MONTE CARLO STUDIES OF TWO DIMENSIONAL QUANTUM SPIN SYSTEMS

Thesis by

Miloje S. Makivić

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1991

(Submitted August 2, 1990)

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Michael Cross, for his inspiring guidance, insight and warm and friendly support throughout my graduate studies. I owe him a lot for making the subject of condensed matter physics a fascinating place.

I also wish to express my sincere gratitude to Professor Geoffrey Fox for his constant encouragement and support during my work. I thank him for being exposed to the exciting fields of parallel computing and complex systems.

I am deeply thankful to Professor Peter Weichman for his valuable advice and insight and very generous help. I also thank him for listening to my questions so patiently.

I would like to express my great appreciation to Hong Ding, with whom I worked closely during the last two years. I consider myself fortunate to have had such a fruitful collaboration.

I am indebted to all my friends and colleagues who made life at Caltech so enjoyable and enlightening.

I would like to thank The California Institute of Technology and The Shell Foundation for their financial assistance.

For their love and support I owe a special debt of gratitude to my parents Stanoje and Marica Makivić.

I thank my wife Irina for bringing love, beauty and meaning to everything and I dedicate this thesis to her.

ABSTRACT

Spin-$\frac{1}{2}$ nearest neighbor Heisenberg antiferromagnet and XY model on a square lattice are studied via large scale quantum Monte Carlo simulations using a fast and efficient multispin coding algorithm on the Caltech/JPL MarkIIIfp parallel supercomputer, based on the Suzuki-Trotter transformation. We performed simulations with very good statistics on lattices as large as 128x128 spins, in the temperature range from 0.1 to 2.5 in units of the effective exchange coupling J. We calculated energy, specific heat, magnetic susceptibilities and also spin correlation functions from which we deduce the correlation lengths.

For the Heisenberg model, at temperatures higher than J the results are in excellent agreement with high-temperature series expansion. At low temperatures the long wavelength behavior is essentially classical. Our data show that the correlation length and staggered susceptibility are quantitatively well described by the renormalized classical picture at the 2-loop level of approximation. From the divergence of the correlation length, we deduce the value of the quantum renormalized spin stiffness, $\rho_s/J = 0.199(2)$. We give evidence that the correlation function is of the Ornstein-Zernicke type. By comparing the largest measured correlation lengths with neutron scattering experiments on La_2CuO_4, we deduce the value of effective exchange coupling $J = 1450 \pm 30\ K$. By measuring the imaginary time-dependent correlation functions, we show that the dynamics of the model can be well understood within a Bose liquid-type picture. The spin waves are rather sharp throughout most of the Brillouin zone and the damping is weakly dependent on the wave vector.

In the case of the XY model, convincing numerical evidence is obtained on square lattices as large as 96x96 that the spin-$\frac{1}{2}$ XY model undergoes a Kosterlitz-Thouless (KT) phase transition at kT_c/J=0.350(4). The correlation length and in-plane susceptibility diverge at T_c precisely according to the form predicted by Kosterlitz and Thouless for the classical XY model. The specific heat increases very rapidly on heating near T_c and exhibits a peak around $kT/J = 0.45$. We also measure the spin stiffness and the correlation function exponent below the transition temperature. Within the statistical accuracy of the measurements, the results are well described by the square root singularity (with a nonuniversal amplitude) below T_c, and they have the universal values in agreement with KT theory at T_c.

TABLE OF CONTENTS

TABLE AND FIGURE CAPTIONS

TABLES

1. Temperature, Trotter number, linear size, energy, specific heat, uniform susceptibility, correlation length and exponent λ for 2D Heisenberg antiferromagnet.

2. A sample of spin correlations in S_z direction for the XY model for selected temperatures.

FIGURES

1. (a) The breakup of the Hamiltonian: In x-direction, H_1 includes bonds indicated by the solid links; H_3 includes bonds indicated by broken links. Similarly for H_2, H_4 in y-direction. (b) The products U_1U_2 and U_3U_4 decompose a square lattice into a collection of noninteracting square cells. This property mimics the cell decomposition.

2. A "space" flip. The dashed line denotes a noninteracting plaquette lying in spatial dimensions. The four plaquettes extending in the time direction are interacting ones. After the four spins are flipped, the two worldlines twist around each other.

3. A "time" flip. The dashed line denotes a noninteracting plaquette. The 8 plaquettes surrounding it are interacting ones. After the 8 spins are flipped, the worldline is distorted. Notice that all plaquettes go in the time direction.

4. A "global" flip in the time direction. Only four of the interacting plaquettes which surround a straight worldline are shown.

5. (a) A "global" flip in spatial directions. The dashed line indicates the string of spins being flipped. The plaquettes shown are all interacting. (b) A configuration with a different winding number is reached.

6. The vectorized "time" flips are shown. Spins along the time direction are packed into computer words. The two 32-bit words S1 and S2 contain eight "time" plaquettes, indicated by the dashed lines. The plaquettes shown are all interacting ones.

7. (a) The configuration of the hypercube nodes. In this example, 32 nodes are configured as 4 independent rings, each consisting of 8 nodes. Each ring runs an independent simulation. (b) The decomposition of the physical space of each lattice among the nodes in a ring. (c) The sequence of steps necessary to perform a sweep of "space" loop updates. In Step 1, individual processors update their nonoverlapping domains in parallel. When a boundary is reached, the boundary layer of spins is communicated via a *cshift* call, as shown in Step 2. In Step 3, the communicated spins, which are now local, are updated. The system is shown as a cross section through a single time slice. (d) The memory organization of a ring. A processor's data domain always occupies the lowest memory addresses.

8. Correlation functions on the 32x32 lattice at $T = 0.45$, $M = 24$. Squares denote the run with winding numbers N_x and N_y restricted to 0. Crosses denote the run with nonconstrained winding numbers. The data points clearly overlap. The error bars are of the symbol size.

9. Correlation functions on the 32x32 lattice at $T = 0.45$, with different Trotter numbers M.

Chapter 1

Introduction

The discovery of high temperature superconductors [1] has brought about a resurgence of interest in two-dimensional quantum antiferromagnets. There are experimental and theoretical indications that spin dynamics plays a crucial role in the new superconducting mechanism which is believed to originate from purely electronic degrees of freedom [2-4]. Neutron scattering experiments on the parent compound La_2CuO_4 reveal a rich magnetic structure [4,5]. Over a wide temperature range, copper spins in Cu-O planes exhibit strong two-dimensional antiferromagnetic correlations, but without broken symmetry.

The simplest theoretical description of the system is provided by the spin-1/2 antiferromagnetic Heisenberg model (AFHM), which is also the strong coupling limit of the Hubbard model at half-filling (*i.e.*, with charge fluctuations integrated out):

$$H = J \sum_{\langle ij \rangle} (S_i^x S_j^x + S_i^y S_j^y + S_i^z S_j^z) \tag{1.1}$$

The isotropic model of Eq. (1) is a special case of the general anisotropic Heisenberg Hamiltonian with nearest neighbor interaction:

$$H = \sum_{\langle ij \rangle} (J_x S_i^x S_j^x + J_y S_i^y S_j^y + J_z S_i^z S_j^z) \tag{1.2}$$

The summation $\sum_{\langle ij \rangle}$ goes over all the nearest neighbor pairs on a square lattice and S_i is the spin operator at the i-th site. The energy scales are set by the effective exchange couplings J_x, J_y, and J_z.

Much of the renewed interest in 2D antiferromagnets is rooted in P. W. Anderson's suggestion [2] that novel kinds of excitations which exist in strongly fluctuating spin systems may ultimately lead to superconductivity in doped materials such as $La_{2-x}Sr_xCuO_4$. It is important, therefore, to understand the magnetic properties of the stoichiometric compound (La_2CuO_4), which is an antiferromagnetic insulator. The immediate goal is to understand the nature of spin fluctuations in CuO_2 layers, which seem to be responsible for superconductivity.

It is very plausible that the intervening oxygen ions mediate an antiferromagnetic coupling via the superexchange mechanism, leading to the $S = 1/2$ Heisenberg model of Eq. (1). Previous studies and our work show that, to a large extent, static spin correlations measured in neutron scattering experiments may be adequately described by an $S = 1/2$ nearest neighbor (NN) quantum Heisenberg model (QHAF) on a square lattice, with large isotropic antiferromagnetic effective exchange J of the order of 1500 K. Interplanar coupling and spin anisotropies are rather small [3,4]. Ising-like anisotropies could lead to a qualitatively different behavior, but they are likely to be less important than the interlayer coupling. Effective interlayer coupling J^1 is about five orders of magnitude smaller than intralayer exchange J, which is not surprising, considering the layered structure of La_2CuO_4. Although small, when combined with large regions of strongly correlated spins in a layer, this coupling drives the system into a unique 3D ordered state. This transition occurs at a finite temperature (Neél temperature, T_N), and is entirely due to the interlayer coupling and symmetry-breaking perturbations, since the pure 2D Heisenberg antiferromagnet cannot have a finite temperature transition to an ordered state. If the spins in a layer are correlated over a region of linear size ξ, the Néel temperature may be estimated by: $k_B T_N \simeq J^1 \xi^2$. Using the experimental values for $T_N \simeq 200K$ and $\xi \simeq 100$ lattice spacings [4], one arrives at the estimate $J^1/J \simeq 10^{-5}$. Above the crossover region close

to T_N, the 2D correlations should be relatively insensitive to such a small coupling. Also, the staggered magnetization in the ground state is practically unaffected by this coupling.

A very difficult question to answer is the effect of next nearest neighbor (NNN) or even further exchanges. It is likely that, as long as the ground state is ordered, the long wavelength properties are going to be unaffected. However, for sufficiently strong NNN antiferromagnetic interaction, the ground state presumably loses the long-range order. The nature of such a state is highly controversial [6]. The path integral Monte Carlo method, as developed here, is not suitable for studying such a system. Since the NN model can be very successfully studied by the present method, and, as it turns out, some very important experiments are adequately explained by the model, we will restrict ourselves to NN Hamiltonian only.

Now that the choice of the model is somewhat justified, the first question one might attempt to answer is the nature of the ground state. It will determine, to some extent, the nature of the low lying excitations, which, in turn, determine the low temperature behavior of the model, as observed in the experiments.

Two-dimensional systems in the extreme quantum limit present formidable challenges for a theorist. Strong fluctuations due to low dimensionality are coupled with pronounced quantum fluctuations. Consequently, it is difficult to design reasonably controlled perturbative approaches. One of the most profound fluctuation effects is the absence of long-range order at finite temperatures, which is rigorously established by the Mermin-Wagner theorem [7]. On the other hand, the exact results, analogous to those that exist in one dimensional systems are lacking.

Analytical approaches employ different perturbation schemes (*e.g.*, spin wave theory, renormalization group approaches, series expansions, large-N expansions, etc.), variational

treatments, or exact diagonalizations for small systems.

On the other hand, Monte Carlo methods can provide nonperturbative results for finite systems. If the system size is large enough, quantitatively reliable conclusions can be drawn about the thermodynamic limit. Monte Carlo methods have been successfully applied to study both ground state and finite temperature properties of AFHM.

Recent gains in computational power, particularly the advance of parallel supercomputers, made it feasible to perform simulations on systems which are almost two orders of magnitude larger than those achieved before [8]. Statistical physics problems of this kind, due to its regularity and short-range interactions, are particularly suited for distributed memory, medium grain-size architectures with high-speed processing elements, like MarkIIIfp Caltech/JPL hypercube [9].

This thesis presents development and applications of a Monte Carlo algorithm, based on a generalized path integral approach to quantum statistics, for this parallel computer. The algorithm is designed with speed and efficiency on this particular architecture in mind. A general model given by Eq. (2) can be studied, but the major interest exists for two specific cases: the isotropic model of Eq. (1), and the XY model, defined by $J_x = J_y = J$ and $J_z = 0$. The classical 2D XY model exhibits a finite temperature phase transition to a state with no long-range order, but with a diverging correlation length. This is the celebrated Kosterlitz-Thouless transition [10]. It is interesting to examine the effects of quantum fluctuations on a subtle transition like this one, particularly in the light of controversial results coming from numerical studies [11] and a lack of reliable analytical results for the quantum model [12]. Large-scale simulations were carried out for these two models. A rather comprehensive picture of their finite temperature behavior is obtained. The contact with current theoretical and experimental understanding is successfully established.

Chapter 2 is devoted to a discussion of the spin wave picture, which is the natural perturbation-theoretic framework to study spin models. It is used to formulate a qualitative low temperature picture of the system. The physical quantities useful in characterization of spin systems are defined. Chapter 3 is an exposition of the Suzuki-Trotter transformation [13], which is the essential part of the method used in this work to study the model nonperturbatively. The implementation of this formalism to the two spin models is discussed in detail, with particular emphasis on the effect of conservation laws on the Monte Carlo updating procedure. The algorithmic issues regarding spin packing and parallelization are given in Chapter 4. The methods of measurement of interesting physical quantities and technical details of the simulations are given in Chapter 5. The results and the comparison with experiment and theory for the Heisenberg model are presented in Chapter 6, and in Chapter 7, for the XY model. Appropriately commented parts of the source code are given in the Appendix.

References

[1] J. G. Bendorz and K. A. Müller, Z. Phys. **B64**, 189 (1986).

[2] P. W. Anderson, Science **235**, 1196 (1987); T. M. Rice, Z. Phys. B **67**, 141 (1987) and references therein.

[3] G. Shirane, Y. Endoh, R. J. Birgenau, M. A. Kastner, Y. Hidaka, M. Oda, M. Suzuki, and T. Murakami, Phys. Rev. Lett. **59**, 1613 (1987); D. Vaknin, S. K. Sinha, D. E. Moncton, D. C. Johnston, J. M. Newsam, C. R. Safinya, and H. E. King, Jr., Phys. Rev. Lett. **58**, 2802 (1987); K. B. Lyons, P. A. Fleury, J. P. Remeika, A. S. Cooper, and T. J. Negran, Phys. Rev. B **37**, 2353 (1988).

[4] Y. Endoh, K. Yamada, R. J. Birgenau, D. R. Gabbe, H. P. Jenssen, M. A. Kastner, C. J. Peters, P. J. Picone, T. R. Thurston, J. M. Tranquada, G. Shirane, Y. Hidaka, M. Oda, Y. Enomoto, M. Suzuki, and T. Murakami, Phys. Rev. B **37**, 7443 (1988).

[5] G. Aeppli, S. M. Hayden, H. A. Mook, Z. Fisk, S-W. Cheong, D. Rytz, J. P. Remeika, G. P. Espinosa, and A. S. Cooper, Phys. Rev. Lett. **62**, 2052 (1989).

[6] See for example, I. Affleck, Phys. Rev. **B 37**, 5186 (1988).

[7] N.D. Mermin and H. Wagner, Phys. Rev. Lett **17**, 1133 (1966).

[8] H.-Q. Ding and M. S. Makivic, Phys. Rev. Lett. **64**, 1449 (1990).

[9] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood Cliffs, New Jersey (1988).

[10] J.M. Kosterlitz and D.J. Thouless, J. Phys. **C6**, 1181 (1973); J.M. Kosterlitz, J. Phys. **C7**, 1046 (1974).

[11] H. De Raedt and A. Lagendijk, Phys. Rev. **B33**, 5102 (1986); E. Loh, Jr., D.J. Scalapino and P.M. Grant, Phys. Rev. **B33**, 5014 (1986).

[12] J. Rogiers and R. Dekeyser, Phys. Rev **B** 13, 4886 (1976); D. D. Betts and M. Plischke, Can. J. Phys. 54, 1553 (1976); T. Tatsumi, Prog. Theor. Phys. 65, 451 (1981); H. Takano and M. Suzuki, J. Stat. Phys. 26, 635 (1981).

[13] M. Suzuki, Prog. Theor. Phys. **56**, 1457 (1976); M. Suzuki, J. Stat. Phys. **43**, 883 (1986).

Chapter 2

The Spin Wave Theory

If the magnitude of the spin S is increased, so that the product JS^2 is kept fixed at a finite value, while the quantum mechanical operator $\mathbf{S}/S$ tends to a classical unit vector $\mathbf{\Omega}$, we arrive at the classical analog of QAHM. This procedure implies that $1/S$ is the natural expansion parameter as one goes from the classical to the quantum limit.

The ground state is trivial for the classical counterpart. The square lattice is a bipartite lattice, *i.e.* we can identify two sublattices A and B, such that nearest neighbors of a site in A belong to sublattice B, and vice versa. The classical energy is minimized by the spin configuration where all spins on sublattice A point in a single direction, while the spins in B are also aligned but in the opposite direction. This is the, so-called Néel state. The system has finite sublattice magnetization, which, in this case, has maximum possible value per spin, $N = S$. As soon as the quantum parameter $1/S$ is turned on from 0, this is not an eigenstate of the quantum mechanical Hamiltonian. The situation is quite different in the ferromagnet, where the effective exchange J is negative. The spins of a ferromagnet in the ground state are all aligned in the same direction, yielding the maximum possible value for magnetization per spin, $M = S$. The ferromagnet is frozen at $M = S$ for *any* value of $1/S$.

Knowing that the Néel state is no longer the ground state, the natural question is how different is the true ground state from the Néel state as $1/S$ goes from zero to the extreme quantum limit, $1/S = 2$. In 1D, as soon as $1/S$ is turned on, the average

sublattice magnetization, N, becomes zero immediately [1]. Thus, the ground state is qualitatively different from the Nèel state, when quantum fluctuations are included. In 2D, for any bipartite lattice, there exists a rigorous proof that the ground state is Nèel like, *i.e.* N is finite, for sufficiently large spins, $S \geq 1$ [2]. Unfortunately, the proof is not extended to $S = 1/2$. Assuming that the ground state is ordered, a natural perturbation theoretic framework is provided by the spin wave picture, where the system is mapped onto a collection of weakly interacting quasiparticles, after the quasi-harmonic precession of spins around the direction of ordering is quantized [3].

The situation is much more problematic in the absence of a Néel-type picture to begin with, since there are no reliable analytic methods to treat strongly interacting systems. In the absence of classical order in the ground state, the nature of excitations is unclear, the continuum limit of the theory is questionable, and subtle topological effects may alter the picture qualitatively [4]. It is possible that these topological considerations lead to qualitatively different pictures for integer and half-odd integer spins [4,5], particularly if the Lieb-Shultz-Mattis theorem [6], proved in 1D, can be extended to 2D [4]. By now, there is overwhelming numerical evidence that the ground state of $S = 1/2$ QAFM possesses Néel-type order [7-9]. To drive the system away from classical ordering, one must introduce frustration in some form to amplify fluctuations, for example through next nearest neighbor coupling.

Spin wave theory attempts to construct a perturbation expansion in powers of $1/S$, around the classical Néel state defined by:

$$S_i^z = \begin{cases} -S & \text{if } i \in A; \\ +S & \text{if } i \in B; \end{cases} \tag{2.1}$$

There is a class of transformations which establish correspondence between the spin operators and a set of boson spin deviation operators [10]. On sublattice A, one applies the

operator transformation:

$$\begin{aligned} S_i^+ &= \sqrt{2S} a_i^\dagger (1 - a_i^\dagger a_i/(2S))^\alpha \\ S_i^- &= \sqrt{2S} (1 - a_i^\dagger a_i/(2S))^{1-\alpha} a_i \\ S_i^z &= -S + a_i^\dagger a_i \end{aligned} \tag{2.2}$$

where the operators $a_i^\dagger$ and a_i satisfy the usual boson commutator relations. On sublattice B, one introduces another set of bosonic operators $b_i^\dagger$ and b_i, via the transformation

$$\begin{aligned} S_j^+ &= \sqrt{2S} (1 - b_j^\dagger b_j/(2S))^{1-\beta} b_j \\ S_j^- &= \sqrt{2S} b_j^\dagger (1 - b_j^\dagger b_j/(2S))^\beta \\ S_j^z &= S - b_j^\dagger b_j \end{aligned} \tag{2.3}$$

Numbers α and β may be chosen independently. The famous Holstein-Primakoff (HP) [11] transformation corresponds to $\alpha = \beta = 1/2$, while Dyson-Maleev [12] formalism corresponds to $\alpha = 1$, $\beta = 0$ (or vice versa). HP formalism preserves the mutual adjoint relationship between spin raising and lowering operators, at a price of introducing the square root of an operator. When expanded in a Taylor series, the square root leads to an infinite number of terms in the Hamiltonian. In DM formalism the transformed Hamiltonian is not hermitian, but it has only terms which are quadratic and quartic in bose operators. This property frequently leads to a better controlled behavior in perturbation theory [13].

Mapping onto a bosonic Hamiltonian is certainly advantageous because of simple commutators, which lead to simple rules for constructing perturbation theory, based on Wick's theorem [14]. The problem is that the ordinary Fock space is much larger than the spin space. It can be divided into two parts: (1) $n_i \leq 2S$ and (2) $n_i > 2S$. To dispose of

unphysical boson states, corresponding to the occupation numbers $n_i > 2S$, one introduces the projectors onto the physical subspace [15]

$$P(S)|n_i>=\begin{cases}|n_i> & \text{if } n_i \leq 2S \\ 0 & \text{if } n_i > 2S\end{cases} \tag{2.4}$$

which may be expressed in terms of bose operators

$$\begin{aligned} P(S) &= \sum_{n=0}^{2S} P_n \\ P_0 &= \sum_{j=0}^{\infty} \frac{(-1)^j}{j!}(a^\dagger)^j a^j \\ P_n &= \frac{1}{n!}(a^\dagger)^n P_0 a^n \end{aligned} \tag{2.5}$$

The "restricted" bose operators $a(S)$ and $a^\dagger(S)$ may be used in place of ordinary bose operators:

$$a \Rightarrow a(S) = aP(S) \qquad a^\dagger \Rightarrow a^\dagger(S) = P(S)a^\dagger \tag{2.6}$$

to achieve the exact mapping between spin and boson Hilbert spaces. When the projectors are expanded, new terms appear in the Hamiltonian, which go under the name of "kinematical" interaction [12]. It was shown by F. J. Dyson [12] that, for a ferromagnet, there exists a gap between the physical states and lowest lying unphysical state. This justifies the approximation, common for *both* ferromagnets and antiferromagnets, which neglects the effects of kinematical interactions by replacing the projector with unity: $P(S) \simeq 1$. An analogous proof does not exist for antiferromagnets, but the approximation is justifiable as long as $\langle N_i \rangle = \langle a_i^\dagger a_i \rangle \ll 2S$. This is certainly true for large spins and low temperatures, but cannot be known *a priori*. This condition, if satisfied, then serves as a check of the self-consistency of a calculation.

The original QAFM Hamiltonian becomes, using the DM formalism, neglecting the kinematic interactions, and Fourier transforming the boson operators [10]:

$$\begin{aligned} H =& H_0 + V \\ H_0 =& \frac{K}{2} \sum_{\mathbf{q}} \psi^\dagger_{\mathbf{q}} (\mathbf{I} + \gamma_{\mathbf{q}} \sigma_1) \psi_{\mathbf{q}} \\ V =& \frac{K}{2S} \sum_{\mathbf{q}_1 \mathbf{q}_2 \mathbf{q}_3 \mathbf{q}_4} \psi^\dagger_{\mathbf{q}_1,\alpha} \psi^\dagger_{\mathbf{q}_2,\beta} V^{\alpha\beta\gamma\delta}(\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4) \psi_{\mathbf{q}_3,\gamma} \psi_{\mathbf{q}_4,\delta} \end{aligned} \tag{2.6}$$

where $\psi^\dagger_{\mathbf{q}} = (a^\dagger_{\mathbf{q}}, b_{-\mathbf{q}})$ is a two-component operator, σ_i is a Pauli matrix, and $\mathbf{I}$ is the identity matrix. For a lattice with coordination number z, K is defined as $K = 2JzS$. If the coordinates of nearest neghbors of a lattice site $\mathbf{r}_i$ are denoted by $\mathbf{r}_j$, then the Fourier transform of the unit cell is

$$\gamma_{\mathbf{q}} = \frac{1}{z} \sum_j e^{i\mathbf{q}(\mathbf{r}_i - \mathbf{r}_j)} \tag{2.7}$$

The quadratic part of the Hamiltonian, H_0, corresponds to a system of noninteracting bosons, *i.e.*, spin waves. Let us, for a moment, leave aside the interaction part, V. The noninteracting time-ordered Green's function is simply:

$$G^0(\mathbf{q}, \omega) = [\omega \sigma_3 - K(\mathbf{I} + \gamma_{\mathbf{q}} \sigma_1)]^{-1} \tag{2.8}$$

Its poles give the spectrum of elementary excitations: $\epsilon(\mathbf{q}) = K\sqrt{1 - \gamma^2_{\mathbf{q}}}$. Due to the symmetry under the interchange of sublattices A and B, there are 2 degenerate spin wave modes. For small momenta, the spectrum is linear: $\lim_{q \to 0} \epsilon(q) = cq$, where the *noninteracting* spin wave velocity is $c = 2S\sqrt{2}Ja/\hbar$ for a square lattice with lattice constant a.

The noninteracting spin wave Hamiltonian H_0 is easily diagonalized via the Bogoliubov transformation [16]:

$$\phi_{\mathbf{q}} = (\mathbf{I} \cosh(\frac{1}{2} \operatorname{atanh}(\gamma_{\mathbf{q}}) - \sigma_1 \sinh(\frac{1}{2} \operatorname{atanh}(\gamma_{\mathbf{q}})) \psi_{\mathbf{q}} \tag{2.9}$$

where the new bose operator set $\phi_{\mathbf{q}}^{\dagger} = (\alpha_{\mathbf{q}}^{\dagger}, \beta_{-\mathbf{q}})$ is just a rotated original one. The Hamiltonian becomes

$$H_0 = \frac{1}{2} \sum_{\mathbf{q}} \epsilon_{\mathbf{q}} \phi_{\mathbf{q}}^{\dagger} \phi_{\mathbf{q}} \tag{2.10}$$

This Hamiltonian is not normal-ordered. When rewritten explicitly in terms of α and β operators, and casting the operators in normal order, the spin wave Hamiltonian will include, in addition to quartic terms, a constant term which represents the shift of the classical energy even when no spin waves are present:

$$H_0 = E_0 + \frac{1}{2} \sum_{\mathbf{q}} \epsilon_{\mathbf{q}} (\alpha_{\mathbf{q}}^{\dagger} \alpha_{\mathbf{q}} + \beta_{\mathbf{q}}^{\dagger} \beta_{\mathbf{q}}) \tag{2.11}$$

where $E_0 = 1/2 \sum_{\mathbf{q}} \epsilon_{\mathbf{q}}$. This term comes entirely from zero-point motion, and is a trivial example of renormalization effects due to quantum fluctuations.

Another quantity which is renormalized even at the level of noninteracting spin waves is the staggered magnetization, N. In the spin wave ground state, it is equal to $N = \langle S_z \rangle = S - \langle a_i^{\dagger} a_i \rangle = S - \langle b_i^{\dagger} b_i \rangle = S - 0.197$ for the square lattice and spin $S = 1/2$ [17]. The fact that this renormalization is finite hints that the assumption of Néel order in the ground state is not unjustified, but certainly does not prove it. It is also easy to calculate the uniform magnetic susceptibility $\chi_{\perp}$, in the direction perpendicular to the sublattice magnetization (in units were $g\mu_B \hbar = 1$): $\chi_{\perp} = \hbar^2 / 8Ja^2$.

One of the most important properties that characterize the ground state is the spin stiffness, which measures the rigidity of the spin assembly when a slow twist is imposed in the direction of the order parameter. For a system of arbitrary dimensionality, and for any temperature, it can be rigorously defined in terms of static equilibrium properties in the following manner [18]: Suppose that a system is confined to a long cylinder of cross sectional area A and length L. Introduce the "wall" potential at one end of the cylinder, which forces the order parameter to align with phase angle θ. At the other end of the

cylinder, we imagine two possibilities: we may introduce "wall" potential that will enforce a phase angle θ or $-\theta$. The first one will give a uniform system, the second one will induce a slow twist in the phase of the order parameter as one goes from one end of the cylinder to the opposite one. If we denote the free energy in the uniform case by $F(+\theta,+\theta)$, and the free energy in the twisted case by $F(+\theta,-\theta)$, then the spin stiffness ρ_s and the helicity modulus Υ are defined by:

$$\rho_s(T) = \frac{m^2}{\hbar^2}\Upsilon = lim_{A,L\to\infty}(\frac{L}{2\theta^2 A})[F(+\theta,-\theta) - F(+\theta,+\theta)] \tag{2.12}$$

For calculational purposes, it is more convenient to use a different definition [19]. We imagine that a long wavelength twist of wave number k_0 is introduced in the order parameter. Then, the thermodynamic limit is taken first. The free energy will depend on both temperature and k_0. Since it cannot depend on the "handedness" of the twist, to lowest order the free energy will have a term $\propto k_0^2$. The spin stiffness then may be defined as:

$$\rho_s(T) = \frac{m^2}{\hbar^2}\Upsilon = lim_{k_0\to 0}\frac{\partial^2 F(T,k_0)}{\partial k_0^2} \tag{2.13}$$

It is easy to show that, for the spin wave Hamiltonian H_0, $\rho_s = c^2\chi_\perp = JS^2$.

So far the interaction part V of the Hamiltonian (2.6), which is of order $1/S$, was neglected. It leads to scattering and finite lifetime of the spin waves. When treated in perturbation theory, it leads to renormalization of the physical quantities defined above, but does not change the spin wave picture drastically. Spin waves remain relatively well-defined excitations, especially the long wavelength ones. In 3D, a tedious analysis [13] of the most important contributions to the self energy $\Sigma(\mathbf{q},\omega(\mathbf{q}))$ on resonance, at low temperatures leads to the decay rate which scales as $\tau_k \propto k^2$, while the dispersion, although renormalized, remains linear at low momenta, $\epsilon_k \propto k$. The situation is not so clear in 2D. Renormalization of a physical quantity due to spin wave interactions can be expressed as

a series in powers of $1/S$. Its convergence properties are not well understood. The series is probably asymptotic and is missing out terms like $e^{-\alpha S}$. It is imperative to calculate quantities by alternate methods, particularly for small spins.

Whatever the method of calculation, as long as the spin wave picture persists, the effects of quantum fluctuations can be expressed by the multiplicative renormalization factors [20]:

$$\begin{aligned} c &= \frac{2S\sqrt{2}Ja}{\hbar} Z_c(S) \\ \chi_\perp &= \frac{\hbar^2}{8Ja^2} Z_\chi(S) \\ \rho_s &= JS^2 Z_{\rho_s}(S) \end{aligned} \tag{2.14}$$

To lowest nontrivial order in spin wave theory [17]

$$\begin{aligned} Z_c &= 1 + 0.158/2S + O((1/2S)^2) \\ Z_\chi(S) &= 1 - 0.552/2S + O((1/2S)^2) \end{aligned} \tag{2.15}$$

and, using the hydrodynamic relation $\rho_s = c^2\chi_\perp$ [20], we have

$$Z_{\rho_s}(S) = JS^2(1 + 0.158/2S)^2(1 - 0.552/2S) \tag{2.16}$$

Singh and Huse [7] have developed a method that goes far beyond spin wave theory to calculate the ground state properties. They start from the Ising limit of the model, *i.e.* they set $J_x = J_y = 0$. The ground state is Néel state again. Then $J_x = J_y = J_\perp$ is turned on and a perturbation theory is based on the expansion in powers of $J_\perp$. To reach the isotropic Heisenberg limit $J_\perp = J_z = J$ they do a Padé extrapolation. They obtain the following numbers: $Z_c = 1.18 \pm 0.02$, $Z_\chi = 0.52 \pm 0.03$ and $Z_{\rho_s} = 0.72 \pm 0.02$. For the staggered magnetization they find $N = 0.302 \pm 0.007$. The spin wave results agree reasonably with these values: $Z_c = 1.158$, $Z_\chi = 0.448$, $Z_{\rho_s} = 0.60$ and $N = 0.303$. It is interesting that the $O(1/S)$ term in the spin wave expansion of N is identically zero [17]. The remarkable agreement with the result of Singh and Huse, which is also

verified by Monte Carlo [9] and variational calculations [21], suggests that the next term may be 0 as well. So far, there is a body of evidence, coming from numerical studies, like zero and finite temperature Monte Carlo [9], exact diagonalizations for small lattices [8] and variational calculations [21] that the ground state of QAFM is ordered and can be adequately understood within the spin wave picture. Of course, there are quantities that depend on the details of spin wave spectra and lifetimes, particularly near the zone boundary, like Raman scattering intensities [22], where it is essential to go beyond the spin wave theory.

The extension of these results to finite temperatures is not quite straightforward. The major obstacle is the fact that there is no ordered state to begin with. The Mermin-Wagner theorem precludes the possibility of having finite staggered magnetization at any finite temperature. But the existence of long-range order in the ground state suggests that spins at nonzero temperatures become correlated over large and larger distances as temperature is lowered [20,23]. It makes sense to talk about *long* short-range order. On length scales shorter than the correlation length $\xi(T)$, it is impossible to distinguish it from real long-range order. Thus, the spin waves which satisfy $k\xi \gg 1$ (*i. e.*, $\lambda \ll \xi$), are still going to be well-defined excitations. They should not interact strongly at low momenta. However, as $k\xi \simeq 1$, they become overdamped, and the behavior becomes diffusive as $k\xi \to 0$. Since the order parameter is not conserved, at $k = 0$, the relaxation rate is finite, and probably not very different from $k = \xi^{-1}$. This damping rate should then set the frequency scale at low temperatures. The renormalization group analysis suggests that the spin wave running coupling constant in 2D grows as the length scale increases, leading to an increase of multimagnon excitations. Furthermore, the excitations are not Goldstone bosons as $k \to 0$, since the finite correlation length necessarily implies a gap in the spectrum [23]. As temperature is lowered, the gap goes to zero since the correlation

length diverges, recovering the usual $T = 0$ spin waves, which are Goldstone bosons arising from the broken rotational symmetry of the Néel state.

To verify the assumptions behind this qualitative picture we need a nonperturbative calculation of static and dynamic spin correlation functions. Finite temperature Monte Carlo methods based on the Suzuki-Trotter transformation [24] provide a convenient nonperturbative framework. The Suzuki-Trotter transformation is a powerful method to treat nonfrustrated quantum spin systems. It is a first principles calculation on the model, and the systematic errors are easily controlled. The next chapters are devoted to the exposition of the method and the results. The numerical results are then compared with perturbative approaches [20,23,25] based on the spin wave picture and neutron scattering experiments [26].

References

[1] E. H. Lieb and D. C. Mattis, in *Mathematical Physics in One Dimension*, (Academic Press, New York, 1966), Ch. 6.

[2] E. J. Neves and J. F. Peres, Phys. Lett. **114A**, 331 (1986); T. Kennedy, E. H. Lieb, and B. S. Shastry, J. Stat. Phys. **53**, 1019 (1988).

[3] P. W. Anderson, Phys. Rev. **86**, 694 (1952).

[4] I. Affleck, Phys. Rev. **B 37**, 5186 (1988).

[5] F. D. M. Haldane, Phys. Rev. Lett. **61**, 1029 (1988).

[6] E. Lieb, T. Schultz, and D. Mattis, Ann. Phys. (N. Y.) **16**, 407 (1961).

[7] R. R. P. Singh, Phys. Rev. B **39**, 9760 (1989); R. R. P. Singh and D. A. Huse, Phys. Rev. B **40**, 7247 (1989).

[8] J. Oitmaa and D. D. Betts, Can. J. Phys. **56**, 897 (1978); J. E. Hirsch and S. Tang, UCSD preprint.

[9] J. D. Reger and A. P. Young, Phys. Rev. B **37**, 5978 (1988); M. Gross, E. Sanchez-Velasco, and E. Siggia, Phys. Rev. B **39**, 2484 (1989); T. Barnes and A. S. Swanson, Phys. Rev. B **37**, 9405 (1988).

[10] W. L. Ridgeway, Phys. Rev. B. **25**, 1931 (1982).

[11] T. Holstein and H. Primakoff, Phys. Rev. **58**, 1098 (1940);

[12] F. J. Dyson, Phys. Rev. **102**, 1217 (1956); **102**, 1230 (1956); S. V. Maleev, Sov. Phys., JETP **64**, 654 (1958).

[13] A. B. Harris, D. Kumar, B. I. Halperin, and P. C. Hohenberg, Phys. Rev. B **3**, 961 (1971).

[14] J. W. Negele, H. Orland, in *Quantum Many-Particle Systems* (Addison Wesley, New York 1988).

[15] D. C. Mattis, in *The Theory of Magnetism*, (Harper & Row, New York, 1965), Ch. 6; P. B. Weichman, Ph.D. Thesis, Cornell University, 1985.

[16] N. Bogoliubov, J. Phys. USSR **11**, 23 (1947).

[17] T. Oguchi, Phys. Rev. **117**, 117 (1960).

[18] M. E. Fisher, M. N. Barber, and D. Jasnow, Phys. Rev. A **8**, 1111 (1973).

[19] J. Rudnick and D. Jasnow, Phys. Rev. B **16**, 2032 (1977).

[20] S. Chakravarty, B. I. Halperin, and D. Nelson, Phys. Rev. Lett. **60**, 1057 (1988); Phys. Rev. B **39**, 2344 (1989).

[21] D. Huse and V. Elser, Phys. Rev. Lett. **60**, 2531 (1988);

[22] R. R. P. Singh, P. A. Fleury, K. B. Lyons and P. E. Sulewski, Phys. Rev. Lett. **62**, 2736 (1989).

[23] M. Takahashi, Phys. Rev. B. **40**, 2494 (1989).

[24] M. Suzuki, Prog. Theor. Phys. **56**, 1457 (1976).

[25] A. Auerbach and D. P. Arovas, Phys. Rev. Lett. **61**, 617 (1988).

[26] Y. Endoh, K. Yamada, R. J. Birgenau, D. R. Gabbe, H. P. Jenssen, M. A. Kastner, C. J. Peters, P. J. Picone, T. R. Thurston, J. M. Tranquada, G. Shirane, Y. Hidaka, M. Oda, Y. Enomoto, M. Suzuki, and T. Murakami, Phys. Rev. B **37**, 7443 (1988).

Chapter 3

Computational Method

3.1. The Suzuki-Trotter Transformation

The Suzuki-Trotter method [1] is based on M. Suzuki's generalization of the famous Trotter formula [2] for exponential operators:

$$e^{A+B} = \lim_{n\to\infty}(e^{A/n}\cdot e^{B/n})^n \tag{3.1}$$

In statistical physics, one is interested in evaluating the trace of the density operator, $\rho = exp(-\beta H)$, where $\beta = 1/k_B T$ is the inverse temperature and H is the Hamiltonian of the system. The first step towards obtaining a controlled approximation of this operator is to write the exponential as a product of exponentals with "small" arguments. In other words, thinking of ρ as a propagator in imaginary time, and of β as of the imaginary time interval, one breaks the whole time interval into a large number of short time slices, and rewrites the finite time propagator as a product of large number of short time propagators. This is the essential and exact step behind any path integral formulation. In the next step, one approximates the short-term propagators in such a manner that the matrix elements are readily evaluated. There is a considerable freedom in choosing the approximation, but they can be classified according to their error dependence on the number of time slices M, for large values of M. This number is commonly known as the Trotter number.

This can be formally written as [1]:

$$e^{-\beta H} = (e^{-\beta H/M})^M = lim_{M\to\infty}(f_n[-\beta H/M])^M \tag{3.2}$$

where the n-th order approximant f_n satisfies:

$$e^{-\beta H/M} = f_n[-\beta H/M] + O(\frac{1}{M^n}) \tag{3.3}$$

This relation then implies for the density operator:

$$e^{-\beta H} = (f_n[-\beta H/M])^M + O(\frac{1}{M^{n-1}}) \tag{3.4a}$$

and the partition function:

$$Z = Tre^{-\beta H} = Tr(f_n[-\beta H/M])^M + O(\frac{1}{M^{n-1}}) \tag{3.4b}$$

It can be shown that for the exponent of a sum of operators [1]:

$$f_2[\sum_{j=1}^{p} A_j] = \prod_{j=1}^{p} e^{A_j} \tag{3.5}$$

It is not difficult to generate much higher order approximants. For example, one can formally obtain any large order approximant [3] using

$$f_{n+1}(A/M, B/M) = e^{A/M} e^{B/M} e^{C_2/M^2} \cdots e^{C_n/M^n} \tag{3.6}$$

if the coefficients are chosen in the following manner:

$$C_2 = \frac{1}{2}[B, A], \quad C_3 = \frac{1}{3}[C_2, A + 2B], \cdots \tag{3.7a}$$

or, in general,

$$C_q = \frac{1}{q!}\{\frac{\partial^q}{\partial \lambda^q}[e^{-\lambda^{q-1} C_{q-1}} \cdots e^{-\lambda^2 C_2} e^{-\lambda B} e^{-\lambda A} e^{\lambda(A+B)}]\}|_{\lambda=0} \tag{3.7b}$$

One additional method of generating higher order approximants is borrowed from high temperature expansions [4], *i.e.*,

$$f_n[-\beta H/M] = \sum_{k=0}^{n} \frac{1}{k!}[-\beta H/M]^k \tag{3.8}$$

Since the expression on the right-hand side involves different powers of volume, it is better to go to the cumulant expansion [5], *i.e.*, make a resummation of the following form:

$$(f_n[-\beta H/M])_{pq} = exp(f_{pq}^{(n)}[-\beta H/M]) - (1 - \delta_{pq}) \tag{3.9}$$

It is easy to check order by order that the matrix elements of the new approximant are:

$$\begin{aligned} f_{pq}^{(1)}[A] &= < p|A|q > \\ f_{pq}^{(2)}[A] &= f_{pq}^{(1)}[A] + \frac{1}{2}(< p|A^2|q > - < p|A|q >^2) \end{aligned} \tag{3.10}$$

and similarly for higher order terms. It is known [6] that the convergence behavior of this type of approximant is worse for large system sizes, than if the Hamiltonian is broken into pieces and then Eq. (3.5) is used. For that reason, we used a method based on Eq. (3.5).

It is fortunate that the traces of decomposed operators, under rather general conditions, have better convergence properties than what Eq. (3.4) suggests. If the approximant $f_n(\{A_j\})$ of the type (3.5) satisfies the condition:

$$f_n(\{-A_j\})^{-1} = f_n(\{A_j\})^T \tag{3.11}$$

which is certainly true if the operators A_j are symmetric, $A_j^T = A_j$, then the approximant for the partition function is an even function of M [2]:

$$Z_n(M) = Tr[f_n\{(A_j/M)\}]^M = Z_n(-M) \tag{3.12a}$$

Therefore,

$$Z = Z_{2n}(M) + O(1/M^{2n}) \tag{3.12b}$$

A very similar result is valid if one calculates average values of operators which satisfy $Q^T = Q$, namely

$$< Q >= Q_{2n}(M) + O(1/M^{2n}) \tag{3.13}$$

where

$$Q_{2n}(M) = Tr(Q[f_{2n}(\{A_j\})]^M)/Z_{2n}(M) \tag{3.14}$$

Therefore, the systematic error of using a finite value of M, which is what one has to do in practice, is of the order of $1/M^2$ [2,6]. The exact result then may be obtained by calculating the quantities of interest for a few large values of M, and then extrapolating to $M \to \infty$, using the formula:

$$Q(M) = Q(\infty) + a/(MT)^2 + b/(MT)^4 + ... \tag{3.15}$$

where $Q(\infty)$ is the correct value.

To use formula (3.5) we break the Hamiltonian into 4 pieces [1], $H = \sum_{i=1}^{4} H_i$, each containing a commuting subset of nearest neighbor bonds on a square lattice as shown in Fig.1a. H_1 contains odd bonds in x-direction, H_2 contains odd bonds in y-direction, H_3 contains even bonds in x-direction, and H_4 contains even bonds in y-direction. The subHamiltonians H_i themselves do not commute, but it is important that their constituent bonds do. Then, to obtain the partition function, we apply the results (3.2) and (3.5):

$$Z = \mathrm{Tr}e^{-H/T} = \mathrm{Tr}(e^{-\Delta\tau H})^M = \mathrm{Tr}\lim_{M\to\infty}(\prod_{i=1}^{4} e^{-\Delta\tau H_i})^M \tag{3.16}$$

where M is the Trotter number and $\Delta\tau = \beta/M$. After $4M$ resolutions of unity are inserted between adjacent exponentials, we obtain:

$$Z = \lim_{\Delta\tau\to 0}\sum_{\{C\}}\langle C_1|e^{-\Delta\tau H_1}|C_2\rangle\langle C_2|e^{-\Delta\tau H_2}|C_3\rangle\cdots\langle C_{4M}|e^{-\Delta\tau H_4}|C_1\rangle \tag{3.17}$$

The intermediate states, C_j, which are chosen to diagonalize S_z, may be regarded as belonging to different time slices. Therefore, the original 2D quantum system of size LxL is mapped onto equivalent 3D classical system of size $LxLx4M$, where the labels of intermediate quantum states, being ordinary numbers, now serve as classical degrees of

freedom. Since subHamiltonians involve subsets of bonds that do not share common spins, the exponential $exp(-\beta H_j/M)$ decomposes into product of exponentials involving only 2 spins on a bond [7]. Hence, with this approximation of the short-term propagators one has to solve only a two-body problem, which is, of course, trivial. The Hilbert space of a bond has 4 states, and we have to calculate 16 matrix elements of the operator:

$$\rho_{ij} = e^{-\Delta\tau(J_x S_i^x S_j^x + J_y S_i^y S_j^y + J_z S_i^z S_j^z)} \tag{3.18}$$

between two adjacent time slices labeled by τ and $\tau + 1$:

$$\rho_{ij}(\tau, \tau+1) = \langle S_{i,\tau}^z S_{j,\tau}^z | \rho_{ij} | S_{i,\tau+1}^z S_{j,\tau+1}^z \rangle \tag{3.19}$$

where spatial indices i, j correspond to the particular bond. The four spin labels associated with this transfer matrix element may be identified with four classical spins sitting on the corners of an interacting 4-spin plaquette, which is then the basic building block of the interaction in the equivalent classical system. The "local density matrix" $W = \rho_{ij}$, which, in this context is the "local Boltzman factor," has the following nonzero matrix elements [7,8]:

$$\begin{aligned}
W_{++,++} &= e^{-\beta E(1)} = \lambda(1 + X_3) \\
W_{--,--} &= e^{-\beta E(2)} = \lambda(1 + X_3) \\
W_{+-,+-} &= e^{-\beta E(3)} = \lambda(1 - X_3) \\
W_{-+,-+} &= e^{-\beta E(4)} = \lambda(1 - X_3) \\
W_{+-,-+} &= e^{-\beta E(5)} = \lambda(X_1 + X_2) \\
W_{-+,+-} &= e^{-\beta E(6)} = \lambda(X_1 + X_2) \\
W_{++,--} &= e^{-\beta E(7)} = \lambda(X_1 - X_2) \\
W_{--,++} &= e^{-\beta E(8)} = \lambda(X_1 - X_2)
\end{aligned} \tag{3.20}$$

where "+" stands for spin up, "−" stands for spin down, and:

$$\begin{aligned}\lambda &= \cosh(K_x)\cosh(K_y)\cosh(K_z) - \sinh(K_x)\sinh(K_y)\sinh(K_z)\\ X_1 &= \frac{\tanh(K_x) - \tanh(K_y)\tanh(K_z)}{1 - \tanh(K_x)\tanh(K_y)\tanh(K_z)}\\ K_{x,y,z} &= \frac{J_{x,y,z}}{Mk_BT}\end{aligned} \tag{3.21}$$

and X_2 and X_3 are obtained by cyclic permutations in x, y and z.

We are interested in the models which have higher symmetry than the general anisotropic model. In the models with xy and full rotational symmetry, S_z is conserved. Because of that, it is easy to see that $X_1 = X_2$, and only 6 out of 16 spin configurations of a 4-spin plaquette have nonzero Boltzmann weights, *i.e.*, energies $E(7)$ and $E(8)$ become infinite. In the case of XY model, it is sometimes advantageous to work in the basis which diagonalizes S_y. Since S_y is conserved only modulo 2, there will be 8 nonzero matrix elements.

