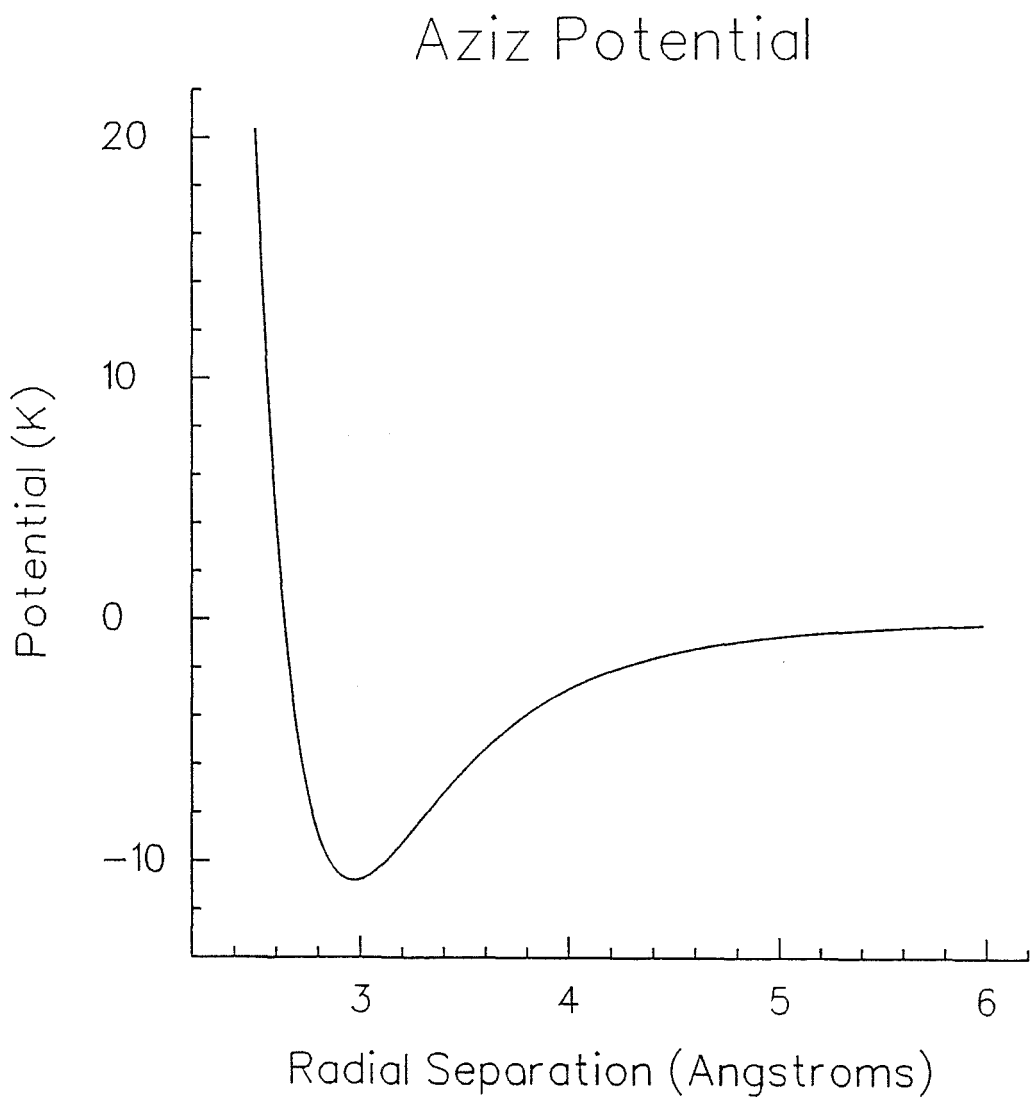

Aziz Potential
20
10
0
−10
Potential (K)
3
4
5
6
Radial Separation (Angstroms)

Figure 29