In order to interpret the matrix elements as local Boltzman factors, it is necessary to render them all real and positive. It is easy to see that for the antiferromagnet, the matrix elements corresponding to energies $E(5)$ and $E(6)$ are negative. These are the off-diagonal matrix elements, and they bring the signature of quantum fluctuations. The problem is easily solved for a bipartite lattice. To obtain positive matrix elements, it is sufficient to perform a cannonical transformation on one of the sublattices, say B, [9]:

$$H \Rightarrow UHU^{\dagger} = H = J\sum_{\langle ij\rangle}(-S_i^xS_j^x - S_i^yS_j^y + 2S_i^zS_j^z) \tag{3.22}$$

where the unitary operator U=exp($i\pi\sum_{i\in B}S_i^z$) maps $S^x \Rightarrow -S^x$ and $S^y \Rightarrow -S^y$. The energies of the allowed spin configurations for the Heisenberg antiferromagnet are given by:

$$\begin{aligned}\beta E(1) &= \beta E(2) = K = 1/4Mk_BT\\ \beta E(3) &= \beta E(4) = -K + \ln(\cosh(2K))\\ \beta E(5) &= \beta E(6) = -K + \ln(\sinh(2K))\end{aligned} \tag{3.23}$$

The negative sign of local Boltzman factors cannot be removed in the same fashion for a nonbipartite lattice, or a Hamiltonian that couples antiferromagnetically spins both between sublattices and within a sublattice, as is the case with the Hamiltonian having NN and NNN interactions. This is the major obstacle to performing simulations of frustrated spin models. It is presumably possible to find a representation that would not involve this problem, but it is likely that some of the features of these simple representations that work for nonfrustrated systems, like locality, are going to be lost.

Ferromagnetic and antiferromagnetic XY models are unitarily equivalent, through the cannonical map (3.22). The corresponding energies for the ferromagnetic XY model are:

$$\begin{aligned} \beta E(1) &= \beta E(2) = 0 \\ \beta E(3) &= \beta E(4) = \ln(\cosh(2K)) \\ \beta E(5) &= \beta E(6) = \ln(\sinh(2K)) \end{aligned} \tag{3.24}$$

The final result of the Suzuki-Trotter decomposition is an equvalent 3D classical system of Ising spins with rather unusual interactions. It may be regarded as a collection of 4-spin interacting plaquettes, denoted by shaded squares in Fig. 1b. Although we started from a highly isotropic quantum system, the equivalent classical system is very anisotropic. The couplings in the classical system also depend on temperature and Trotter number.

3.2. The Monte Carlo Method

The equilibrium averages of the equivalent classical system are evaluated in a stochastic computer simulation using the Metropolis algorithm [10]. If a spin configuration of the

whole system is denoted by $\mathbf{\Omega}$, its energy by $E(\mathbf{\Omega})$, and the quantity we are interested in is $Q(\mathbf{\Omega})$, we want to evaluate many-body integrals of this form:

$$< Q >= \frac{\int D\mathbf{\Omega} Q(\mathbf{\Omega}) e^{-\beta E(\mathbf{\Omega})}}{\int D\mathbf{\Omega} e^{-\beta E(\mathbf{\Omega})}} \tag{3.25}$$

Major contributions to the integral come from relatively small parts of the otherwise huge phase space. Therefore, to evaluate the multidimensional integral (3.25) efficiently, it is essential to sample different regions of phase space according to their contribution to the partition sum, *i.e.*, to perform importance sampling. If a discrete set of N phase space points $\mathbf{\Omega}_\nu$, $\nu = 1, \ldots, N$, is drawn according to its probability of occurring in the partition sum, $P(\mathbf{\Omega}) = exp(-\beta E(\mathbf{\Omega}))$, the integral (3.25) is approximated by:

$$< Q >_{MC}= \frac{1}{N} \sum_{\nu=1}^{N} Q(\mathbf{\Omega}_\nu) \tag{3.26}$$

and the error can be controlled by increasing the size of the sample, since, for large sample sizes, the central limit theorem assures that

$$< Q >_{MC}=< Q > +O(N^{-1/2}) \tag{3.27}$$

The method of generating the equilibrium distribution is borrowed from kinetic theory. A system at equilibrium has no memory of its history prior to equilibrium, and its equilibrium is independent of the details of dynamics, as long as the dynamics fulfills the detailed balance condition:

$$P(\mathbf{\Omega}_1) W(\mathbf{\Omega}_1 \rightarrow \mathbf{\Omega}_2) = P(\mathbf{\Omega}_2) W(\mathbf{\Omega}_2 \rightarrow \mathbf{\Omega}_1) \tag{3.28}$$

and is ergodic. One constructs a random walk through the phase space, which is essentially a Markov chain, so that the conditions of detailed balance and ergodicity are satisfied. The distribution of points approaches the fixed point, the equilibrium distribution, after a sufficiently long relaxation time, which depends on the dynamics. In practice, one starts

from an arbitrary initial spin configuration, generates trial moves (updates) in such a manner that whole of the phase space can be explored, and accepts or rejects the moves according to the chosen dynamics W. An experimentally determined number of steps at the beginning of a run are discarded to allow the system to thermalize, and then the measurements are taken. The constraints on the dynamics do not specify it uniquely. The one we used is the celebrated Metropolis algorithm:

$$W(\Omega_1 \rightarrow \Omega_2) = \begin{cases} exp(-\beta(E(\Omega_2) - E(\Omega_1)) & \text{if } E(\Omega_2) \geq E(\Omega_1) \\ 1 & \text{if } E(\Omega_2) < E(\Omega_1) \end{cases} \tag{3.29}$$

3.3. The Updating Procedure

Special care has to be exercised to ensure ergodicity for quantum problems [11,12]. In particular, we must not violate the quantum conservation laws. For an ordinary Ising system, the updating of spin configurations is trivial. Any spin configuration can be achieved through a sequence of one spin flips. This is not true for the equivalent classical system (3.17). Actually, a single spin flip *always* generates a configuration with infinite energy. A look at the matrix elements (3.20) reveals that an interacting 4-spin plaquette can have either 2 or 4 spins updated at a time. A 4-spin flip can always be achieved as a product of 2 2-spin flips on different edges of the plaquette. A 2-spin flip on a diagonal of the plaquette is a product of 2 2-spin flips, one of which is on a horizontal edge and the other on a vertical edge. Thus, *any* allowed update can be achieved as a sequence of updates of pairs of spins on plaquette edges. Since each spin is shared by 2 plaquettes, it follows immediately that one has to flip a *closed loop of spins*. This would be sufficient in a case where there are 8 nonzero matrix elements (*e.g.*, XY model in S_y representation). In the Heisenberg case, the conservation of S_z imposes the following constraints. If two spins on a horizontal edge of an interacting plaquette are being updated (both spins on the same time slice), the conservation law requires that the two spins be opposite. If two

spins on a vertical edge of an interacting plaquette are involved, one may update the spins only if they are both up or both down.

The ergodicity requirement then reduces to the problem of generating all possible closed loops. This could be done stochastically, in principle, by generating loops of arbitrary shape and length on a computer, but it is not efficient. A better way to generate all possible closed loops under these constraints is to exponentiate a set of 4 types of elementary updates which mimic the generators of the fundamental homotopy group on a torus, with the conservation law built in. The toroidal geometry is the consequence of spatial periodic boundary conditions (to preserve translational invariance) and periodic boundary conditions in the time direction (required by the trace operation).

We have two types of local updates and two types of global updates. The local updates are homotopic to the unit loop. The simpler local update ("space" flip) is shown in Fig. 2. One searches for a noninteracting loop ("space" loop), bounded by the edges of four interacting plaquettes. A "space" loop belongs to a single time slice. Due to the constraint, one may attempt to flip the four spins on a "space" loop (according to the Boltzmann weight in Eq. (3.23) or (3.24)) only if they are in a Néel configuration, as shown in Fig. 2.

Another local update ("time" flip) is shown in Fig. 3. It involves a noninteracting loop, extending in the time direction ("time" loop), bounded by eight interacting plaquettes. There are eight spins involved in the flip. Again, the flip is attempted only if the spins are in a Néel configuration, *i.e.*, the four spins on one vertical edge are all the same and opposite from the four spins on the other edge, as shown in Fig. 3.

The global update in time direction is shown in Fig. 4. One searches for a straight line of all up or all down spins, and flips them all. This move is responsible for generating

fluctuations of the total magnetization. The efficiency of this type of update depends on how many straight lines of up (or down) spins are available. Naively, one would expect that it is increasingly difficult to find such lines as temperature is lowered and number of time slices increased. This is not true, however, since the spatial correlation length grows faster than the system width in the "time" direction, L_t. Hence, at low temperatures, in the renormalized classical regime, temporal correlation length always saturates around $\xi_t \simeq L_t$. Indeed, we found that the straight lines are abundant enough at all temperatures we used in the simulations. The plaquette configurations corresponding to off-diagonal matrix elements $(E(5), E(6))$ are energetically costly in a Néel-like environment, their probability being $\propto 1/M$ for large Trotter numbers. However, once the Néel environment is "melted" a bit by these fluctuations, they are more likely to be accepted. The relative abundance of straight lines shows a slight increase with size for very small systems ($L < 12$), in agreement with the preceding arguments, since the singlet fluctuations are stronger for smaller systems. Also, once a straight line is found, it is feasible to accept a flip, because for large Trotter numbers the flip probability behaves as $\propto (e^{-a/M})^M = e^{-a}$, where a is a finite positive number. This enabled us to obtain correct results for the uniform susceptibility, which we calculated from magnetization fluctuations generated by these straight line flips.

In Fig. 5a, the other type of global update, extending in either spatial direction, is shown. Assuming that the numbering of time slices starts from zero, let us choose two adjacent time slices, denoted by t_1 and t_2, such that t_1 is even and t_2 is odd. Consider now a pair of neighboring straight lines, belonging to t_1 and t_2 respectively, such that they run in the y direction, *i.e.*, $x(t_1) = x(t_2) = const.$ It is easy to see that this pair consists of segments that involve only edges of interacting plaquettes. The same is true for a pair $y(t_1) = y(t_2) = const$, but in this case, the lower time slice t_1 must be odd and the upper one t_2 must be even. Therefore, global flips in the x and y directions are interleaved.

Furthermore, to satisfy the conservation law, the two lines must have the same, Néel-like spin configurations, as shown in Fig. 5a.

By employing the connection between spin and boson algebra discused in Chapter 2, this updating procedure can be given a useful alternative interpretation. Using a projector $P_i(0)$ onto a state with 0 particles on site i, we defined the restricted Bose operators $a_i = P_i(0)b_i$, $a_i^\dagger = b_i^\dagger P_i(0)$ which vanish outside the subspace with 0 or 1 particle. After the identification $|1> \Rightarrow |+>$ and $|0> \Rightarrow |->$, one has the equivalence $a^\dagger \Rightarrow \sigma_-$ and $a \Rightarrow \sigma_+$, where σ_- and σ_+ are spin lowering and raising operators. Since $N = a^+a = (1 - \sigma^z)/2$, the conservation of S_z may be interpreted as number conservation in this language.

By connecting the sites with spins down (or up, due to particle-hole symmetry), one obtains the worldlines of bosons with infinite on-site repulsion (hard core provided by the projector), propagating in imaginary time. There is a unique way of doing this, since the hard core prevents two lines from crossing each other. The updating of the spin system is equivalent to generating all possible configurations of world lines.

The elementary moves have a more transparent interpretation in the boson worldline picture. The "time" flip locally distorts a worldline in both spatial directions. Its effect is shown in Fig. 3, and it involves a single worldline. The "space" flip involves two nearby worldlines, and its effect is to exchange them and wrap them around one another, as shown in Fig. 2. Due to this permutation, the worldlines do not necessarily close on themselves, although the spin configuration is periodic in time.

The global move in the time direction destroys or creates static particles (holes) (Fig. 4). It connects subspaces with different particle number (magnetization). This global move is easily generalized to tracing the worldlines of any shape (mobile particles), which probably cannot be avoided in a strongly fluctuating system. It is not very complicated,

and it was done in the case of the XY model, but not for the purpose of updating the system. It is rather fast and is parallelizable, but it certainly affects the speed of the strictly regular algorithm.

Since the worldlines are on a torus they will belong to different homotopy classes, characterized by their winding numbers N_x and N_y. The global move in spatial directions is responsible for connecting these distinct topological subspaces. It breaks and reconstructs a number of worldlines of order L, and its effect is shown in Fig. 5b. Physically, this update is responsible for creating a coherent shift in the center of mass of the worldline assembly, which ultimately leads to finite spin stiffness, as will be discussed later. For periodic boundaries, this shift is of topological significance. This update is essential to move between distinct global topological constraints. However, the important thing is that the "space" flip may be used to emulate the effects of this type of update at the local level, as will be discussed later in the context of XY model.

In the early stages of this work we also experimented with another basis set, the coherent spin states. Being continuous, it is conceivable that they would be more appropriate to capture the dynamics, which is dominated by spin waves. However, they lead to complex transition probabilities and the phase fluctuations destroy the statistics.

References

[1] M. Suzuki, Prog. Theor. Phys. **56**, 1457 (1976); M. Suzuki, J. Stat. Phys. **43**, 883 (1986).

[2] M. Suzuki, Phys. Lett. **113A**, 299 (1985).

[3] M. Suzuki, J. Math. Phys. **26**, 601 (1985).

[4] G. S. Rushbrooke, G. A. Baker, Jr., and P. J. Wood, in *Phase Transitions and Critical Phenomena*, ed. by C. Domb and M. S. Green (Academic, New York, 1974), Vol. 3, Chap. 5.

[5] H. Takano, Prog. Theor. Phys. **73**, 332 (1985).

[6] R. M. Fye, Phys. Rev. B **33**, 6271 (1986).

[7] J. J. Cullen and D. P. Landau, Phys. Rev. B **27**, 297 (1983).

[8] M. Suzuki, S. Miyashita, A. Kuroda, Prog. Theor. Phys. **58**, 1377 (1976).

[9] J. W. Negele, H. Orland, in *Quantum Many-Particle Systems* (Addison Wesley, New York 1988).

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[11] M. Marcu, in *Quantum Monte Carlo Methods in Equilibrium and Nonequilibrium Systems*, M. Suzuki ed. (Springer, Berlin, 1987), p. 64.

[12] M. Gross, E. Sanchez-Velasco, and E. Siggia, Phys. Rev. B **39**, 2484 (1989).

Chapter 4

Algorithm and Parallel Implementation

4.1. Multispin Coding

Ising spins are 1-bit objects. This naturally leads to the idea of some sort of spin packing, since it is wasteful to represent a 1-bit data object with a 32-bit computer word. Spin packing saves memory space, significantly reduces index manipulation and facilitates vectorization. Hence, to achieve a high level of speed and efficiency, we implemented the Suzuki-Trotter method via a multispin coding algorithm [1]. The spins are packed into 32 bit computer words, along the imaginary time direction, which is singled out as the natural choice due to the anisotropy of the equivalent classical system. Packing in spatial directions introduces additional anisotropy between x and y directions, and would ultimately lead to different codes for calculating correlation functions in x and y directions. Furthermore, boundary conditions would be treated differently for different directions. Our choice of packing, on the other hand, results in a very homogeneous code, leads to an extremely simple implementation of the global flip in imaginary time direction, and reduces the amount of internode communication when the program is ported to the parallel machine. In addition, this type of packing is more flexible than packing in spatial directions regarding different types of spatial data decompositions and system sizes. The reason is that any multicoding algorithm is most efficient if the words are completely packed. For spatial packing, this means that one can study only systems with lattice sizes which are multiples of 32. The efficient temporal packing requires that the number of

time slices has to be a multiple of 32 (or, alternatively, the Trotter number, M, should be a multiple of 8). This leaves much more flexibility at low temperatures where Trotter numbers are necessarily large. On the other hand, one attempts to increase the lattice size in a more continuous fashion than in steps of 32 as temperature goes down, to keep the system large compared with the correlation length. Excessively big systems are very difficult to thermalize and tend to stay frozen in metastable regions as the temperature is lowered.

All the necessary checks and updates can be implemented through bitwise logical operations on words. In a single CPU operation, one retrieves information about 64 spins residing in 2 words. Also, index calculation is done once for a spin word, instead of calculating it for each spin separately. The same principles are applied for both local and global moves, but it is easier to illustrate them for local moves, as shown in Fig. 6.

A pair of adjacent words contains eight "time" loops (see Fig. 6). Because every two adjacent "time" loops share an interacting plaquette, we update all four odd "time" loops simultaneously in a vectorized fashion. The other four even "time" loops are updated next. Many of the useful quantities obtained in updating the four odd ones will also be used for the four even ones. We want to update the odd "time" loops 1, 3, 5 and 7 of the spin words S1 and S2 in Fig. 6. We first compute F = S1 [XOR] S2 , and then W = F [AND] MASK1, where MASK1 has "1"s located at the proper position of the "time" loop 1: MASK1=(0$\cdots$01111). The flip of "time" loop 1 is allowed if W [AND] MASK1 = MASK1 and (S1 [AND] MASK1) + (S2 [AND] MASK1) = 16 (which means that all four spins in S1 must be down and the four spins in S2 must be up, or vice versa). S1 is also XOR-ed with N1, N6 and N5 to obtain E1, E6 and E5, the information needed to compute the energy due to the three interacting plaquettes on S1 side (see Fig. 3). Note that the conservation law is now working to our advantage. Although a plaquette

may have 16 spin configurations, we know which one it is after a single logical operation. The reason is that 10 possible configurations are ruled out by the conservation law. Out of the remaining 6, once we know that the "time" flip is allowed, as established by the appropriate entry in W, we are left with 2 possibilities only: either the plaquette has antiferromagnetic configuration ($E(3)$ or $E(4)$, which are degenerate in the absence of a magnetic field) or ferromagnetic configuration ($E(1)$ or $E(2)$, which are again degenerate in the absence of a magnetic field). Thus, an XOR operation suffices to establish whether the spins on the lower edge of the plaquette are equal or opposite. In the simulation of the XY model in S_y basis, we do need the information about the spins on the vertical edge of the plaquette, since they are allowed to be both equal or opposite. To retrieve that piece of information, it is enough to XOR the word with itself, after it was right-shifted for a single bit. Similarly, S2 is XOR-ed with N2, N3 and N4. This piece of information does not need to be recomputed after the odd "time" loops are updated, since it is related to the spin configuration in the neighboring words. Finally, S1 is XOR-ed with (S1 [right-shift] 1) to obtain C which contains the information about the upper and lower interacting plaquettes (which are shared by adjacent "time" loops). This part has to be recomputed for the "even" loops. After masking N1-N6 and C with appropriate masks, we SHIFT, OR them together, to obtain X1 and X2 which contain the information about the eight "time" loops shared by S1 and S2. Notice that N1-N6 are used only *once* for all of the eight "time" loops.

To retrieve the information specific to the "time" loop 1, we calculate I1 = X1 [AND] MASK1 and I2 = X2 [AND] MASK1. (I1,I2) is a pair of small integers ($I1, I2 < 16$) in one-to-one correspondence with the spin configuration on a specific plaquette chosen by MASK1, and they uniquely determine the transition probability. Thus (I1,I2) is used as an index to fetch the transition probability stored in a small lookup table calculated at the beginning. Upon acceptance, the proper four spins in S1 are flipped by S1 = S1 [XOR]

MASK1, and similarly for S2. The update of the "time" loop 3 proceeds in the same manner as for loop 1, after left-shifting MASK1 for 8 bits. Now the 4 1-bits in MASK1 are located at the position of "time" loop 3, and they pick up the information about it from X1 and X2 in exactly the same fashion as for "time" loop 1. The procedure is completely analogous for loops 5 and 7. Once "time" loops 1, 3, 5, and 7 are completed, we need to recalculate C only, and the entire process is repeated for even loops. Notice that the only floating point operations in these updates are random number comparisons, required for the Metropolis accept/reject test.

Four adjacent words contain eight "space" loops, which can be even more easily vectorized. They can be updated without alternating even and odd ones, since they are decoupled. Because the detailed explanation of the procedure does not introduce anything substantially new with respect to the "time" flip, it will be left in the form of comments in the source code.

The global move in the time direction is very easy to implement with this type of spin packing. One has to check whether bits are all either 0's or 1's, then to XOR the word to be flipped with four neighboring words to get the transition probability. The same principles are used to implement the global flip in spatial directions, but the actual procedure is much more complicated. One has to worry whether the flip is in the x or y direction. Furthermore, to check whether the flip is allowed or not, one must scan L spin words, which should be contrasted to 2 in the case of "time" flip or 4 in the case of "space" flip. Determination of transition probability for the flip in the y-direction requires knowledge about spins residing in different processors, due to the nature of data decomposition (see below). Thus, unlike other updates, this cannot be done completely in parallel, with communication required for boundary spins only. The way we deal with it is to gather the data spaces residing in different processors into a complete copy in each

of the processors. Then the update is done in node 0 only, and results are broadcasted to other nodes. This is inefficient, but we are not particularly worried about it. The reason is that even when we use this update, it is done infrequently, thus diminishing the true overhead due to gather/scatter procedure. More importantly, for most of the runs we disposed of this update completely, because the results are unaffected by its inclusion for large systems.

It is desirable to have the simplest possible spin interaction in order to minimize the complexity of the various tests needed to determine the transition probability. For this reason, we believe that a "bond-type" decomposition is preferable due to the simplicity of spin interactions, although the spin packing could be done with any other decomposition, such as "cell-type" breakup, which leads to more complicated 8-spin interactions [2].

4.2. Parallel Implementation

The code was adapted for a parallel supercomputer, the 32-node Caltech/JPL MarkIIIfp Hypercube [3,4,5] at Caltech. MarkIIIfp is an assembly of rather sophisticated processors (nodes) which are connected as a 5-dimensional hypercube and are managed by a separate computer (host). The node board consists of 3 processors: the I/O processor managing the communications between nodes, the data processor (both are MC68020 chips) and the WEITEK processor to enhance floating point performance. In addition to local fast memories, the 3 processors can access 4 megabytes of DRAM [3]. Without spin packing, this memory cannot accomodate the complete spin system for most of the lattice sizes we simulated. The node system is managed by a Counterpoint 19K computer via a custom built interface. The actual computation is performed in the WEITEK chip, which is in a master-slave relation with the data processor. In our application, the data

processor is used only to initiate a call to the WEITEK chip, which has superior numerical capabilities. The programming model consists of two interlocked programs: the host program which serves the requests from the node assembly and the node program which is the actual application program [4,5]. The node program is downloaded to the nodes by the host.

The hypercube topology has a rather rich, but practically feasible, interconnection structure [5]. Nodes reside on vertices of a D dimensional hypercube and communicate by passing messages to its nearest neighbors. Thus each node has D communication channels. In each of the hypercube dimensions, a node can be assigned either coordinate 0 or coordinate 1. Hence, a binary number with D digits is enough to uniquely specify the location of the node in the assembly. A very appealing property of hypercubic topology is that it can be mapped onto Cartesian grids of lower dimensionality. A two-dimensional Cartesian grid is the natural data space topology for our simulation. The communication between nodes is handled by the Crystalline Operating System (CrOS) [5]. It is a very efficient environment for regular applications with short-range interactions. The communactions are very rigid in the sense that only nearest neighbors communicate and both message-sending and message-receiving nodes *must* expect the communication. The same applies to the host-node communication. This expected communication introduces an implicit synchronization in the working of the processor assembly. This is known under the name of loose synchronicity [5]. Since a processor blocks if its communication call is not complemented by another communication call on the other side of the channel, it is essential that the amount of computation between scheduled communications be evenly distributed among the nodes. This is an extremely sensitive issue in a variety of inherently irregular problems, but is, fortunately, irrelevant for our simulation. The homogeneity of the spin system assures that, when pieces of the system are divided among nodes in a geometrically uniform fashion, the computational load will also be even. This holds for the majority of

equilibrium Monte Carlo calculations, where one Monte Carlo sweep essentially defines a global internal clock.

It is possible to design a more general and flexible message-passing environment, but it is going to be far less efficient than CrOS. All the interprocessor communications in our simulation are handled by the CrOS calls. This implies that the mapping onto a Cartesian grid should preserve the nearest neighbor connections existing in the hypercubic topology. It is easy to see that nearest neighbor's coordinates differ by a single bit. Therefore, the nearest neighbors on the Cartesian grid should also differ by a single bit. This is easily achieved using a Grey code [5]. There is a set of utilities that performs this mapping transparently to the user:

$$gridinit(gdim, num)$$

$$gridcoord(proc, coord)$$

$$gridchan(proc, dir, sign)$$

Gridinit is used to initialize the Cartesian grid of dimension $gdim$. Num is an array of size $gdim$, and each element in the array gives the number of nodes assigned to the corresponding dimension of the grid. The number of nodes in each dimension should be a power of 2 in order to achieve periodic boundary conditions automatically. The routine *gridcoord* returns an array *coord* of Cartesian coordinates of a processor number *proc*. A processor selects one of its neighbors for communication by specifying the corresponding channel mask to the channel hardware. Processor number *proc*, residing in a Cartesian grid, will communicate with a processor in positive ($sign = 1$) or negative ($sign = -1$) direction *dir*, through the channel whose mask is returned after a call to the function *gridchan*.

The nearest neighbor interaction in the spin system allowed for an efficient parallelization. The hypercube topology of the processor node system is first mapped onto a

2D grid. Then, the processor nodes were configured as disconnected rings, by selecting a subset of channel masks. Each processor uses only 2 channel masks to communicate with processor to the left and the one to the right in its ring. The rest of the channels are nonexistent for the purpose of simulation. Of course, the system communication routines employ all channels. Each ring corresponds to an independent system, so we achieve trivial parallelism just by configuring the node assembly in this way (see Fig. 7a).

Each processor is assigned a piece of the 3D lattice, consisting of a number of (x,τ)-planes, as shown in Fig. 7b. When projected onto the original 2D lattice of the quantum system, we divide it into stripes running in the y-direction. The communication between processors is required only if the boundary spins are updated and involves only nearest neighbor processors as is required by CrOS. Another data decomposition that seems natural in this context is to break the latice into square patches instead of our decomposition into stripes. This would be a truly 2D decomposition, while ours is 1D or ring decomposition. Theoretically, 2D decomposition has the advantage that it is extensive, *i.e.*, if we want to increase the system size, we just increase the number of processors, while keeping the size of the patch assigned to individual processors constant. Thus, the computation time is independent of system size since the load per processor is constant and communication is strictly local. This actually does not hold in practice, since the number of nodes is limited to 32. In our application, the width of the stripe in the x-direction is fixed, but the length in the y-direction is the same as the system size. Thus the computation time scales linearly with system size, since we usually fix the width of the stripe to 4 or 6 layers ((x,τ) planes). Although this seems to favor the 2D decomposition, one should keep in mind that the efficiency of implementation strongly depends on the amount of communication, which is a bottleneck, particularly in an architecture where the node is

very fast, and it takes 20 times longer for the channel hardware to transmit a word than for the CPU to perform an operation. The efficiency of a program is defined as [5]

$$\epsilon = \frac{T_{seq}}{NT_{conc}(N)} = (1 + f_C)^{-1} \tag{4.1}$$

where T_{seq} is the time required to perform the computation on a sequential computer, while $T_{conc}(N)$ is the time required for the same task on a concurrent processor consisting of N nodes. The quantity f_C is called the overhead. In our case the communication overhead is the major contributor. It scales with the data volume n as $n^{1/d}$, where d is the topological dimension of the application. Apparently, the higher the dimensionality of a decomposition the more communication is required, since the previous formula is just surface-to-volume ratio. Communication itself is slow, but also the software overhead to setup a channel transfer is significant. Experimentally, to send a message consisting of N computer words, the required amount of time is [6]

$$T_{comm}(N) = (200 + 2 \cdot N)\mu s$$

Obviously, one should attempt to communicate rarely, and whenever a message is passed, it should be as long as possible if the communication time is to scale linearly with the length of the message. Both of these criteria are met with our ring decomposition. There is a significant amount of computation within a stripe, before a boundary is reached and communication is necessitated. As seen in Fig. 7c, the communication occurs only at two boundaries of a stripe, while it would be necessary to communicate on four boundaries of a square in the 2D decomposition. Also, a communication involves $LM/8$ computer words at a time (M=Trotter number, L= system size). In the 2D decomposition it would involve $bM/8$ words, where b is the size of the square patch assigned to the lattice. Experimentally, the efficiency of the application is very high (around 90%) [7].

There are additional considerations against the 2D decomposition. A close examination of the details of communication reveals that the minimum lengths of the messages are different for the x and y directions. Also, the *order* of communications in x and y directions is important, and the order of updates within a node matters, since the updates of the boundary spins of a node sometimes require knowledge about spins residing in *next* nearest neighbors. Since such a communication cannot be handled directly in CrOS, one must carefully schedule the order of message exchanges along the x and y directions in the grid. Finally, the input and output buffers have to be handled differently for communications along the x and y directions. All of these problems which complicate programming and debugging are absent with the ring decomposition.

Sometimes, due to slow dynamics, systems tend to get stuck in nonequilibrium regions of phase space. A good way to ensure that this does not happen is to have another run with a completely different random number sequence. It is considered better to have a few shorter runs than a single long one [8]. We always decompose a system among processors so that we have at least 4 independent rings. This serves to guard against metastability as well as against possible hardware errors. We have not encountered hardware errors, and we detected metastability only at $T = 0.25J$, for system size 160x160. At this point we discontinued running.

The communication is implemented via one of the library routines:

$$cshift(inbuf, inmask, insize, outbuf, outmask, outsiz)$$

Insize bytes are written from a *single* input channel specified by *inmask* into the buffer pointed to by *inbuf*. Also, *outsiz* bytes from a buffer pointed to by *outbuf* are written into output channels specified by *outmask*.

Memory management is rather simple. Each node operates on its own piece of the lattice, but the lattice coordinates are shifted according to the position of the node in a ring. This is depicted in Fig. 7d. Hence, the spin words are stored in an array so that the spin words corresponding to the processors own subspace occupy labels from 1 to $bLM/8$. Each word carries 32 spins stacked atop each other along imaginary time. Since the decomposition breaks the Hamiltonian into 4 pieces, the number of full spin words stacked on top of each original 2D lattice site is $M/8$, where M is the Trotter number. There are L lattice sites in the y-direction, and b denotes the number of (x, τ) planes alotted to a processor. Next $LM/8$ array elements are used to store the boundary layer of spin words from the processor in the positive x direction along the ring, and additional $LM/8$ elements are used to store the boundary layer from the neighbor in the negative x direction on the ring. These are the array elements pointed to by $inbuf$ in a communication call. These boundary layers are all that a processor needs to know about its neighbors to perform local updates and global updates in the time direction, and for the measurements of thermodynamic quantities.

The situation is different for winding number updates and measurements of correlation function, since these involve global operations on the lattice. This is handled in a pipelined manner by a pair of *gather/scatter* routines written in CrOS calls with syntax:

$$gather(buf, dim, size)$$

$$scatter(buf, dim, size)$$

Gather will pick up *size* bytes at the location pointed to by buf, and transfer them along direction 0, if $dim = 1$, so that after $nproc[dir]$ steps($nproc[dir]$ is the number of processors along dir) a copy of the complete spin system will reside in each node. If $dim = 2$, this routine may also accomodate the 2D decomposition. It will first gather the square patches

of the lattice from all processors with the same coordinate in $dir = 0$. This exchange is along dir 1 and is similar to the one described above, except that one must use the library routine $vshift$ instead of $cshift$, since the data to be transferred are not adjacent in address space. After that, each processor has a stripe extending in dir 1. Hence, the situation is now analogous to the ring decomposition, and one just repeats the steps appropriate to $dim = 1$. This exchange goes along dir 1 and uses $cshift$. The coordinates of each node's subspace always begin with 0.

After the global exchanges, each node computes *all* correlation functions, but for *its own* spins. So the measurement is done in parallel. After that, the partial results from the nodes are combined into a global result using the library routine

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

This function uses a binary tree algorithm to combine *nitems* of size *size* bytes stored at the location pointed to by buf, according to a commutative and associative function pointed to by $func$. The final result is placed in buf and is achieved in $logN$ steps, where N is the number of nodes. After the global measurement *scatter* is not used, since nothing was changed in the spin configuration. *Scatter* is used only after the winding number update is performed on the node with ring coordinate 0. Then, the updated spin configuration is scattered to all the nodes from the source node, which is assumed to be node 0. The overhead associated with these global communication routines is not significant, since the measurements are relatively rarely done, typically after 5 to 10 sweeps of each of the types of moves in turn.

The measurement of correlation functions is slightly different in the program that calculates dynamical correlations. Since we ultimately want to calculate Fourier transformed functions for a certain number of wave vectors, it is better to measure a subset of

correlation functions, different for each processor, but on the *whole* spin system residing in each processor's memory after a call to *gather*. The results are not combined until the very end of a run. Then each processor calculates its own contribution to the Fourier transform, with its own subset of spatial correlations, and then the results are combined. Note that we can separate the measurements from the communications, since we can place the complete data domain in a single processor. Otherwise, the measurements would be interspersed with communication calls which would bring data subsets from different nodes in the ring one at a time (procedure call *rotate* in Chapter 9 of Ref. [5]). That probably slightly increases the communication overhead.

In the simulation we used a parallel version of the generalized Fibonacci additive random numbers generator [5,6] which has a period longer than 2^{127}. The computations of static properties of Heisenberg and XY models took about 2000 hours of CPU time on a 32-node hypercube (roughly equivalent to about 2400 hours of Cray-XMP after the code is further vectorized).

References

[1] M. S. Makivic and H.-Q. Ding, submitted to Phys. Rev. B.

[2] E. Loh, D. J. Scalapino and P. M. Grant, Phys. Rev. **B31**, 4712 (1985).

[3] J. Tuazon, J. Peterson, and M. Pniel, in *The Third Conference on Concurrent Computers and Applications*, Volume **I**, ed. by G. Fox, ACM Press, New York (1988), p. 71.

[4] P. Burns, J. Crichton, D. Curkendall, B. Eng, C. Goodhart, R. Lee, R. Livingston, J. Peterson, M. Pniel, J. Tuazon, and B. Zimmerman, in *The Third Conference on Concurrent Computers and Applications*, Volume **I**, ed. by G. Fox, ACM Press, New York (1988), p. 872.

[5] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors*, Prentice Hall, Englewood Cliffs, New Jersey (1988).

[6] H.-Q. Ding, Caltech Report C3P-799 (August 1989).

[7] H.-Q. Ding and M.S. Makivic, presented in the 5th Distributed Memory Computing Conference, Charleston, S. Carolina, April 1990, (Caltech report C3P-845).

[8] K. Binder, in *Applications of the Monte Carlo Method in Statistical Physics*, ed. K. Binder, Springer-Verlag, Berlin (1987).

Chapter 5

Simulation and Measurements

5.1. Simulation

We tested the algorithm on a 4x4 system, where the exact results are available [1]. It is necessary to include *all* types of updates to obtain correct results, since the system is far away from the thermodynamic limit. However, in Table 1, we show the results with and without the spatial global moves for a much larger system (32x32). We also plot the correlation functions for the 2 cases (Fig. 8). It is apparent that this system size is large enough to eliminate any systematic differences.

The equivalence between the quantum system and the classical one is exact only in the limit of infinite M. In practice one works with finite values of M (or, equivalently, $\Delta\tau \neq 0$), which is a source of systematic error. The error introduced is small and well controlled. As discussed in Chapter 3, it is of order $(\Delta\tau)^2$, and it is volume-independent for sufficiently large systems, being proportional to the norm of commutators $[H_i, H_j]$ [2]. For a wide class of observables, one may use the extrapolation:

$$Q(M) = Q(\infty) + a/(MT)^2 + b/(MT)^4 + ... \tag{5.1}$$

where $Q(\infty)$ is the correct value. Commonly, only the leading order term is retained, so one extrapolates linearly in $(1/mT)^2$. If one works with small values of M, or if one suspects that the coefficients a or b exhibit a singular dependence on temperature, which might happen if $T = 0$ is the critical temperature, a more careful procedure might be in

order. One should attempt to fit the results with polynomials in $1/(MT)$ of increasing order, until the fits stabilize. A different approach, the one we adopted, is to fix the value of $(1/MT)^2 = (\Delta\tau)^2$ to a very small number at every temperature, by choosing appropriate values of M [3]. The systematic error is then the same at every temperature. Our choice of $\Delta\tau \leq 0.07$ is small enough to ensure that the systematic error is within the statistical errors of the simulation. To prove this, at $T = 0.45$, we used 3 different values of the Trotter number: $M = 16$, $M = 24$, and $M = 32$. The results show a very weak M-dependence and the differences between the extrapolated values and those obtained with the largest value of $M = 32$ are within the statistical errors. Since we have only 3 values of M, it is difficult to show quantitatively that the data scale as M^{-2}, but we assume that it is true. In Table 1 we show the calculated values of energy, susceptibility and specific heat for $M = 16$ and $M = 32$. Additional checks are given below.

Simulations of the Heisenberg model were done on (2+1)-dimensional systems as large as 128x128x192 spins, in the temperature range from $T = 2.5J$ to $T = 0.25J$. For the XY model, the largest system size was 96x96, and the temperatures range from $T = 0.1J$ to $T = 1.0J$. During the simulations, we have monitored the thermal relaxation and the autocorrelation lengths. At each T we did several sufficiently long runs. For example, at $T = 0.3$, on the 96x96 lattice, we did 4x350000 sweeps (4 independent runs, 350000 sweeps each). The thermalization took about 5000 sweeps and the autocorrelation length is about 4000 sweeps. (This all refers to global observables such as the spin correlation functions; for local observables such as energy, both relaxation and autocorrelations times are much shorter.) At higher temperatures and for smaller systems, the runs are slightly shorter.

A note is in order here about the singular coupling problem [4]. At high temperatures, we use $M = 16$, leading to $MT \simeq 30$. Such a large value of MT implies rather high energies $E(5)$ and $E(6)$ (Eq. (3.23)). The author of Ref. [4] speculates that such a strong coupling

may lead to a breakdown of the importance sampling. One of the consequences may be the "premature convergence" problem, where the observables, upon increasing M exceed the correct value, and become increasingly inaccurate as M is further increased. The author [4] attributed that to the singular coupling in the classical problem, but it seems that it is just a problem of poor statistics in his simulation. Although we are working in the strong coupling regime, we do not encounter the "premature convergence" problem.

The global move which changes the magnetization significantly improves the relaxation rate, by about a factor of 3, and its acceptance is quite reasonable (10-50%, depending on T), as discussed before, contrary to many reports in the literature. In all the simulations, we use this global move. On the other hand, we find that the global move which changes the winding numbers is not efficient and does not improve significantly the relaxation time, so we dropped it in our simulations. It follows from standard thermodynamic arguments that both the global moves may be dropped, since the averages will not be affected in the infinite volume limit. There is, however, a case when one cannot dispose of the winding number update without penalty. This will be discussed below. It should be pointed out that finite size effects may be affected by inclusion or exclusion of global moves. We do not have to worry about that, since we are working on rather large systems.

5.2. Methods of Measurement

There are, in general, two types of quantities that can be easily measured with a path integral Monte Carlo method: those that can be obtained as the derivatives of the partition function, and those which are diagonal in the chosen representation. We calculate energy, specific heat and uniform susceptibility, as the derivatives of the partition function. They are given by the following averages [5]:

$$E = \left\langle \sum_p F(j_p) \right\rangle \tag{5.2}$$

$$CT^2 = \left\langle \sum_p (F(j_p)^2 - G(j_p)) \right\rangle - \left\langle \sum_p F(j_p) \right\rangle^2 \tag{5.3}$$

$$\chi T = \left\langle (\sum_p N(j_p))^2 \right\rangle \tag{5.4}$$

The summation goes over all the interacting 4-spin plaquettes, labeled by p, and j_p is a label of a particular spin configuration on a plaquette. The functions $F(j_p)$, $G(j_p)$ and $N(j_p)$ represent contributions of individual plaquettes to the total average. Note that the quantity that is measured in Eq. (5.2) is not the same energy that appears in the exponent, Eqs. (3.20, 3.21). This peculiarity is a consequence of the Suzuki-Trotter decomposition, where the energy levels of the equivalent classical system depend on temperature and its size in the imaginary time direction. For the same reason, the specific heat has an additional contribution coming from the function $G(j_p)$. Only the uniform susceptibility does not have the anomalous terms, since it is diagonal in the representation. The functions F, G and N for the allowed 4-spin configurations in the S_z representation are:

$$\begin{aligned} F(j) &= \frac{\partial(\beta E(j))}{\partial\beta} \\ F(1) &= F(2) = K/\beta \\ F(3) &= F(4) = \frac{1}{\beta}(-K - \ln(\cosh(2K))) \\ F(5) &= F(6) = \frac{1}{\beta}(-K - \ln(\sinh(2K))) \end{aligned} \tag{5.5}$$

$$\begin{aligned} G(j) &= \frac{\partial F(j)}{\partial\beta} \\ G(1) &= G(2) = 0 \\ G(3) &= G(4) = \left[\frac{2K}{\beta}\,\mathrm{sech}(2K)\right] \\ G(5) &= G(6) = \left[\frac{2K}{\beta}\,\mathrm{csch}(2K)\right] \end{aligned} \tag{5.6}$$

$$\begin{aligned} N(3) &= N(4) = N(5) = N(6) = 0 \\ N(1) &= \frac{1}{4M} \\ N(2) &= -\frac{1}{4M} \end{aligned} \tag{5.7}$$

More detailed information about the system may be acquired by studying spin correlations. We first measured the static staggered spin correlation function

$$C(\mathbf{r}) = (-1)^{r_x+r_y}\frac{4}{L^2}\sum_{\mathbf{n}}\langle S^z_{\mathbf{n}} S^z_{\mathbf{n}+\mathbf{r}}\rangle \tag{5.8}$$

along the x and y directions (a factor of 4 appears in the definition due to the normalization $C(0) = 1$). This quantity still provides a somewhat limited picture, but it takes much less computer time to measure than the full set of spin correlations. Nevertheless, it allows us to infer the correlation length, which can be compared with experiments. This correlation function is a diagonal operator, hence one can choose any time slice for the measurement, make a mask which has a single 1-bit at the location of that time slice, and just XOR the two spin words at positions (n_x, n_y) and $(n_x + r_x, n_y + r_y)$. We used a slightly more complicated procedure. Due to the decomposition, the translational invariance of the Hamiltonian is broken and we want to partially restore it during the measurement. This effect, of course, vanishes in the limit of infinite Trotter number. Since the Hamiltonian is decomposed into 4 pieces, the spin interactions are periodic in time with period 4, so we calculate the correlation function at 4 adjacent time slices, corresponding to the lowest 4 bits in a spin word. All the Trotter numbers in the simulation are larger than or equal to 16, thus we have at least 2 words stacked on top of each 2D lattice site. To improve statistics, we do such a measurement for each word and then average them. Therefore, we do the measurement on a fraction (1/8) of the total number of time slices. The correlations are calculated separately along the x and y axes, and then averaged. We check whether they agree within error bars, to insure that the result is indeed isotropic and as an additional test of thermalization.

Furthermore, we check whether the finite value of the Trotter number will lead to significant systematic effects. At $T = 0.45$, we calculated the correlation function using 3 different values of M, $M = 16, 24, 32$. In Fig. 9 we show these correlation

functions. The correlation function has very weak M-dependence for large values of M. The apparent nonmonotonic behavior in Fig. 9 is due to the statistical errors combined with a weak Trotter number dependence. We calculated the extrapolated correlation function, from these 3 Trotter numbers using the extrapolation formula of Eq. (5.1). The extrapolated correlation function and the one with the largest value of $M = 32$, are practically indistinguishable. At $T = 0.35$ and $T = 0.3$ we calculated correlation functions with two different values of M, $M = 24$ ($\Delta\tau \simeq 0.14$) and $M = 48$ ($\Delta\tau \simeq 0.07$). In Fig. 10 we show the correlation functions at $T = 0.35$. Although the width of the time slice is halved, the correlation function differences are very small and within the error bars. The results at $T = 0.3$ are quite similar. The parameters of the fits to these data are given in Table 1. The correlation lengths are practically the same. These results clearly show that the values of $\Delta\tau = 1/MT$ we are using are small enough to ensure that the systematic errors are buried within the statistical errors of the simulation, and that the extrapolation to $M = \infty$ is unnecessary.

We also calculate the staggered susceptibility for the antiferromagnet,

$$\chi_{st} = \frac{1}{L^2}\langle(\sum_{\mathbf{r}}(-1)^{r_x+r_y}S^z_{\mathbf{r}})^2\rangle \tag{5.9}$$

following the same approach as outlined for the correlation function. Note that we define the staggered susceptibility to be simply equal to the antiferromagnetic structure factor $C(\pi,\pi)$. The reason is that it is more suitable for a scaling analysis than the thermodynamic susceptibility. They are different in the quantum model because the staggered magnetization is not a conserved quantity.

The calculation of the thermodynamic staggered susceptibility, which is the true response function, requires knowledge of the dynamic structure factor. The dynamic structure factor contains much more information about the system, particularly about the

excitations. It is directly measured in inelastic neutron scattering experiments. It requires the amount of measurement that scales with the volume of the system, unlike the function of Eq. (5.8) which scales with the linear size of the system.

The dynamic structure factor $S(\mathbf{q},\omega)$ is the Fourier transform of the time-dependent correlation function:

$$S(\mathbf{q};\omega) = \sum_{\mathbf{r}} \int_{-\infty}^{\infty} e^{i(\mathbf{q}\cdot\mathbf{r}-\omega t)} S(\mathbf{r};t)dt \tag{5.10}$$

where

$$S(\mathbf{r};t) = \frac{1}{L^2} \sum_{\mathbf{r}_j} \langle S_z(\mathbf{r}_j+\mathbf{r};t) S_z(\mathbf{r}_j;0)\rangle \tag{5.11}$$

and

$$S_z(\mathbf{r};t) = e^{iHt} S_z(\mathbf{r}) e^{-iHt} \tag{5.12}$$

is the spin operator in the Heisenberg picture.

Real-time correlation functions are notoriously difficult to measure. The source of the difficulty is the oscillatory behavior of the integrand, which also plagues analytic calculations. The established way to go around it is to calculate dynamic correlations on the imaginary time axis (Wick's rotation in field theoretic language), and then to analytically continue to the real time.

The imaginary time correlation function is defined as:

$$S(\mathbf{r};\tau) = \frac{1}{L^2} \sum_{\mathbf{r}_j} \langle S_z(\mathbf{r}_j+\mathbf{r};\tau) S_z(\mathbf{r}_j;0)\rangle \tag{5.13}$$

where

$$S_z(\mathbf{r};\tau) = e^{H\tau} S_z(\mathbf{r}) e^{-H\tau} \tag{5.14}$$

After a Fourier transform in real space, one obtains the imaginary time correlation function in momentum representation:

$$S(\mathbf{q};\tau) = \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} S(\mathbf{r};\tau) \tag{5.15}$$

It is related to the structure factor via a Laplace transform:

$$S(\mathbf{q};\tau) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-\omega\tau} S(\mathbf{q};\omega)d\omega \tag{5.16}$$

The imaginary time correlation function $S(\mathbf{q};\tau)$ is very easily measured in our simulation, since it is diagonal in S_z. The only difference with respect to the measurement of the static correlation function is that for $\tau \neq 0$ one has first to shift the corresponding spin word for τ time slices to pick up the value of $S_z(\mathbf{r}_j + \mathbf{r};\tau)$. Also, the method of distributing the calculation among the processors is slightly different, as explained in the previous chapter. Most interesting $\mathbf{q}$ values are around the incipient antiferromagnetic order wave vector $\mathbf{q} = (\pi,\pi)$, in units of inverse lattice spacing a^{-1}, and around the zone center, $\mathbf{q} = (0,0)$. Because of that, we calculate the Fourier transforms for the allowed discrete values of wave vector along the triangle in the Brillouin zone defined by $(0,0) \Rightarrow (\pi,\pi) \Rightarrow (\pi,0) \Rightarrow (0,0)$.

To calculate the frequency-dependent structure factor $S(\mathbf{q};\omega)$ one has to invert the Laplace transform of Eq. (5.16). This operation is ill defined for noisy data in the time domain, and is a major obstacle for calculating dynamical properties of quantum systems reliably. Recently, significant progress has been achieved in this area with maximum entropy methods used for image reconstruction [6]. We are currently using that method to analytically continue our data to the real frequency domain [7]. That work is not yet complete, and will not be presented here. There is, however, another possibility, if one knows what type of real frequency behavior is to be expected from other calculations. One can formulate a guess for the structure factor in real frequency domain with a set of variational parameters, $S_g(\mathbf{q};\omega;\alpha)$, then perform the Laplace transform (5.16) to the imaginary times, and then determine the parameters by minimizing appropriate cost function. Most frequently we perform a χ^2 fit, *i.e.*, the variational parmeters are determined by minimizing:

$$L = \sum_{\tau} \frac{1}{(\sigma_\tau)^2}(S_d(\mathbf{q};\tau) - \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-\omega\tau} S_g(\mathbf{q};\omega;\alpha)d\omega)^2 \tag{5.17}$$

The data are given by S_d, the Laplace transform of the the variational guess with parameters α is represented by S_g, and σ_τ represents the statistical noise in the data. The preliminary results using this approach will be given in the following chapter.

For the XY model we calculated additional quantities. The topological order in the model may be quantified by vortex density. An intuitive picture of vortices in the quantum model is not without problems, but it is possible to construct operators that have the correct eigenvalues in classical vortex configurations. One definition of the vortex density operator is due to Betts [8]:

$$V = \frac{1}{4}(\sigma_1^x\sigma_2^y - \sigma_2^y\sigma_3^x + \sigma_3^x\sigma_4^y - \sigma_4^y\sigma_1^x) \tag{5.18}$$

The labeling of spins used to define vorticity is shown in Fig. 38. Time reversal symmetry implies that $< V >= 0$, *i.e.*, the densities of vortices and antivortices are equal. The total density of vortices and antivortices

$$< V^2 >= \frac{1}{4}(1 - 2 < \sigma_1^x\sigma_3^x > + < \sigma_1^x\sigma_2^y\sigma_3^x\sigma_4^y >) \tag{5.19}$$

is in general nonzero. The averaging is performed over all elementary squares on the lattice.

It is not straightforward to measure this quantity, since the operators cannot be simultaneously diagonalized by a single basis for the whole lattice. An attractive possibility is to chose an S_x basis on sublattice A and an S_y basis on sublattice B. It is easy to see that the vortex density operator is diagonal in this basis, but a more serious problem appears. The introduction of a phase shift between sublattices leads to imaginary matrix elements for the local Boltzman factors. This eliminates the basis change as a possible solution.

In general, this problem is solved in the following manner [9]. We start from the Suzuki-Trotter approximant for the expectation value:

$$< Q >= Z^{-1}\mathrm{Tr}Qe^{-H/T} = Z^{-1}\mathrm{Tr}Q(e^{-\Delta\tau H})^M = Z^{-1}\mathrm{Tr}\lim_{M\to\infty} Q(\prod_{i=1}^{4} e^{-\Delta\tau H_i})^M \quad (5.20)$$

We must now insert $4M+1$ resolutions of unity between adjacent operators, to obtain:

$$< Q >= Z^{-1}\lim_{\Delta\tau\to 0}\sum_{\{C\}}\langle C_0|Q|C_1\rangle\langle C_1|e^{-\Delta\tau H_1}|C_2\rangle\langle C_2|e^{-\Delta\tau H_2}|C_3\rangle\cdots\langle C_{4M}|e^{-\Delta\tau H_4}|C_0\rangle \quad (5.21)$$

For the sake of brevity, let us denote:

$$U_i = e^{-\Delta\tau H_i} \quad (5.22)$$

Then the previous equation can be rewritten as:

$$< Q >= \lim_{\Delta\tau\to 0}\frac{\sum_{\{C\}}\langle C_0|Q|C_1\rangle\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_0\rangle}{\sum_{\{C\}}\langle C_0|C_1\rangle\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_0\rangle} \quad (5.23)$$

or equivalently

$$< Q >= \lim_{\Delta\tau\to 0}\frac{\mathrm{Tr}\langle C_0|Q|C_1\rangle\rho_M(C_0,C_1)}{\mathrm{Tr}\langle C_0|C_1\rangle\rho_M(C_0,C_1)} \quad (5.24)$$

where the new density matrix $\rho_M(C_0,C_1)$ is:

$$\rho_M(C_0,C_1) = \frac{\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_0\rangle}{\sum_{\{C\}\neq C_0,C_1}\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_0\rangle} \quad (5.25)$$

This approach would work for any operator, but it has very serious disadvantages. It breaks the homogeneity in time of the equivalent classical system by introducing an extra time slice for the measurement. In a worldline picture, this also implies that one has to break them, since there is no propagator to connect time slices C_0 and C_1. Furthermore, the measurement can be done at a single time slice only. If the worldlines are not broken, we can do the measurement at any time slice. This significantly improves the statistics of the measurement. Finally, spin packing and vectorization cannot be done efficiently, because the additional time slice has to be treated differently from the rest of the system.

Fortunately, there is a class of operators that is not diagonal but still can be measured without breaking the worldlines. These operators have the same selection rules as the propagators U_i or their products, U_iU_j. In the case of the vortex density operator, it is easy to see that products U_1U_2 and U_3U_4 decompose the lattice into disjoint square cells, as shown in Fig. 1c. The measurement of vortex density is performed on these elementary squares. The expectation value of the vortex density operator can be written as:

$$< V^2 >= \lim_{\Delta\tau\to 0} \frac{\sum_{\{C\}}\langle C_1|V^2U_1U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_1\rangle}{\sum_{\{C\}}\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_0\rangle} \tag{5.26}$$

One can insert identity into the numerator, to obtain

$$< V^2 >= \lim_{\Delta\tau\to 0} \frac{\sum_{\{C\}} \frac{\langle C_1|V^2U_1U_2|C_3\rangle}{\langle C_1|U_1U_2|C_3\rangle}\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_1\rangle}{\sum_{\{C\}}\langle C_1|U_1|C_2\rangle\langle C_2|U_2|C_3\rangle\cdots\langle C_{4M}|U_4|C_1\rangle} \tag{5.27}$$

This formula implies that one has to measure the expectation value of the quantity

$$\frac{\langle C_\tau|V^2U_1U_2|C_{\tau+2\Delta\tau}\rangle}{\langle C_\tau|U_1U_2|C_{\tau+2\Delta\tau}\rangle} \tag{5.28}$$

between any pair of time slices which are $2\Delta\tau$ apart, but with the respect to the *original* density operator with unperturbed worldlines. Of course, this works only if the ratio of matrix elements in (5.28) exists. In general, it may happen that an operator QU_1U_2 will connect states which are not connected by the propagators U_1U_2. Fortunately, $V^2U_1U_2$ has the same selection rules as U_1U_2. In addition to that, if this formula is to be practical, then the required matrix elements should be easy to calculate. Because of that, it is crucial that U_1U_2 and U_3U_4 decouple the lattice. This trick wouldn't work with any other bond decomposition. Other choices of nonoverlapping bond subsets do not decouple the square lattice. This is an additional reason to choose this particular decomposition. In addition to the simplicity of spin interactions when compared to the square cell decomposition, which is true for any bond decomposition, *all* operators that can be easily measured with the cell decomposition can also be measured here.

The anticipated Kosterlitz-Thouless transition is characterized by a universal jump in the spin stiffness at T_c. We measure the spin stiffness ρ_s using the relation between ρ_s and winding number fluctuations [10]. In 2D, this relation reads:

$$\rho_s = \frac{1}{2} k_B T \langle W^2 \rangle \tag{5.29}$$

The winding number, for a system of linear size L, is defined as:

$$\sum_{i=1}^{N} (\mathbf{r}_{P_i} - \mathbf{r}_i) = L\mathbf{W} \tag{5.30}$$

where $\mathbf{r}_i$ is the position of a worldline at $\tau = 0$, $\mathbf{r}_{P_i}$ is its position at $\tau = \beta$, and the summation is performed over all worldlines. The subscript P_i denotes that the propagation from 0 to β, if the worldline does not close on itself, entails a permutation of particles, since it must connect to another worldline. Our algorithm accomplishes this reconnection of the worldlines via the "space" loop flip.

Physically, the winding number measures the coherent motion of a collection of worldlines as the system of particles evolves from 0 to β. It describes the average shift of the center of mass of the worldlines. This coherent motion, if it persists in the thermodynamic limit, leads to superfluidity or finite spin stiffness. If the system has periodic boundary conditions, then this shift in the center of mass translates into a topological constraint, implying that the particles have wound around the torus.

The fact that particle number and phase are cannonicaly conjugate operators, translates into the following formula [10]:

$$\frac{Z(\theta)}{Z(0)} = \langle e^{i\theta W} \rangle \tag{5.31}$$

where $Z(\theta)$ is the partition function of the system with a twist θ introduced in the order parameter, and $Z(0)$ is the partition sum without the twist. But, according to Eq. (2.12),

in the limit $\theta \rightarrow 0$, we can also write

$$\frac{Z(\theta)}{Z(0)} = e^{-\theta^2 \beta \rho_s / 2} \tag{5.32}$$

In the thermodynamic limit, the probability distribution of W is Gaussian, which combined with Eqs. (5.31) and (5.32) leads to Eq. (5.29).

It is apparent that in a simulation with periodic boundary conditions, we must introduce global flips that will introduce winding number fluctuations. The problem is that such a global flip is extremely inefficient for large system sizes. In practice, we find that it is not possible to generate a sufficient number of these flips within a reasonable amount of computer time, for systems bigger than 8x8. This is certainly not enough to guarantee small finite size effects.

The solution is rather simple. As discussed before, if one restricts oneself to a particular winding number subspace, it may be regarded as a choice of the statistical ensemble. Following the same logic that derives the cannonical ensemble considering a subsystem within a system which is microcannonical, we can work with a system having a winding number 0, but we isolate a subsystem, which cannot have such a topological constraint. In other words, the large system with $W = 0$, is "microcannonical," but it is just the heat bath for its subsystem, which is "cannonical." The winding number for the subsystem is *not* a topological constraint, since it has open boundary conditions. It is interpreted as an ordinary shift in the center of mass of its worldlines as they propagate from 0 to β. Thus, running a large system without winding number flips, but measuring the coherent motion of the worldlines in a subsystem, we can still calculate spin stiffness efficiently. The subsystem size, of course, cannot be larger than 1/2 original system size. Running a single very large system, and measuring concurrently $\langle W^2 \rangle$, for various subsystem sizes, one can acquire enough information to extrapolate to infinite system size in a single run. This may compensate for taking measurements only on a portion of the large system.

References

[1] M. C. Cross, private communication.

[2] R. M. Fye, Phys. Rev. B **33**, 6271 (1986).

[3] E. Loh, D. J. Scalapino and P. M. Grant, Phys. Rev. **B31**, 4712 (1985).

[4] A. Wiesler, Phys. Lett. **89A**, 359 (1982).

[5] J. J. Cullen and D. P. Landau, Phys. Rev. B **27**, 297 (1983).

[6] M. Jarrel, J. Gubernatis, R. Silver, N. Sivia, in preparation.

[7] M. Makivic and M. Jarrel, in preparation.

[8] D. D. Betts, F. C. Salevsky, and J. Rogiers, J. Phys. A **14**, 531 (1981).

[9] J.E.Hirsch, D.J.Scalapino, R.L.Sugar, and R.Blankenbecler, Phys.Rev. B **26**, 5033 (1982).

[10] E. L Pollock and D. M. Ceperley, Phys.Rev. B **36**, **36**, 8343 (1987).

Chapter 6

Quantum Heisenberg Antiferromagnet in Two Dimensions

6.1. Thermodynamics

The energy as a function of temperature is given in Fig. 11. The size-dependence of the energy is negligible. The results are in good agreement with the calculations performed by other authors on much smaller lattices. The rotational invariance of the Heisenberg Hamiltonian should be preserved by the Monte Carlo simulation. We check this requirement by calculating the energy in two ways: using Eq. (5.2) and from the nearest neighbor spin correlation $\langle S_i^z S_j^z \rangle$ assuming full isotropy in spin space. This rotational invariance requirement is satisfied within error bars for all of our data points.

We compare the results of the simulation with two analytic calculations for two opposite limits. Takahashi developed a modified spin wave theory, which is expected to be valid in the low temperature regime, by constructing a variational spin wave density matrix [1]. The density matrix is that of a noninteracting spin wave system, as discussed in Chapter 2. The difference from a $T = 0$ spin wave theory is that the parameters of the Bogoliubov transformation are determined by minimizing the free energy, but with the constraint that $\langle S_{\mathbf{r}}^z \rangle = 0$. This implies that the Mermin-Wagner theorem is enforced by a global constraint. In the limit of $T \to 0$, the ordinary spin wave theory is recovered. At finite temperatures, spin waves become excitations with a gap. The theory does not lead to the damping of excitations. The spin wave energy is:

$$\epsilon_{\mathbf{q}} = \lambda(1 - \eta^2 \gamma_{\mathbf{q}}^2)^{1/2} \tag{6.1}$$

where λ and η are obtained from the self-consistency requirement:

$$S + \frac{1}{2} = \frac{2}{N} \sum_{\mathbf{q}} \frac{1}{2} \frac{1}{(1 - \eta^2 \gamma_{\mathbf{q}}^2)^{1/2}} \coth\{\frac{\lambda}{2T}(1 - \eta^2 \gamma_{\mathbf{q}}^2)^{1/2}\} \tag{6.2a}$$

and

$$\frac{\eta\lambda}{Jz} = \frac{2}{N} \sum_{\mathbf{q}} \frac{1}{2} \frac{\eta\gamma_{\mathbf{q}}^2}{(1 - \eta^2 \gamma_{\mathbf{q}}^2)^{1/2}} \coth\{\frac{\lambda}{2T}(1 - \eta^2 \gamma_{\mathbf{q}}^2)^{1/2}\} \tag{6.2b}$$

The agreement with our calculation is rather good for $T \leq 0.6J$. In the high temperature limit, accurate results can be obtained with high temperature expansion [2]. We plot our data along with Takahashi's calculation for a 64x64 lattice and high temperature expansion up to $x^{10}(x = J/k_B T)$. Our data smoothly interpolate between these two asymptotic regimes. At $T > J$, the agreement with the high temperature expansion is excellent.

The data for the uniform susceptibility are presented in Fig. 12. The susceptibility saturates around $T \simeq J$, and then connects smoothly to the high temperature form. Again, the agreement with high-T expansion is excellent for $T > J$. For $T \leq 0.6J$, we obtain reasonable agreement with Takahashi's result:

$$\chi = \frac{1}{3T} \cdot \frac{2}{N} \sum_{\mathbf{q}} (n_{\mathbf{q}}^2 + n_{\mathbf{q}}) \tag{6.3}$$

where $n_{\mathbf{q}}$ is the thermal occupation number for spin waves with wave vector $\mathbf{q}$. A major source of the small systematic difference is the incorrect temperature-dependence of the variational parameter that determines the spin wave gap (parameter η). Note that at $T = 0.35J$ and $0.40J$ the system size is exactly the same as in his calculation, 64x64 spins, so this cannot be attributed to a systematic size-dependence. As one goes to higher temperatures, spin wave theory fails. The magnetization fluctuations are overestimated, because the ideal density of states overemphasizes the short wavelength spin waves. This is not felt at lower temperatures, since temperature acts as a cutoff.

At temperatures higher than $T = 0.25J$, we attempt to work on system sizes large enough to practically eliminate the finite size effects. The finite size effects are expected to get worse at lower temperatures. At $T = 0.25J$, we deliberately simulate a 32x32 lattice, which is 16 times smaller than the lattice used at $T = 0.27J$ (128x128). The sharp drop in the susceptibility is the indicator of a significant finite size effect for the lattice of this size. This demonstrates that quantitatively reliable results for the zero temperature limit cannot be obtained by studying relatively small systems (up to 16x16), as was attempted in recent literature [19].

The specific heat data are shown in Fig. 13. It peaks around $T = 0.6J$. The agreement with the high temperature expansion, in the range $T \geq J$ is excellent. Specific heat is expected to behave as $\propto T^2$, from spin wave theory, in the low temperature regime, but we are not going low enough to be able to accurately extract the proportionality constant. Again, we point out that quantitatively accurate results cannot be obtained on small lattices for the $T = 0$ limit. We illustrate this again by plotting the data point for a 32x32 lattice at $T = 0.25J$, next to the 128x128 lattice at $T = 0.27J$. We should point out that our thermodynamic data agree well with other calculations [19], performed on smaller systems, for temperatures higher than $T = 0.35J$. The disagreement is noticeable at lower temperatures, particularly in the case of the specific heat. This implies that one should use large lattices, but it is very difficult to obtain accurate results on large lattices in the extreme low temperature limit due to thermalization problems.

6.2. Static Spin Correlations

In an infinite system, the correlation function behaves asymptotically as:

$$\lim_{r\to\infty} C_\infty(r) = Ar^{-\lambda}e^{-r/\xi} \tag{6.4}$$

This expression may be regarded as a definition of the correlation length ξ. In Ornstein-Zernicke theory, for a d-dimensional system, the exponent λ is equal to $(d-1)/2$. In order to incorporate periodic boundary conditions, for a system of linear size L, we use the following symmetrized form:

$$C_L(r) = A(r^{-\lambda}e^{-r/\xi} + (L-r)^{-\lambda}e^{-(L-r)/\xi}). \tag{6.5}$$

The correlation length ξ, the exponent λ and the amplitude A are treated as fitting parameters. We are not aware of any previous attempt to infer λ for the present model from Monte Carlo data.

We start on a 24x24 lattice at $T = 2.5$, where the correlation length is less than a lattice spacing. As the temperature is lowered, we increase the lattice size up to 128x128, to keep $L \geq 5\xi$. With a procedure like this we hope to keep finite size effects negligible. The correlation functions at few moderate temperatures are plotted in Fig. 14a, along with the best fits. The results of the best fits for ξ and λ are summarized in Table 1. In Figs. 14b, c, d and e, we plot correlation functions for the lower temperatures and larger lattices.

The shortest range correlations were not included in the fit, since they are not described by the asymptotic form of Eq. (6.4). For larger correlation lengths, usually 4 or 5 shortest range correlations were excluded, while for shorter correlation lengths we discard 1 or 2 points. Also a few points close to half the system size were discarded since their

error bars are too large. The criterion was to keep those points which have a signal/noise ratio of at least 5. After these points are discarded, the fits are excellent and are very stable for a rather wide span of both long-range and short-range cutoffs.

Since the measured values of the exponent λ are very close to $(d-1)/2 = 1/2$, we believe that the correlation functions are of a Ornstein-Zernicke (OZ) type, as expected on general grounds. This indicates that the puzzling result $\lambda = 1$ (as in 3D), predicted by Schwinger boson mean field theory [3] and by modified spin wave theory [1], may be an artifact of the approximations involved.

For temperatures less than $T \simeq 0.35$, λ drops slightly below 1/2. This is not unexpected, since the OZ behavior is found only asymptotically, for $r/\xi \gg 1$. This condition is no longer true for our data points, as T is lowered: ξ becomes large and $r_{max} \simeq 2\xi$. The surprisingly good fit to Eq. (6.5) shows that an effective value of λ can accommodate these not-so-distant points. This gradual decrease of effective λ is, of course, consistent with $\lambda = 0$ in an ordered ground state. In principle, with the data of significantly higher statistical quality, at finite temperatures we should be able to measure the effects of logarithmic corrections to OZ function in momentum space, where the expected scaling form is

$$S(q) = S(0)\frac{1 + \frac{1}{2}\ln(1 + (q\xi)^2)}{1 + (q\xi)^2} \tag{6.6}$$

Apparently, our fitting form with "running" exponent λ takes care of these corrections in an effective way. On the other hand, for extremely low temperatures with respect to lattice size L, when the staggered magnetization has finite value, we should be able to separate [4] the spin wave correction to scaling at $T = 0$ by a fit of the form

$$C(r) \propto (M^{\dagger} + \frac{A}{r})^2 \tag{6.7}$$

for distances $a \ll r \ll \xi_J$, where a is lattice spacing and ξ_J is Josephson's correlation length [5]. For larger distances, this should be smoothly matched onto OZ form. We attempted to do this, but the system sizes we are working with require much lower temperatures than those we used, to measure statistically significant staggered magnetization $M^\dagger$. This leads to very unstable fits to the form of Eq. (6.7).

In previous calculations [6,7], the correlation functions were described by pure exponentials. Our approach allows for a more consistent comparison with experiment, since the correlation lengths in neutron scattering experiments [8] were obtained by fitting the structure factor data to Lorentzians.

The correlation length as a function of inverse temperature is shown in Fig. 15. The data points fall onto a straight line throughout the whole temperature range. This naturally leads to the exponential fitting form :

$$\xi(T) = Ae^{2\pi\rho_s/T} \tag{6.8}$$

Similar behavior is found in the classical Heisenberg model in 2D [9] and is typical for systems at lower critical dimensionality. The fit is indeed very good (χ^2 per degree of freedom is 0.62). The parameters of the fit are listed below (see Eq. 6.9).

It was argued by Chakravarty, Halperin and Nelson (CHN) [5] that, in the parameter regime where the ground state is ordered, the hydrodynamic description is provided by the nonlinear σ model, which has the correct symmetry and excitation spectrum. A perturbative renormalization group calculation carried out on this model leads to a classical picture, but with the parameters renormalized by quantum fluctuations. Essentially the same result is obtained by Arovas and Auerbach [3] who study the model within the framework of a large N expansion, but the connection with the classical model is not so transparent. Takahashi's variational spin wave theory also turns out to be equivalent to

the 1-loop result [5]. However, we show that the 1-loop result is *not* quantitatively correct. Our calculation is in very good agreement with the renormalized classical picture. Besides verifying the qualitative picture, we provide accurate quantitative results.

In particular, we find that the long-wavelength spin stiffness ρ_s is significantly reduced due to quantum effects, which leads to a much slower growth of the correlation length when compared to the classical case. This renormalization of the spin stiffness for a quantum spin S is most conveniently expressed in terms of the renormalization factor $Z_\xi^{(S)}$, defined by: $\rho_s = JS(S+1)Z_\xi^{(S)}$.

The spin stiffness governs the leading exponential divergence, but its value can be accurately calculated only if the leading temperature behavior of the preexponential factor $A(T)$ is known. In analytic calculations, $A(T)$ depends on the level of approximation. For instance, in the classical 2D $NL\sigma M$, at the 1-loop level this prefactor is temperature independent, $A(T) \propto const$, while at the 2-loop level $A(T) \propto T$.

In a numerical simulation, the extraction of correct temperature-dependence is computationally very demanding, since it requires spanning a wide range of correlation lengths and a high statistical quality of data. For this reason, our calculation is the first one that can clearly identify the functional $form$ of the correlation length from the numerical data. Since the correlation lengths we measure range from 1 to 28, with statistical uncertainties always smaller than 5%, we can clearly distinguish between different powers of T in front of the exponential. Our data show that the preexponential factor has to be a constant. This follows from a very good quality of the fit given in Eq. (6.8). The parameters of the fit are:

$$A = 0.276(6), \quad \rho_s = 0.199(2)J \ (Z_\xi = 0.265(2)) \tag{6.9}$$

These considerations are illustrated in Fig. 15, where we show our data, along with the best fits obtained with different powers T^α, $\alpha = -1, 0, 1$. Clearly, only the constant prefactor renders a good description of the data. We also fit the data with a general form

$$\xi = AT^\alpha e^{2\pi\rho_s/T} \tag{6.10}$$

regarding α as an additional free parameter. From such a fit, we obtain $\alpha \simeq 0.03$, which may be regarded as an estimate of the upper bound on this exponent. This fit is also shown in Fig. 15, although it is completely indistinguishable from the straight line.

This is in complete agreement with the calculation of CHN [5], where the classical 2-loop term and $1/T$ factor coming from quantum to thermal crossover conspire to give a constant prefactor. The analysis of CHN assumes that the long-wavelength sector of the antiferromagnet can be described by a quantum generalized nonlinear σ model with effective action (written in dimensionless coordinates)

$$S_{eff} = \frac{1}{2g_0}\int_0^{\beta\hbar c_0\Lambda} du \int d^2\mathbf{y}\{|\partial_{\mathbf{y}}\mathbf{\Omega}|^2 + |\partial_u\mathbf{\Omega}|^2\} \tag{6.11}$$

where the dimensionless coupling constant $g_0 = \hbar c_0\Lambda/\rho_s^0$ is determined by local spin wave velocity c_0, short-range cutoff Λ and local spin stiffness ρ_s^0. A momentum shell renormalization group calculation leads to a set of recursion relations which show that the system has no finite temperature transition. For any nonzero temperature, the flow is toward high temperatures, but the correlation length grows exponentially as T is lowered, just like in the classical case.

The quantum nature of the model just leads to a finite size effect due to the finite slab "width" in the imaginary time direction. The one loop calculation reveals that the only difference with respect to the classical case is that the short wavelength cutoff for the

quantum model is not the lattice spacing a, but the thermal de Broglie wavelength, which is equal to $\lambda_{TH} = \hbar c/T$, where c is the spin wave velocity. Thus the one-loop result is:

$$\xi_{1-loop} \propto \frac{\hbar c}{T} e^{2\pi\rho_s/T} \tag{6.12}$$

Quantum dynamics thus leads to a different cutoff and renormalized spin stiffness, but does not affect the classical picture qualitatively. With this in mind, it is natural in the next step to take over the 2-loop β function from the classical model, but with these changes made. This leads to exactly the same results as when the quantum fluctuations are explicitly integrated out in the $2+1$ model of Eq. (6.11). The 2-loop result, given by Eq. (6.8), compares very favorably with our computation, as shown above. The excellent fit to Eq. (6.8) up to $T \simeq 1.0$ indicates that the renormalized classical region is quite wide. This picture remains valid up to a crossover temperature where the Josephson and thermal length scales become compatible [5]. This implies that $2\pi\rho_s \simeq T_{cr}$, *i.e.*, Eq. (6.8) should be valid while the argument of the exponential is of order 1 or less.

Let us compare the renormalized spin stiffness we obtained to those obtained in previous Monte Carlo simulations [6,7] on much smaller lattices ($\leq$ 20x20). Manousakis and Salvador [6] gave $\rho_s \simeq 0.22$. Gomez-Santos, Joannopoulos and Negele [7] obtained $\rho_s \simeq 0.159$. At lower temperatures, the former will overestimate the growth of the correlation length, while the latter will underestimate it. Both of these simulations are based on the Handscomb Monte Carlo method, which is applicable to spin-1/2 only. The method is, essentially, stochastic evaluation of linked cluster diagrams of perturbation theory in powers of βJ. It is more difficult to perform sampling in the abstract graph space than in the worldline configuration space, and we believe that the method may be rather inefficient and the simulations might be plagued with metastability. In the earlier simulation of [6], the growth of ξ was overestimated, which led to a possibility that the correlation length diverged at a finite temperature. This would imply a rather unlikely analogy with the

defect-driven KT transition for the XY model. This result was brought into question by the latter simulation [7]. Our result ruled out the possibility of a finite temperature phase transition.

Singh and Huse [10] estimated the spin stiffness constant by expanding around the Ising limit (see Chapter 2). They obtain $\rho_s \simeq 0.18(1)$, in reasonable agreement with our calculation. Auerbach and Arovas [3] obtained $\rho_s \simeq 0.185$, quite close to our result. This is a little surprising considering the mean field nature of their theory. They keep the rotational invariance of the Heisenberg Hamiltonian by generalizing it to a model with $SU(N)$ symmetry, and then doing a mean field calculation which is exact in the $N \to \infty$ limit. They do not calculate Gaussian corrections to their result, so it is difficult to know the convergence properties of the theory. It is interesting that our calculation gives a *higher* value than the mean field theory, indicating that the inclusion of fluctuations of all orders in $1/N$ actually *increases* the spin stiffness.

The results for the staggered susceptibility are plotted in Fig. 16. The Ornstein-Zernicke form of staggered spin correlation function implies that $\chi_{st}(T) \propto b(T)\xi^2$. The function $b(T)$ is known to behave as $b(T) \propto T^2$ in the classical 2D $NL\sigma M$ at low temperatures [11]. Our data show that this scaling form applies to the quantum Heisenberg model as well. We plot the staggered susceptibility as a function of $T^2\xi^2$. The data points are well described by a straight line, indicating that $\chi_{st} \propto T^2\xi^2$. The slight curvature at larger correlation lengths is most likely a finite size effect. The constant of proportionality can be related to various quantum renormalization factors [5].

6.3. Comparison with Experiments

It is believed that the Heisenberg model can provide a good description of the undoped high-T_c materials in the insulating phase (Chapter 1). We investigate this by

comparing the correlation lengths from our calculation with those obtained in neutron scattering experiments [8].

The effective exchange coupling J and the lattice spacing a are the free parameters in our calculation. Although the Heisenberg Hamiltonian is an effective model, the most straightforward approach is to identify the microscopic length scale a with the distance between copper atoms on Cu-O planes (neglecting the orthorombic distortion): $a = a_H = 3.79\text{Å}$. The only unknown parameter remains the exchange coupling J, which is to be determined. Choosing $J = 1450\ K$, we plot our data points, the best fit (Eqs. (6.8, 6.9)) and the data from neutron scattering experiments [8] in Fig. 17. The agreement is very good, since our data points are nested between the experimental points. This is a strong indication that the magnetic behavior is indeed dominated by the the nearest neighbor Heisenberg Hamiltonian. Thus, we provide, by a direct comparison with experiment, an independent determination of the exchange constant J:

$$J = 1450 \pm 30\ K$$

The uncertainty of $\pm 30K$ is estimated by plotting our data points for various values of J, until the deviation from the experimental data points becomes noticable. This value of J is in very good agreement with the one derived from the spin pair Raman spectrum, by Singh *et al.* [12]: $J = 1480 \pm 70\ K$. The agreement between estimates derived from both short- and long-wavelength physics is another indication that the Heisenberg model captures the essential physics. Also, the measurements of spin wave velocity [13] on a different sample, combined with the quantum renormalization factor for spin wave velocity [10], yield a close estimate, $J \simeq 1550\ K$.

The fitting form of Eqs. (6.8) and (6.9), with this value of J,

$$\xi(T) = 0.276 \cdot a_H \cdot e^{1.25 \cdot 1450/T[K]}\text{Å} \tag{6.13}$$

can reproduce some of the experimental data at lower temperatures, which are not accessible to direct simulation. Our choice of J yields a very good agreement with experiment over a wide temperature range and achieves the best agreement between our *data points* and experimental data points. Note, however, that as the 3D Néel temperature is approached, the theoretical curve gives systematically larger correlation lengths than experiment. A multitude of small effects like anisotropies, interlayer coupling etc., enter the picture in the vicinity of T_N. T_N itself is a manifestation of these effects, because the pure Heisenberg model does not have a finite temperature transition.

The correlation lengths are very sensitive to the value of spin stiffness, due to the exponential dependence. However, if an error is made in the calculation of ρ_s in units of J, it is possible to adjust the amplitude A and the exchange J, and still obtain reasonable agreement with experimental data. Because of that, the values of J that were used in the literature to fit the experimental data vary from 900 K to 1600 K. We believe that our calculation provides an accurate determination of J, in the sense that it is the optimal value, as long as one is satisfied with the description of the real system via the single coupling model. It is a first principles calculation on the model, with all possible sources of error under control. Our estimate is derived from the data of very high statistical quality, and is based on a direct comparison with experiment. Furthermore, our values of ρ_s and J are in very good agreement with two *independent* series estimates of ρ_s [10] and J [12].

We do not perform the calculation for higher spins, but in order to appreciate the renormalization effects of quantum fluctuations in the spin-1/2 case, it is very instructive to make a comparison with the extensively studied spin-1 system K_2NiF_4 [14] with $J = 104K$. Recently, Birgenau [15] very successfully fitted the measured correlation lengths to

$$\xi(T)/a = 0.123e^{5.31J/T}/(1 + T/5.31J) \tag{6.14}$$

The factor $(1 + T/5.31J)$ comes from integration of the 2-loop β-function without taking the $T \to 0$ limit and could be neglected if T is very close to 0. The spin wave value (see Chapter 2) is:

$$2\pi\rho_s = 2\pi JS^2(1 + 0.158/2S)^2(1 - 0.552/2S) \tag{6.15}$$

For S=1, $2\pi\rho_s = 5.30$, fits the experiment quite well (Fig. 18), whereas for S=1/2, the spin wave value $2\pi\rho_s = 0.944$ differs significantly from the correct value $2\pi\rho_s = 1.25$. This indicates that the large quantum fluctuations in a spin-$\frac{1}{2}$ system are not adequately accounted for in the spin wave theory, whereas for a spin-1 system, they are. Similar results follow from Raman spectra experiments and calculations [12].

6.4. Preliminary Dynamical Calculations

The dynamical structure factor for a noninteracting spin wave system can be easily calculated [16]. The result can be written in the following form:

$$S(\mathbf{q},\omega) = \frac{1}{2}\frac{\omega\chi(\mathbf{q})}{(1 - e^{-\beta\omega})}F(\mathbf{q},\omega) \tag{6.16}$$

where the function F contains only two delta functions coming from sharp spin waves. The exponential factor guarantees that the detailed balance condition is satisfied, *i.e.*,

$$S(\mathbf{q},\omega) = S(\mathbf{q},-\omega)e^{\beta\omega} \tag{6.17}$$

while $\chi(\mathbf{q})$ is the static wave vector dependent susceptibility. The arguments at the end of Chapter 2 imply a natural guess for F in the case of the real system. For large correlation lengths the quasi-long range order can support well-defined spin waves, but they will be smeared mostly by quantum fluctuations and then due to thermal effects. We attempt to represent that by introducing finite widths for the spin wave peaks. It is reasonable then to take over the expression familiar from the elementary theory of damped harmonic

oscillator and to attempt to fit dynamic correlations by a sum of two Lorentzians centered on the expected spin wave frequencies $\omega(\mathbf{q})$. The spin wave frequencies and the widths of the spin wave peaks $\gamma(\mathbf{q})$ may be regarded as fitting parameters.

Hence, we calculate the Laplace transform (5.16) of the following variational guess for the dynamic structure factor:

$$S_g(\mathbf{q},\omega) = A(\mathbf{q},T)\frac{\beta\omega}{(1-e^{-\beta\omega})}\{\frac{1}{((\omega-\omega_\mathbf{q})^2)+\gamma_\mathbf{q}^2}+\frac{1}{((\omega+\omega_\mathbf{q})^2)+\gamma_\mathbf{q}^2}\} \qquad (6.18)$$

and determine A, $\omega_\mathbf{q}$ and $\gamma_\mathbf{q}$ from the χ^2 fit to the imaginary time data from the simulation (see previous chapter). At the moment, we have the results for 2 temperatures, $T = 0.5J$ and $T = 0.45J$. In both cases, we used Trotter number $M = 32$ and a 32x32 lattice. It is probably possible to use the same lattice at $T = 0.4J$ where we know from static calculations that the correlation length is $\xi \simeq 10$, but the major advantage of using a bigger lattice is smaller spacing between the allowed wave vectors in the Brillouin zone. The data on imaginary time axis are, expectedly, correlated and a more careful analysis than what is suggested by Eq. (5.17) is actually required. We diagonalize the full covariance matrix and perform the rotation in data space to obtain uncorrelated data before doing the χ^2 fit.

The fit to a sum of Lorentzians is very good and is shown in Figs. (20,21) for selected wave vectors. The value of χ^2 per degree of freedom is typically $\simeq 1$. This is true for all $\mathbf{q}$ points that we analyzed (shown in Fig. 19). The linewidths are typically much smaller than the peak frequencies. Therefore, spin waves are well-defined excitations and are weakly damped for most of the zone. From the locations of the peaks we derive the dispersion of spin waves at low temperatures. This is shown in Fig. (22) and the linewidths are plotted in Fig. (23). We also show the linear part of the spin wave spectrum for renormalized spin waves, using the result of Singh *et al.* with $Z_c = 1.18$ for the spin wave velocity.

Apparently, finite temperatures contribute weakly to the renormalization of spin waves. At the Bragg point, the excitations develop a gap, related to the finite correlation length, as is obvious from the above figures.

The gap decreases with decreasing temperature. The damping at the Bragg point is nonzero, since the staggered magnetization is not a constant of motion. The q-dependence of the damping is not dramatic. It grows as the wave vectors move away from the Bragg point, but it is rather weakly dependent on the wavevector along Γ direction. The wavy structure that is seen in Fig. 23 is more likely the artifact of fitting, rather than a real effect. It is apparent that the relaxation rate of the staggered order parameter at $\mathbf{k} = (\pi,\pi) - \mathbf{q} = 0$ and at $k\xi \propto 1$ are of the same magnitude. This relaxation rate should set the frequency scale for $\omega_{\mathbf{q}}$ and $\gamma_{\mathbf{q}}$. It is then plausible that the momentum and frequency dependent quantities, like $S(\mathbf{q},\omega)$ should be expressible in terms of the scaling functions which depend on momenta and frequencies through scaled arguments only:

$$S(\mathbf{q},\omega) = \omega_s^{-1} S(q)\Phi(\mathbf{Q},\Omega) \tag{6.19}$$

where

$$\mathbf{Q} = \mathbf{q}\xi, \qquad \Omega = \frac{\omega}{\omega_s} \tag{6.20}$$

The spin wave damping at the Bragg point should be determined by the spin wave frequency of the mode with wave vector $\propto \xi^{-1}$:

$$\omega_s \simeq \omega(\xi^{-1}) = (\frac{\rho_s(\xi)}{\chi_\perp(\xi)})^{1/2}\xi^{-1} \tag{6.21}$$

The second equality comes from spin wave hydrodynamics [5]. The spin stiffness and transversal susceptibility which enter this relation are defined on length scale $\propto \xi$. Using the 1-loop momentum shell RG results for these quantities, one arrives at CHN's suggestion for the scaling frequency [5]

$$\omega_s \propto \frac{c}{\xi}(\frac{T}{2\pi\rho_s})^{1/2} \tag{6.22}$$

These conjectures can be tested by performing calculations for various temperatures. While some sort of dynamic scaling is very probable, the scaling frequency is not necessarily given by (6.22). CHN and others [17] also give certain asymptotic forms for spin wave frequencies and damping in the limit $k\xi \gg 1$. However, the results coming from different approximations are contradictory. Numerical simulation will be able to test the validity of different microscopic analytical approaches. What is even more interesting is the regime $k\xi \simeq 1$, which is difficult to access analytically. The spin stiffness is vanishing on this length scale, and topological defects may be important. A more sophisticated analysis using maximum entropy methods with "informed" default models that incorporate three sum rules [18] is underway. This will enable us to go beyond the simple sum of Lorentzians as the structure factor. In principle, by collapsing the data from various temperatures onto a single curve, we should be able to extract the exact scaling function $\Phi(\mathbf{Q}, \Omega)$.

References

[1] M. Takahashi, Phys. Rev. B. **40**, 2494 (1989).

[2] G. S. Rushbrooke, G. A. Baker, Jr., and P. J. Wood, in *Phase Transitions and Critical Phenomena*, ed. by C. Domb and M. S. Green (Academic, New York, 1974), Vol. 3, Chap. 5.

[3] A. Auerbach and D. P. Arovas, Phys. Rev. Lett. **61**, 617 (1988); D. P. Arovas and A. Auerbach, Phys. Rev. B **38**, 316 (1988).

[4] P. B. Weichman, private communication.

[5] S. Chakravarty, B. I. Halperin, and D. Nelson, Phys. Rev. Lett. **60**, 1057 (1988); Phys. Rev. B **39**, 2344 (1989).

[6] E. Manousakis and R. Salvador, Phys. Rev. Lett. **60**, 840 (1988); Phys. Rev. B **39**, 575 (1989). Their correlation lengths are close to ours for $T \geq 0.5J$.

[7] D. H. Lee, J. D. Joannopoulos and J. W. Negele, Phys. Rev. B **30**, 1599 (1984); G. Gomez-Santos, J. D. Joannopoulos and J. W. Negele, Phys. Rev. B **39**, 4435 (1989).

[8] Y. Endoh, K. Yamada, R. J. Birgenau, D. R. Gabbe, H. P. Jenssen, M. A. Kastner, C. J. Peters, P. J. Picone, T. R. Thurston, J. M. Tranquada, G. Shirane, Y. Hidaka, M. Oda, Y. Enomoto, M. Suzuki, and T. Murakami, Phys. Rev. B **37**, 7443 (1988).

[9] S. H. Shenker and J. Tobochnik, Phys. Rev. B **22**, 4462 (1980).

[10] R. R. P. Singh, Phys. Rev. B **39**, 9760 (1989); R. R. P. Singh and D. A. Huse, Phys. Rev. B **40**, 7247 (1989).

[11] D. J. Amit, in *Field Theory, the Renormalization Group, and Critical Phenomena*, (World Scientific, Singapore, 1984).

[12] R. R. P. Singh, P. A. Fleury, K. B. Lyons and P. E. Sulewski, Phys. Rev. Lett. **62**, 2736 (1989).

[13] G. Aeppli, S. M. Hayden, H. A. Mook, Z. Fisk, S.-W. Cheong, D. Rytz, J. P. Remeika, G. P. Espinosa, and A. S. Cooper, Phys. Rev. Lett. **62**, 2052 (1989).

[14] R.J. Birgeneau, J. Skalyo, Jr., and G. Shirane, Phys. Rev. B **3**, 1736 (1971).

[15] R.J. Birgeneau, Phys. Rev. B **41**, 2514 (1989).

[16] W. Jones and N. H. March, in *Theoretical Solid State Physics*, Chapter 10, (Dover Publications, Inc., New York, 1973), vol. 1.

[17] T. Becher and G. Reiter, Phys. Rev. Lett. **63**, 1004 (1989).

[18] P. C. Hohenberg and W. F. Brinkman, Phys. Rev. B **10**, 128 (1974).

[19] Y. Okabe and M. Kikuchi, J. Phys. Soc. Jpn. **57**, 4351 (1988).

Chapter 7

Quantum XY Model in Two Dimensions

It is well known that the two-dimensional (2D) classical XY model undergoes Kosterlitz-Thouless (KT) [1] transition at $kT_c/J = 0.898$ [2,3], characterized by exponentially divergent correlation length and in-plane susceptibility. The transition, due to the unbinding of vortex-antivortex pairs is weak. The specific heat has a finite peak above T_c and it is a smooth function of temperature. The transition cannot be established by measuring specific heat which is a smooth function of temperature, due to the presence of essential singularity.

The natural question is whether the 2D quantum XY magnet will exhibit a phase transition, and if the answer is yes, what is the type of the transition. The answers are relevant to a wide class of 2D problems such as magnetic insulators, superfluidity, melting, and possibly to the recently discovered high-T_c superconducting transition. Physics in two dimensions is characterized by large fluctuations even at the classical level. Changing from the classical model to the quantum model, additional quantum fluctuations (which are particularly strong in the case of spin-1/2) may alter the physics significantly. A direct consequence is that the already weak KT transition could be washed out completely.

Two decades ago, high temperature series studies [4] raised the possibility of a divergent susceptibility for the two-dimensional model. For the classical model, the remarkable

theory of Kosterlitz and Thouless [1] provided a clear physical picture and correctly predicted a number of important properties. We first briefly summarize the most important results. The ground state of the model is a perfectly aligned ferromagnetic state. At low temperatures, the Mermin-Wagner theorem precludes the possibility of long-range order, but, just like in the case of the $O(3)$ symmetry, if the correlation length is large, the pseudo-long-range order may support spin waves, now defined on lengthscales shorter than ξ. What matters is that the nearby spins are always almost aligned although the average direction is not defined. Then, it is natural to begin with the quadratic approximation to the original XY Hamiltonian

$$H = J \sum_{\langle ij \rangle} (S_i^x S_j^x + S_i^y S_j^y) \tag{7.1}$$

Since we are discussing the classical model, spin components are pure numbers. We are mostly interested in the long wavelength properties, so we will turn lattice sums into gradients, and we arrive at the continuum $O(2)$ spin wave Hamiltonian where the local degree of freedom $\theta(\mathbf{r})$ is confined to lie on the unit circle:

$$H = \frac{J}{2} \int d^2\mathbf{r} |\partial_{\mathbf{r}} \theta|^2 \tag{7.2}$$

The analysis of the spin wave Hamiltonian leads to a rather interesting result. Asymptotic spin correlations, instead of exponentially decaying, as is typical in the absence of conventional long-range order, exhibit the power law

$$C(r) \propto r^{-\eta} \tag{7.3}$$

where the exponent η is temperature-dependent. Within the spin wave approximation it is given by:

$$\eta(T) = \frac{T}{2\pi J} \tag{7.4}$$

The power law is typical at a critical point. Spin wave approximation thus leads to a *line* of critical points. This line extends to infinite temperatures if only spin waves are kept. The remarkable insight of Kosterlitz and Thouless was that the inclusion of topological defect configurations, vortices, ultimately leads to the termination of this fixed line. At low temperatures, vortices tend to stay together, tightly bound in pairs and clusters. Because of this, they cancel each other at larger distances, and do not alter qualitatively the spin wave correlations. Above T_c, they begin appearing isolated, and this leads to the exponential decay of correlations above the KT transition.

According to KT theory, the correlation length increases exponentially above T_c, following Eq. (7.11). This leads to an essential singularity in the free energy ($\propto \xi^{-2}$) and specific heat. A very important prediction of the theory is the universal jump in spin stiffness [5]. Although the system has no finite magnetization, at T_c the spin stiffness acquires a nonzero value. This is consistent with the Mermin-Wagner theorem. In 3D, both spin stiffness and magnetization become nonzero below T_c. Spin stiffness and the exponent η are related by

$$\rho_s = \frac{T}{2\pi\eta(T)} \tag{7.5}$$

This exact result is essentially a generalization of Eq. (7.4) with fully renormalized spin stiffness ρ_s replacing the bare spin stiffness J. At T_c the ratio of spin stiffness to the transition temperature has the universal value:

$$\frac{\rho_s(T_c^-)}{T_c} = \frac{2}{\pi} \tag{7.6}$$

This also implies that the exponent η has the universal value

$$\eta(T_c^-) = \frac{1}{4} \tag{7.7}$$

It is also a simple consequence of the recursion relations [5] that near T_c spin stiffness and η exhibit a nonuniversal square root singularity.

However, much less is known about the quantum model. In fact, its behavior has been controversial. Using a large order high temperature expansion, Betts *et al.* [6] suggested a second order transition at $kT_c/J = 0.64$ for spin-1/2. Later, real-space renormalization group analysis was applied [7] to the model with results contradictory and inconclusive. DeRaedt, DeRaedt, Fivez and Lagendijk [8] then presented an exact solution for the pair product approximation [9] and a Monte Carlo simulation. Both were based on the Suzuki-Trotter transformation with small Trotter number M. Their results, both analytical and numerical, supported an *Ising*-like (second order) transition at the Ising point $kT_c/J = 1/2log(1+\sqrt{2}) = 0.567$, with a logarithmically divergent specific heat. Loh, Scalapino and Grant [10] simulated the system with an improved technique [11]. They found that specific peak remains finite and argued that a phase transition occurs at T_c=0.4-0.5 by measuring the change of the "twist energy" from the 4x4 lattice to the 8x8 lattice. The subsequent dispute [12] between DeRaedt, *et al.*, and Loh, *et al.*, centered on the importance of using a large Trotter number M and the global updates in small size systems, which move the system from one topological subspace to another. Recent path integral Monte Carlo simulations [13] still measure only thermodynamic quantities, and cannot offer firm evidence for *any* kind of transition. A decoupled cell Monte Carlo method [14] indicates an ordinary transition, with correlation length diverging as $\xi \propto t^{-\nu}$, and $\nu \simeq 0.7$.

The key to pinning down the existence and the type of transition lies in the study of correlation length and in-plane susceptibility because their divergences constitute the most direct evidence of a phase transition. These quantities are much more difficult to measure and large lattices are required in order to avoid finite size effects. These key points are lacking in the previous works, and are the focus of our study. We report simulations on much bigger lattices with much better statistics. Due to the algorithmic advances and extensive use (equivalent to about 1100 Cray hours) of the Caltech/JPL MarkIIIfp parallel supercomputer, we are able to measure spin correlations and thermodynamic quantities

accurately on very large lattices (96x96). We report convincing evidence that a phase transition does occur at finite temperature in the extreme quantum case, spin-1/2. At the transition point, $kT_c/J = 0.35 \pm 0.01$, the correlation length and susceptibility diverge exactly according to the form of Kosterlitz-Thouless [1] (see Eq. (7.11)). If the transition is KT-like, the universal properties should persist in the quantum model as well. To firmly establish the nature of the transition, we measure the spin stiffness and exponent η below T_c. We find that their behavior, within statistical uncertainties of the simulation, agrees very well with KT theory. From a broader perspective, this comprehensive study of the quantum model provides additional numerical support for the well-founded belief that quantum fluctuations are irrelevant for finite temperature critical phenomena.

The quantum XY model is simulated following the Suzuki-Trotter approach. The method is discussed in Chapters 3, 4 and 5. Here and below, both the Boltzmann constant k_B and the exchange coupling J are set to 1. We performed simulations on lattices as large as 96x96 spins and temperatures ranging from $T = 1.0$ to 0.2. The correlation length calculations were done in the following manner. We start at $T = 1.0$. As T is lowered, we systematically increase the lattice size to satisfy $L \geq 4\xi$ at every T, so that finite size effects are small. A similar procedure was used successfully for the classical XY model [3]. Over 95% of CPU time is spent on lattices 64x64 and 96x96 at four temperatures. We did two sufficiently long runs at every T. We used a large Trotter number M=24 for all temperatures. The width of the imaginary time slice is $\Delta\tau \leq 0.1$ for the temperature range we studied. This should be contrasted with DeRaedt *et al.* [8] who used $\Delta\tau \sim 1$ and Loh *et al.* [10] who used $\Delta\tau = 0.25$ (which is already reasonably good).

Our main focus is to compute the static spin correlation function:

$$C(\mathbf{r}) = \frac{4}{L^2} \sum_{\mathbf{n}} \langle S^y_{\mathbf{n}} S^y_{\mathbf{n+r}} \rangle \tag{7.8}$$

and the in-plane susceptibility

$$\chi = \langle (\sum_{\mathbf{r}} S^x_{\mathbf{r}})^2 + (\sum_{\mathbf{r}} S^y_{\mathbf{r}})^2 \rangle / 2L^2 = \langle (\sum_{\mathbf{r}} S^y_{\mathbf{r}})^2 \rangle / L^2. \tag{7.9}$$

where L is the linear size of the system. At large r, $C(r)$ has the asymptotic form:

$$F(r) = Ar^{-\lambda} e^{-r/\xi} \tag{7.10}$$

where ξ is the correlation length, η is the algebraic exponent. In practice, we fit to $C(r) = F(r) + F(L-r)$ to incorporate periodic boundaries. The fits to this form are excellent, quite similar to those for the Heisenberg model. The best fits for ξ, λ and χ are plotted in Figs. 24, 25, and 26. Selected correlation functions and the fits are plotted in Fig. 27.

Clearly, ξ and χ increase very fast as T is lowered. They will diverge at some finite T_c. We fit them to the form predicted by Kosterlitz and Thouless for the classical model [1]

$$\xi(T) = Ae^{B/(T-T_c)^\nu}, \ \nu = \frac{1}{2} \tag{7.11}$$

The fit is indeed very good (χ^2 per degree of freedom is 0.81), as shown in Fig. 24. The fit for correlation length gives

$$A_\xi = 0.27(3), B_\xi = 1.18(6), T_c = 0.350(4) \tag{7.12}$$

A similar fit for susceptibility χ is also very good (χ^2/DOF=1.06):

$$A_\chi = 0.060(5), B_\chi = 2.08(6), T_c = 0.343(3) \tag{7.13}$$

as shown in Fig. 26. The good quality of both fits and the closeness of T_c's obtained with two independent fits are among the main results of this work. The fact that these fits also reproduce the expected scaling behavior $\chi \propto \xi^{2-\eta}$ with

$$\eta = 2 - B_\chi / B_\xi = 0.24 \pm 0.10 \tag{7.14}$$

is a further consistency check. These results strongly indicate that spin-1/2 XY model undergoes Kosterlitz-Thouless phase transition at $T_c = 0.350 \pm 0.004$. The exponent λ for temperatures above T_c is plotted in Fig. 25. λ is consistent with the Ornstein-Zernike exponent (d-1)/2=1/2 at higher T. As ξ grows, we observe a decrease, which is quite analogous to the Heisenberg case. Due to the violation of the asymptotic requirement $r \gg \xi$, we are seeing corrections to the OZ form from shorter length scales. Close to T_c the correlation function crosses over to the power law (7.3). Below T_c we use the pure power law to extract η. Those results will be discussed later.

We measured energy and specific heat C_v (for $T \leq 0.41$ we used 32x32 lattice). The value of energy is in general agreement with previous work [10,13]. The specific heat is shown in Fig. 28. We found that C_v has a peak above T_c, at around $T = 0.45$. The peak clearly shifts away from $T = 0.52$ on the much smaller 8x8 lattice [10]. De Raedt *et al.* [8] suggested a logarithmically divergent C_v in their simulation, which is likely an artifact of their small M values. The strong anisotropy in temporal direction probably pushes the system toward the Ising limit. It is possible that approximations with different Trotter numbers belong to different universality classes. A classical equivalent with a small Trotter number is then qualitatively different from the original quantum model. One striking feature in Fig. 28 is a very steep increase of C_v at $T \approx T_c$. The shape of the curve is asymmetric near the peak. These features of the C_v curve differ from that in the classical XY model [2,3].

The evidence so far strongly suggests that the quantum model undergoes classical KT transition. Quantum fluctuations are capable of pushing the transition point from T_c=0.898 [3] in the classical model down to T_c=0.35 in the quantum spin-1/2 case, although not strong enough to push it down to zero. If this were the case, it would probably be more difficult to establish the map between the quantum system and the classical one with

identical long wavelength physics. However, the experience with the Heisenberg model shows that even that is still intuitively simple if the ground state is qualitatively similar to the classical ground state. The constant B_ξ is reduced from 1.67 in the classical case [3] to 1.18 in the spin-1/2 case. This is expected, since the amplitude of the square root is nonuniversal. This amounts to a multiplicative renormalization of temperature scale. An analogous thing happens with the quantum $NL\sigma$ model discussed in the previous chapter, after quantum fluctuations are integrated out.

The next step toward establishing the nature of the transition is to test the universal predictions of the theory. For that purpose, we first measure the correlation functions below T_c and deduce temperature-dependence of the exponent η. The correlation functions are measured on system sizes ranging from 12x12 to 48x48. They are fitted by a pure power law, altered to incorporate periodic boundaries:

$$C(r) = A(r^{-\eta} + (L-r)^{-\eta}) \tag{7.15}$$

One should expect logarithmic correction to the power law at T_c, but our data are very well fitted with this form within statistical uncertainty. Some of the fits are shown in Figs. 29-31. The size-dependence of the exponent η, as inferred from these fits is negligible. We show η as a function of lattice size at $T = 0.2$, $T = 0.3$ and $T = 0.36$ in Fig. (32). For larger lattice sizes, η seems to increase slightly, which is consistent with the expected decrease of spin stiffness as the lattice size grows, but it is difficult to quantify it, since the differences are within error bars. It is apparent from the figures that the smallest lattices tend to have larger η. This comes again from the shorter length scales where correlations decrease more quickly. As expected, the exponent values are also more sensitive to the short-range cutoffs than in the case of larger lattices. In Fig. (33) we show $\eta(T)$ for all lattice sizes. We attempt to fit $\eta(T)$ with the following form:

$$\eta(T) = A + B(T - T_c)^\alpha \tag{7.16}$$

Since there are too many variational parameters for the amount and quality of data, we fix the parameter α to the value predicted by KT theory, $\alpha = 1/2$. Then we change the value of T_c until the fits give the expected universal value of $A = \eta(T_c) = 1/4$. This procedure gives $T_c = 0.321$ and the square root amplitude $B = 0.343$ for the 32x32 lattice. On the 24x24 lattice, the same approach yields $T_c = 0.324$ and $B = 0.254$. While the amplitudes differ by about 25%, the estimated transition temperatures are very close. They also agree well with the value of T_c estimated from susceptibility and correlation length measurements. While these estimates are not quantitatively as accurate as the correlation length measurements above T_c, we obtain a perfectly consistent picture. These results imply that, within the accuracy of the simulation, $\eta(T_c)$ is equal to the universal KT result of 1/4. They also show that below T_c, its behavior is consistent with the nonuniversal square root cusp, in accordance with KT picture. The fits are shown in Figs. 34a and b.

Using the exact relation (7.5) we can go one step further. We can do an independent calculation of spin stiffness, using the method described in Chapter 5, and then calculate η from Eq. (7.5). The results should agree with those calculated from the correlation function within error bars.

The results for spin stiffness are currently available for only two lattice sizes, 16x16, and 24x24. They are shown in Figs. 35 and 36. Qualitatively, they agree with everything one expects to see in a KT transition. The finiteness of ρ_s above $T = 0.36$ is a finite size effect. Finite size is also responsible for the smooth appearance near T_c, but it is apparent that the "tail" above T_c shrinks as lattice size increases, and the increase around T_c is steeper. In the thermodynamic limit, this will become a jump. In Figs. 37a and b we show the exponent η calculated in the 2 different ways. The agreement is very good below T_c and, as expected, breaks down sharply above. Hence, this final consistency check

also points to the KT transition. Spin stiffness is also well fitted with Eq. (7.16). The procedure is the same as for the exponent η. The parameter A is now kept at the value $2/\pi T_c$, and $\alpha = 1/2$. The fit is shown in Fig. 36.

The quantum fluctuations lead to a finite density of topological defects even in the ground state. This leads to small additional depletion of spin stiffness at $T = 0$. A way to quantify the defects in the model is to measure the vortex density. The method is given in Chapter 5. We use both the operator suggested by Betts *et al.* [15], as described in Chapter 5, and Swendsen's [16] "vortex detector:"

$$V_S = \sum_{\mathbf{p}} (1 - \sigma_1^x \sigma_3^x - \sigma_1^y \sigma_3^y)(1 - \sigma_2^x \sigma_4^x - \sigma_2^y \sigma_4^y) \tag{7.17}$$

The summation goes over all elementary squares on the lattice, as depicted in Fig. 38. Both operators lead to the same qualitative picture, with a sharp rise of vortex density near T_c. The results are plotted in Fig. 39. The results show negligible size-dependence, since they measure short-range multispin correlations. Although one cannot quite naively relate these microscopic vortex configurations to the vortices of KT theory, this is in general agreement with the idea of vortex driven transition. Deep into the spin wave region, vortex density is very small and practically temperature-independent. We use this weak temperature-dependence to estimate the ground state vortex density: $V_B \simeq 0.0236(1)$. This is in excellent agreement with the result of Betts *et al.*, obtained by extrapolating to thermodynamic limit exact diagonalization results on clusters of up to 18 spins: $V_B \simeq 0.025$.

We also measured spin correlations along S_z directions in spin space. They arise only as a consequence of the spin algebra in the quantum model. They are weak and short-ranged (see Table 2), which is consistent with the idea that the long wavelength sector of the quantum model is accounted for by a classical model living on a S^1 manifold.

The mapping onto such a model should not present any conceptual problems. Given that the correlation length diverges at a finite temperature, the situation is even simpler than for the Heisenberg model. This is what one expects on rather general grounds, since the renormalization group transformation eliminates all nonzero Matsubara frequencies at finite T, thus eliminating quantum dynamics.

Therefore, one can formulate the following qualitative picture. The low energy sector of the lattice quantum XY model may be described by a quantum field theory. The exact form of the theory is not known, but for the critical properties it is not even important. Finite temperatures lead to the finite thickness $\beta\hbar$ in imaginary time direction which incorporates quantum fluctuations. We can explicitly integrate out quantum fluctuations, until the system becomes truly two-dimensional, just like Chakravarty *et al.* did for the quantum $NL\sigma$ model. The numerical results suggest that the effective model we arrive at is in the same universality class as the classical model. It is then plausible that the dominant fluctuations are described by the following dual effective action of a sine-Gordon type [17,5]:

$$S_{eff} = \frac{1}{2}K_0 \int_{\mathbf{x}} |\partial_{\mathbf{x}} S|^2 d\mathbf{x} + 2y_0 a^{-2} \int_{\mathbf{x}} \cos(2\pi S(\mathbf{x})) d\mathbf{x} \tag{7.18}$$

The memory of the original quantum model on the lattice is burried in the model-dependent quantities K_0, related to the bare spin stiffness and y_0 which controls the self-interaction of vortices. Under the length rescaling transformation of the renormalization group, this effective model is mapped onto a model with different couplings $K(l)$ and $y(l)$, depending on the lengthscale defined by l. A momentum shell renormalization group technique leads to the following flow equations to order y^2[5,17]:

$$\begin{aligned} \frac{\mathrm{d}y^2(l)}{\mathrm{d}l} &= (4 - 2\pi K(l))y^2(l) \\ \frac{\mathrm{d}K^{-1}(l)}{\mathrm{d}l} &= 4\pi^3 y^2(l) \end{aligned} \tag{7.19}$$

These equations are valid in the regime with small density of defects. Our simulation indicates that such an assumption is justified below the transition temperature. The initial conditions are given by $K(0) = K_0$ and $y(0) = y_0$.

Below the transition temperature, one has the important relation: $\rho_s(T)/T = \lim_{l\to\infty} K(l)$. This relation, combined with the flow equations leads to the prediction of the universal jump in spin stiffness at T_c and the nonuniversal square root cusp, which is verified in the simulation. The important fact is that these results do not depend on the *initial* conditions K_0 and y_0.

As we move away from T_c, the behavior depends on the particular model. In principle, given the model-dependent initial conditions, the flow equations may be numerically integrated to obtain $K(\infty)$ and spin stiffness. This was done for the classical XY model, where the proper treatment of spin waves leads to the following choice [117]:

$$\begin{aligned} K^{-1}(0) &= (\beta J)^{-1}(1 + (2\beta J)^{-1}) \\ y(0) &= e^{-K(0)\pi^2/2} \end{aligned} \tag{7.20}$$

This choice leads to a linear decrease in ρ_s from the bare value J at $T = 0$. It is apparent from our simulation that the behavior of ρ_s below T_c in the quantum model is rather flat (see Fig. 36). This indicates that a different choice of the initial conditions is required. The flatness of spin stiffness in the quantum model suggests that a more appropriate model-dependence might be a generalization of the model choice of Nelson and Kosterlitz [5]:

$$\begin{aligned} K(0) &= C_1 \beta J S^2 \\ y(0) &= e^{-C_2 K(0)} \end{aligned} \tag{7.21}$$

The nonuniversal constants C_1 and C_2 can be fixed by measuring the spin stiffness in the ground state, or at very low temperatures. It should be noted that the observed behavior of the spin stiffness is also consistent with a T^3 power law. The power law T^{d+1}, where d is the dimensionality of a quantum spin system, is obtained by an application of the naive

spin wave theory [18]. In order to resolve this issue, we are currently trying to push the simulation down to $T = 0.05J$ and to calculate $T = 0$ spin stiffness by a variational Monte Carlo approach.

While the simulation of the static properties of the quantum XY model fully agrees with the well-established and understood Kosterlitz-Thouless picture, the understanding of dynamical properties is not as satisfactory. Computer time permitting, it will be very interesting to investigate dynamic correlations in the XY model, particularly in the vicinity of T_c. At this point, the spin stiffness vanishes and the defects are expected to substantially affect the line shape. Very little is known about the dynamics of these objects, particularly in the context of a quantum model.

References

[1] J.M. Kosterlitz and D.J. Thouless, J. Phys. **C6**, 1181 (1973); J.M. Kosterlitz, J. Phys. **C7**, 1046 (1974).

[2] S. Miyashita, H. Nishimori, A. Kuroda and M. Suzuki, Prog. Theo. Phys. **60**, 1669 (1978); J.Tobochnik and G.V. Chester, Phys. Rev. **B20**, 3761 (1980); J. van Himbergen and S. Chakravarty, Phys. Rev. **B23**, 359 (1981); J.F. Fernandez, M.F. Ferreira and J. Stankiewicz, Phys. Rev. **B34**, 292 (1986).

[3] R. Gupta, J. DeLapp, G. Batrouni, G.C. Fox, C.F. Baillie and J. Apostolakis, Phys. Rev. Lett. **61**, 1996 (1988); U. Wolff, Nucl. Phys **B322**, 759 (1989).

[4] H.E. Stanley, Phys. Rev. Lett. **20**, 589 (1968); M.A. Moore, Phys. Lett. **B5**, 65 (1969).

[5] D. R. Nelson and J. M. Kosterlitz, Phys. Rev. Lett. **39**, 1201 (1977); J. V. José, L. P. Kadanoff, S. Kirkpatrick, and D. R. Nelson, Phys. Rev. B **16**, 1217 (1977).

[6] D.D. Betts, in *Phase Transition and Critical Phenomena*, ed. C. Domb and M.S. Green, (Academic Press, New York, 1974), Vol. 3, p. 569; J. Rogiers, E.W. Grundke and D.D. Betts, Can. J. Phys., **57**, 1719 (1979).

[7] J. Rogiers and R. Dekeyser, Phys. Rev **B13**, 4886 (1976); D.D.Betts and M. Plischke, Can. J. Phys. **54**, 1553 (1976); R. Dekeyser, M. Reynaert and M.H. Lee, Physica **86-88B**, 627 (1977); T. Tatsumi, Prog. Theor. Phys. 65, 451 (1981); H. Takano and M. Suzuki, J. Stat. Phys. 26, 635 (1981).

[8] H. De Raedt, B. De Raedt, J. Fivez and A. Lagendijk, Phys. Lett **104A**, 430 (1984); H. De Raedt, B. De Raedt and A. Lagendijk, Z. Phys. **B57**, 209 (1984); Also see, M. Suzuki, S. Miyashita, A. Kuroda and C. Kawabata, Phys. Lett. **A60**, 478 (1977).

[9] M. Suzuki, J. Stat. Phys. **43**, 833 (1986).

[10] E. Loh, Jr., D. J. Scalapino and P. M. Grant, Phys. Rev. **B31**, 4712 (1985).

[11] J. E. Hirsch, D. J. Scalapino, R. L. Sugar and R. Blankenbecler, Phys. Rev. **B26**, 5033 (1982).

[12] H. De Raedt and A. Lagendijk, Phys. Rev. **B33**, 5102 (1986); E. Loh, Jr., D. J. Scalapino and P. M. Grant, Phys. Rev. **B33**, 5014 (1986).

[13] Y. Okabe and M. Kikuchi, J. Phys. Soc. Jpn. **57**, 4351 (1988).

[14] S. Homma, T. Horiki, H. Matsuda and N. Ogita, Nagoya University preprint.

[15] D. D. Betts, F. C. Salevsky, and J. Rogiers, J. Phys. A **14**, 531 (1981).

[16] R. H. Swendsen, Phys. Rev. Lett. **49**, 1302 (1982).

[17] T. Ohta and D. Jasnow, Phys. Rev. B **20**, 139 (1979).

APPENDIX

Source Code

The most important parts of the source code are listed here. Following a subroutine name is a brief explanation of its purpose. The comments which clarify certain pieces of the code are inserted within the body of a subroutine.

```
/*****************************************************************
size.h : include file, defines
         maximum sizes of the lattice and some masks
*****************************************************************/

#define  MXnx            128              /* max size in x-dir*/
#define  MXny            128              /* max size in y-dir*/
#define  MXtrot          56               /* max Trotter number*/
#define  MXword          (MXtrot/8)       /* max number of 32-bit words*/


#define  HIGH_1          0x80000000
#define  HIGH_3          0xe0000000
#define  HIGH_4          0xf0000000
#define  LOW_3           7
#define  LOW_4           15
#define  M_TIME          0x80
#define  M_SPACE         0x40
#define  M_GLOB          0x20
#define  M_WIND          0x10

/*****************************************************************
spdef.h : include file with global variables,
          defines arrays and parameters
*****************************************************************/
#include "size.h"

unsigned int tlines[MXnx*MXny*MXword]; /* spin words */
unsigned int bits[32], low_bits[32], high_bits[33]; /* masks */
unsigned int mask_dn[4],mask_md[4],mask_up[4],mask_hi[4];
unsigned int t_mask[32],s_mask[4];

int fwdt[40], bckt[40]; /* arrays to include periodic boundaries */
int fwdx[4*MXnx], bckx[4*MXnx],fwdy[4*MXny],bcky[4*MXny];
int wr[3*MXword];
float t_prob[16][16], s_prob[5], g_prob[MXword*32+1]; /* probabilities*/

/* hypercube environment       */
int doc,procnum,totproc,cpmask;
int nprocs[2],perbc[2],recpnum[2],nextproc[2][2],dim;

/* parameters */
int upflag,totswp,mratio,tmratio,blksiz,iseed,init_flag,accpt,wrtsiz;
int nx,ny,nword,totspin,nspins,mmax;
float temp, f0,f1,f2,f3,f20,f30,f32,e0,e1,e2,e3,g2,g3;

char outname[40];

float kosinus[MXnx*MXny*2];
float e_data[60];
int taccpt, saccpt, gaccpt, waccpt;
float taacc, saacc, gaacc, waacc;
```

```
/*************************************************************
globline: global flip in time direction. This routine will
          sweep through the lattice, and for any given spin
          word identify its neighbors. The actual update is
          performed in globflipr().
*************************************************************/

#include "spext.h"
#include "cros.h"
#include "ih.h"

/*********************************************************************
                        glob line geometry
                                          y
                 *                        ^
                 ylow                     |
                                          |
         *xhgh   *cen    *xlow            +------> x

                 yhgh
                 *

*********************************************************************/
/*********************************************************************
                        glob line memory management

              |    |  |  |  |  |       |  |    |         |
              |    |  |  |  |  |       |  |    |         |
              |    |  |  |  |  | ..... |  |    |         |
              |    |  |  |  |  |       |  |    |         |

      space_left  0  1  2  3  4  ...  6  7 space_right0 space_right1
                  |<-----  local  ------>|

*********************************************************************/

glob_lines()
{
int x,y,len,space_right,space_left,y0,y1,y2;
int xfwd, xbck, yfwd, ybck,cen,in1,out1;

len = nx*nword*4;
space_right = ny ;
space_left  = ny+1;

/* copy y=ny-1 plane from left to right node */
y0 = space_left;
out1 = (ny-1)*nx*nword;
in1 = y0*nx*nword;
cshift(&tlines[in1],nextproc[0][NEG],len,&tlines[out1],
                                    nextproc[0][POS],len);
cflush(&tlines[in1],len);

in1 = nword*space_right*nx;
out1 = nword*0*nx;

for(y=0;y<ny;y++){
y1=y+1;
y2=(y == 0) ? space_left : (y-1);
if(y1 == ny) {
/* pass tlines from right node to left */
cshift(&tlines[in1],nextproc[0][POS],len,&tlines[out1],
                                   nextproc[0][NEG],len);
cflush(&tlines[in1],len);
}
```

```
for(x=0;x<nx;x++){
        xfwd= nword*(fwdx[x] + y*nx) ;
        xbck= nword*(bckx[x] + y*nx) ;
        yfwd= nword*(x + y1*nx) ;
        ybck= nword*(x + y2*nx) ;
        cen = nword*(x + y*nx) ;
/* identify the neighbors and pass their pointers to the actual
flipping routine                                                  */
        if(y%2 == 0) {
            if(x%2 == 0)
                glob_flipr(xfwd,cen,xbck,yfwd,ybck);
            else
                glob_flipr(xbck,cen,xfwd,yfwd,ybck);

        } else {
            if(x%2 == 0)
                glob_flipr(xfwd,cen,xbck,ybck,yfwd);
            else
                glob_flipr(xbck,cen,xfwd,ybck,yfwd);
        }
}
}

}

/*************************************************************
globflipr: Calculate energy of the flip and perform update.
           Spins along trotter (z) direction are packed into
           words of 32 spins each.
*************************************************************/

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

/*************************************************************

                +
                | +
                |   +
                |   +
          +++++++ + +
          +     +   +
          + +   +
          + +-+--
            +   +
            +   |
              + |
                ++ +++
                |    +
                |    +
                ++ +++    --> X

*************************************************************/
glob_flipr(xlow,cen,xhgh,ylow,yhgh)
int xlow,cen,xhgh,ylow,yhgh;
{
unsigned int xlw,xhg,ylw,yhg,x0,x1,y0,y1,i;
unsigned int mskx0,mskx1,msky0,msky1,cenln,flag0,flag1;
int  sum, mw;

/* check if all spins are up or all are down                 */
flag0 = flag1= 1;
for(mw=0;mw<nword;mw++){
        cenln = tlines[cen+mw];
        flag0 &= (cenln == 0)? 1:0 ;
```

```
        flag1 &= (cenln == 0xffffffff)? 1:0 ;
}

sum = 0;
if( (flag0 | flag1) == 1){  /* available for flip  */
    for(mw=0;mw<nword;mw++){
        cenln = tlines[cen+mw];
        xlw = cenln ^ tlines[xlow+mw]  ;
        xhg = cenln ^ tlines[xhgh+mw]  ;
        ylw = cenln ^ tlines[ylow+mw]  ;
        yhg = cenln ^ tlines[yhgh+mw]  ;
/* XOR a spin word with its 4 neighbors                                  */

        for(i=0;i<32;i += 4){
        /*    mskx0 = bits[i];
            mskx1 = bits[i+2];
            msky0 = bits[i+1];
            msky1 = bits[i+3];  */
            x0 = (xlw >> i     ) & 1;
            x1 = (xhg >> (i+2) ) & 1;
            y0 = (ylw >> (i+1) ) & 1;
            y1 = (yhg >> (i+3) ) & 1;
/* pick up the correct bit at every time slice                           */
            sum += (int)(x0 + x1 + y0 + y1);
        }
    }
/* calculate the energy                                                  */
                                /* flip all spins */
        e1 = 2.0*f20*(float)(sum - nword*16) ;
    if(randf() < exp(e1) ) {
        for(mw=0;mw<nword;mw++) tlines[cen+mw] = ~tlines[cen+mw];
        gaccpt ++;
    }
}
} /* end of glob_flipr()  */
```

```
/*************************************************************
wind_lines: Identifies spin words needed for winding number
            updates and calculates transition probabilities.
            If between layers (0 and 1) and (2 and 3)
            they update Ny, if between layers (1 and 2) and
            (3 and 0) they update Nx.
            It is called in node 0 only, after a call to gather().
*************************************************************/

#include "spext.h"
#include "math.h"
/*************************************************************************
        Layer=0,2 goes in y-directions
        Layer=1,3 goes in x-directions
*************************************************************************/

/***  winding_line goes in x (or y) direction:

       +---+   +---+   +---+   +---+   +---+
           |   |   |   |   |   |   |   |   |
           +---+   +---+   +---+   +---+   +---+


**********/
wind_lines()
{
unsigned int x,y,x1,y1,layer,line,line1,vert[MXnx][MXword],t,mask;
unsigned int flag1,flag_even,flag_odd,mask1,mask2;
unsigned int start,flpmsk,flpmsk1,lt,flt,blt;
int engx,engy,mw,mw0,mw1;
float delta;

for(layer=0; layer<4; layer += 2){      /*  goes in y-dir  */
  for(x=0;x<nx;x++){
    if(layer == 0){
              if(x%2 == 0) x1 = fwdx[x];
              else x1 = bckx[x];
    } else{
              if(x%2 == 0) x1 = bckx[x];
              else x1 = fwdx[x];
    }
    for(y=0;y<nx;y++) for(mw=0;mw<nword;mw++){
        line = tlines[nword*(x+y*nx)+mw];
        line1 = tlines[nword*(x+y*nx)+wr[mw+1]];
        vert[y][mw] = line^( (1&line1) << 31 | line>>1);
    }
    for(mw=0;mw<nword;mw++){ /* loop over all Trotter layers */
      flpmsk = 0;
      for(t=0;t<32;t += 4){
        lt = layer + t;
        mask = bits[lt];
        flag_even=flag_odd=flag1=0;
        for(y=0;y<nx;y += 2){
/* this piece calculates contributions from plaqs which are
running in the same direction as the winding line flip */
            flag_even += (tlines[nword*(x+y*nx)+mw] >> lt ) & 1  ;
            flag_odd += (tlines[nword*(x+fwdx[y]*nx)+mw] >> lt ) & 1  ;
            flag1 += ((vert[y][mw] >>lt) & 1)+((vert[fwdx[y]][mw] >>lt) & 1) ;
        }

        if(((flag_even == (nx/2)) && (flag_odd == 0))||
                ((flag_even == 0) && (flag_odd == (nx/2)))){
                /* check if update allowed */
                /*  all 1's or all 0's  */
        if( flag1 == 0 ){ /*  all 0's  */
/* need to treat different layer separately, or shift before flip */
```

```
        mw1 = (lt+1 > 31)? wr[mw+1] : mw ; /* if lt+1>31 use upper word */
        mw0 = (lt-1 <  0)? wr[nword+mw-1] : mw ; /* if lt<1  use lower word */
        flt = fwdt[lt];
        blt = bckt[lt];
        mask1 = bits[flt];
        mask2 = bits[blt];
/* engy:  interacting plaqs in y direction */
        engy = 0;
        start = (layer==0) ? 0 : 1 ;
        for(y=start;  y<nx;y += 2)
          engy += ((vert[y][mw1] >> flt)& 1)+((vert[fwdx[y]][mw0] >> blt) & 1);
/* engx:  interacting plaqs in x direction */
/* these are plaquettes which are sticking out in x-dir from the
winding line flip */
/* they do not need shifts since they are on the same time slice */
        engx = 0;
        for(y=0;y<nx;y += 1)
        engx +=
            ((tlines[nword*(x+y*nx)+mw]^tlines[nword*(x1+y*nx)+mw]) >>lt )&1;
        delta = (engy - nx/2)*2.0*f32 + (engx - nx/2)*2.0*f20;
        if(randf() < exp(delta)) flpmsk = flpmsk | bits[lt] | bits[flt];
        } /* if */
        } /* if */
      } /* t */
      for(y=0;y<nx;y++) tlines[nword*(x+y*nx)+mw] ^= flpmsk;
    }   /* mw loop */
  }     /* x */
}       /* layer */

/* the same as above, but these are running in x-dir and they
update Nx                                                    */

for(layer=1; layer<4; layer += 2){      /*  goes in x-dir  */
  for(y=0;y<nx;y++){
    if(layer == 1){
    /*        if(y%2 == 0) y1 = fwdy[y]; */
              if(y%2 == 0) y1 = fwdx[y];
              else y1 = bckx[y];
    } else{
              if(y%2 == 0) y1 = bckx[y];
              else y1 = fwdx[y];
    }
    for(x=0;x<nx;x++) for(mw=0;mw<nword;mw++){
        line = tlines[nword*(x+y*nx)+mw];
        line1 = tlines[nword*(x+y*nx)+wr[mw+1]];
        vert[x][mw] = line^( (1&line1) << 31 | line>>1);
    }
    for(mw=0;mw<nword;mw++){
      flpmsk = 0;
      flpmsk1 = 0;
      for(t=0;t<32;t += 4){
        lt = layer + t;
        mask = bits[lt];
        flag_even=flag_odd=flag1=0;
        for(x=0;x<nx;x += 2){
            flag_even += (tlines[nword*(x+y*nx)+mw] >> lt) & 1 ;
            flag_odd += (tlines[nword*(fwdx[x]+y*nx)+mw] >> lt) & 1 ;
            flag1 += ((vert[x][mw] >>lt)&1) + ((vert[fwdx[x]][mw] >>lt)&1) ;
        }
        if(((flag_even == (nx/2)) && (flag_odd == 0))||
                ((flag_even == 0) && (flag_odd == (nx/2)))){
                                /*  all 1's or all 0's  */
        if( flag1 == 0 ){ /*  all 0's  */
/*      time boundary condition comes here if any */
        mw1 = (lt+1 > 31)? wr[mw+1] : mw ; /* if lt+1>31 use upper word */
        mw0 = (lt-1 <  0)? wr[nword+mw-1] : mw ; /* if lt<1  use lower word */
```

```
            flt = fwdt[lt];
            blt = bckt[lt];
            mask1 = bits[flt];
            mask2 = bits[blt];
            engx = 0;
            start = (layer==1) ? 1 : 0 ;
            for(x=start;  x<nx;x += 2)
                engx += ((vert[x][mw1] >> flt)& 1)+((vert[fwdx[x]][mw0] >> blt)& 1);
            engy = 0;
            for(x=0;x<nx;x++)
            engy += ((tlines[nword*(x+y*nx)+mw]^tlines[nword*(x+y1*nx)+mw])>>lt)& 1;
            delta = (engx - nx/2)*2.0*f32 + (engy - nx/2)*2.0*f20;
            if(randf() < exp(delta)) {
                if(lt == 31 ) {/* layer=3, top bits in upper word  */
                    flpmsk  = flpmsk | bits[lt] ;
                    flpmsk1 = flpmsk1 | bits[flt];
                } else
                    flpmsk = flpmsk | bits[lt] | bits[flt];
            }
            } /* if */
            } /* if */
          }   /* t  */
          for(x=0;x<nx;x++) if(layer == 1)
            tlines[nword*(x+y*nx)+mw] ^= flpmsk;
          else {
            tlines[nword*(x+y*nx)+mw] ^= flpmsk;
            tlines[nword*(x+y*nx)+wr[mw+1]] ^= flpmsk1;
          }
        }     /* mw */
      }       /* y */
    }         /*  layer loop  */

}
```

```
/*********************************************************
space_flipr: The addresses of 4 spin lines involved in a
             space flip are provided by spaceplaq(). This
             routine will calculate the energies of the
             flips and then update the spin lines
*********************************************************/
#include "spext.h"

/*************************************************************************
        even layers:
                                          y
              3*--up---*2                                layer=0,2
               |       |                  ^
               down    down               |
               |       |                  |
              0*--up---*1                 -------> x


space_flip(s0,s1,s2,s3,layer)

NOTE: by the way the energies computed
      even/odd plaqs makes no difference.


         3 ----- 3
           |     |
           |     |
           |     |
    +++ 2 ----- 2
    +      |     |
    up     |     |
    +      |     |
    +++ 1 ----- 1 +++
           |     |    +
           |     |   down
           |     |    +
         0 ----- 0 +++

  layer=0 down plaq and layer=3 up plaq  must rotate the words
  since the top and bottom spins are in adjacent words.
  Unlike the time_plaq, here we don't shift t-lines.

***************/
space_flip(ss0,ss1,ss2,ss3,layer)
int ss0,ss1,ss2,ss3,layer;
{
unsigned int eng_up0,eng_up1,eng_down0,eng_down1,tot_eng;
unsigned int eng_up0h,eng_up1h,eng_down0h,eng_down1h;
unsigned int flag,f01,f23,f03,f12,aux0,aux1,flpmsk, mask;
unsigned int i, s0, s1, s2, s3, i1;
int mw;
float delta;


/* calculation is done one spin word in time direction at a time
until all spin words stacked upon a 2D lattice site are processed */

for(mw=0; mw<nword; mw++){
s0 = tlines[ss0+mw];
s1 = tlines[ss1+mw];
s2 = tlines[ss2+mw];
s3 = tlines[ss3+mw];
f01 = s0 ^ s1  ;
f23 = s2 ^ s3  ;
f03 = s0 ^ s3  ;
f12 = s1 ^ s2  ;
```

```
/* XOR the 4 spin words, to pick up the spins at a single
time slice                                                    */

if(layer == 0){

/* the bottom of the lowest space loop plaquette is in the word
below, so we must patch the 2 spin words                      */

        eng_up0 = (s0^ (s0 >>1) )&s_mask[layer];
        eng_up1 = (s2^ (s2 >>1) )&s_mask[layer];
        /**  perform a rotate << 1  function  **/
        aux0 = ( (HIGH_1 & tlines[ss0+wr[nword+mw-1]]) >> 31 ) | (s0<< 1) ;
        aux1 = ( (HIGH_1 & tlines[ss2+wr[nword+mw-1]]) >> 31 ) | (s2<< 1) ;

/* XOR the spin word with the same word but shifted. This is not
necessary for the Heisenberg model due to the conservation law.
We now know what is the spin configuration.                   */

        eng_down0 = (s0 ^ aux0)&s_mask[layer];
        eng_down1 = (s2 ^ aux1)&s_mask[layer];
} else if(layer==3){

/* the top of the uppermost space loop plaquette is in the word
above, so we must patch the 2 spin words. The rest is the same
as above                                                      */

        /**  perform a rotate >> 1  function  **/
/*      aux0 = ( (1 & s0) << 31 ) | (s0>> 1) ; */
        aux0 = ( (1 & tlines[ss0+wr[mw+1]]) << 31 ) | (s0>> 1) ;
        aux1 = ( (1 & tlines[ss2+wr[mw+1]]) << 31 ) | (s2>> 1) ;
        eng_up0 = (s0^ aux0 )&s_mask[layer];
        eng_up1 = (s2^ aux1 )&s_mask[layer];
        eng_down0 = (s0 ^ (s0 <<1))&s_mask[layer];
        eng_down1 = (s2 ^ (s2 <<1))&s_mask[layer];
} else {
        eng_up0 = (s0^ (s0 >>1))&s_mask[layer];
        eng_up1 = (s2^ (s2 >>1))&s_mask[layer];
        eng_down0 = (s0 ^ (s0 <<1))&s_mask[layer];
        eng_down1 = (s2 ^ (s2 <<1))&s_mask[layer];
}

        eng_up0h = f01&s_mask[layer];
        eng_up1h = f23&s_mask[layer];
        eng_down0h = f03&s_mask[layer];
        eng_down1h = f12&s_mask[layer];

/* shift everything, so that the relevant bits are the lowest  */

if( layer > 0){
eng_up0   >>=  layer;
eng_up1   >>=  layer;
eng_down0 >>=  layer;
eng_down1 >>=  layer;
eng_up0h   >>=  layer;
eng_up1h   >>=  layer;
eng_down0h >>=  layer;
eng_down1h >>=  layer;
}

eng_up0   |=  (eng_up0h << 1);
eng_up1   |=  (eng_up1h << 1);
eng_down0   |=  (eng_down0h << 1);
eng_down1   |=  (eng_down1h << 1);

/* get the energies                                          */
```

```
flpmsk = 0;
for(i=0;i<32;i+=4){
/* get the relevant bits and fetch transition probability   */
    mask = bits[layer + i];
    delta  = s_prob[(eng_up0 >> i) & LOW_2]*s_prob[(eng_up1 >> i) & LOW_2] ;
    delta *= s_prob[(eng_down0 >>i)& LOW_2]*s_prob[(eng_down1 >>i)& LOW_2] ;
    if(randf() < delta)  flpmsk = flpmsk | mask;
}


/**  flip the 4 t-lines  **/
tlines[ss0+mw] = s0^ flpmsk ;
tlines[ss1+mw] = s1^ flpmsk ;
tlines[ss2+mw] = s2^ flpmsk ;
tlines[ss3+mw] = s3^ flpmsk ;

} /* mw loop (loop over all spin words) */

} /* end of spaceflipr() */
/*************************************************************
space_plaqs: Calculates the addresses of the spin words for
             space loop flips, as it sweeps through the lattice,
             and pases them to spaceflipr(), which does the updates.
*************************************************************************/

#include "spext.h"
#include "cros.h"
#include "ih.h"

/*************************************************************************
        even plaq:

                                        y
              3*--up---*2                       layer=0,2
               |       |                ^
              down    down              |
               |       |                |
              0*--up---*1               -------> x

*************************************************************************/

space_plaqs()
{
int x,y,y1;
int xfwd, cen, yfwd, xyfwd;
int in, len;

in = ny*nx*nword;
len = nx*nword*4;

/* layers 1 and 2 can be updated stictly locally   */
/*  layer=1 and 2  */
for(y=0;y<ny;y += 2 ){
y1 = y+1;
for(x=0;x<nx;x++){
        xfwd= nword*(fwdx[x] + y*nx) ;
        yfwd= nword*(x + y1*nx) ;
        xyfwd=nword*(fwdx[x] + y1*nx) ;
        cen = nword*(x + y*nx) ;
            if(x%2 == 0)
                space_flip(cen,yfwd,xyfwd,xfwd,1);
            else
                space_flip(cen,xfwd,xyfwd,yfwd,2);
}
}

/* layers 0 and 3 need boundary layers from neighbors, so first
```

```
call cshift                                                          */
/*  layer=0 and 3  */
cshift(tlines+in,nextproc[0][POS],len,tlines,nextproc[0][NEG],len);
cflush(tlines+in,len);
for(y=1;y<ny;y += 2 ){
y1 = y+1;
for(x=0;x<nx;x++){
        xfwd= nword*(fwdx[x] + y*nx) ;
        yfwd= nword*(x + y1*nx) ;
        xyfwd=nword*(fwdx[x] + y1*nx) ;
        cen = nword*(x + y*nx) ;
            if(x%2 == 0)
                space_flip(cen,xfwd,xyfwd,yfwd,0);
            else
                space_flip(cen,yfwd,xyfwd,xfwd,3);
}
}

/* now return the updated spin layer to the neighbor                  */
cshift(tlines,nextproc[0][NEG],len,tlines+in,nextproc[0][POS],len);
cflush(tlines,len);


}
```

```
/*************************************************************
time_flipr: Time loop updates performed by this routine. The
            addresses of the 6 spin words involved in the flip
            supplied by timeplaq().
*************************************************************/

#include "spext.h"

/****      Mtrotter = nword*8  ******/

/*
Energy information is contained in 2 words, which are then shifted
and their 4 lowest order bits are read to decide if flip or not.
*/
/*****************************************************************
        Layer=0
                plaqs in y direction, both even and odd ones.
        Layer=1
                plaqs in x direction, both even and odd ones.
                boundary condition(z) for the upper-most one.
        Layer=2
                plaqs in y direction, both even and odd ones.
                boundary condition(z) for the upper-most one.
        Layer=3
                plaqs in x direction, both even and odd ones.
                boundary condition(z) for the upper-most one.
*****************************************************************/

/********       even plaq:
                                        y                       x
                *       *                 layer=0                 layer=1
                down1   down2           ^                       ^
                                        |                       |
        *left   *cen1   *cen2   *right  -------> x      y <-----

                up1     up2
                *       *

 Odd plaq: exchange down <=> up

**********/

/*****************
1st do 0 2 4 6, then do 1 3 5 7 (all on the same t-line).

          +++++++       most sig bit
          +     +
          +     +               ^
        3 ----- 4               |
          |     | upper
          |     |
    +++ 2 ----- 5 +++
    +     |     |     +
    +     |     |     +
    +++ 1 ----- 6 +++
          |     |               |
          |     | down          V
        0 ----- 7
        cen1  cen2      least sig bit
          +     +
          +++++++

e3    = top int_plaq
e4    = bottom int_plaq
e0, e7= two down plaqs
e1, e6= two middle plaqs
```

```
eng3 = (cen1[mw] ^ aux0) & mask_hi[layer] ;
eng4 = (cen1[mw] ^ aux1) & mask_dn[layer] ;
eng4 = (eng4 << 3) | (HIGH_3 & eng4) >> 29 ;

tot_eng1 = eng012[mw] | eng3;
tot_eng2 = eng567[mw] | eng4;

start=0;
flpmsk = 0;

for(i=start;i<32;i+=8)
{
        li = layer + i;
        mask = t_mask[li];
        if((flag&mask)== 0)
        if((cen1[mw]&mask) == 0 || (cen2[mw]&mask) == 0) {
              lft = (tot_eng1 >> li) & LOW_4;
              rght = (tot_eng2 >> li) & LOW_4;

/*lft and rght contain information about the energies, just use
them as indices to the look-up table.                          */

              delta = t_prob[lft][rght];
              if(randf() < delta) {
                flpmsk = flpmsk | mask;
                taccpt ++;
              }
        }
}

        /**  flip the 0 2 4 6 time_plaqs  **/
cen1[mw] = cen1[mw] ^ flpmsk;
cen2[mw] = cen2[mw] ^ flpmsk;


/**********    do 1 3 5 7 (all on the same t-line).  *************/
/**  Only eng3 and eng4 needs to be recomputed,  since the plaqs
might have been flipped                                        **/

aux0 = (1 & cen1[wr[mw+1]]) << 31  | (cen1[mw] >> 1)  ;
aux1 = (HIGH_1 & cen1[wr[nword+mw-1]]) >> 31 | (cen1[mw] << 1)  ;
eng3 = (cen1[mw] ^ aux0) & mask_hi[layer] ;
eng4 = (cen1[mw] ^ aux1) & mask_dn[layer] ;
eng4 = (eng4 << 3) | (HIGH_3 & eng4) >> 29 ; /* to align to eng3 only */

tot_eng1 = eng012[mw] | eng3;
tot_eng2 = eng567[mw] | eng4;

start=4;
flpmsk=0;

for(i=start;i<32;i+=8) {
        li = layer + i;
        mask = t_mask[li];
        if( (flag&mask) == 0)
        if( (cen1[mw]&mask) == 0 || (cen2[mw]&mask) == 0) {
            lft = (tot_eng1 >> li) & LOW_4;
            rght = (tot_eng2 >> li) & LOW_4;
            delta = t_prob[lft][rght];
            if(randf() < delta) {
                flpmsk = flpmsk | mask;
                taccpt ++ ;
            }
        }
}
```

```
e2, e5= two upper plaqs

***************/
int time_flipr(left,cp1,cp2,right,down1,down2,up1,up2,layer)
int left, cp1, cp2, right, down1, down2, up1, up2, layer;
{
unsigned int eng0,eng1,eng2,eng3,eng4,eng5,eng6,eng7;
unsigned int flag,aux0,aux1,flpmsk, mask, low, high, back;
unsigned int cen1[MXword], cen2[MXword], e012[MXword], e567[MXword];
unsigned int eng012[MXword], eng567[MXword], tot_eng1, tot_eng2, lft, rght;
int mw, lyr, layer1, start, li, i, wr1;
float delta,r1;

layer1 = layer ;

/* First compute eng012 and eng567  */
/* These are the energy contributions coming from the neighbors
of the 2 spin worfs which are being flipped. These energies are
useful for both even and odd time loops.                        */
/* Just XOR the words and pick up the appropriate bits by masking*/

for(mw=0;mw<nword;mw++){
  eng0 = (tlines[cp1+mw] ^ tlines[down1+mw]) & mask_dn[layer] ;
  eng1 = (tlines[cp1+mw] ^ tlines[left+mw])  & mask_md[layer]  ;
  eng2 = (tlines[cp1+mw] ^ tlines[up1+mw])   & mask_up[layer]  ;
  eng5 = (tlines[cp2+mw] ^ tlines[up2+mw])   & mask_up[layer]   ;
  eng6 = (tlines[cp2+mw] ^ tlines[right+mw]) & mask_md[layer] ;
  eng7 = (tlines[cp2+mw] ^ tlines[down2+mw]) & mask_dn[layer];
  e012[mw] = eng0 | eng1 |eng2;
  e567[mw] = eng5 | eng6 |eng7;
}

/*  Rotate eng012, cen1 etc. for different layers to layer=0 position  */

lyr=layer;
back = 32 - lyr;
if(lyr > 0){
    low = low_bits[lyr];
    for(mw=0;mw<nword;mw++) {
        wr1 = wr[mw+1] ;
        eng012[mw] = e012[mw] >> lyr | e012[wr1] << back;
        eng567[mw] = e567[mw] >> lyr | e567[wr1] << back;
        cen1[mw]= tlines[cp1+mw]>>lyr | tlines[cp1+wr1] <<back ;
        cen2[mw]= tlines[cp2+mw]>>lyr | tlines[cp2+wr1] <<back ;
    }
} else for(mw=0;mw<nword;mw++) {
        eng012[mw] = e012[mw];
        eng567[mw] = e567[mw];
        cen1[mw]= tlines[cp1+mw];
        cen2[mw]= tlines[cp2+mw];
}

layer=0;        /* now layer=0, because already shifted eng012,cen1 */
                /* loop over multiple words of a single trotter line */
for(mw=0;mw<nword;mw++){

flag = ~(cen1[mw] ^ cen2[mw]) ;

/* First do 0th 2nd 4th 6th time-plaqs, then do odd ones */

/* To calculate contributions from plaquettes within the 2 words
cen1 and cen2, need to shift them and then XOR. When shifting, must
pickup the bits residing in words above and below.              */

aux0 = (1 & cen1[wr[mw+1]]) << 31  | (cen1[mw] >> 1)  ;
aux1 = (HIGH_1 & cen1[wr[nword+mw-1]]) >> 31 | (cen1[mw] << 1)  ;
```

```
        /**  flip the 1 3 5 7 time_plaqs and put back in t-lines **/
 cen1[mw] = cen1[mw] ^ flpmsk;
 cen2[mw] = cen2[mw] ^ flpmsk;
 } /* mw loop */

 if(lyr > 0){
     high = high_bits[lyr];
     for(mw=0;mw<nword;mw++) {
         wr1 = wr[nword+mw-1];
         tlines[cp1+mw]= cen1[mw]<<lyr | cen1[wr1] >>back;
         tlines[cp2+mw]= cen2[mw]<<lyr | cen2[wr1] >>back;
     }
 } else for(mw=0;mw<nword;mw++) {
         tlines[cp1+mw]= cen1[mw];
         tlines[cp2+mw]= cen2[mw];
 }

 } /* end of timeflipr() */
 /*************************************************************
 time_plaqs: Calculates adresses of spin words involved in time
             loop flips. Sweeps through the lattice, passing
             the adresses to the routine timeflipr(), which does
             the updates.
 *************************************************************/
 /*************************************************************
         Layer=0
                 plaqs in y direction, both even and odd ones.
         Layer=1
                 plaqs in x direction, both even and odd ones.
                 boundary condition(z) for the upper-most one.
         Layer=2
                 plaqs in y direction, both even and odd ones.
                 boundary condition(z) for the upper-most one.
         Layer=3
                 plaqs in x direction, both even and odd ones.
                 boundary condition(z) for the upper-most one.
 *************************************************************/

 /********       even plaq:

                  *       *                  y                          x
                  down1   down2                layer=0                    layer=1
                                               ^                          ^
                                               |                          |
         *left    *cen1   *cen2   *right       -------> x       y <-----

                  up1     up2
                  *       *

  Odd plaq: exchange down <=> up

 **********/


 /****************


  |         |   |   |   |   |         |   |         |                |
  |         |   |   |   |   |         |   |         |                |
  |         |   |   |   |   |         |   |         |                |
  |         |   |   |   |   |         |   |         |                |

 spc_left0 0   1   2   3   4   ...   6   7   space_right0   space_right1
           |<-----   local   ------>|

 *****************/
```

```
#include "spext.h"
#include "cros.h"
#include "ih.h"
time_plaqs()
{
int xeven,yeven,layer,left,right,down1,down2,up1,up2;
int cp1, cp2;
int x,x1,x00,y00,y,y0,y1,y2,len,space_left0,space_right0,space_right1;
int in1,in2,out1,out2;

len = nx*nword*4;

/* these are the memory locations for the boundary layers from
the node to the left and to the right                          */
space_right0 = ny ;
space_right1 = ny+1 ;
space_left0  = ny+2;

layer=0;
y00=1;
xeven=0;
/* find out the addresses of spin words which share time loops
that start on layer 0                                          */
for(y=y00;y<ny;y +=2){
    y1=y+1;
    y2=y+2;
    if( y1 == ny ) {
/* pass tlines from right node to left */
        in1 = nword*y1*nx;
        in2 = nword*y2*nx;
        out1 = nword*0*nx;
        out2 = nword*1*nx;
        cshift(&tlines[in1],nextproc[0][POS],len,
               &tlines[out1],nextproc[0][NEG],len);
        cshift(&tlines[in2],nextproc[0][POS],len,
               &tlines[out2],nextproc[0][NEG],len);
        cflush(&tlines[in1],len);
        cflush(&tlines[in2],len);
    }
    for(x=0;x<nx;x++){
        right= nword*(x + y2*nx);
        left = nword*(x + bcky[y]*nx);
        if(x%2 == xeven){ /** even plaq  **/
            down1 = nword*(fwdx[x] + y*nx);
            down2 = nword*(fwdx[x] + y1*nx);
            up1 = nword*(bckx[x] + y*nx);
            up2 = nword*(bckx[x] + y1*nx);
        } else {
            up1 = nword*(fwdx[x] + y*nx);
            up2 = nword*(fwdx[x] + y1*nx);
            down1 = nword*(bckx[x] + y*nx);
            down2 = nword*(bckx[x] + y1*nx);
        }
        cp1=nword*(x+y*nx);
        cp2=nword*(x+y1*nx);
        time_flipr(left,cp1,cp2,right,down1,down2,up1,up2,layer);
    }
}

layer=1;
x00=0;
yeven=0;
/* find out the addresses of spin words which share time loops
that start on layer 1                                          */
for(y = ny-1; y >= 0 ; y--){
```

```
    y1=y+1;
    y0=y-1;
    if(y == ny-1 ){
        y1=space_right0;
        /* because the plane is in the left node in layer 0  */
    } else if (y == 1) {
        /* put y=0 plane back, from left to right  */
        out1 = space_right0*nx*nword;
        in1 = 0*nx*nword;
        cshift(&tlines[in1],nextproc[0][NEG],len,
               &tlines[out1],nextproc[0][POS],len);
        cflush(&tlines[in1],len);
    } else if (y == 0) {
        /* because y0 is -1 if y==0. So y==0 is different from y==1 */
        y0 = space_left0 ;
        /* copy y=ny-1 plane from left to right node */
        out1 = (ny-1)*nx*nword;
        in1 = y0*nx*nword;
        cshift(&tlines[in1],nextproc[0][NEG],len,
               &tlines[out1],nextproc[0][POS],len);
        cflush(&tlines[in1],len);
    }
for(x=x00;x<nx;x+=2){
    x1=fwdx[x];
/*  for(y=0;y<ny;y++){  */
        left = nword*(bckx[x] + y*nx);
        right = nword*(fwdx[x1] + y*nx);
        if(y%2 == yeven){ /** even plaq  **/
            down1 = nword*(x + y1*nx);
            down2 = nword*(x1 + y1*nx);
            up1 = nword*(x + y0*nx);
            up2 = nword*(x1 + y0*nx);
        } else {
            up1 = nword*(x + y1*nx);
            up2 = nword*(x1 + y1*nx);
            down1 = nword*(x + y0*nx);
            down2 = nword*(x1 + y0*nx);
        }
        cp1=nword*(x+y*nx);
        cp2=nword*(x1+y*nx);
        time_flipr(left,cp1,cp2,right,down1,down2,up1,up2,layer);
    }
}

layer=2;
y00=0;
xeven=1;
/* find out the addresses of spin words which share time loops
that start on layer 2                                        */
for(y=y00;y<ny;y +=2){
    y0=y-1;
    y1=y+1;
    y2=y+2;
    if(y == 0 ){
        y0=space_left0;
    } else if( y2 == ny) {
        y2=space_right0 ;
        /* pass tlines from right node to left */
        in1   = y2*nx*nword;
        out1  = 0*nx*nword;
        cshift(&tlines[in1],nextproc[0][POS],len,
               &tlines[out1],nextproc[0][NEG],len);
        cflush(&tlines[in1],len);
    }
    for(x=0;x<nx;x++){
        right = nword*(x + y2*nx);
```

```
        left = nword*(x + y0*nx);
        if(x%2 == xeven){ /** even plaq  **/
            down1 = nword*(fwdx[x] + y*nx);
            down2 = nword*(fwdx[x] + y1*nx);
            up1 = nword*(bckx[x] + y*nx);
            up2 = nword*(bckx[x] + y1*nx);
        } else {
            up1 = nword*(fwdx[x] + y*nx);
            up2 = nword*(fwdx[x] + y1*nx);
            down1 = nword*(bckx[x] + y*nx);
            down2 = nword*(bckx[x] + y1*nx);
        }
        cp1=nword*(x+y*nx);
        cp2=nword*(x+y1*nx);
        time_flipr(left,cp1,cp2,right,down1,down2,up1,up2,layer);
    }
}

layer=3;
x00=1;
yeven=1;
/* find out the addresses of spin words which share time loops
that start on layer 3                                          */
for(y=ny-1;y>=0;y--){
    y1=y+1;
    y0=y-1;
    if(y == ny-1 ){
        y1=space_right0;
     /* because the plane is in the left node in layer 0  */
    } else if (y == 0) {
        y0 = space_left0 ;
        /* copy y=ny-1 plane from left to right  */
        out1 = (ny-1)*nx*nword;
        in1  = y0*nx*nword;
        cshift(&tlines[in1],nextproc[0][NEG],len,
               &tlines[out1],nextproc[0][POS],len);
        cflush(&tlines[in1],len);
    }
  for(x=x00;x<nx;x+=2){
    x1=fwdx[x];
        left = nword*(bckx[x] + y*nx);
        right= nword*(fwdx[x1] + y*nx);
        if(y%2 == yeven){ /** even plaq  **/
            down1 = nword*(x + y1*nx);
            down2 = nword*(x1+ y1*nx);
            up1   = nword*(x + y0*nx);
            up2   = nword*(x1+ y0*nx);
        } else {
            up1   = nword*(x + y1*nx);
            up2   = nword*(x1+ y1*nx);
            down1 = nword*(x + y0*nx);
            down2 = nword*(x1+ y0*nx);
        }
        cp1=nword*(x+y*nx);
        cp2=nword*(x1+y*nx);
        time_flipr(left,cp1,cp2,right,down1,down2,up1,up2,layer);
  }
}

}
```

```
/*************************************************************
weightsv: Calculates transition probabilities for elementary
          flips in the XY model in Sy representation. It also
          calculates tables which give matrix elements for
          the operators U1U2 and U3U4, needed for vorticity
          calculations. It is called in init(). The indices
          to the tables u1u2, s1s3, s1s3u1u2, s0s1s2s3 and
          s0s1s2s3u1u2 are calculated in meas_vortx().
*************************************************************/
#include "spext.h"
#include <math.h>


weightsv(J1)
float J1;
{
float exchk,sinhk,coshk,expk;
float u1,u2,a,b,c,d,sigma13,sigma0123;
float u1u2[16][16],s1s3[16][16],s1s3u1u2[16][16];
float s0s1s2s3[16][16],s0s1s2s3u1u2[16][16];
float uu1[16][16], uu2[16][16];
unsigned int i,j,k,m,i1,j1,k1,m1,down,up;

exchk = 1./(4.*(nword*8)*temp);
sinhk = sinh(exchk);
coshk = cosh(exchk);
expk = exp(exchk);

a=expk*coshk;
b=expk*sinhk;
c=coshk/expk;
d=sinhk/expk;

for(i=0;i<=1;i++)
 for(j=0;j<=1;j++)
  for(k=0;k<=1;k++)
   for(m=0;m<=1;m++)
    for(i1=0;i1<=1;i1++)
     for(j1=0;j1<=1;j1++)
      for(k1=0;k1<=1;k1++)
       for(m1=0;m1<=1;m1++) {
        down = 15 - ((m << 3) | (k << 2) | (j << 1) | i);
        up = 15 - ((m1 << 3) | (k1 << 2) | (j1 << 1) | i1);

        if((i == i1) && (j == j1)) {
             if(i == j) u2=a;
             else       u2=c;
             goto P2U2;
        }

        if((i != i1) && (j != j1)) {
             if(i == j) u2=b;
             else       u2=d;
             goto P2U2;
        }

        u2=0.;
        goto P1U1;

P2U2:

        if((k == k1) && (m == m1)) {
             if(k == m) u2 *= a;
             else       u2 *= c;
             goto P1U1;
        }
```

```
        if((k != k1) && (m != m1)) {
             if(k == m) u2 *= b;
             else       u2 *= d;
             goto P1U1;
        }

        u2=0.;


P1U1:

        if((j == j1) && (k == k1)) {
             if(j == k) u1 = a;
             else       u1 = c;
             goto P2U1;
        }

        if((j != j1) && (k != k1)) {
             if(j == k) u1 = b;
             else       u1 = d;
             goto P2U1;
        }

        u1=0.;
        goto S1S3;


P2U1:

        if((m == m1) && (i == i1)) {
             if(m == i) u1 *= a;
             else       u1 *= c;
             goto S1S3;
        }

        if((m != m1) && (i != i1)) {
             if(m == i) u1 *= b;
             else       u1 *= d;
             goto S1S3;
        }

        u1=0.;


S1S3:

        if((i == i1) && (k == k1) && (j != j1) && (m != m1)) sigma13=1.;
        else sigma13=0.;

        if((i != i1) && (k != k1) && (j != j1) && (m != m1)) sigma0123=1.;
        else sigma0123=0.;

        uu1[down][up] = u1;
        uu2[down][up] = u2;
        s1s3[down][up] = sigma13;
        s0s1s2s3[down][up] = sigma0123;

      }

for(i=0;i<16;i++)
 for(j=0;j<16;j++) {
 u1u2[i][j] = 0.;
 for(k=0;k<16;k++) u1u2[i][j] += uu1[i][k] * uu2[k][j];
 }
```

```
for(i=0;i<16;i++)
 for(j=0;j<16;j++) {
  s1s3u1u2[i][j] = 0.;
  s0s1s2s3u1u2[i][j] = 0.;

  for(k=0;k<16;k++) s1s3u1u2[i][j] += s1s3[i][k] * u1u2[k][j];
  for(k=0;k<16;k++) s0s1s2s3u1u2[i][j] += s0s1s2s3[i][k] * u1u2[k][j];

  if(s1s3u1u2[i][j] == 0.) x1x3[i][j]=0.;
  else x1x3[i][j] = s1s3u1u2[i][j]/u1u2[i][j];

  if(s0s1s2s3u1u2[i][j] == 0.) x0x1x2x3[i][j]=0.;
  else x0x1x2x3[i][j] = s0s1s2s3u1u2[i][j]/u1u2[i][j];
 }


z0z2[0] = 1.;
z0z2[1] = -1.;

}

/*****************************************************************
prob_tab: Calculates transition probabilities for all possible
          time,space, and glob flip updates. Called at the
          beginning of a run by init().
*****************************************************************/

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


prob_tab(f20,f32)
float f20,f32;
{
 unsigned int shift;
 int delta1,delta2,j,k;

/* This is a table of probabilities for time and space flips.
   There are 5 possible space flip configurations and 16 * 16
   possible time flip configurations, stored in s_prob[5] and
   t_prob[16][16]                                              */

/* Numbers 0,1,2,3,4, which are indices to s_prob[], are
obtained in bit manipulations performed by spaceflipr()    */

 s_prob[0] = exp((-4.) * f32);
 s_prob[1] = exp((-2.) * f32);
 s_prob[2] = 1.0;
 s_prob[3] = exp(2. * f32);
 s_prob[4] = exp(4. * f32);

/* Numbers j,k, which are indices to t_prob[][], are
obtained in bit manipulations performed by timeflipr()    */

 for(j=0; j<16; j++)
 {
  for(k=0; k<16; k++)
  {
   delta1 = 0;
   delta2 = 0;

  for(shift = 0;shift < 3; shift++) {
   if( ((j >> shift) & 1) == 1)
                delta1++;
   else
                delta1--;
```

```
    if( ((k >> shift) & 1) == 1)
                    delta1++;
    else
                    delta1--;
   }

    if(((j >> 3) & 1) == 1)
                    delta2++;
    else
                    delta2--;

    if(((k >>3) & 1) == 1)
                    delta2++;
    else
                    delta2--;

    t_prob[j][k] = exp(delta1*f20 + delta2*f32);
   }
  }

/* to calculate the probability of a glob flip in time dir,
one needs the number of antiferromagnetic plaquettes, atteched
to a particular string of tlines above a lattice site. This is j
and is obtained by globflipr()                                  */

 for(j=0;j<=nword*32;j++)  g_prob[j] = exp((double)(j-nword*16)*2.0*f20);

}
```

```
/*************************************************************
spmain_wtk: node program, running on the WEITEK
*************************************************************/


#include <math.h>
#include <cros.h>
#include "ih.h"
#include "spdef.h"


int ack, param[10],cmd ;    /* Subroutine parameters */
int time1,time2 ;
int no_op() ;
int combelt1();

spmain_wtk()
{

        int i,j;
/* get cube environment */
        cros3hack();

/* server loop, complemented by the request loop in the host
program                                                      */

while(1) {

/* get the function to be performed */

    bcastelt(&cmd,sizeof(int)) ;
    cflush(&cmd,sizeof(int));
    ack = -cmd ;
    printf(" cmd = %d \n",cmd);
    if(procnum == 0) cwrite(&ack,cpmask,4);

    switch(cmd) {
        case SEED:

/* initialize random number generator  */

                bcastelt(&iseed,sizeof(int)) ;
                cflush(&iseed,4);
                rm    = iseed ;
                ack = -iseed;
                dumpelt(&ack,4);
                fib_init(procnum,iseed);
                break;

/* get the parameters of the run            */

        case SETPAR:
                recpar() ;
                break ;

/* initialize the spin configuration,
generate masks, probabilities and 2D grid  */

        case INITIALIZE:
                bcastelt(param,sizeof(int)) ;
                cflush(param,4);
                init_flag = param[0] ;
                ack = - param[0] ;
                dumpelt(&ack,4);
                init(init_flag) ;
        /*      set_dist();     */
```

```
                break ;

/* perform updates and measurements          */

        case UPDATE:
                time1 = clock();
                update();
                time2 = clock();
                time2 = (time2-time1)/1000;
                dumpelt(&time2,sizeof(int));
                break;

/* perform a single measurement, no updates  */

        case MEASURE:
                measure();
                break;

/* dump the spin system to the host          */

        case DUMPLAT:
                dumplat();
                break;

        case QUIT:
                exit(1) ;

        default :
                break ;
    } /*  switch  */
}
exit(1);
} /* main routine */

/* just for testing purposes     */

int (*Dabort)();
ccabort()
{
calldp(Dabort,0);
}


inc_or(a,b,size)
int *a,*b ;
int size ;
{
        *a |= *b ;
        return 0 ;
}
inc_and(a,b,size)
int *a,*b ;
int size ;
{
        *a &= *b ;
        return 0 ;
}

add_f(a,b,size)
float *a, *b ;
int size ;
{
        *a += *b ;
        return 1 ;
}
```

```
no_op(a,b,size)
int *a, *b ;
int size ;
{
        return 0 ;
}
/*************************************************************
This is the main program running in the host. It sends
commands to be performed by the nodes and serves the
requests coming from the nodes. It is in lockstep with
the complementary routine running in the nodes.
*************************************************************/

#include <cros.h>
#include <stdio.h>
#include "ih.h"
#include "ihspdef.h"

#define order(N) cmd = N ;
                 printf("brcast return %d\n",bcastcp(&cmd,4)) ;
                 printf(" ordered : N \n") ; chk(N) ;

main(argc,argv)
int argc;
char *argv[];
{
int cmd,param[10],i,j,iflg,pflag,mode;
char ctemp, measfile[40], qqqfile[40] ;
int  time0,time1,epoch() ;
int  msec[128],secmap[129];
float rate;
FILE *outfile;

printf("Dimension of cube is ?  \n");
scanf("%d",&doc); /* Dimension of cube    */
printf("doc = %d\n",doc);
cubeldl(doc,argv[1],NULLPTR);
printf(" Finished download \n");

totproc = 1<<doc;   /*  Total number of active nodes.*/
pflag = 0;      /*      output to terminal      */
while ((ctemp=getc(stdin)) != 'q') {
switch(ctemp) {
    case 'b':
        order(DUMPLAT) ;
        dumplat() ;
        break;
    case 'r':
        order(SEED) ;
        printf("INPUT seed : \n");
        scanf("%d",param);
        bcastcp(param,sizeof(int)) ;
        mdumpcp(bckbuf,sizeof(int),bufmap);
        for(i=0 ; i<totproc ; i++)
                if(bckbuf[i] != -param[0]) error_out("seed");
        break;
    case 'i':
        order(INITIALIZE) ;
        printf("INPUT(ferro_mag=0  Neel=1  random=2  old=3): \n");
        scanf("%d",param) ;
        bcastcp(param,sizeof(int)) ;
        mdumpcp(bckbuf,sizeof(int),bufmap);
        for(i=0 ; i<totproc ; i++)
                if(bckbuf[i] != -param[0]) error_out("init");
        if(param[0] == 3) loadlat();
        break;
```

```
      case 's':
          order(SETPAR) ;
          read_file("par.in");
          setpar() ;
          printf("Parameters set up.\n");
          fflush(stdout);
          break;
      case 'f':
          pflag = 1;       /*  output to files     */
          break;
      case 'u':
          order(UPDATE) ;
          time0 = epoch() ;
          update(pflag) ;
          time1 = epoch() ;
          elapsed(time0,time1) ;
          fflush(stdout);
          break;
      case 'm':
          order(MEASURE) ;
          measure_rec(pflag) ;
          fflush(stdout);
          break;
      case '?':

                  printf("Commands are:\n");
                  printf("=================================\n\n") ;
                  printf("q :              Quit\n") ;
                  printf("i :              Initialize\n");
                  printf("u :              Update \n");
                  printf("s :              Set parameters\n");
                  printf("r :              Set random number seeds\n");
                  printf("f :              Output into files\n");
                  printf("b :              Back up lattice \n");
          break;
      default:
          printf("INPUT (? for help) ");
}
}
/* Received quit signal..... */

order(QUIT) ;
fclose(outfile);
printf("computation completed.\n") ;
exit(0) ;
} /* main() ends */

chk(given)
int given ;
{
        int i,ack=555 ;
        i = combcp(&ack,4,1) ;
        printf("combcp return %d \n",i) ;
        if(given != -ack) printf("Sent %d  received %d\n",given,ack) ;
}

/*************************************************************
init_conf: Initialize the spin configuration. If flag = 0,
           initial configuration is ferromagnetic, flag=1
           is for Neel configuration, flag = 2 is for random
           initial configuration, and flag=3 is configuration
           from a previous run, kept in a file.
           There is also a version of this routine which performs
           different initializations for different processor
           rings.
*************************************************************/
```

```
#include "spext.h"
#include "stdio.h"
#include "cros.h"
#include "math.h"
extern int cycle[130], r_table[127];
int temporal[65536];

init_conf(flag)
int flag;
{
int x,y,mw;
 switch(flag)
 {
  case 0:       /*  ferro magnetic  */
    for(x = 0; x < nx; x++) for(y = 0; y < ny; y++)
    for(mw=0; mw< nword; mw++) tlines[nword*(x + y*nx)+mw] = 0;
    break;
  case 1:       /* Neel state  */
    for(x = 0; x < nx; x++) for(y = 0; y < ny; y++)
      if(((x + y) % 2) == 0)
        for(mw=0; mw< nword; mw++) tlines[nword*(x + y*nx)+mw] = 0;
      else
        for(mw=0; mw< nword; mw++) tlines[nword*(x + y*nx)+mw] = 0xffffffff;
    break;
  case 2:       /*  random  */
    for(x = 0; x < nx; x++) for(y = 0; y < ny; y++)
      if(randf() < 0.5)
        for(mw=0; mw< nword; mw++) tlines[nword*(x + y*nx)+mw] = 0;
      else
        for(mw=0; mw< nword; mw++) tlines[nword*(x + y*nx)+mw] = 0xffffffff;
    break;
  case 3:       /*  old configuration, read from a file via
                a call to loadelt()                          */
                loadelt(tlines,nx*ny*nword*4);
                cflush (tlines,nx*ny*nword*4);

                loadelt(&cycle[128],2*4);
                cflush (&cycle[128],2*4);

                loadelt(r_table,127*4);
                cflush (r_table,127*4);
    break;
  default:
        printf(" error in init!\n");
        exit(1);
    break;
 }

}

/*************************************************************
dumplat: dump the spin system onto host computer's file system
*************************************************************/
dumplat()
{
                dumpelt(tlines,nx*ny*nword*4);
                dumpelt(&cycle[128],2*4);
                dumpelt(r_table,127*4);
}

/*********************************************************
init_masks: For various purposes we have to pick up only
            certain bits from spin words and words resulting
            from operations on spin words. This is done using the
            masks defined here.
*********************************************************/
```

```
init_masks()
{
 int layer, i;

 for(i = 0; i < 32; i++) bits[i] = (1 << i);
 for(i = 0; i < 32; i++) low_bits[i] = 0;
 for(i = 1; i < 32; i++) low_bits[i] = low_bits[i-1] | bits[i-1];
/* low[0]=0,low[1]=1,low[2]=0x3, low[31]=0x7fffffff  */
 for(i = 0; i < 32; i++) high_bits[i] = 0;
 for(i = 1; i < 32; i++) high_bits[i] = high_bits[i-1] | bits[32-i];
 mask_dn[0] = 0x11111111;
 for(layer = 1; layer < 4; layer++) mask_dn[layer] = mask_dn[0] << layer;
 for(layer = 0; layer < 4; layer++)
 mask_md[layer] = ((0x80000000 & mask_dn[layer]) >> 31) |
                                  (mask_dn[layer] << 1);
 for(layer = 0; layer < 4; layer++)
 mask_up[layer] = ((0x80000000 & mask_md[layer]) >> 31) |
                                  (mask_md[layer] << 1);
 for(layer = 0; layer < 4; layer++)
 mask_hi[layer] = ((0x80000000 & mask_up[layer]) >> 31) |
                                  (mask_up[layer] << 1);
 t_mask[0] = 15;
 for(i = 1; i < 32; i++)
 t_mask[i] = ((0x80000000 & t_mask[i-1]) >> 31) | (t_mask[i-1] << 1);
 s_mask[0] = s_mask[1] = s_mask[2] = s_mask[3] = 0;
 for(i = 0; i < 32; i += 4)
  for(layer = 0; layer < 4; layer++)
            s_mask[layer] = s_mask[layer] | bits[layer + i];
}

/************************************************************
boundary: Defines vectors wich are to be used to properly
          handle periodic boundary conditions.
************************************************************/
boundary()
{
int i;
 for(i=0;i<nx;i++)fwdx[i]=i+1;
 for(i=0;i<6;i++) fwdx[nx-1+i] = i;
 /*  fwdx[nx-1] = 0;    fwd[fwd[i]] = fwd[i+1]  */

 for(i=0;i<nx;i++)bckx[i]=i-1;
 bckx[0] = nx-1;

 for(i=0;i<ny;i++)fwdy[i]=i+1;
 for(i=0;i<6;i++) fwdy[ny-1+i] = i;

 for(i=0;i<ny;i++)bcky[i]=i-1;
 bcky[0] = ny-1;

 for(i=0;i<32;i++) fwdt[i] = i+1 ;
 for(i=0;i<6;i++) fwdt[31+i] = i ;

 for(i=0;i<32;i++) bckt[i] = i-1 ;
 bckt[0] = 31 ;

 for(i=0;i<3*nword;i++) wr[i] = i%nword;

}

/************************************************************
kos: Defines a mask used to calculate time dependent correlation
     functions and the allowed wave vectors in the Brillouin zone.
```

```
*****************************************************************/
kos()
{
int i,j,k,b,bmax,imax,jmax;
float factor;
imax= 2*nx*(nx/2 + 1);
factor=6.28318530717959/(float)nx;
for(i=0;i<imax;i++) kosinus[i]=cos((float)i * factor);

bmax=16;
jmax = 1 << bmax;
for(j=0;j<jmax;j++) {
 k=0;
 for(b=0;b<bmax;b++) k += 1-2*((j>>b) & 1);
 temporal[j] = k;
}

}
/************************************************************
init: Called at the beginning of a run, after the node
      program and parameters are downloaded to the nodes.
      Performs all the necessary initializations, by calling
      appropriate routines.
************************************************************/
#include "spext.h"
#include "stdio.h"
#include "math.h"


init(flag)
int flag;
{
float J1,J3;

J3 = -1.0;
J1 = 1.0;
weights(J1,J3);

srand(iseed);
fib_init(procnum,iseed);

init_masks();
init_conf(flag);
boundary();

prob_tab(f20,f32);
kos();

}       /* end of init() */
/************************************************************
recpar: Receive the parameters of a run from the host. The
        version given here accomodates different parameter
        sets for different rings.
************************************************************/

#include "cros.h"
#include "spext.h"
#include <stdio.h>
#include <math.h>


extern int inc_or(), combelt1();
int err,bckbuf[100],param[100] ;

recpar()
{
```

```
        int cfg,f2,plen,i,eng_len,corr_len;

geometry:
/* Load the sizes of the lattices, the parameters of the
Cartesian grid, number of temperatures and initialization flag. */

        plen = 6;
        bcastelt(param,plen*sizeof(int) ) ;
        cflush(param,plen*sizeof(int));
        for(i=0;i<plen;i++) bckbuf[i] = -param[i];
        f2 = dumpelt(bckbuf,plen*sizeof(int)) ;

        nx      = param[0];
        ny      = param[1];
        ntemp   = param[2];
        init_flag= param[3];
        nprocs[0] = param[4];
        nprocs[1] = param[5];
/* configure the node system into a 2D grid                    */
        cfg=config();

/* load temperatures to the rings                               */
temperatures:
        plen = ntemp;
        bcastelt(param,plen*sizeof(int));
        cflush(param,plen*sizeof(int));
        for(i=0;i<plen;i++) bckbuf[i] = -param[i];
        f2 = dumpelt(bckbuf,plen*sizeof(int)) ;
        temp = ((float)param[recpnum[1]])/10000.;
        /*
        dumpelt(&temp,sizeof(float));
        dumpelt(recpnum,sizeof(int));
        dumpelt(recpnum+1,sizeof(int));
        */

/* load Trotter numbers to the rings                            */
Trotter:
        plen = ntemp;
        bcastelt(param,plen*sizeof(int));
        cflush(param,plen*sizeof(int));
        for(i=0;i<plen;i++) bckbuf[i] = -param[i];
        f2 = dumpelt(bckbuf,plen*sizeof(int)) ;
        nword = param[recpnum[1]];
        /*
        dumpelt(&nword,sizeof(int));
        dumpelt(recpnum,sizeof(int));
        dumpelt(recpnum+1,sizeof(int));
        */


config_test:
        plen=4;
        bckbuf[0] = cfg;
        bckbuf[1] = dim ;
        bckbuf[2] = recpnum[0];
        bckbuf[3] = recpnum[1];
        f2 = dumpelt(bckbuf,plen*sizeof(int)) ;



/* load the parameters of updates, like sweep number, seeds,
bin size, and how many bytes are dumped to the host when
the spin configuration is backed up                          */
updates:
        plen = 7;
        bcastelt(param,plen*sizeof(int) ) ;
```

```
        cflush(param,plen*sizeof(int));
        for(i=0;i<plen;i++) bckbuf[i] = -param[i];
        f2 = dumpelt(bckbuf,plen*sizeof(int)) ;

        ntemp  = param[0];
        iseed  = param[1];
        upflag = param[2];
        totswp = param[3];
        blksiz = param[4];
        mratio = param[5];
        wrtsiz = param[6];      /*  write out to host. in # of blocks  */

        nspins = nx*ny;

/*      init(init_flag);        */

ndir = 1;
/*
eng_len = 4*wrtsiz;
corr_len = ndir*wrtsiz*(nx/2 + 1);

if((e_data = (float *) malloc(4*eng_len)) == NULL) exit(1);
if((c_data = (float *) malloc(4*corr_len)) == NULL) exit(1);
*/
        return(0) ;
}

/*************************************************************
config: sets up and configures communication channel numbers.
*************************************************************/

#include "cros.h"
#include "spext.h"


int config()
{
int  error;
        perbc[0] = perbc[1] = 1 ; /* periodic boundaries */
        dim = -1;
        /* determine the dimensionality of the Cartesian grid     */
        if(nprocs[0] == totproc )
          dim=1;
        else
          if((nprocs[0]<totproc)&&(nprocs[0]*nprocs[1]==totproc))
                                    dim=2;

        error = whereami(procnum,nprocs,perbc,recpnum,nextproc,dim) ;
        /*    map onto a Cartesian grid of dimensionality dim      */

return(error) ;
}

cros3hack()
{
    struct cubenv ce;
    cparam(&ce);
    doc = ce.doc;
    procnum = ce.procnum;
    totproc = ce.nproc;
    cpmask = ce.cpmask;
}
/*************************************************************
whereami: Maps hypercube topology onto the data decomposition grid.
          Called in config().
*************************************************************/
```

```
#include <cros.h>
#include "spext.h"
#define ERR_1   -1
#define OK_1    0


int whereami(theprocnum,nprocs,perbc,recpnum,nextproc,dims)

int theprocnum, dims;
int nprocs[],perbc[],recpnum[] ;
int nextproc[][2] ;
{
   int i;
   int initerr,coorderr,chanlerr,chan2err;


   initerr =  ( gridinit(dims,nprocs) == ERR_1 ) ;

   coorderr =  ( gridcoord(theprocnum,recpnum) == ERR_1 ) ;

   for  (i=0 ; i<dims ; ++i) {
     chanlerr =  ( (nextproc[i][0] = gridchan(theprocnum,i,-1) )== ERR_1 ) ;
     chan2err =  ( (nextproc[i][1] = gridchan(theprocnum,i, 1) )== ERR_1 ) ;
   }


   if( initerr || coorderr || chanlerr || chan2err )
        return( ERR_1 );
   else
        return(OK_1);
}
```

```
/*****************************************************************
gather: gather lattice pieces distributed among nodes in a ring
        into a copy of the whole lattice in each node. Necessary
        step before a global winding number update or correlation
        function measurement.
*****************************************************************/
#include "spext.h"
#include "cros.h"
#include "ih.h"

gather(buf,dim)
int buf[],dim;
/*  buf should be in DRAM  */
{
int in,out,len0,len1,len2,len3,len4;
int Nx,coord0,coord1;

len0 = nx*nword;
Nx=ny*nprocs[1];
len1 = Nx*ny*nword;
len2 = nx*nword*4;
len3 = Nx*nword*4;
len4 = 4*len1;

if(dim == 1) {
 if(dim == 2) {
/* for 2D decomposition, first exchange spin words along x-direction in
data space. A processor writes a chunk of data to the one below it, and
reads another chunk into the memory location next to to the location of
the written out chunk, from the node above it. Then the memory locations
are shifted again, and the same communication procedure is repeated   */
   for(coord1=1;coord1<nprocs[1];coord1++) {
     in = len0*coord1;
     out = len0*(coord1-1);
     vshift(buf+in,nextproc[1][POS],len2,len3,ny,buf+out,
            nextproc[1][NEG],len2,len3,ny);
   }
   cflush(buf,len4);
 }

/* After the exchange in x-direction, the data are exactly in the form
required for the ring decomposition in y-direction, i.e each processor
has a stripe of the underlying two dimensional lattice. This is the part
that is actually used in the computation since our decomposition is 1D.
The procedure is exactly the same as for the exchange in x-direction.
The only difference is that vshift is not necessary, since the
communicated data occupy contigous memory locations.                */

 for(coord0=1;coord0<nprocs[0];coord0++) {
       in  = len1*coord0;
       out = len1*(coord0-1);
       cshift(buf+in,nextproc[0][POS],len4,buf+out,nextproc[0][NEG],len4);
 }
 cflush(buf,len4);
}

/*****************************************************************
scatter: the inverse of gather. Distribute the lattice to all other
         nodes from node 0.
*****************************************************************/
scatter(buf,dim)
int buf[], dim;
/*  buf should be in DRAM  */
{
int r_len, in, out ,w_len, dist_0, dist_1;
int len0, len1, len2, len3, len4, Nx;
```

```
dist_0 = recpnum[0];
dist_1 = recpnum[1];
in = 0;
out = len1;

/* Source node is assumed to be 0. It will first send stripes of
lattice along direction 0. It will send nprocs[0]-1 stripes(chunks) to
its neighbor, processor with coordinate 1 in 0-direction. This processor
will send nprocs[0]-2 stripes to its neighbor with coordinate 2, and
so on, until the processor with largest 0-coordinate receives only
1 stripe. If the decomposition is the ring decomposition, this is
enough.                                                              */
w_len= (nprocs[0]-dist_0-1)*len4;
r_len=  (nprocs[0]-dist_0)*len4;
len0 = nx*nword;
Nx=ny*nprocs[1];
len1 = Nx*ny*nword;
len2 = nx*nword*4;
len3 = Nx*nword*4;
len4 = 4*len1;

if( dist_0 == 0 ) {
        cwrite(buf+out,nextproc[0][POS],w_len);
} else if( dist_0 == (nprocs[0] - 1)) {
        cread(buf+in,nextproc[0][NEG],0,r_len);
} else {
        cread(buf+in,nextproc[0][NEG],0,r_len);
        cwrite(buf+out,nextproc[0][POS],w_len);
}
cflush(buf,len4*nprocs[0]);

/* For a 2D decomposition, only processors with coordinate 0 in
direction 1, are updated. They will now pass chunks of the lattice
along direction 1 to other processors. Now each processor with
coordinate 0 in direction 1 is a source. The procedure is the same
as for direction 0, but we must use vshift since the memory locations
are not consecutive. A processor with coordinate 1 in direction 1 will
receive nprocs[1]-1 chunks (square patches of the lattice), while the
last one along direction 1 receives only its own patch.              */
if(dim == 2) {
   in = 0;
   out = len0;
   w_len= (nprocs[1]-dist_1-1)*len0;
   r_len=  (nprocs[1]-dist_1)*len0;

   if( dist_1 == 0 ) {
        vwrite(buf+out,nextproc[1][POS],4*w_len,len3,ny);
   } else if( dist_1 == (nprocs[1] - 1)) {
           vread(buf+in,nextproc[1][NEG],0,4*r_len,len3,ny);
   } else {
           vread(buf+in,nextproc[1][NEG],0,4*r_len,len3,ny);
           vwrite(buf+out,nextproc[1][POS],4*w_len,len3,ny);
   }
   cflush(buf,len4);
}
}
```

```
/*************************************************************
update: This routine initiates MC sweeps, calls all the necessary
        routines, and periodically dumps the results to the
        host. It is complemented by a server routine in the
        host. The version shown here involves measurements
        of vortex density, thermodynamics and static correlations.
        Very similar versions handle dynamic
        correlations and spin stiffness as well.
*************************************************************/
#include "spext.h"
#include "stdio.h"
float cbuf[10];

update()
{
register int sweep,i,k,j,ptr,c_ptr,max;
int dim0,dim1;
int xcorr[MXnx],ycorr[MXnx];
float xcr[MXnx],  ycr[MXnx];
float en,en2,mag,mag2,su,sg,smagntz,factor;
float vfactor,blknorm,nx2,stm1,stmag,stmag2;
float v13,v24,v1234;

/* needed for combine routines   */
 dim0 = 0;
 dim1 = log2(nprocs[0]);
/* various normalization factors */
 factor = 1./(float)(4*4*nx*nx*nword*blksiz);
 vfactor=1./(float)(nx*nx);
 blknorm = 1./(float)blksiz;
 nx2 = (float)(nx*nx);
 max = nx/2;
/* initialize cumulative data variables */
 en=0.; en2=0.;
 mag=0.; mag2=0.;
 stmag=0.; stmag2=0.;
 ptr = 0; c_ptr = 0;
 v13 = 0.;
 v24 = 0.;
 v1234 = 0.;

for(j=0;j<=max;j++) xcr[j] = 0.;
/* needed for acceptance statistics   */
taccpt = 0;  taacc = 0.0 ;
saccpt = 0;  saacc = 0.0 ;
gaccpt = 0;  gaacc = 0.0 ;
waccpt = 0;  waacc = 0.0 ;

for(sweep=1;sweep<=totswp;sweep++) {

/* upflag says which types of updates to perform */
  if((upflag&M_TIME) == M_TIME ) time_plaqs();
  if((upflag&M_SPACE)== M_SPACE) space_plaqs();
  if((upflag&M_GLOB) == M_GLOB ) glob_lines();

/* get the acceptance statistics                  */
  accpt_update();

/* measure every mratio sweeps                    */

  if(sweep%mratio == 0){
    plaqcnt(cbuf,&cbuf[4]);
    stag_mag(&cbuf[6]);
    vorticity(&cbuf[7]);

    combinesub(cbuf,10,dim0,dim1);
```

```
        su = (cbuf[0]*e0+cbuf[1]*e1+ cbuf[2]*e2+cbuf[3]*e3)/nx2;
        sg = (cbuf[0]*g0+cbuf[1]*g1+ cbuf[2]*g2+cbuf[3]*g3)/(nx2*nx2);
        en   += su;
        en2  += su*su - sg;
        smagntz = (cbuf[4] - cbuf[5])/(32.0*nword*nx2);
        mag  += smagntz;
        mag2 += smagntz*smagntz*nx2/temp;

        stm1 = (cbuf[6])/(4.0*8.*nword*nx2);
        stmag  += stm1;
        stmag2 += stm1*stm1*nx2/temp;

        v13 += cbuf[7];
        v24 += cbuf[8];
        v1234 += cbuf[9];

    /* gather pieces of lattice for C(r) measurement                  */
        gather(tlines,nword*nspins);
        corr_meas(xcorr,ycorr);

    /* winding update, and then scatter from node 0 */
        if((upflag&M_WIND) == M_WIND ) {
            wind_lines();
            scatter(tlines,nword*nspins,0);
            accpt_update();
        }

        for(j=0;j <=max ;j++) {
            xcr[j] += 0.5*(float)(xcorr[j]+ycorr[j]) ;
        }


    /* accumulate measurements into a single bin  */
        if(sweep%(blksiz*mratio) == 0) {
            e_data[ptr+0] = en*blknorm;
            e_data[ptr+1] = en2*blknorm;
            e_data[ptr+2] = mag*blknorm;
            e_data[ptr+3] = mag2*blknorm;
            e_data[ptr+4] = stmag*blknorm;
            e_data[ptr+5] = stmag2*blknorm;
            e_data[ptr+6] = v13*blknorm*vfactor;
            e_data[ptr+7] = v24*blknorm*vfactor;
            e_data[ptr+8] = v1234*blknorm*vfactor;
            for(j=0;j <=max ;j++) {
                    c_data[c_ptr++] = xcr[j]*factor;
                    xcr[j] = 0.0;
            }

            ptr += 9;
            en   = 0.0;
            en2  = 0.0;
            mag  =0.0;
            mag2 =0.0;
            stmag  =0.0;
            stmag2 =0.0;
            v13 = 0.;
            v24 = 0.;
            v1234 = 0.;

    /* when wrtsiz bins accumulated, write to the host file system */
    /* dumpelt() calls complemented by fdumpcp() calls in the host
       server program                                               */

           if( ptr == 9*wrtsiz ) {
             dumpelt(e_data,wrtsiz*9*4);
```

```
            combinesub(c_data,wrtsiz*(max+1),dim0,dim1);
            dumpelt(c_data,wrtsiz*4*(max+1));
            ptr = 0;
            c_ptr = 0;
          }

      }   /* blksiz */
    }     /* mratio */
  }       /* sweep loop  */

          e_data[0] = taacc/(float)(sweep);
          e_data[1] = saacc/(float)(sweep);
          e_data[2] = gaacc/(float)(sweep);
          e_data[3] = waacc*mratio/(float)(sweep);
          combinesub(e_data,4,dim0,dim1);
          dumpelt(e_data,4*4);

} /* update()  */

accpt_update()
{
taacc += (float)(taccpt)/(float)(nx*nx*nword*16);
saacc += (float)(saccpt)/(float)(nx*nx*nword*8);
gaacc += (float)(gaccpt)/(float)(nx*nx);
waacc += (float)(waccpt)/(float)(nx*nword*32);

taccpt = 0;
saccpt = 0;
gaccpt = 0;
waccpt = 0;
}
/*************************************************************
plaqcnt: Measurement of thermodynamic quantities requires
         summing over contributions of individual interacting
         plaquettes, which is the purpose of plaqcnt()
*************************************************************/
#include "spext.h"
#include "stdio.h"
#include "cros.h"
#include "ih.h"
/*************************************************************
        Layer=0,2 plaquettes go in y-direction
        Layer=1,3 plaquettes go in x-direction
*************************************************************/

/***   in y-direction ***************************************

 t=4          +---+   +---+   +---+   +---+   +---+
 layer=3      |   |   |   |   |   |   |   |   |   |
 t=3          +---+   +---+   +---+   +---+   +---+

 t=2      +---+   +---+   +---+   +---+   +---+
 layer=1  |   |   |   |   |   |   |   |   |   |
 t=1      +---+   +---+   +---+   +---+   +---+


*************************************************************/
plaqcnt(e_conf, m_conf)
float e_conf[4], m_conf[2];
{
unsigned int eng_conf[4], mag_conf[2];
unsigned int x_line[MXnx][MXword], y_line[MXny][MXword];
unsigned int x_vert[MXny][MXword], y_vert[MXnx][MXword];
unsigned int x_horzn[MXnx][MXword],y_horzn[MXny][MXword];
unsigned int hcmp,vcmp, line, line1, mcmp , ix;
int t,bit, bit2, layer, mw,x,y, len, in;
```

```
eng_conf[0] = 0;       /* 00: hrzn=same,vert=same => plaq 1 or 2  */
eng_conf[1] = 0;       /* 01: hrzn=same,vert=opst => plaq 7 or 8  */
eng_conf[2] = 0;       /* 10: hrzn=opst,vert=same => plaq 3 or 4  */
eng_conf[3] = 0;       /* 11: hrzn=opst,vert=opst => plaq 5 or 6  */

mag_conf[0] = 0;
mag_conf[1] = 0;

/* copy a layer of tlines from right to left  */
in=nword*ny*nx;
len=nx*nword*4;
cshift(&tlines[in],nextproc[0][POS],len,&tlines[0],nextproc[0][NEG],len);
cflush(&tlines[in],len);

/* layer=1, 3:   interacting plaqs go in y-dir  */
layer=1;
for(x=0;x<nx;x++){
    for(y=0;y<ny;y++) for(mw=0;mw<nword; mw++) {
        y_line[y][mw] = line =  tlines[nword*(x+y*nx)+mw];
                  line1 =  tlines[nword*(x+y*nx)+wr[mw+1]];
        y_vert[y][mw] = line^( ( (1&line1) << 31) | line>>1);
    /*  y_horzn[y][mw] = line^tlines[nword*(x + fwdy[y]*nx)+mw];  */
        y_horzn[y][mw] = line^tlines[nword*(x + (y+1)*nx)+mw];
    }

/* after identifying spin words which are connected by interacting
plaquettes, and XOR-ing to see what are the spin configurations,
scan in time direction by shifting the words, and count plaquettes */

    for(t=0;t<32;t += 4)for(mw=0;mw<nword; mw++) {
        bit =layer+t;
        bit2 = bit+2;   /* layer=3 */

        for(y=0;  y<ny;y += 2) {
            hcmp = (y_horzn[y][mw] >> bit ) & 1;
            vcmp =  (y_vert[y][mw] >> bit ) & 1;
            eng_conf[(hcmp<<1)+ vcmp]++;
            if((hcmp + vcmp) == 0) {
              mcmp = (y_line[y][mw] >> bit ) & 1;
              mag_conf[mcmp]++;
            }
        }
        for(y=1;  y<ny;y += 2) {
            hcmp = (y_horzn[y][mw] >> bit2 ) & 1;
            vcmp =  (y_vert[y][mw] >> bit2 ) & 1;
            eng_conf[(hcmp<<1)+ vcmp]++;
            if((hcmp + vcmp) == 0) {
             mcmp = (y_line[y][mw] >> bit2 ) & 1;
             mag_conf[mcmp]++;
            }
        }
    } /* t loop */
}

/***   in x-direction ****************************************

 t=3          +---+   +---+   +---+   +---+   +---+
 layer=2      |   |   |   |   |   |   |   |   |   |
 t=2          +---+   +---+   +---+   +---+   +---+

 t=1      +---+   +---+   +---+   +---+   +---+
 layer=0  |   |   |   |   |   |   |   |   |   |
 t=0      +---+   +---+   +---+   +---+   +---+
*****************************************************************/
```

```
/* repeat the procedure above, but for plaquettes running in
x-direction                                                    */

layer=0;  /* both layer=0 and 2 */
for(y=0;y<ny;y++){
    for(x=0;x<nx;x++) for(mw=0;mw<nword; mw++) {
        x_line[x][mw] = line =  tlines[nword*(x+y*nx) +mw];
                  line1 =  tlines[nword*(x+y*nx) +wr[mw+1]];
        x_vert[x][mw]  = line^( ( (1&line1) << 31 ) | line>>1);
        x_horzn[x][mw] = line^tlines[nword*(fwdx[x] + y*nx) +mw];
    }

    for(t=0;t<32;t += 4) for(mw=0;mw<nword; mw++) {
        bit=layer+t;
        bit2 = bit+2;   /* layer=2 */

        for(x=0;  x<nx;x += 2) {
/* hcmp = horizon compare of two lower spins */
/* vcmp =vertical compare of two left  spins */
            hcmp = (x_horzn[x][mw] >> bit ) & 1;
            vcmp =  (x_vert[x][mw] >> bit ) & 1;
            eng_conf[(hcmp<<1)+ vcmp]++;
            if((hcmp + vcmp) == 0) {
             mcmp = (x_line[x][mw] >> bit ) & 1;
             mag_conf[mcmp]++;
            }
        }
        for(x=1;  x<nx;x += 2) {
            hcmp = (x_horzn[x][mw] >> bit2 ) & 1;
            vcmp =  (x_vert[x][mw] >> bit2 ) & 1;
            eng_conf[(hcmp<<1)+ vcmp]++;
            if((hcmp + vcmp) == 0) {
             mcmp = (x_line[x][mw] >> bit2 ) & 1;
             mag_conf[mcmp]++;
            }
        }
    }   /* t loop */
}

e_conf[0] = (float) eng_conf[0];
e_conf[1] = (float) eng_conf[1];
e_conf[2] = (float) eng_conf[2];
e_conf[3] = (float) eng_conf[3];

m_conf[0] = (float) mag_conf[0];
m_conf[1] = (float) mag_conf[1] ;
}       /*  end of plaqcnt()    */
/*************************************************************
tau_meas: Measure the complete set of correlations
         S(x,y,t). For each value of t, the spin lines are
         shifted, and than two spin lines which differ by (x,y)
         vector, are XOR-ed. To improve statistics and recover
         isotropy, 16 bits are used for measurement. The
         corresponding value of the spin correlation is
         stored in precomputed table temporal[].
         Each processor computes only a piece of the correlation
         function.
*************************************************************/
#include "spext.h"
#include "math.h"
extern int temporal[65536];
unsigned int shifted[MXnx*MXny*MXword];

tau_meas(cofxyt)
float *cofxyt;
{
```

```
int time_len,tau_half,del_t,del_x,del_y,x,x1,y,y1,p,j,j1,cor,NY,NX;
int t,k,d,b,del_tw,del_tb,tm,ynx,wycgrid,offset,mw;
unsigned int word_dn,word_up,word_up1,word_up2,xor;
float factor;
/* assume square lattice, NY = nx, NY = ny * nprocs[0]  */

NY = NX = nx;
tau_half=16*nword;
time_len=1+4*nword;
wycgrid = NY/(nprocs[0]);

/* correlations are calculated on every other spin word in time
directions, since the measurements are correlated            */

tm= ((nword % 2) == 0) ? (nword/2) : (nword/2+1);
factor=1./((float)16*tm*NX*NY);
offset = recpnum[0]*wycgrid;

/* offset defines which correlations will be computed by the
processor with ring coordinate recpnum[0]                    */

for(del_t=0;del_t <= tau_half;del_t += 4){
/* shift the spin words                                       */
     if(del_t == 0) {
        for(y=0;y<NY;y++) {
          for(x=0;x<NX;x++) {
            j=nword*(x+y*nx);
              for(mw=0;mw<nword;mw++) {
                  shifted[j+mw] = tlines[j+mw];
              } /* mw loop */
          } /* x loop */
        } /* y loop */
     }else{
        for(y=0;y<NY;y++) {
          for(x=0;x<NX;x++) {
            j=nword*(x+y*nx);
              for(mw=0;mw<nword;mw++) {
                  shifted[j+mw] = ((1 & shifted[j + wr[mw+1]]) << 31) |
                                  (shifted[j+mw] >> 1);
              } /* mw loop */
          } /* x loop */
        } /* y loop */
     }
  for(del_x=0;del_x<NX;del_x++) {
    for(del_y = 0;del_y < wycgrid;del_y++) {
        cor=0;
        for(y=0;y<NY;y++) {
          for(x=0;x<NX;x++) {
            j=nword*(x+y*nx);
            x1=(x+del_x)%NX;
            y1=(y+offset+del_y)%NY;
            j1=nword*(x1+y1*nx);
              for(mw=0;mw<nword;mw += 2) {
                  word_dn=tlines[j+mw];
                  word_up=shifted[j1+mw];
                  cor += temporal[(word_dn ^ word_up) & (0xffff)];
/* XOR the spin words and accumulate the correlation function    */
              } /* mw loop */
          } /* x loop */
        } /* y loop */
        t=del_t/4;
        cofxyt[t + (del_x+del_y*nx)*time_len] = ((float)cor)*factor;
/* store the measurement in this array, to be FTransformed */
    } /* del_y loop */
  } /* del_x loop */
} /* del_t loop */
```

```
} /* end of tau_meas() */


/*************************************************************
corr_meas: measure correlation functions along x and y
           direction
*************************************************************/
#include "spext.h"
#include "math.h"

corr_meas(xcorr,ycorr)
int *xcorr,*ycorr;
{
register int del_x,del_y,x,x1,y,y1,p,j,j1,cor,NY;
register int ynx,y1nx,nx_half,NY_half,mw,xor,sgn;
int corr_tab[16];
/* assume square lattice, NY = nx, NY = ny * nprocs[0]  */

NY = nx;
NY_half = nx_half = nx/2;
xcorr[0] = ycorr[0] = 16*nword*nspins;

/* pick up lowest 4 bits in a spin word. corr_tab[] converts
this number into the correct value of Sz                    */
 corr_tab[0] = 4;
 corr_tab[1] = 2;
 corr_tab[2] = 2;
 corr_tab[3] = 0;
 corr_tab[4] = 2;
 corr_tab[5] = 0;
 corr_tab[6] = 0;
 corr_tab[7] = -2;
 corr_tab[8] = 2;
 corr_tab[9] = 0;
 corr_tab[10] = 0;
 corr_tab[11] = -2;
 corr_tab[12] = 0;
 corr_tab[13] = -2;
 corr_tab[14] = -2;
 corr_tab[15] = -4;

sgn = 1;
for(del_y = 1;del_y <= NY_half;del_y++) {
    sgn *= (-1);
    cor = 0;
    for(y=0;y<ny;y++) {
        ynx = y*nx;
        y1nx = (y+del_y)*nx;
        for(x=0;x<nx;x++) {
            j = nword*(x+ynx);
            j1 = nword*(x+y1nx);
            for(mw=0;mw<nword;mw++) {
/* XOR the 2 spin words at (x,y) and (x+del_x,y+del_y) */
                xor  = tlines[j + mw] ^ tlines[j1 + mw];
/* pick up lowest 4 bits                                */
                cor += corr_tab[LOW_4 & xor];
/* shift and pick up next 4 bits, 2 Trotter layers
above                                                   */
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
```

```
            }
        }
    }
    ycorr[del_y] = sgn*cor;
}

sgn = 1;
for(del_x = 1;del_x < nx_half;del_x++) {
    sgn *= (-1);
    cor = 0;
    for(x=0;x<nx;x++) {
        x1 = (x + del_x) % nx;
        for(y=0;y<ny;y++) {
            p = nword*y*nx;
            j = nword*x + p;
            j1 = nword*x1 + p;
            for(mw=0;mw<nword;mw++) {
                xor  = tlines[j + mw] ^ tlines[j1 + mw];
                cor += corr_tab[LOW_4 & xor];
/*  every other trotter layer  */
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
            }
        }
    }

    xcorr[del_x] = sgn*cor;
}

    del_x = nx_half;
    sgn *= (-1);
    cor = 0;
    for(y=0;y<ny;y++) {
      p = nword*y*nx;
      for(x=0;x<nx_half;x++) {
        x1 = x + del_x;
        j = p + nword*x;
        j1 = p + nword*x1;
        for(mw=0;mw<nword;mw++) {
         xor  = tlines[j + mw] ^ tlines[j1 + mw];
         cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
                xor = xor >> 8;
                cor += corr_tab[LOW_4 & xor];
        }
      }
    }

    xcorr[nx_half] = 2*sgn*cor;
/* a factor of 2 is introduced since the overall normalization
factor will be 4*nword*nx*NY                                    */

} /* end of corr_meas() */


/*****************************************************************
meas_vortx: This routine measures the vortex density and the
            vortex pair probability. The addresses of spin
```

```
            words needed for the measurement are supplied by
            vorticity().
*****************************************************************/

#include "spext.h"

meas_vortx(ss0,ss1,ss2,ss3,ss5,ss6,layer,vdat)
int ss0,ss1,ss2,ss3,ss5,ss6,layer;
float *vdat;
{
unsigned int sm,lbit0,lbit1,lbit2,lbit3,lbit6,lbit0123;
unsigned int ubit0,ubit1,ubit2,ubit3,ubit6,ubit0123;
unsigned int aux0,aux1,aux2,aux3,aux6,aux02,aux06,aux26;
unsigned int i, s0, s1, s2, s3, s6, i1;
int mw,corr_tab[16],z0z1z2z3[16];
float c13,g13,g02,g0123,g1236,g1306,gx0123,gz0123;

/* these are the tables of 2 and 4 spin correlations in y
direction in spin space. The index to a table is provided
by bit manipulations on neghboring spin words.          */

 corr_tab[0] = 4;
 corr_tab[1] = 2;
 corr_tab[2] = 2;
 corr_tab[3] = 0;
 corr_tab[4] = 2;
 corr_tab[5] = 0;
 corr_tab[6] = 0;
 corr_tab[7] = -2;
 corr_tab[8] = 2;
 corr_tab[9] = 0;
 corr_tab[10] = 0;
 corr_tab[11] = -2;
 corr_tab[12] = 0;
 corr_tab[13] = -2;
 corr_tab[14] = -2;
 corr_tab[15] = -4;


 z0z1z2z3[0] = 1;
 z0z1z2z3[1] = -1;
 z0z1z2z3[2] = -1;
 z0z1z2z3[3] = 1;
 z0z1z2z3[4] = -1;
 z0z1z2z3[5] = 1;
 z0z1z2z3[6] = 1;
 z0z1z2z3[7] = -1;
 z0z1z2z3[8] = -1;
 z0z1z2z3[9] = 1;
 z0z1z2z3[10] = 1;
 z0z1z2z3[11] = -1;
 z0z1z2z3[12] = 1;
 z0z1z2z3[13] = -1;
 z0z1z2z3[14] = -1;
 z0z1z2z3[15] = 1;

g13=0.;
g02=0.;
g0123=0.;
g1236=0.;
g1306=0.;
gx0123=0.;
gz0123=0.;


for(mw=0; mw<nword; mw++){
```

```
/* this piece is very similar to spaceflipr(), since a cluster
of 4 spin words is involved. 4 spin clusters are due to the
decomposition properties of U1U2 and U3U4                    */
s0 = tlines[ss0+mw];
s1 = tlines[ss1+mw];
s2 = tlines[ss2+mw];
s3 = tlines[ss3+mw];
s6 = tlines[ss6+mw];
sm = s_mask[layer];

/* the matrix elements that we need are between spin layers
t and t+2. We manipulate the spin words to get these matrix
elements in the following fashion. A sequence of XORs is needed
to establish the spin configurations, and we pick up the relevant bits
by masking. What we obtain are small integers, which are then
indices to look-up tables calculated by weights()            */

if(layer == 0){
        /**  perform a rotate << 1  function  **/
        /** need a bit from the word below the current one **/
        aux0 = ( (HIGH_1 & tlines[ss0+wr[nword+mw-1]]) >> 31 ) | (s0<< 1) ;
        aux1 = ( (HIGH_1 & tlines[ss1+wr[nword+mw-1]]) >> 31 ) | (s1<< 1) ;
        aux2 = ( (HIGH_1 & tlines[ss2+wr[nword+mw-1]]) >> 31 ) | (s2<< 1) ;
        aux3 = ( (HIGH_1 & tlines[ss3+wr[nword+mw-1]]) >> 31 ) | (s3<< 1) ;
        aux6 = ( (HIGH_1 & tlines[ss6+wr[nword+mw-1]]) >> 31 ) | (s6<< 1) ;
        lbit0 = aux0 & sm;
        lbit1 = aux1 & sm;
        lbit2 = aux2 & sm;
        lbit3 = aux3 & sm;
        lbit6 = aux6 & sm;
        ubit0 = (s0 >>1) & sm;
        ubit1 = (s1 >>1) & sm;
        ubit2 = (s2 >>1) & sm;
        ubit3 = (s3 >>1) & sm;
        aux02 = (aux2 ^ aux0) & sm;
        aux06 = (aux6 ^ aux0) & sm;
        aux26 = (aux2 ^ aux6) & sm;
} else if(layer==3){
        /**  perform a rotate >> 1  function  **/
        /** need a bit from the word above the current one  **/
        aux0 = ( (1 & tlines[ss0+wr[mw+1]]) << 31 ) | (s0>> 1) ;
        aux1 = ( (1 & tlines[ss1+wr[mw+1]]) << 31 ) | (s1>> 1) ;
        aux2 = ( (1 & tlines[ss2+wr[mw+1]]) << 31 ) | (s2>> 1) ;
        aux3 = ( (1 & tlines[ss3+wr[mw+1]]) << 31 ) | (s3>> 1) ;
        ubit0 = aux0 & sm;
        ubit1 = aux1 & sm;
        ubit2 = aux2 & sm;
        ubit3 = aux3 & sm;
        lbit0 = (s0 <<1) & sm;
        lbit1 = (s1 <<1) & sm;
        lbit2 = (s2 <<1) & sm;
        lbit3 = (s3 <<1) & sm;
        lbit6 = (s6 <<1) & sm;
        aux02 = lbit2 ^ lbit0;
        aux06 = lbit6 ^ lbit0;
        aux26 = lbit2 ^ lbit6;
} else {
        ubit0 = (s0 >> 1) & sm;
        ubit1 = (s1 >> 1) & sm;
        ubit2 = (s2 >> 1) & sm;
        ubit3 = (s3 >> 1) & sm;
        lbit0 = (s0 << 1) & sm;
        lbit1 = (s1 << 1) & sm;
        lbit2 = (s2 << 1) & sm;
        lbit3 = (s3 << 1) & sm;
        lbit6 = (s6 << 1) & sm;
```

```
        aux02 = lbit2 ^ lbit0;
        aux06 = lbit6 ^ lbit0;
        aux26 = lbit2 ^ lbit6;
       }


if( layer > 0){
/* shift the words with relevant bits to layer 0 position */
ubit3 >>= layer;
ubit2 >>= layer;
ubit1 >>= layer;
ubit0 >>= layer;
lbit3 >>= layer;
lbit2 >>= layer;
lbit1 >>= layer;
lbit0 >>= layer;
aux02 >>= layer;
aux06 >>= layer;
aux26 >>= layer;
}

/* the bits are overlapping, so we shift them again
and than OR them to get small integers              */
ubit3 <<= 3;
ubit2 <<= 2;
ubit1 <<= 1;
lbit3 <<= 3;
lbit2 <<= 2;
lbit1 <<= 1;
ubit0123 = ubit0 | ubit1 | ubit2 | ubit3;
lbit0123 = lbit0 | lbit1 | lbit2 | lbit3;

/* now just use the look-up tables x1x3 etc. */
for(i=0;i<32;i+=4){
    c13 = x1x3[(lbit0123 >> i) & LOW_4][(ubit0123 >> i) & LOW_4];
    g13 +=  c13;
    g0123 += z0z2[(aux02 >> i) & 1] * c13;
    g1236 += z0z2[(aux26 >> i) & 1] * c13;
    g1306 += z0z2[(aux06 >> i) & 1] * c13;
    g02 += z0z2[(aux02 >> i) & 1];
    gx0123 += x0x1x2x3[(lbit0123 >> i) & LOW_4][(ubit0123 >> i) & LOW_4];
    gz0123 += z0z1z2z3[(lbit0123 >> i) & LOW_4];
    }

/*
    g02 += corr_tab[(s2 ^ s0) & LOW_4] + corr_tab[(s1 ^ s3) & LOW_4];
*/

} /* mw loop */

vdat[0] = g13/(float)(nword*8);
/*
vdat[1] = g02/(float)(nword*4*2);
*/
vdat[1] = g02/(float)(nword*8);
vdat[2] = g0123/(float)(nword*8);
vdat[3] = g1236/(float)(nword*8);
vdat[4] = g1306/(float)(nword*8);
vdat[5] = gx0123/(float)(nword*8);
vdat[6] = gz0123/(float)(nword*8);
/* these pieces will ultimately be combined to give the
vortex density */

} /* end of meas_vortx() */
/*************************************************************
```

```
vorticity: This function identifies elementary squares and
           addresses of spin words which are then provided
           to meas_vortx().
************************************************************/
#include "spext.h"
#include "cros.h"
#include "ih.h"

/*********************************************************
        even plaq:
                                        y
              3*--up---*2                       layer=0,2
               |       |                ^
               down    down             |
               |       |                |
              0*--up---*1               -------> x


**********************************************************/

vorticity(vbuf)
float vbuf[7];
{
int s5,s6;
int x,y,x1,y1,x2,y2;
int xfwd, cen, yfwd, xyfwd;
int in, len;
float buf[7];

in = ny*nx*nword;
len = nx*nword*4;
vbuf[0] = 0.;
vbuf[1] = 0.;
vbuf[2] = 0.;
vbuf[3] = 0.;
vbuf[4] = 0.;
vbuf[5] = 0.;
vbuf[6] = 0.;
vbuf[7] = 0.;

/* very similar to spaceplaq(), since the 4 spin cluster are
the same. Only difference is that here we also calculate addresses
of 2 extra spins needed for vortex pair density. Essentially we
sweep over lattice looking for 6 spin clusters residing on 2
adjacent squares.                                              */

/*  layer=1 and 2  */
for(y=0;y<ny;y += 2 ){
y1 = y+1;
y2 = fwdy[y1];
for(x=0;x<nx;x++){
        xfwd= nword*(fwdx[x] + y*nx) ;
        yfwd= nword*(x + y1*nx) ;
        xyfwd=nword*(fwdx[x] + y1*nx) ;
        cen = nword*(x + y*nx) ;
            if(x%2 == 0){
                x1=fwdx[x];
                s5= nword*(x+y2*nx);
                s6= nword*(x1 + y2*nx);
                meas_vortx(cen,yfwd,xyfwd,xfwd,s5,s6,1,buf);
                vbuf[0] += buf[0];
                vbuf[1] += buf[1];
                vbuf[2] += buf[2];
                vbuf[3] += buf[3];
                vbuf[4] += buf[4];
```

```
                vbuf[5] += buf[5];
                vbuf[6] += buf[6];
                vbuf[7] += buf[7];
            }
            else{
                x2= fwdx[fwdx[x]];
                s5= nword*(x2+y*nx);
                s6= nword*(x2 + y1*nx);
                meas_vortx(cen,xfwd,xyfwd,yfwd,s5,s6,2,buf);
                vbuf[0] += buf[0];
                vbuf[1] += buf[1];
                vbuf[2] += buf[2];
                vbuf[3] += buf[3];
                vbuf[4] += buf[4];
                vbuf[5] += buf[5];
                vbuf[6] += buf[6];
                vbuf[7] += buf[7];
            }
}
}

/*  layer=0 and 3  */
/* the only difference from above is that the boundary
layer from the neighbor is needed                          */
cshift(tlines+in,nextproc[0][POS],len,tlines,nextproc[0][NEG],len);
cflush(tlines+in,len);
for(y=1;y<ny;y += 2 ){
y1 = y+1;
y2 = fwdy[y1];
for(x=0;x<nx;x++){
        xfwd= nword*(fwdx[x] + y*nx) ;
        yfwd= nword*(x + y1*nx) ;
        xyfwd=nword*(fwdx[x] + y1*nx) ;
        cen = nword*(x + y*nx) ;
            if(x%2 == 0) {
                x2= fwdx[fwdx[x]];
                s5= nword*(x2+y*nx);
                s6= nword*(x2 + y1*nx);
                meas_vortx(cen,xfwd,xyfwd,yfwd,s5,s6,0,buf);
                vbuf[0] += buf[0];
                vbuf[1] += buf[1];
                vbuf[2] += buf[2];
                vbuf[3] += buf[3];
                vbuf[4] += buf[4];
                vbuf[5] += buf[5];
                vbuf[6] += buf[6];
                vbuf[7] += buf[7];
            }
            else{
                x1= fwdx[x];
                s5= nword*(x+y2*nx);
                s6= nword*(x1 + y2*nx);
                meas_vortx(cen,yfwd,xyfwd,xfwd,s5,s6,3,buf);
                vbuf[0] += buf[0];
                vbuf[1] += buf[1];
                vbuf[2] += buf[2];
                vbuf[3] += buf[3];
                vbuf[4] += buf[4];
                vbuf[5] += buf[5];
                vbuf[6] += buf[6];
                vbuf[7] += buf[7];
            }
}
}
```

```
}
/*************************************************************
transform: Calculates the FT of the real space dynamic
           correlations. Each processor in a ring has only
           a piece of correlations, but all nodes combined
           cover the whole lattice. They calculate a piece
           of FT for a discrete set of points in BZ, and
           then combine their pieces to get the whole FT.
           This is done only once at the end of the run.
*************************************************************/
#include "spext.h"
#include "math.h"

transform(cofxyt,cofkt)
float *cofxyt, *cofkt;
{
int del_t,x,y,NY,NX;
int wycgrid,offset;
int k_d,yy,jj,th,index,k_d_max;
float factor,cor;
/* assume square lattice, NY = nx, NY = ny * nprocs[0]  */

NY = NX = nx;
k_d_max=NX/2;
th=4*nword+1;
wycgrid = NY/(nprocs[0]);
offset = recpnum[0]*wycgrid;
/*
factor=1./((float)NY*NX);
*/
factor=1.;
jj=0;
for(k_d= 0;k_d <= k_d_max;k_d++) {
    for(del_t=0;del_t < th;del_t++){
        cor=0.;
        for(yy=0;yy<wycgrid;yy++) {
        y = yy + offset;
          for(x=0;x<NX;x++) {
            index=k_d*(x+y);
            cor += cofxyt[del_t + (x+yy*nx)*th] * kosinus[index];
          } /* x loop */
        } /* y loop */
        *(cofkt + (del_t + jj*th)) = factor * cor;
        cor=0.;
     } /* del_t loop */
     jj++;
} /* k_d loop */

for(k_d= k_d_max-1;k_d >= 0;k_d--) {
    for(del_t=0;del_t < th;del_t++){
        cor=0.;
        for(yy=0;yy<wycgrid;yy++) {
        y = yy + offset;
          for(x=0;x<NX;x++) {
            index=k_d*y + k_d_max*x;
            cor += cofxyt[del_t + (x+yy*nx)*th] * kosinus[index];
          } /* x loop */
        } /* y loop */
        *(cofkt + (del_t + jj*th)) = factor * cor;
        cor=0.;
     } /* del_t loop */
     jj++;
} /* k_d loop */

for(k_d= k_d_max-1;k_d > 0;k_d--) {
    for(del_t=0;del_t < th;del_t++){
```

```
        cor=0.;
        for(yy=0;yy<wycgrid;yy++) {
        y = yy + offset;
          for(x=0;x<NX;x++) {
            index=k_d*x;
            cor += cofxyt[del_t + (x+yy*nx)*th] * kosinus[index];
          } /* x loop */
        } /* y loop */
        *(cofkt + (del_t + jj*th)) = factor * cor;
        cor=0.;
      } /* del_t loop */
      jj++;
} /* k_d loop */

} /* end of transform() */


/*************************************************************
winding: Calculates spin stiffness by measuring the average
         drift of the worldlines belonging to a subsystem
         of the original system. Calculates the drift of both
         particles and holes.
*************************************************************/
#include "spext.h"
#include <stdio.h>
#include <math.h>

winding(p_w,h_w)
float p_w[],h_w[];
{
float p_walk[3],h_walk[3], *ptr[2];
int NX,NY,x,y,i,particle[2];
int delta[2],what;
float factor;


NX=NY=nx;
factor= 2./(float)NX;
particle[0]=0;
particle[1]=0;
ptr[0]=h_walk;
ptr[1]=p_walk;

for(i=0;i<=2;i++) {
  p_walk[i]=0.;
  h_walk[i]=0.;
}

for(y=(NY/4);y<(3*NY/4);y++) {
  for(x=(NX/4);x<(3*NX/4);x++) {
    what=wlines(delta,x,y);
    *((*(ptr+what))) += delta[0];
    *((*(ptr+what))+1) += delta[1];
    particle[what]++;
/*
if (what == 0) printf("this is hole\n");
if (what == 1) printf("this is particle\n");
*/
  }
}


h_w[0]=h_walk[0] * factor;
p_w[0]=p_walk[0] * factor;
```

```
h_w[1]=h_walk[1] * factor;
p_w[1]=p_walk[1] * factor;

p_w[2]=p_w[0]*p_w[0] + p_w[1]*p_w[1];
h_w[2]=h_w[0]*h_w[0] + h_w[1]*h_w[1];

}

/*****************************************************************
wlines: This function identifies the individual worldlines, and
        traces them from 0 to beta. It calculates the drift,
        and then winding() accumulates the results for all
        worldlines. With minor modifications, it can calculate
        the probability of destroying an arbitrary worldline.
        Each processor is assigned a different subset of
        worldlines to trace by assigning a different piece of
        the lattice. Since they are uniformly distributed, the
        computational load is balanced on the average.
*****************************************************************/

wlines(del,x0,y0)
int *del,x0,y0;
{
unsigned int line,line_up,prop,line_above;
int  NX,NY,x,y,x_neigh,y_neigh,z,pos,mw,zmax;
int what;
int delx,dely;

delx=0;
dely=0;
zmax = 32*nword;

  NX=NY=nx;
  mw=0;
  pos = nword*(x0+y0*NX);
  z = 0;
  x= x0;
  y= y0;

 if((tlines[pos] & 1) == 1) {
  what = 1;
  while(z < zmax) {
    line = tlines[pos+mw];
    line_up = (line >> 1) | (tlines[pos+wr[mw+1]] << 31);
    line_above=tlines[pos+wr[mw+1]];
    prop = (line ^ line_up) & bits[z%32];
    if(prop == bits[z%32]) {
       if(z%4 == 0) {
           y_neigh = y;
           if(x%2 == 0) {x_neigh = x+1;delx++;}
           else         {x_neigh = x-1;delx--;}
       }
       if(z%4 == 1) {
           x_neigh = x;
           if(y%2 == 0) {y_neigh = y+1;dely++;}
           else         {y_neigh = y-1;dely--;}
       }
       if(z%4 == 2) {
           y_neigh = y;
           if(x%2 == 0) {x_neigh = (x-1+NX)%NX;delx--;}
           else         {x_neigh = (x+1)%NX;delx++;}
       }
       if(z%4 == 3) {
           x_neigh = x;
           if(y%2 == 0) {y_neigh = (y-1+NY)%NY;dely--;}
           else         {y_neigh = (y+1)%NY;dely++;}
```

```
        }
        z++;
        x = x_neigh;
        y = y_neigh;
     }
     else z++;

     pos = nword*(x+y*nx);
     if(z%32 == 0) mw++;
   }
   }

  else {
   what = 0;
   while(z < zmax) {
     line = tlines[pos+mw];
     line_up = (line >> 1) | (tlines[pos+wr[mw+1]] << 31);
     line_above=tlines[pos+wr[mw+1]];
     prop = (line ^ line_up) & bits[z%32];
     if(prop == bits[z%32]) {
        if(z%4 == 0) {
            y_neigh = y;
            if(x%2 == 0) {x_neigh = x+1;delx++;}
            else         {x_neigh = x-1;delx--;}
        }
        if(z%4 == 1) {
            x_neigh = x;
            if(y%2 == 0) {y_neigh = y+1;dely++;}
            else         {y_neigh = y-1;dely--;}
        }
        if(z%4 == 2) {
            y_neigh = y;
            if(x%2 == 0) {x_neigh = (x-1+NX)%NX;delx--;}
            else         {x_neigh = (x+1)%NX;delx++;}
        }
        if(z%4 == 3) {
            x_neigh = x;
            if(y%2 == 0) {y_neigh = (y-1+NY)%NY;dely--;}
            else         {y_neigh = (y+1)%NY;dely++;}
        }
        z++;
        x= x_neigh;
        y= y_neigh;
     }
     else z++;

     pos = nword*(x+y*nx);
     if(z%32 == 0) mw++;
   }
  }

*del=delx;
*(del+1)=dely;
return what;
}
```

Figures and Tables

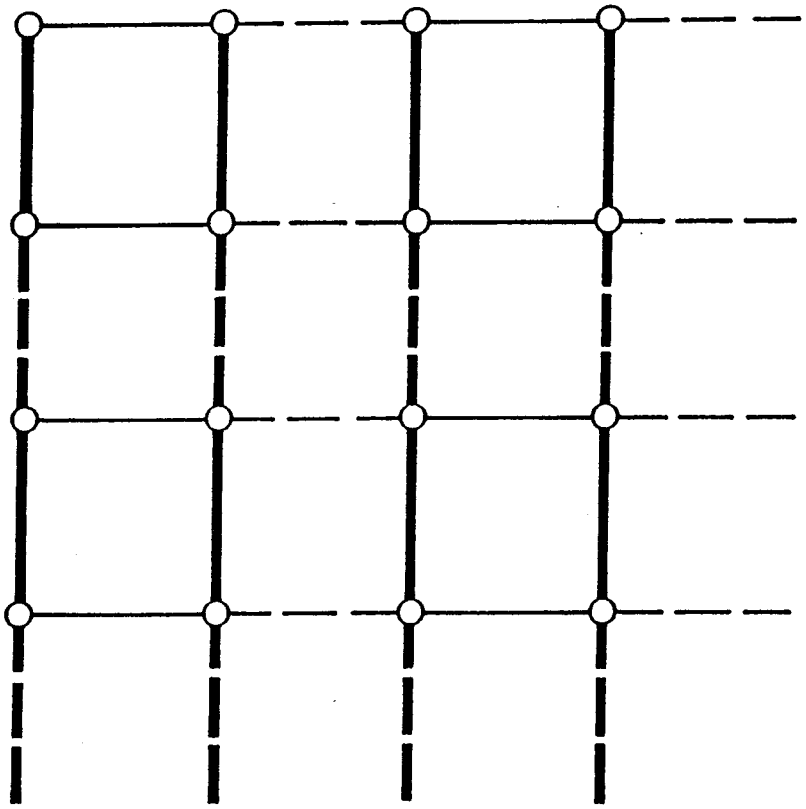
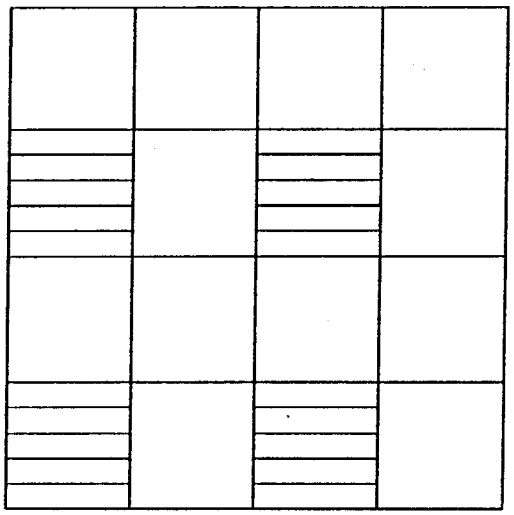


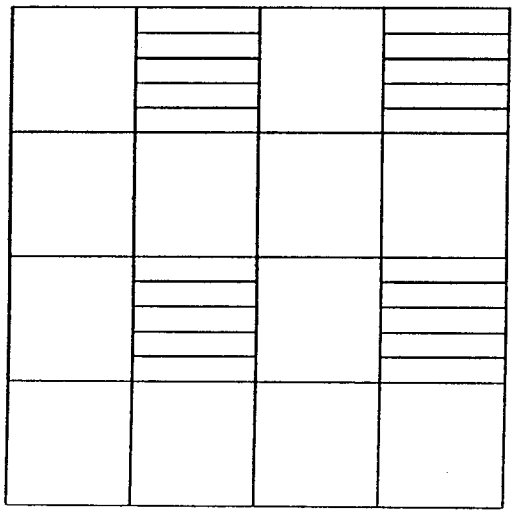
Fig. 1a

Fig. 1b

U_1U_2



U_3U_4



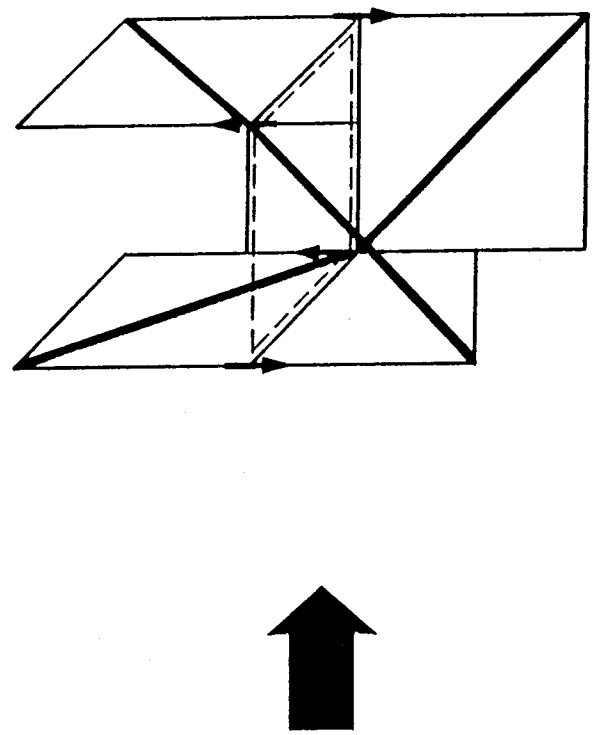


Fig. 2

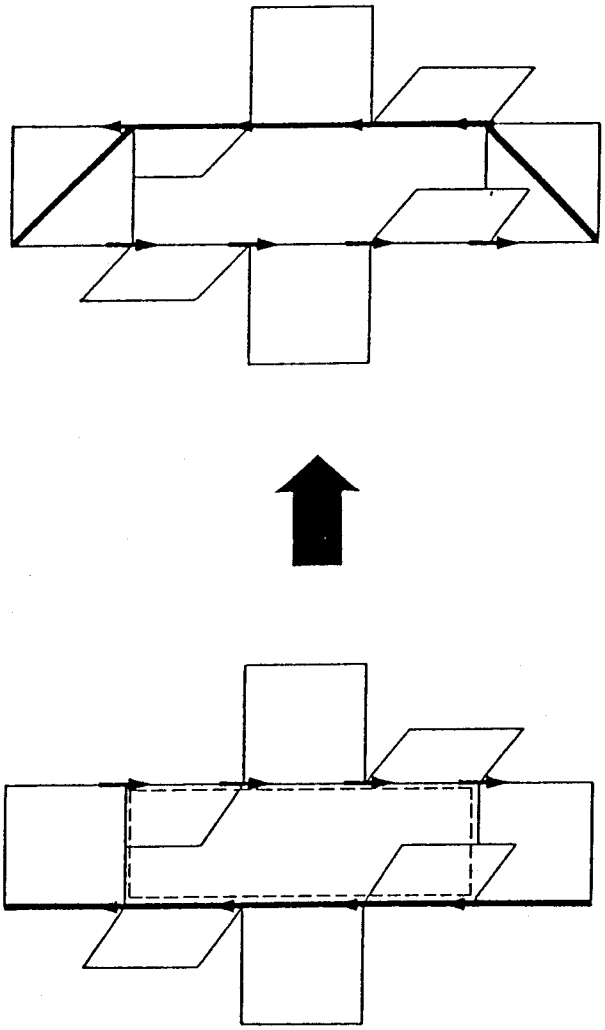


Fig. 3

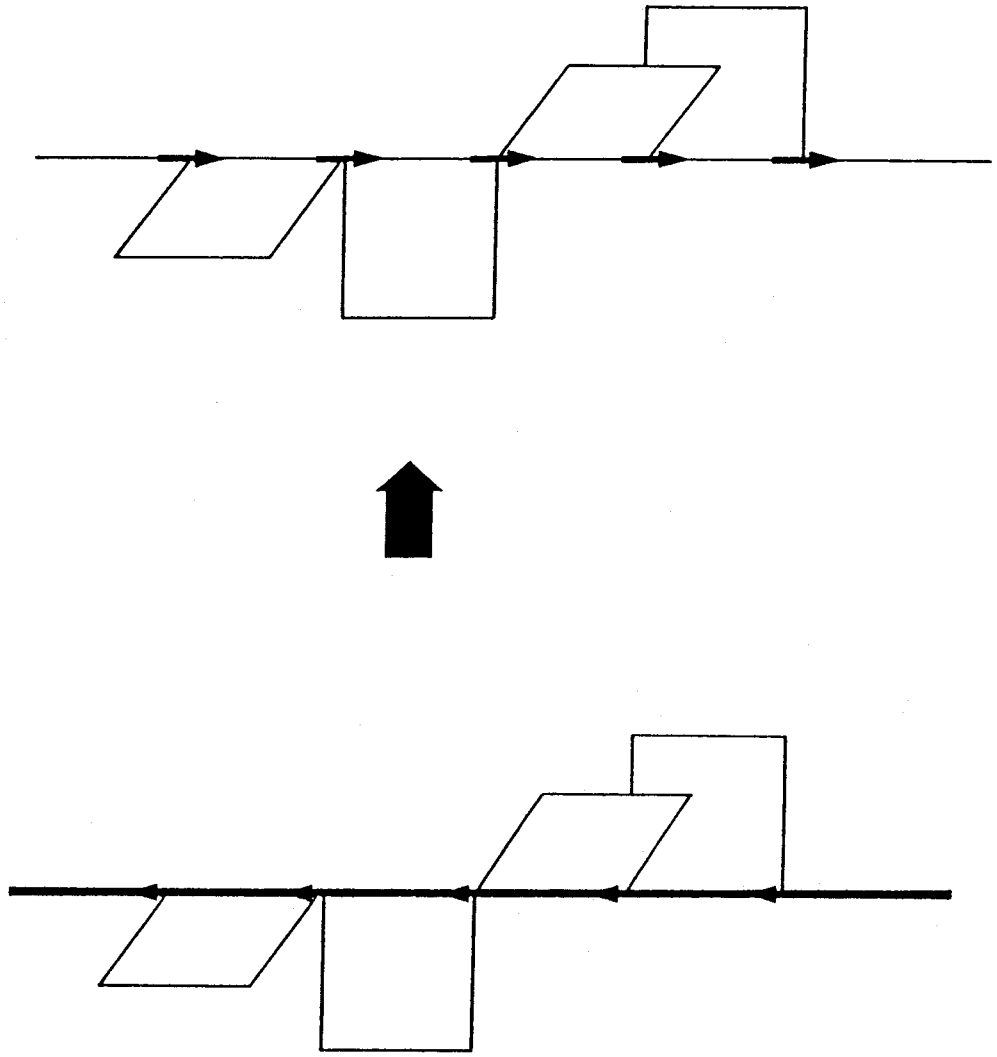
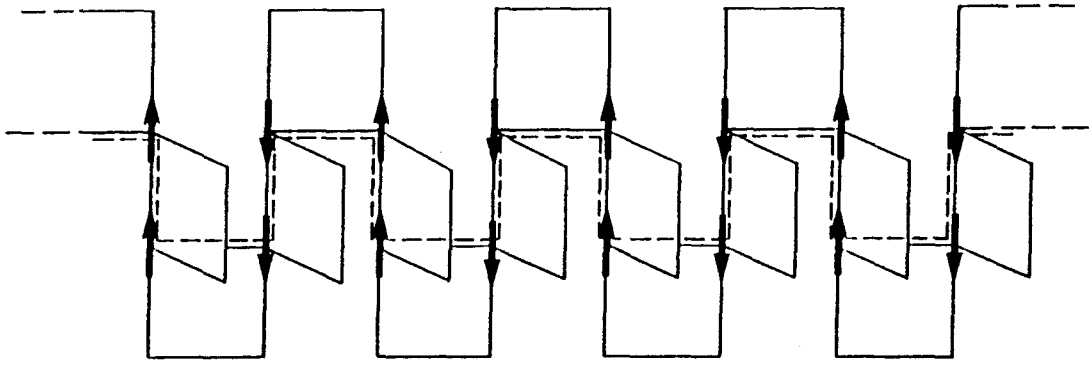
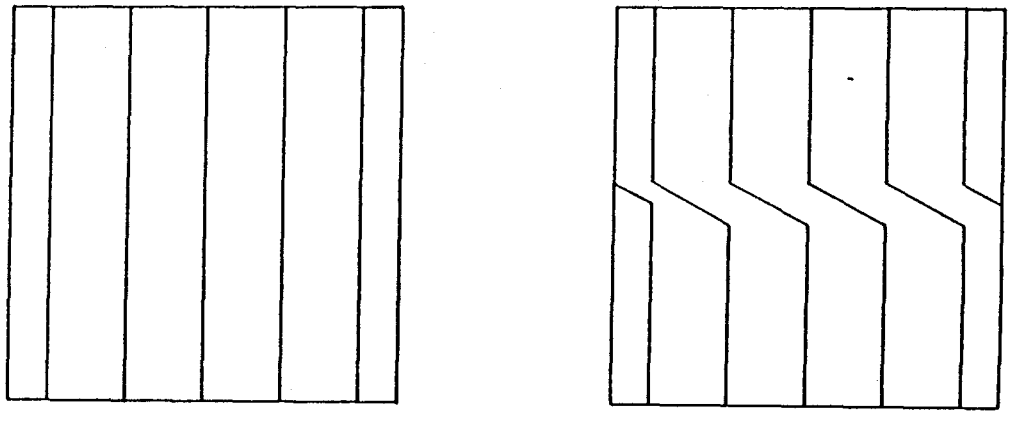


Fig. 4



(a)



(b)

Fig. 5

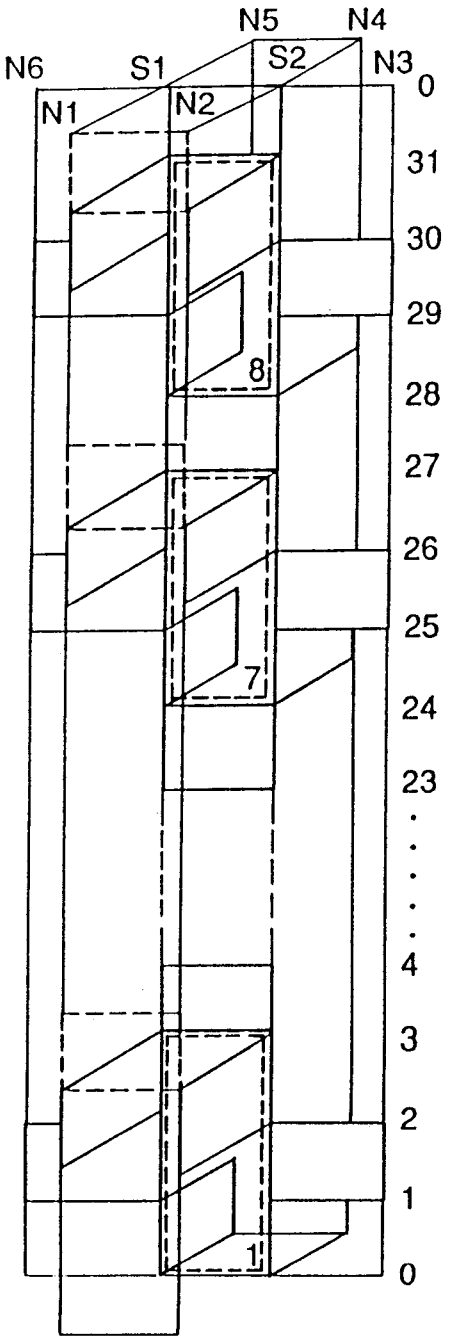


Fig. 6

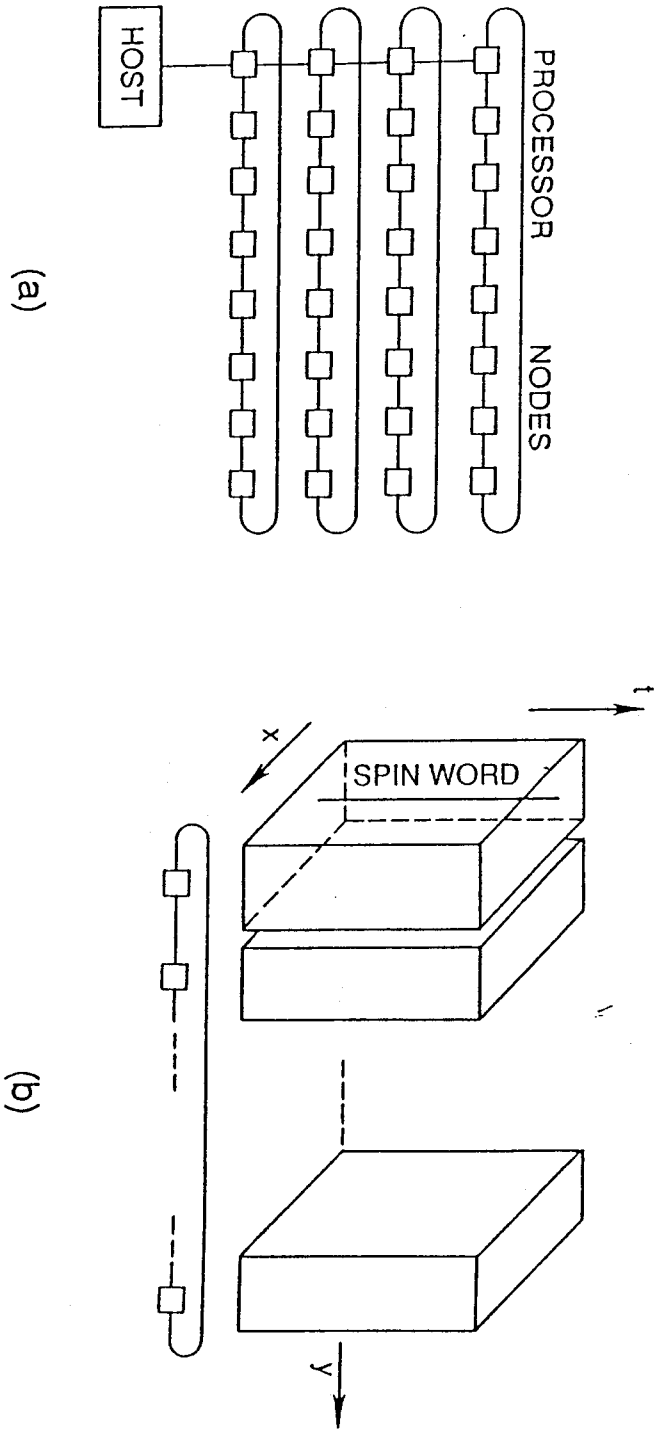
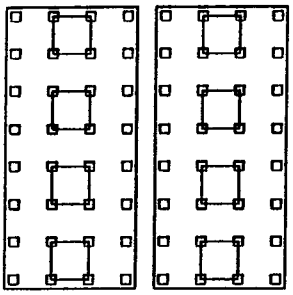


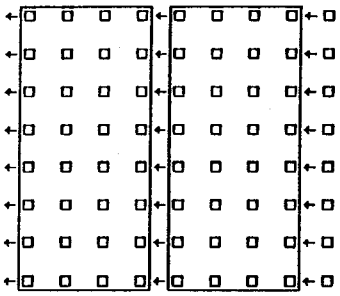
Fig. 7

Fig. 7c

Step 1: Local Update



Step 2: Communication



Step 3: Local Update

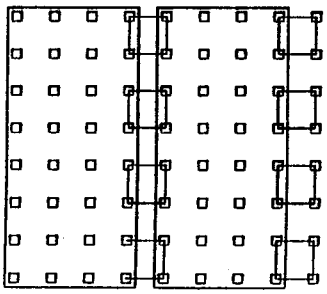


Fig. 7d

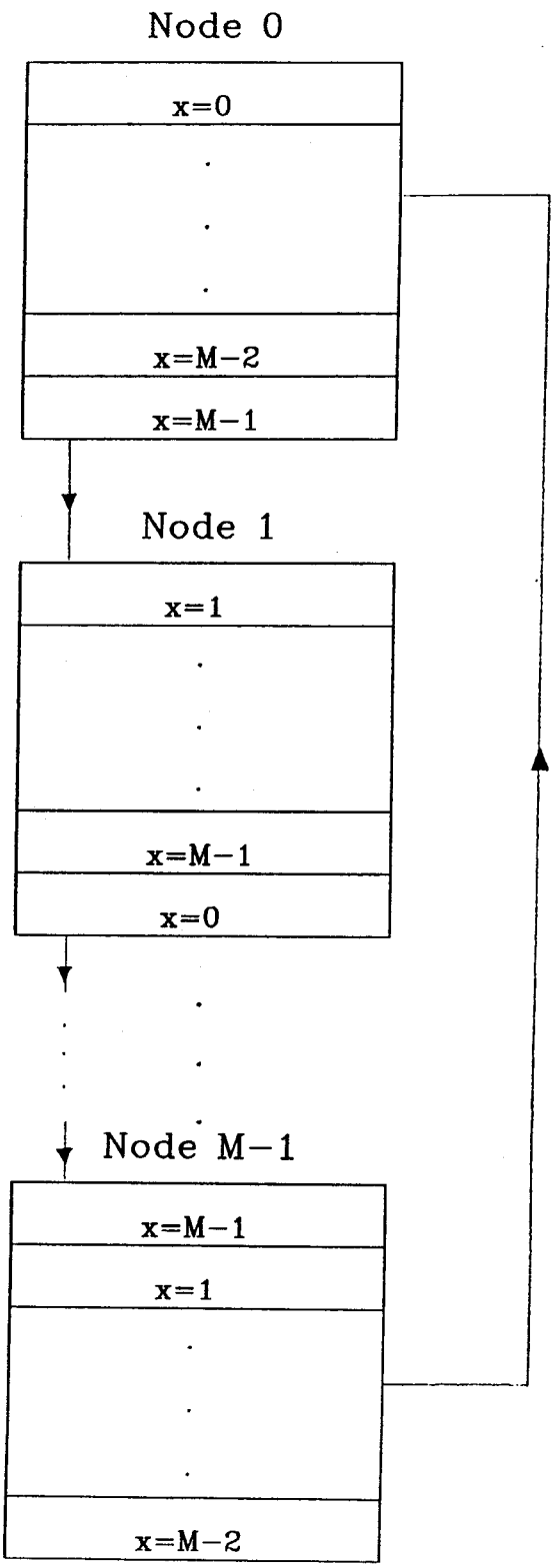


Fig. 8

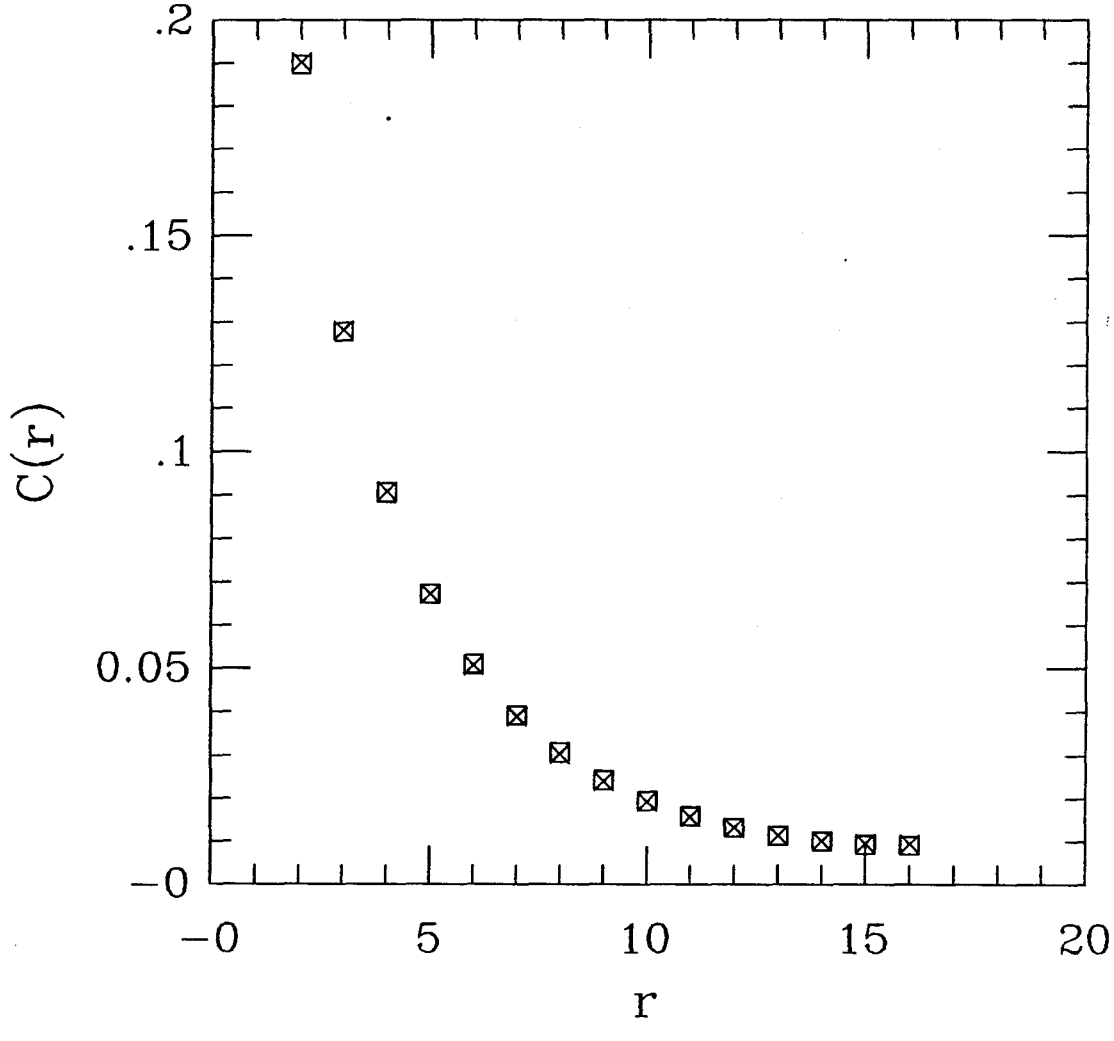


Fig. 9

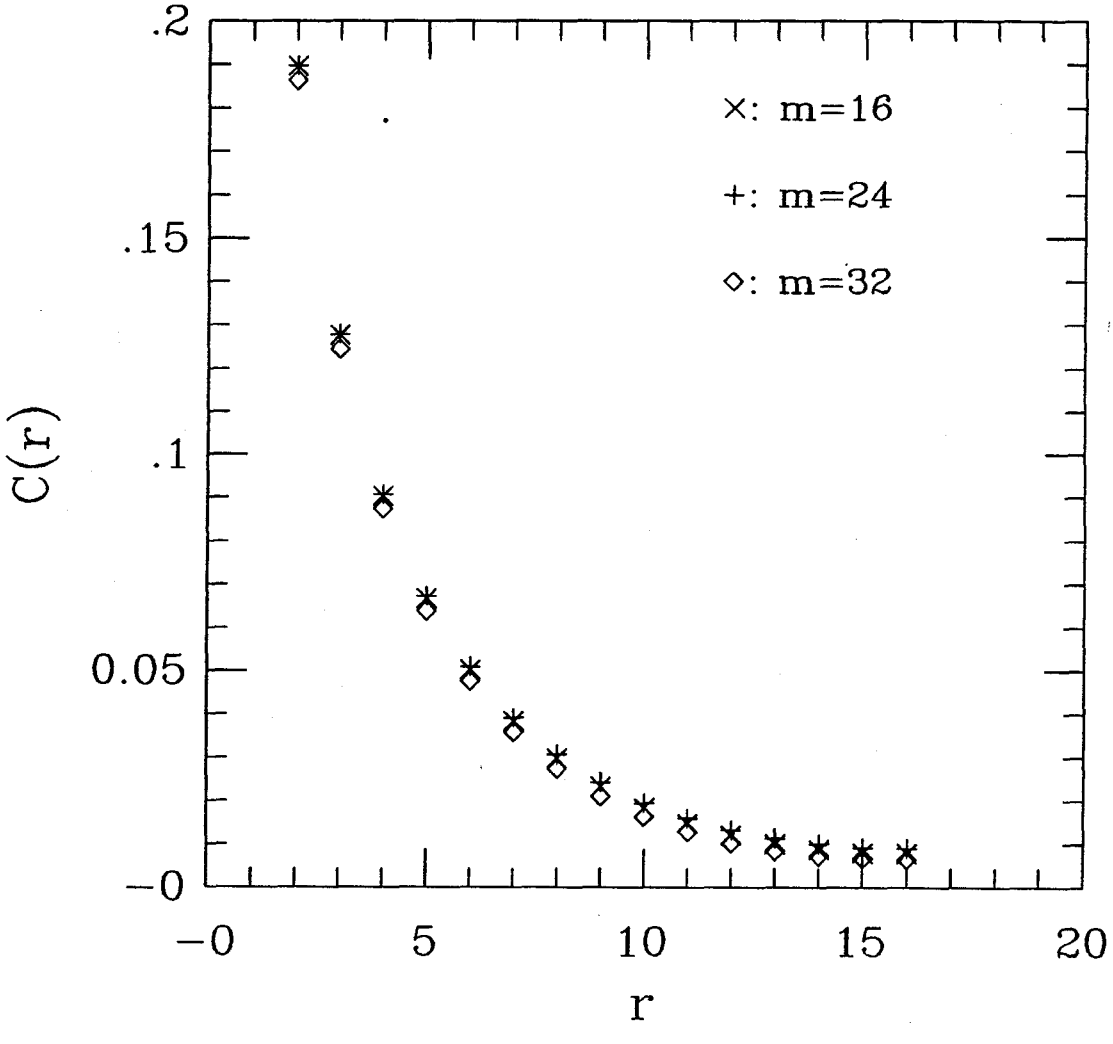


Fig. 10

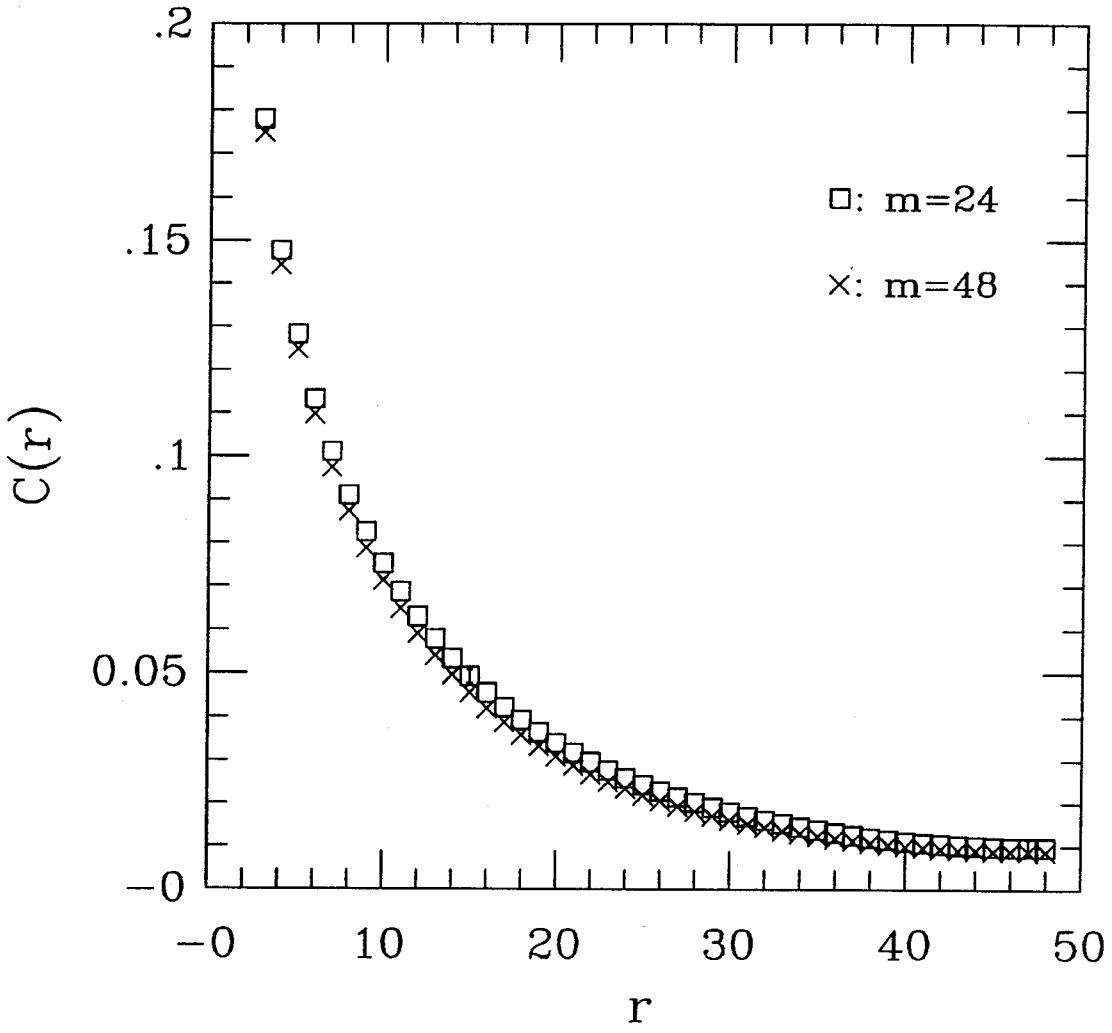


Fig. 11

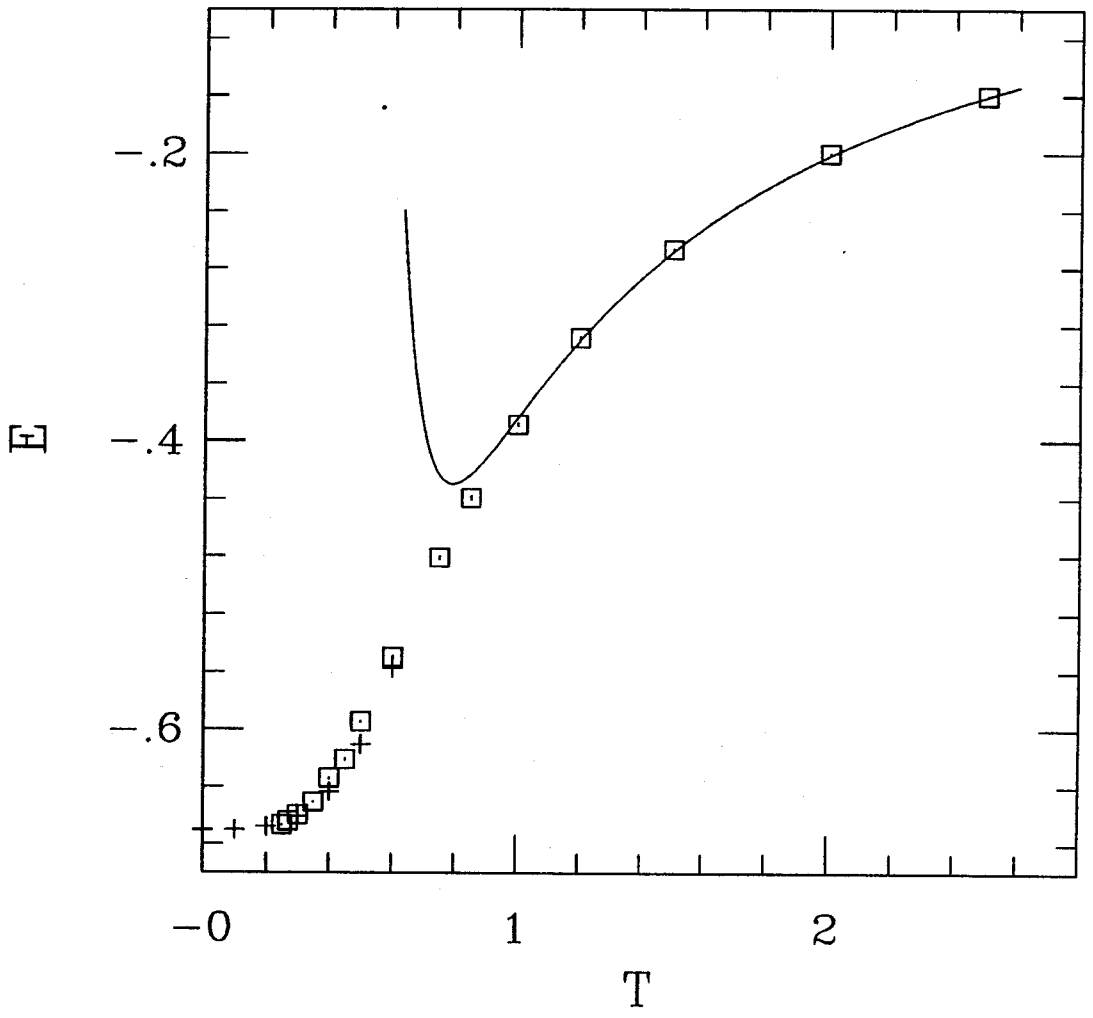


Fig. 12

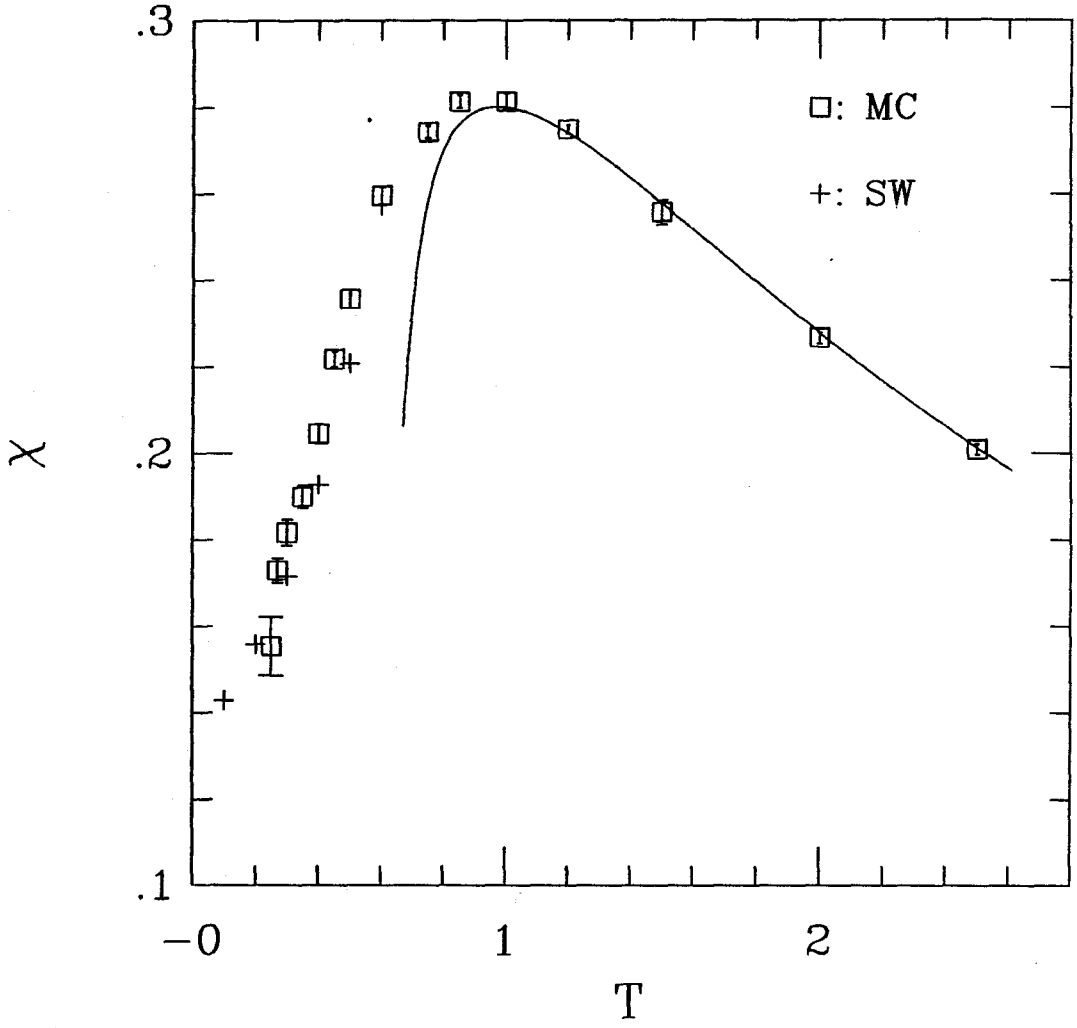


Fig. 13

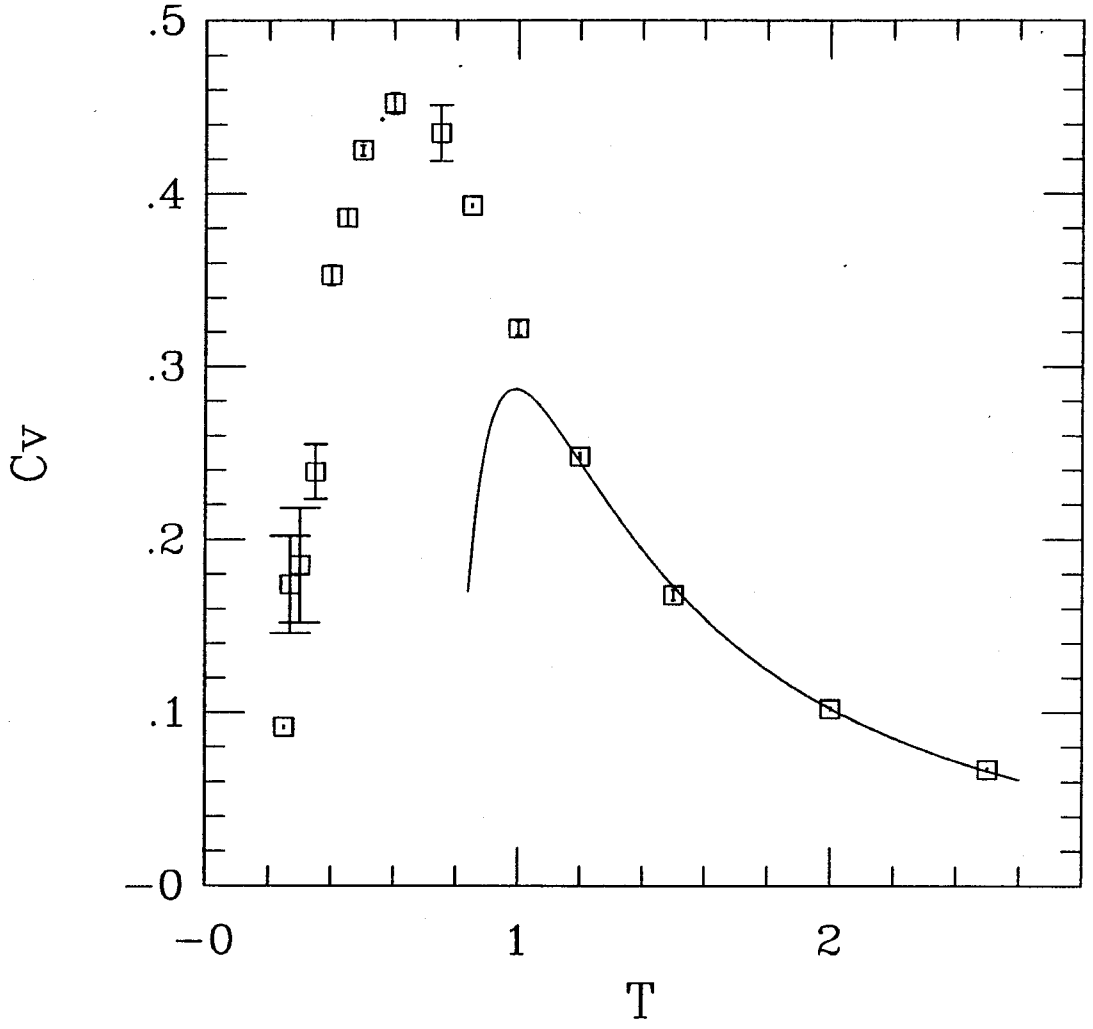


Fig. 14a

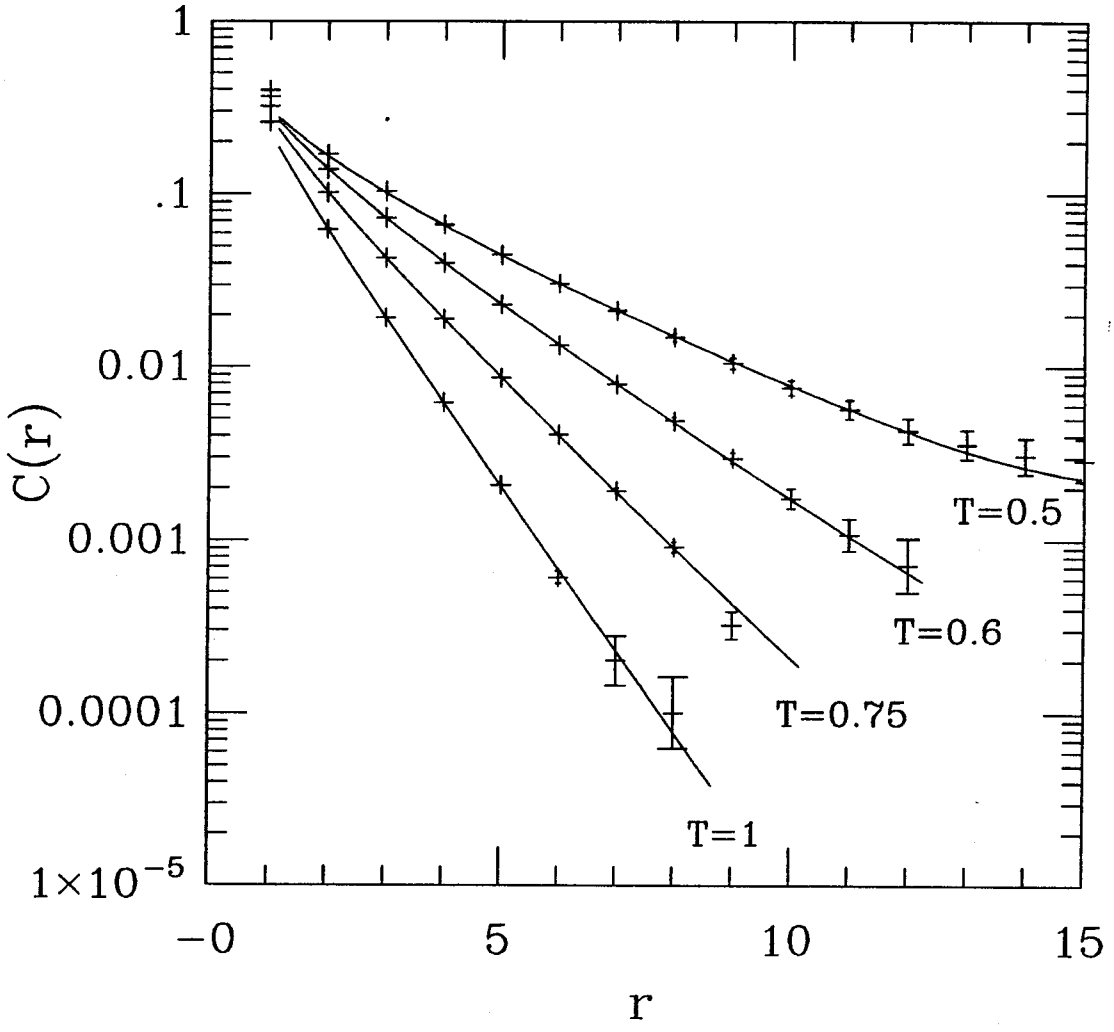


Fig. 14b

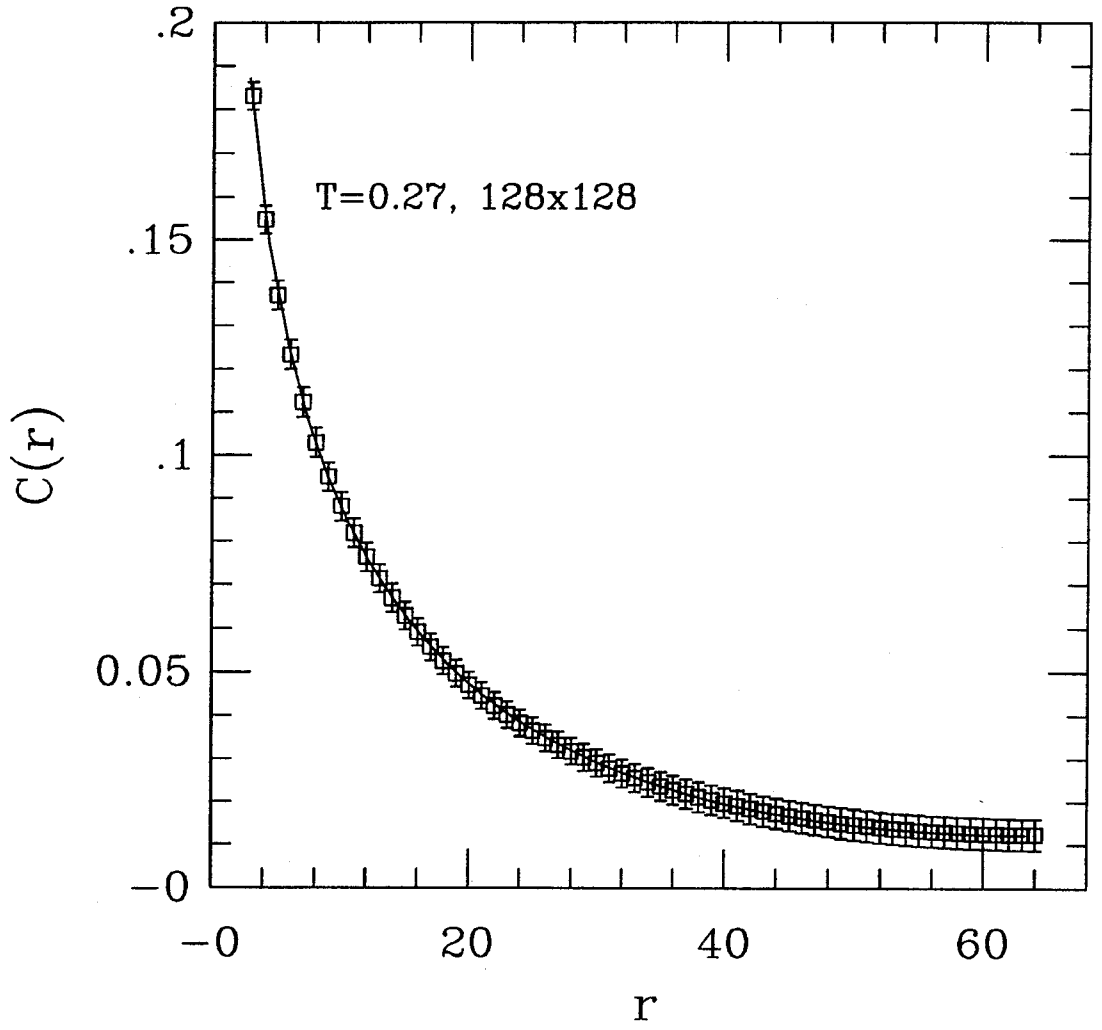


Fig. 14c

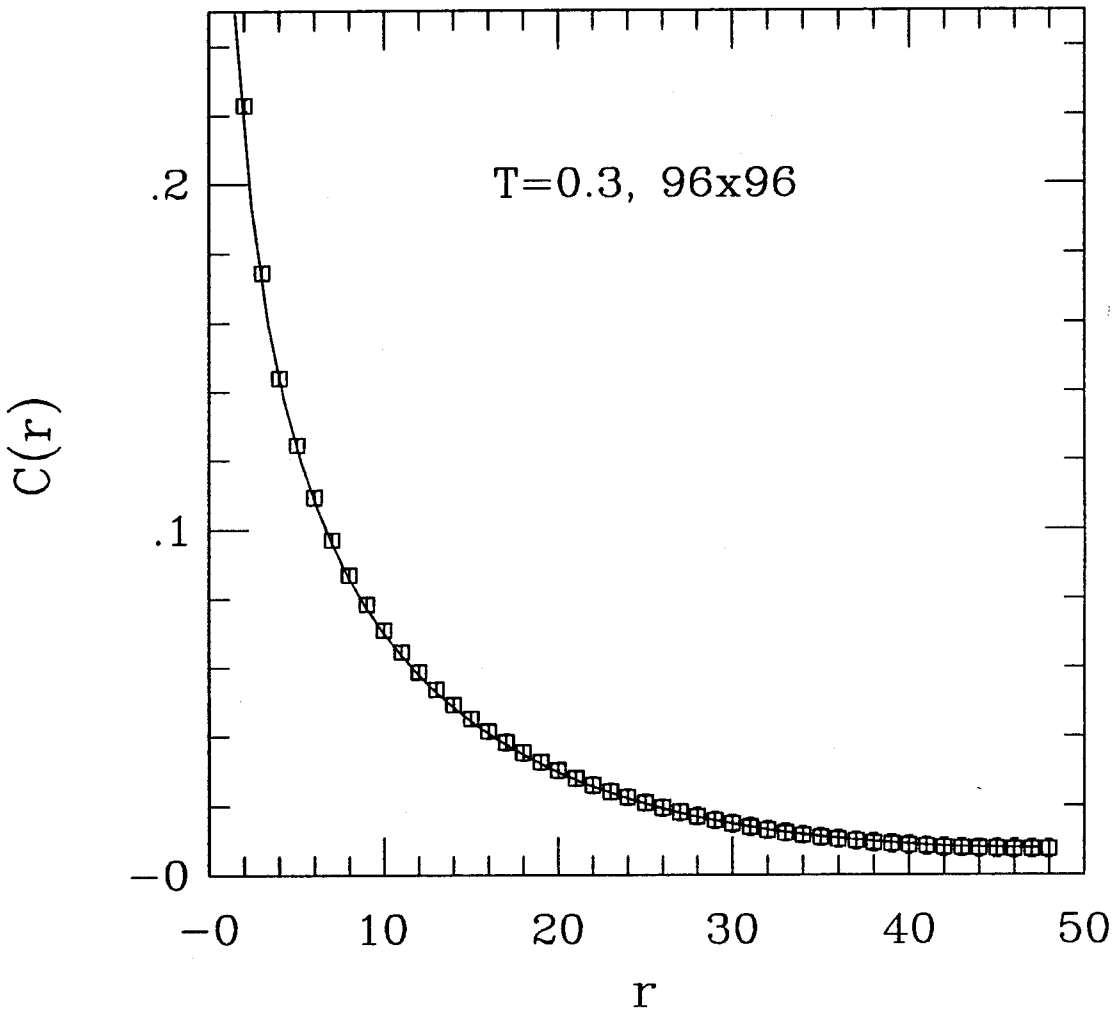


Fig. 14d

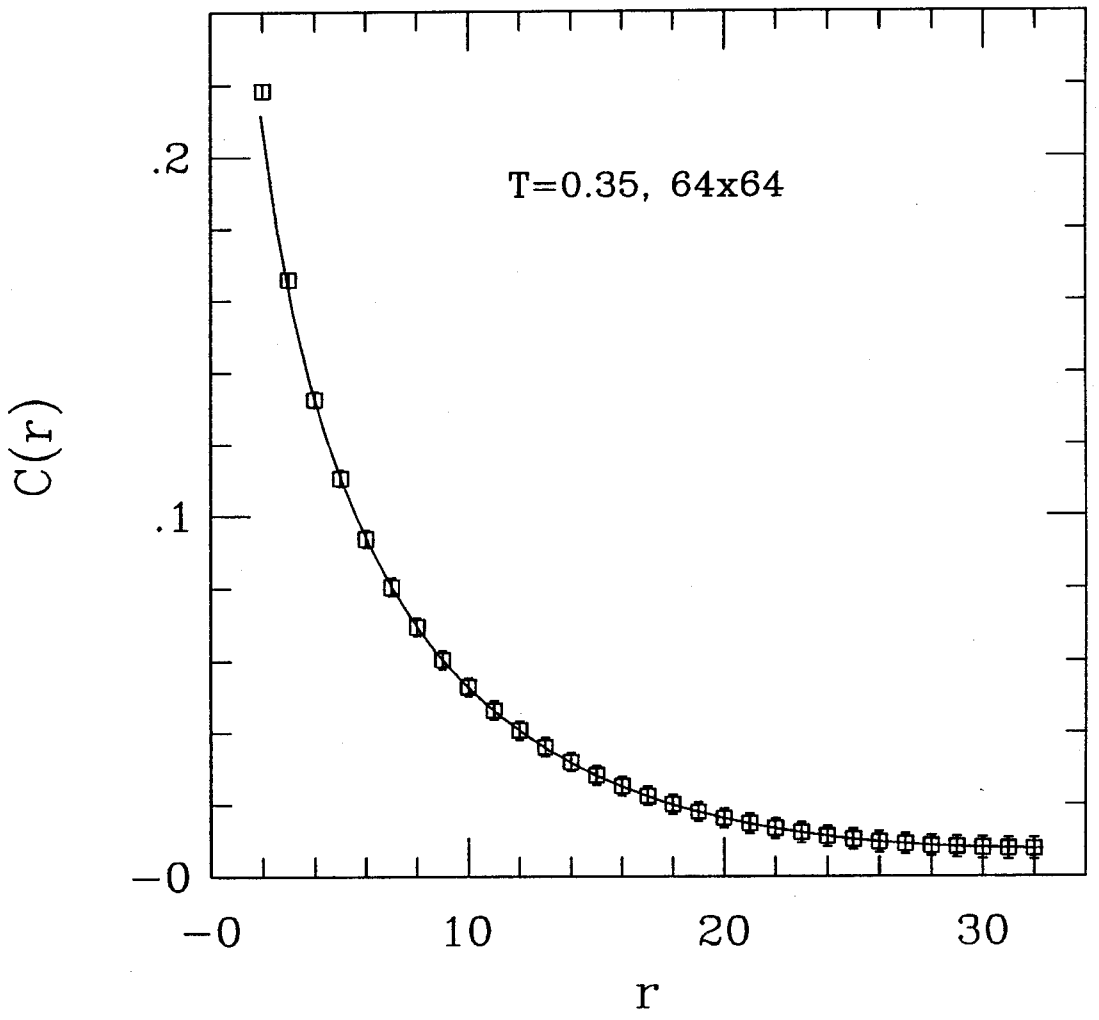


Fig. 14e

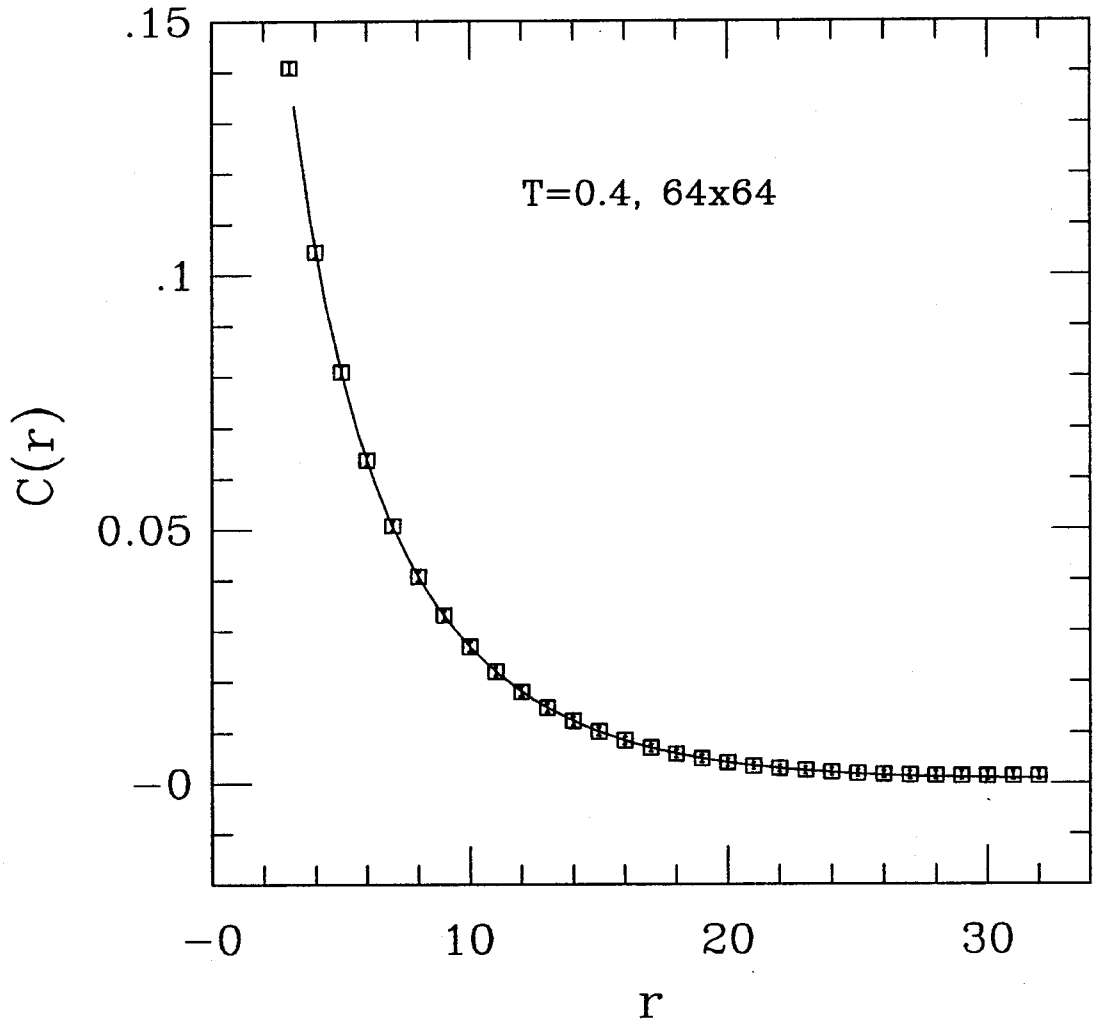


Fig. 15

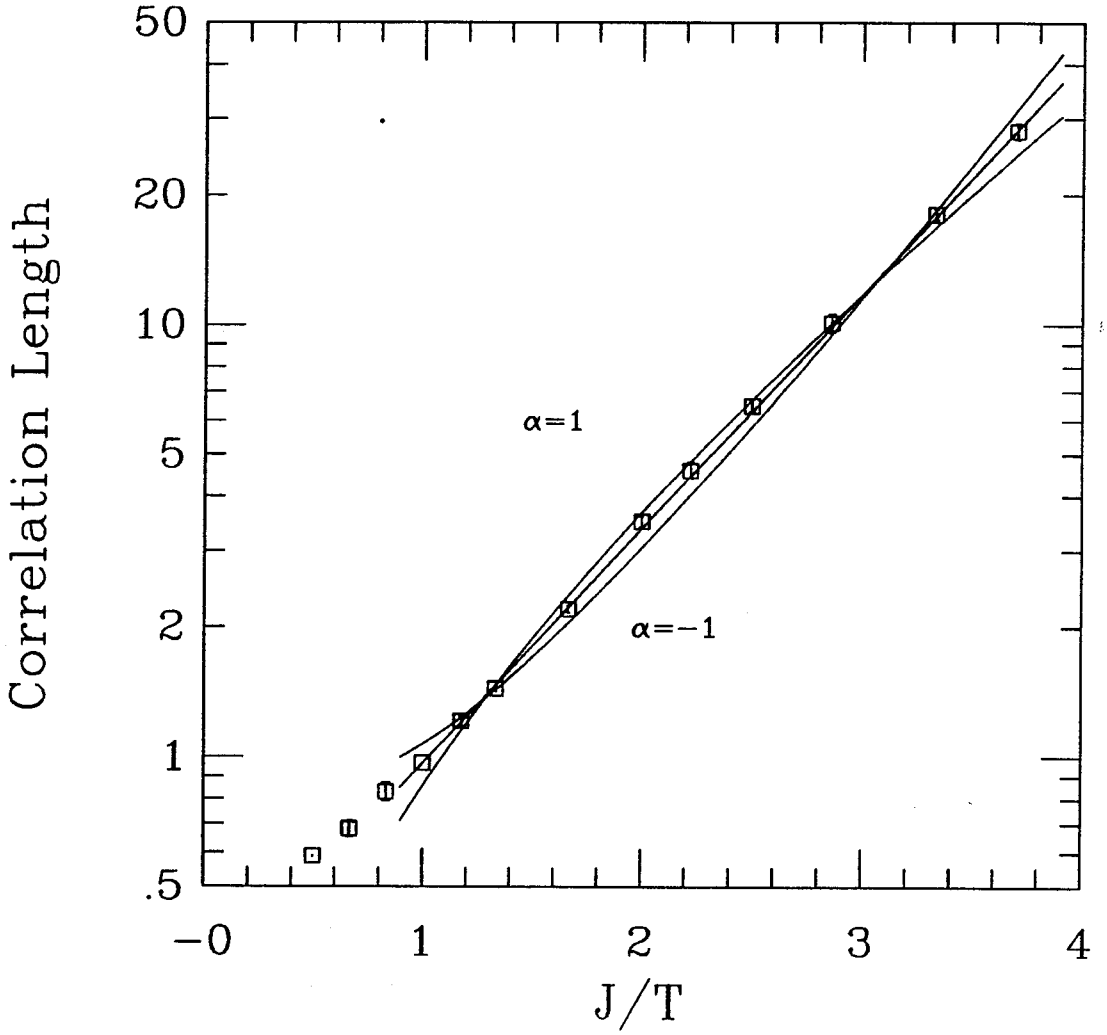


Fig. 16

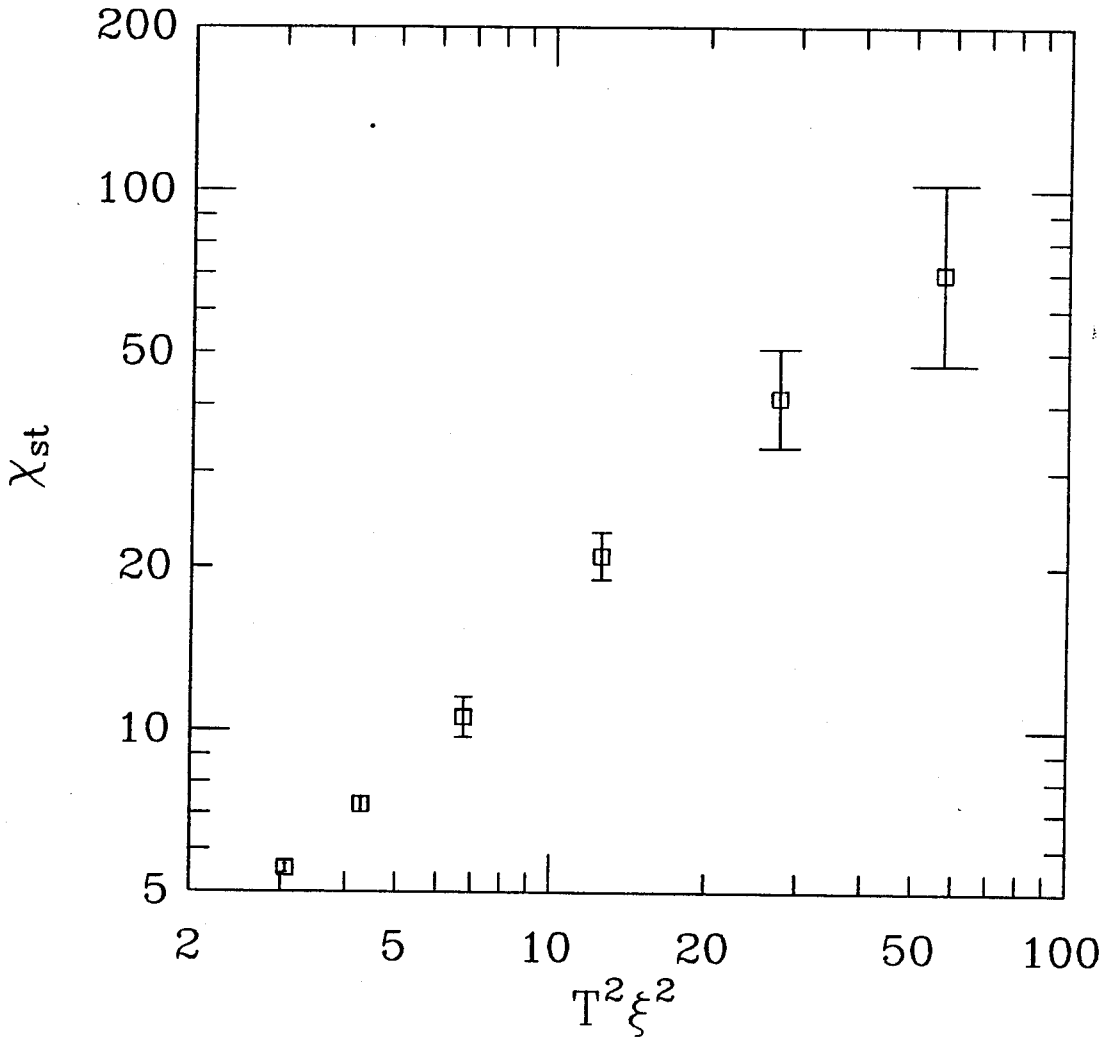
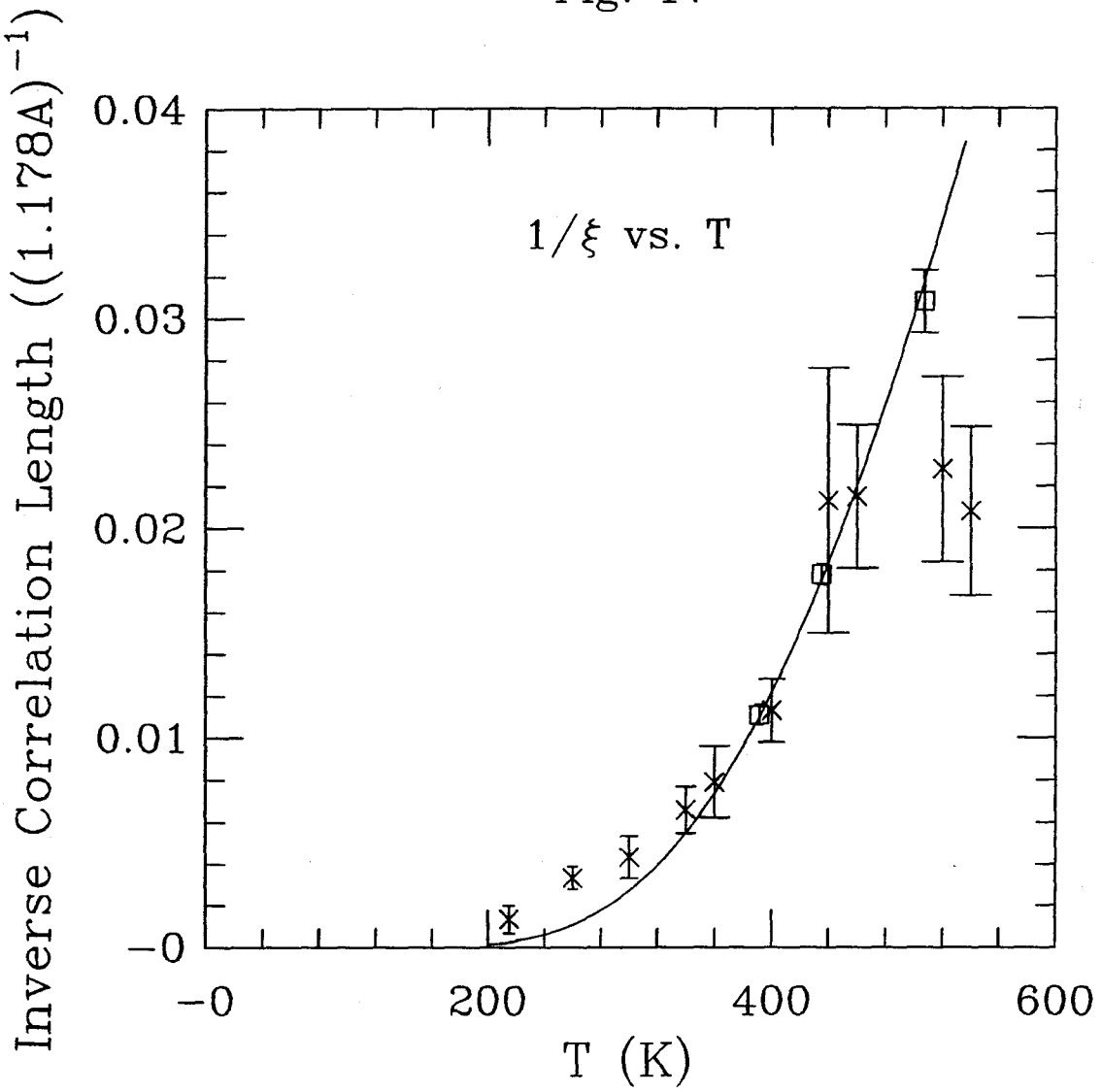


Fig. 17


Inverse Correlation Length ((1.178A)$^{-1}$)
0.04
0.03
0.02
0.01
−0
1/ξ vs. T
−0
200
400
600
T (K)

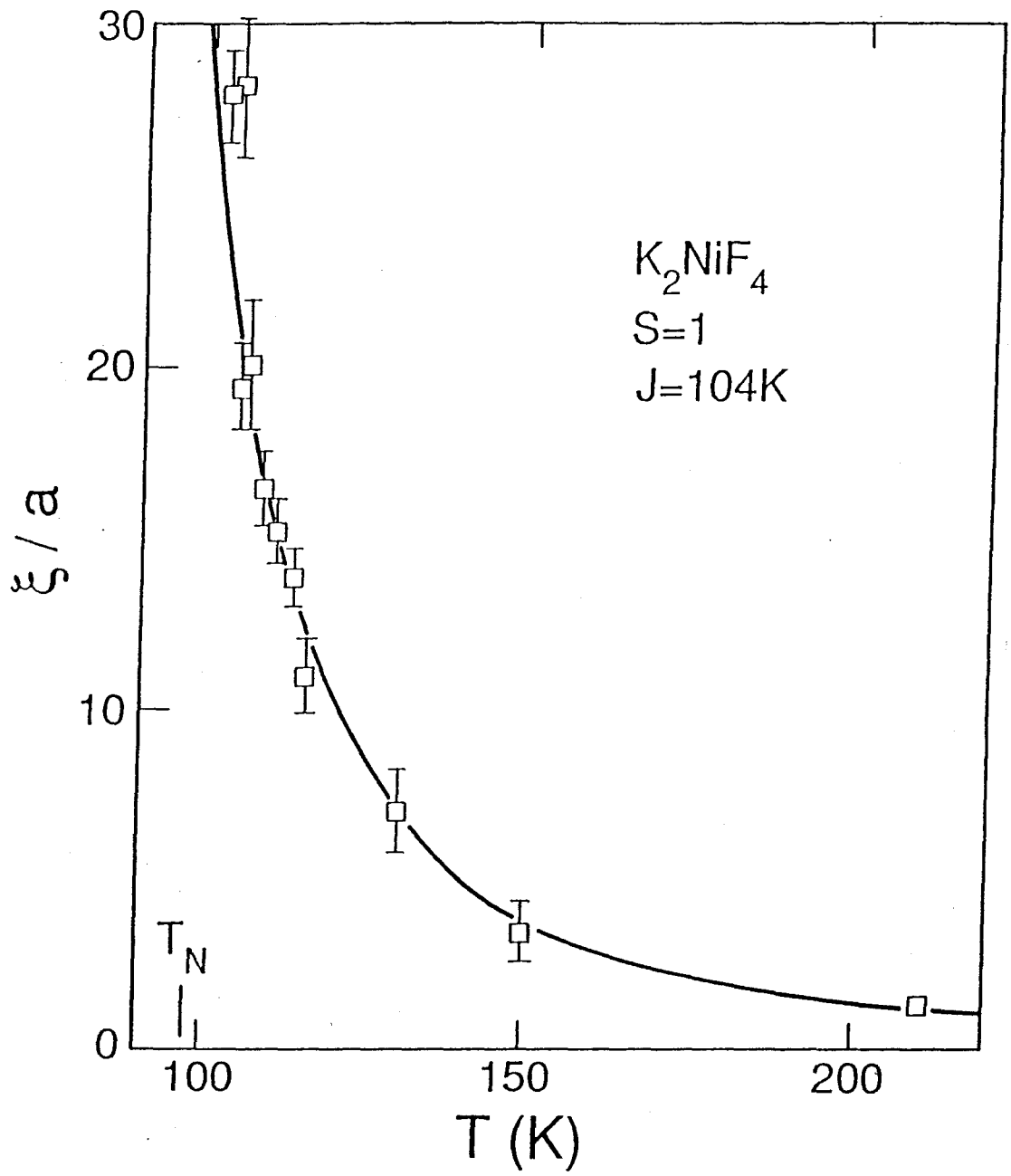


Fig. 18

Fig. 19

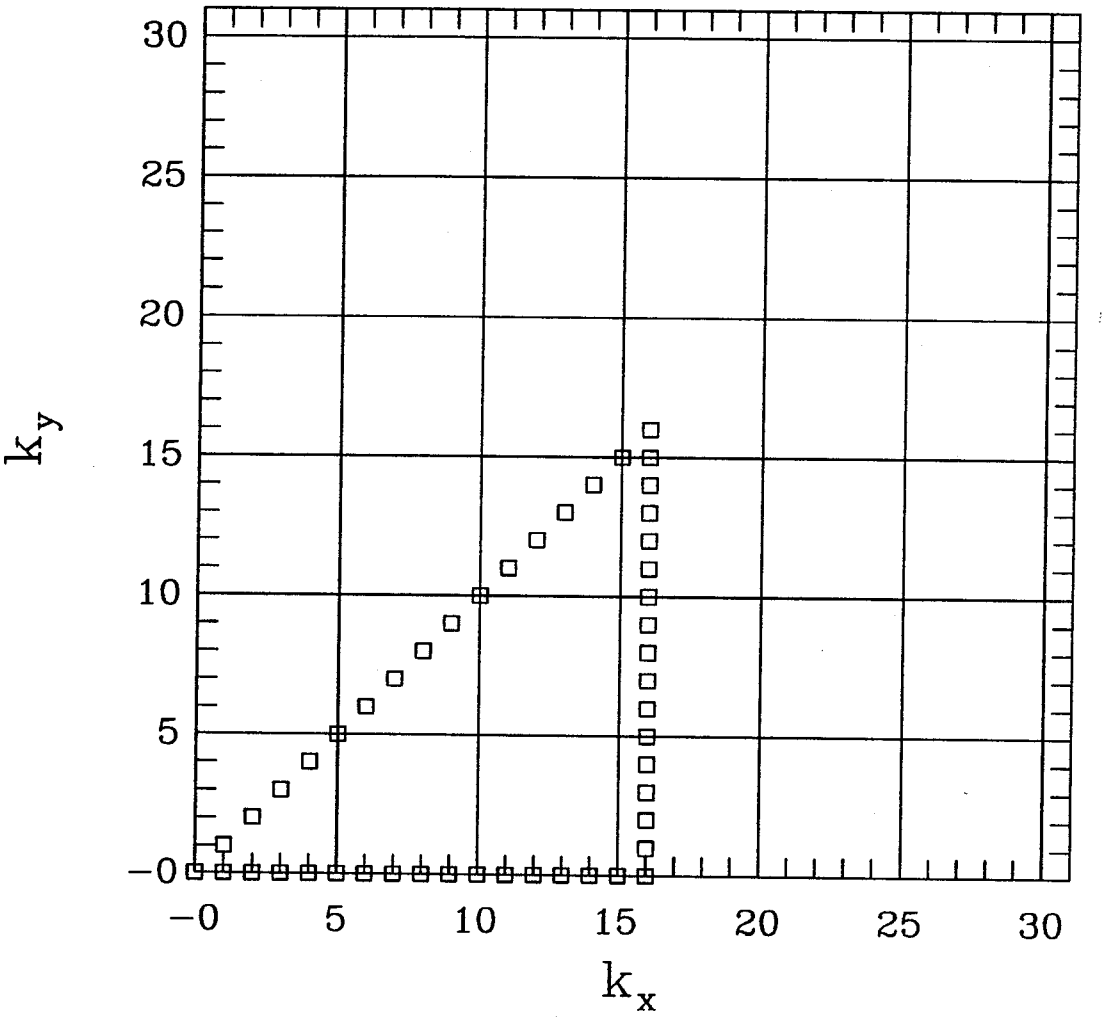


Fig. 20

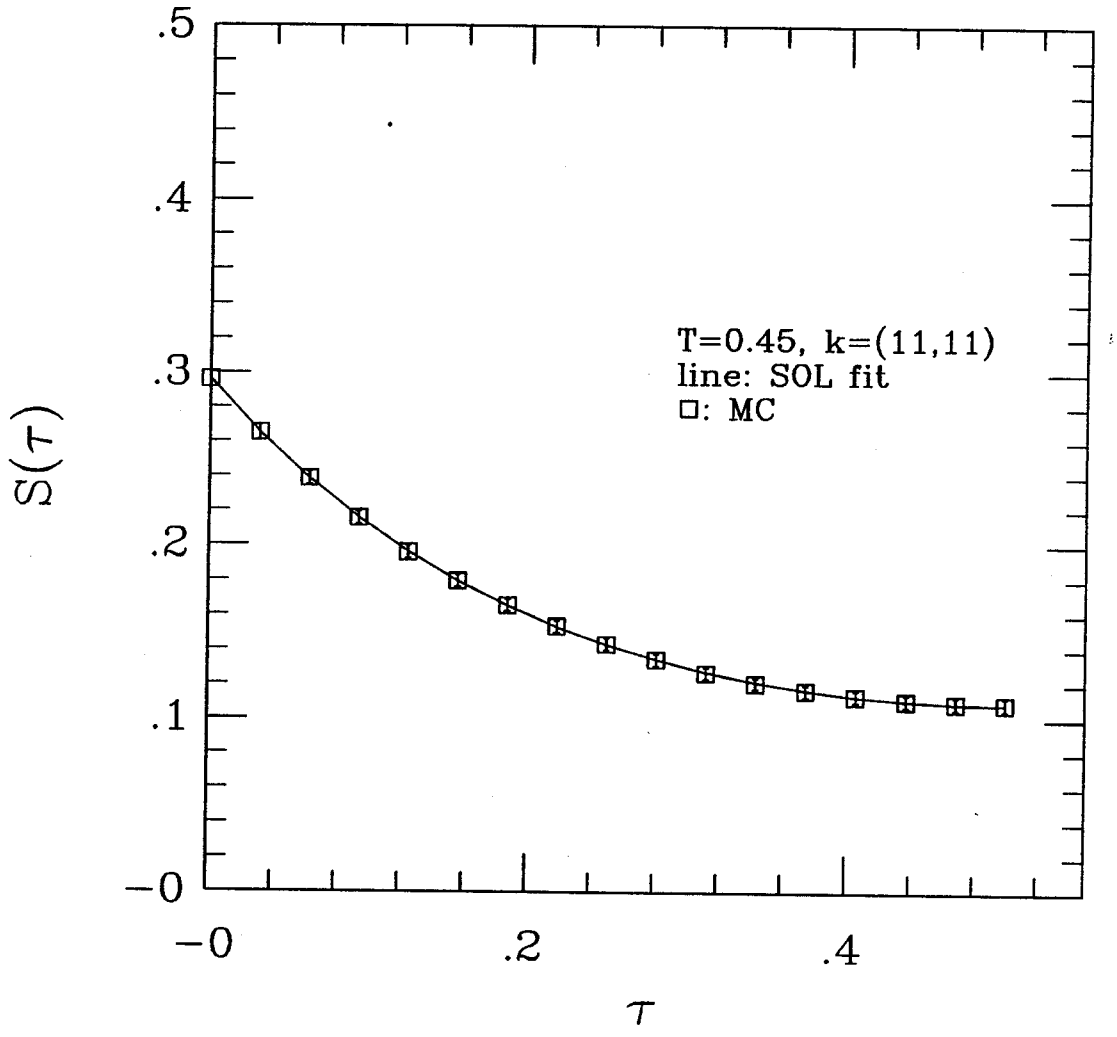


Fig. 21

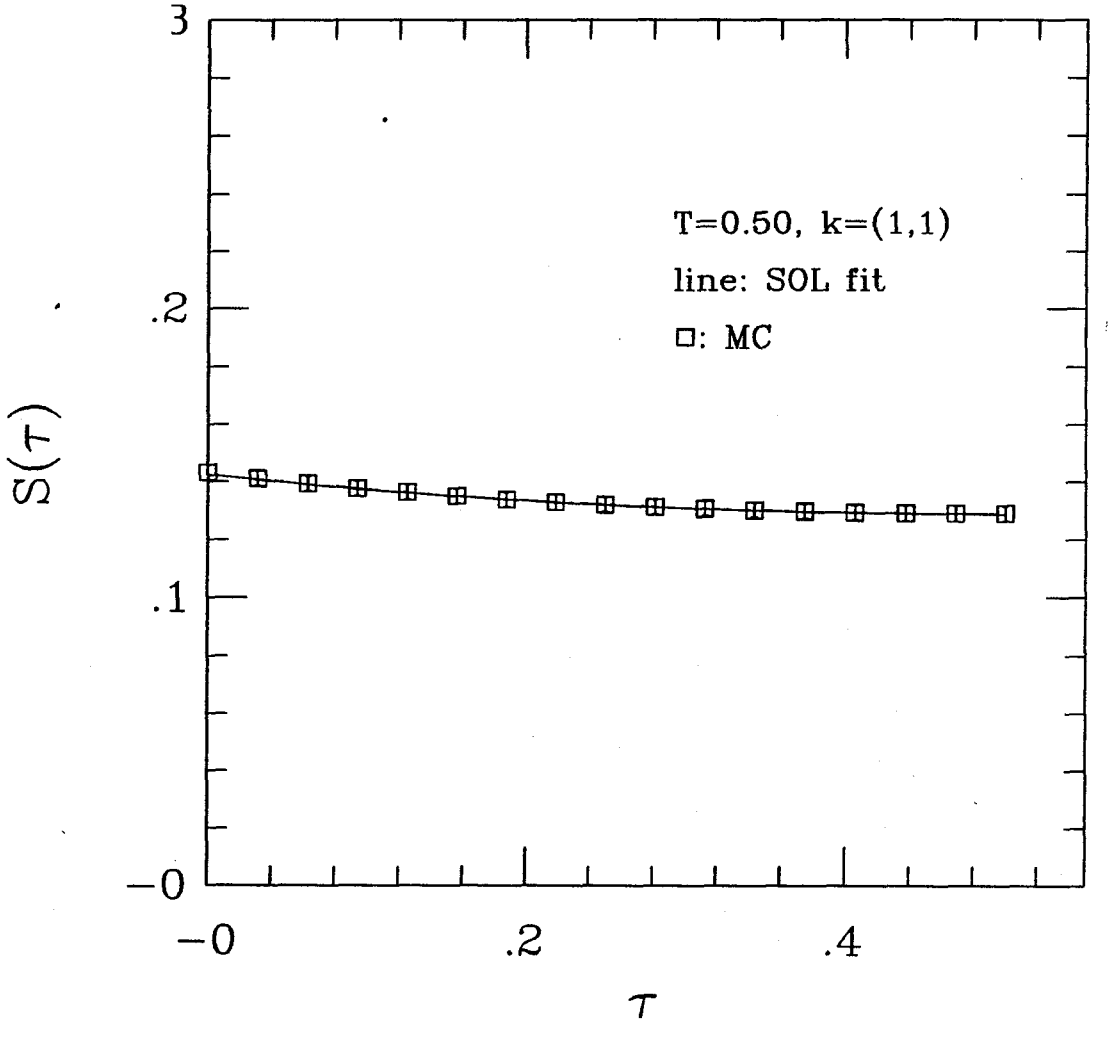


Fig. 22a

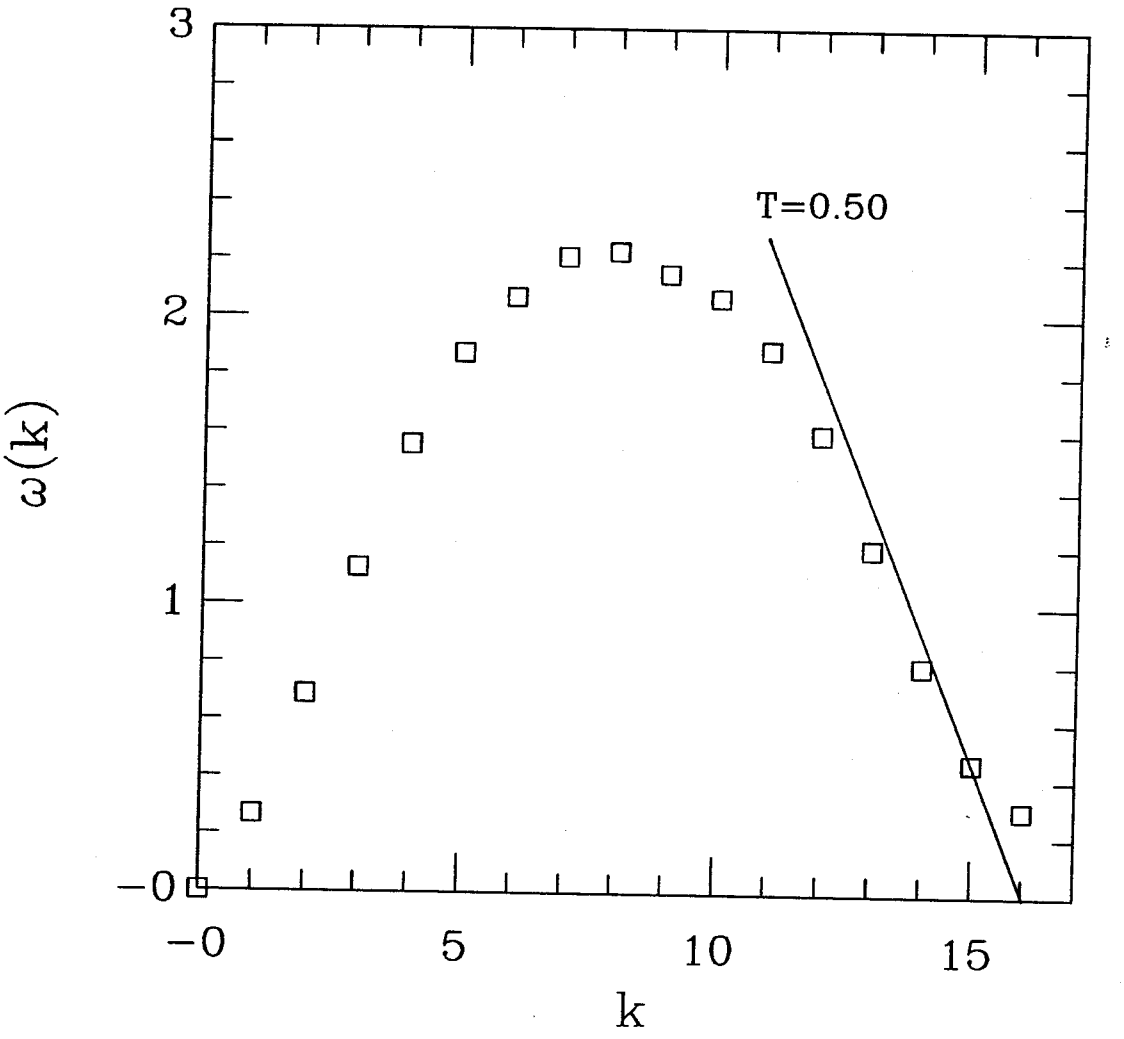


Fig. 22b

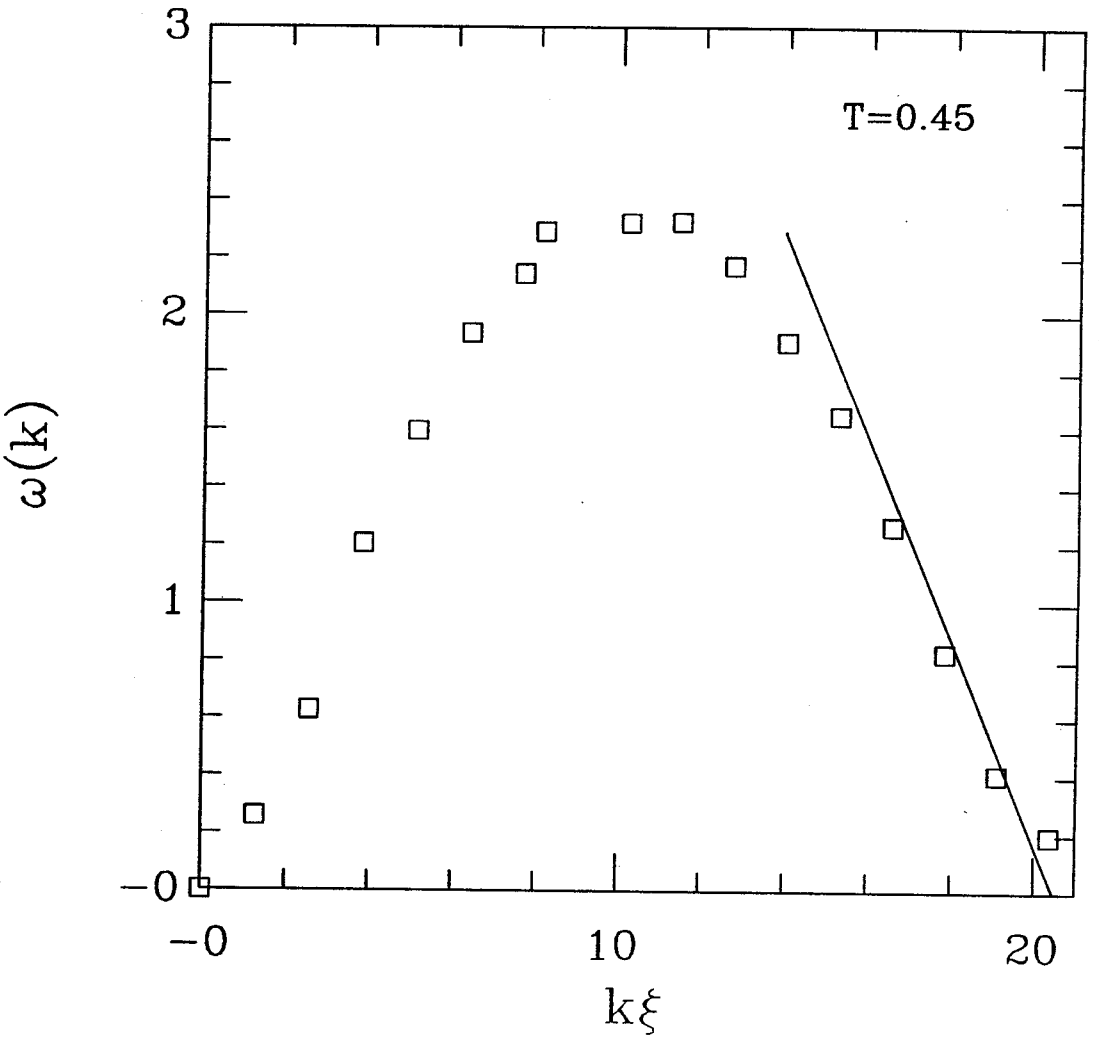


Fig. 23a

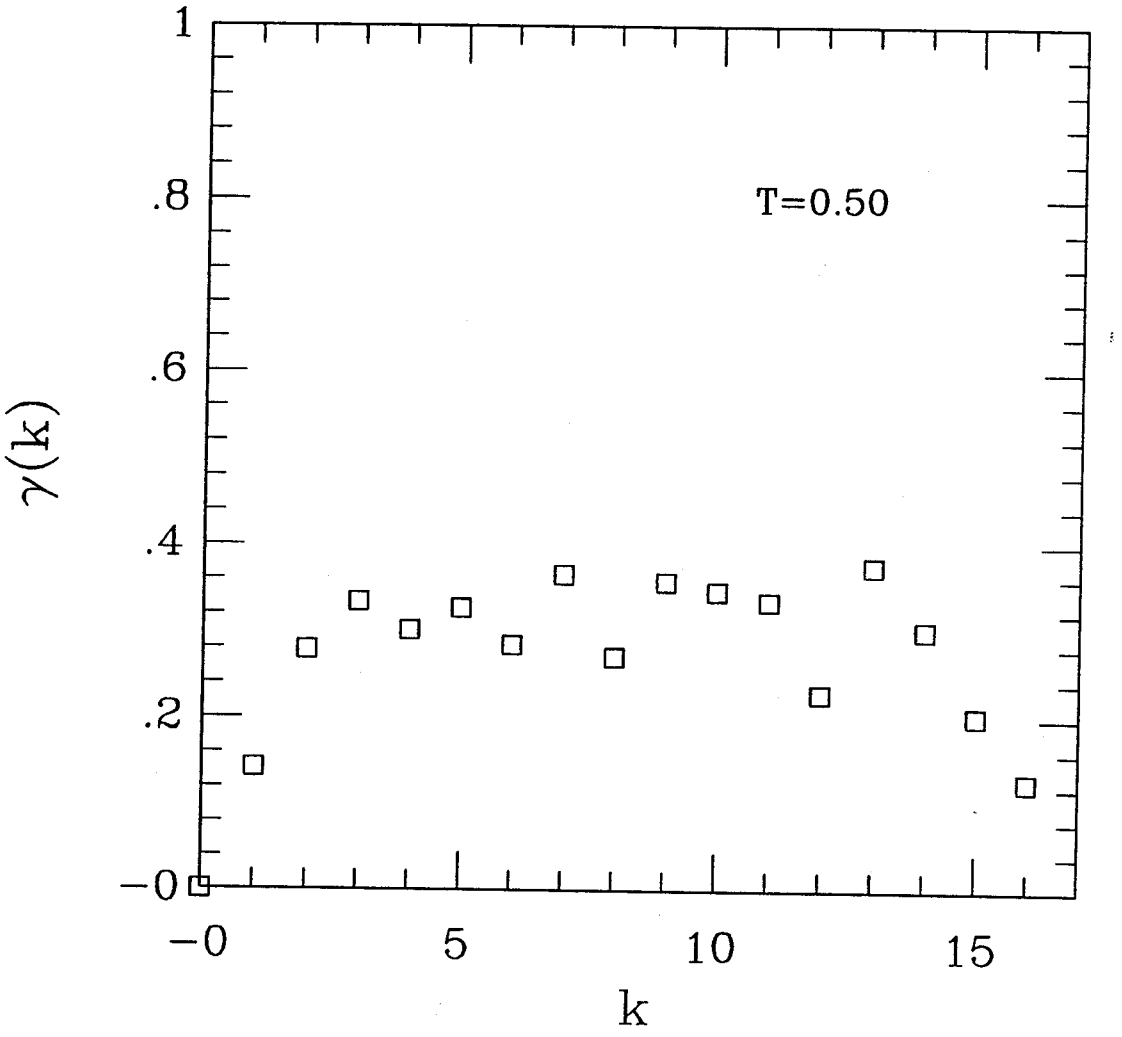


Fig. 23b

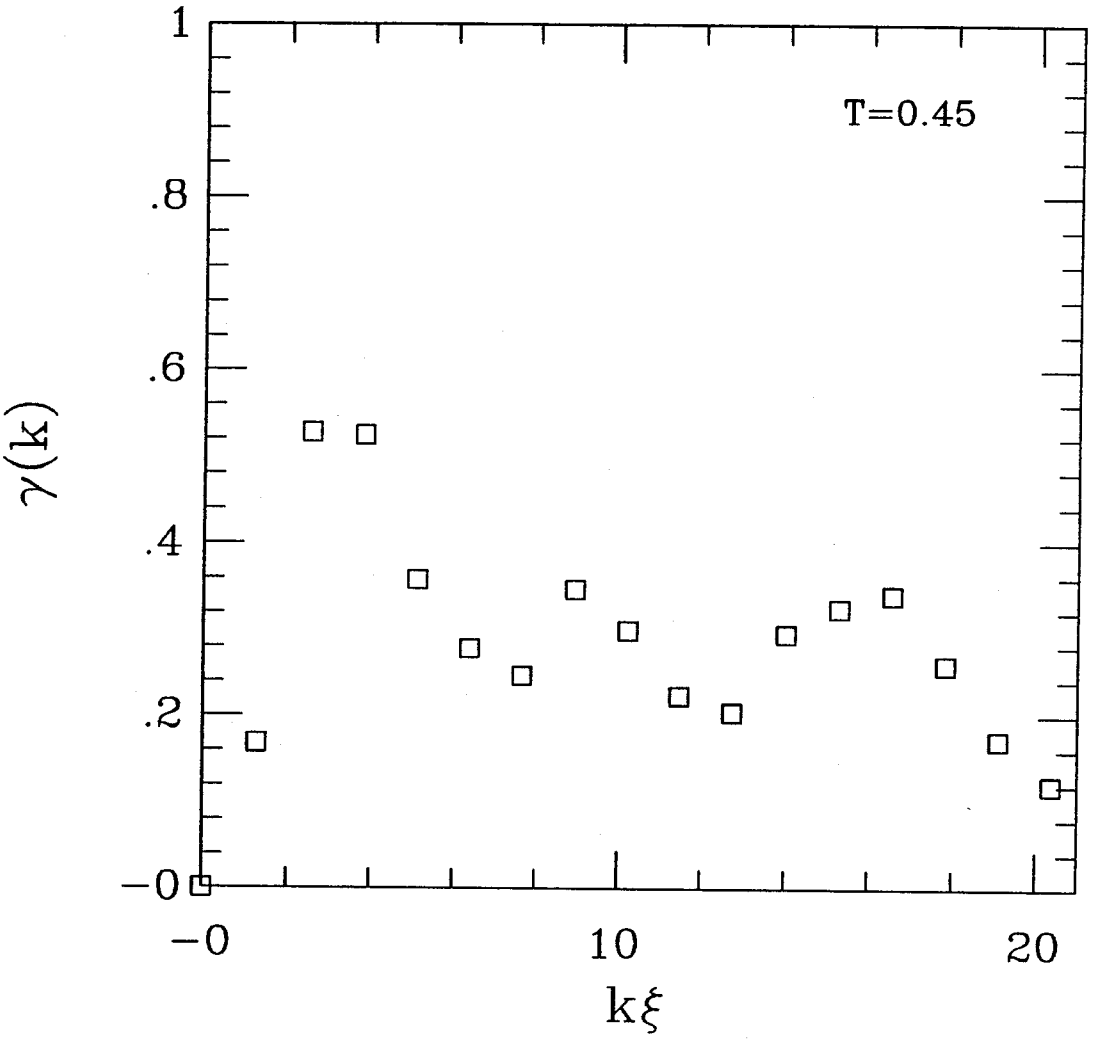


Fig. 24a

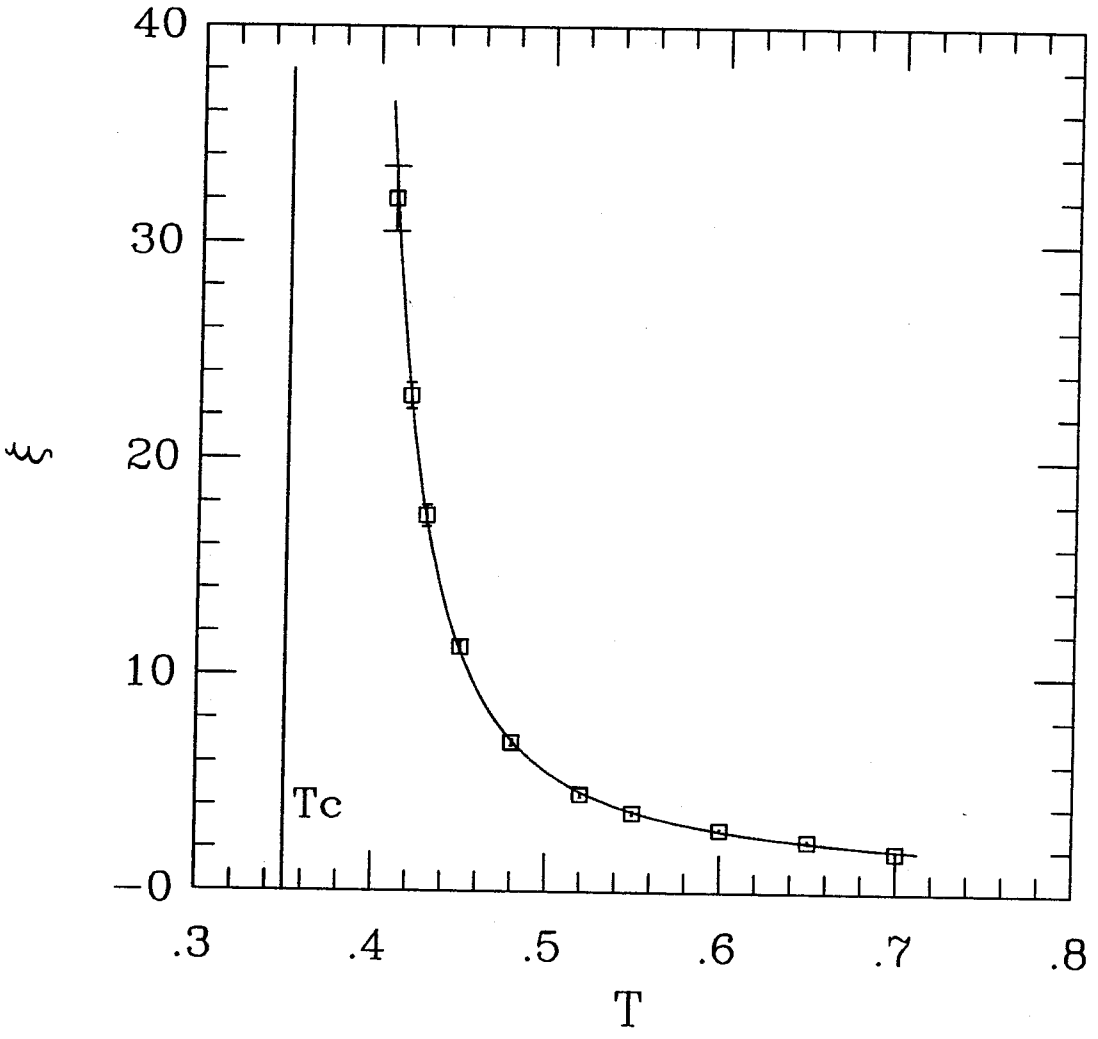


Fig. 24b

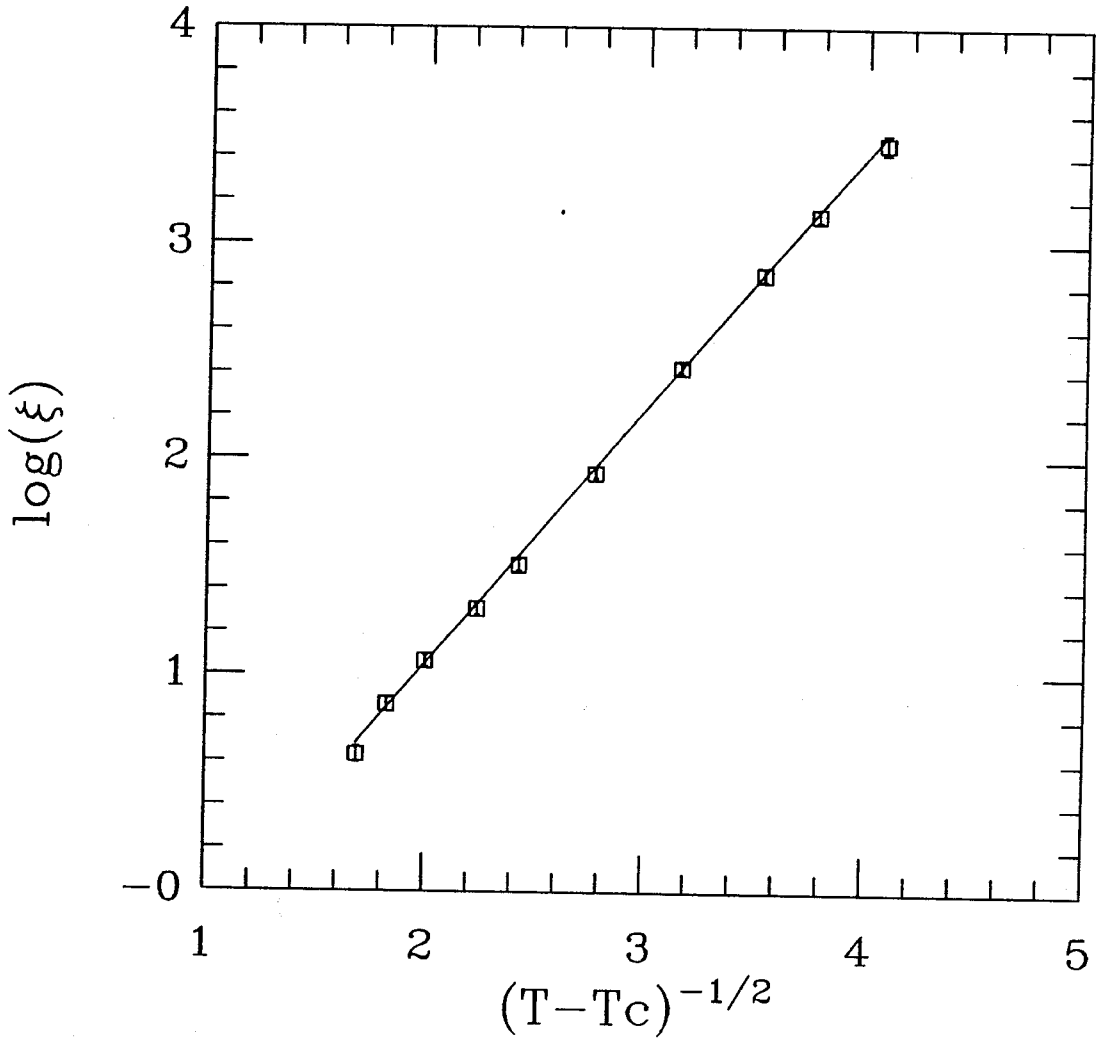


Fig. 25

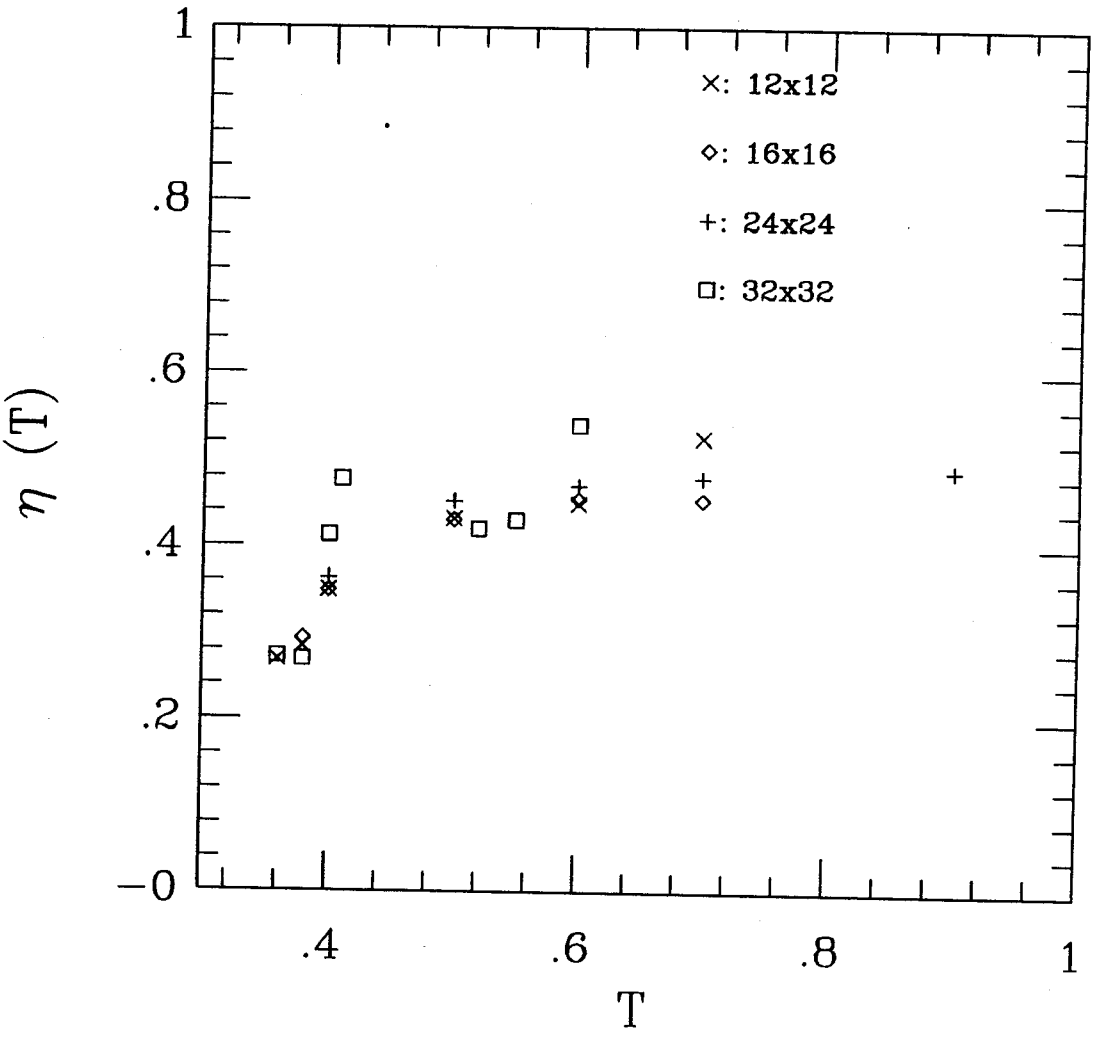


Fig. 26

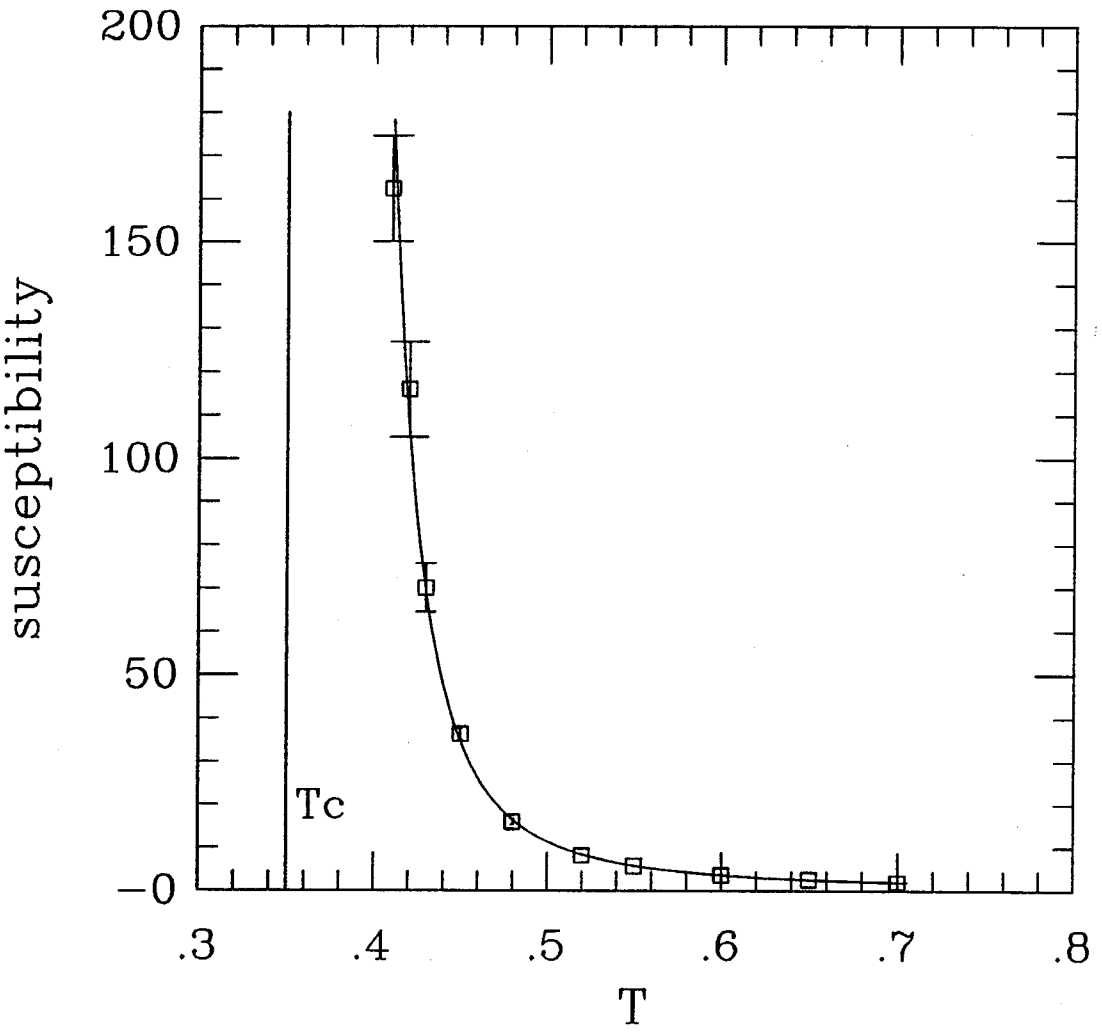


Fig. 27a

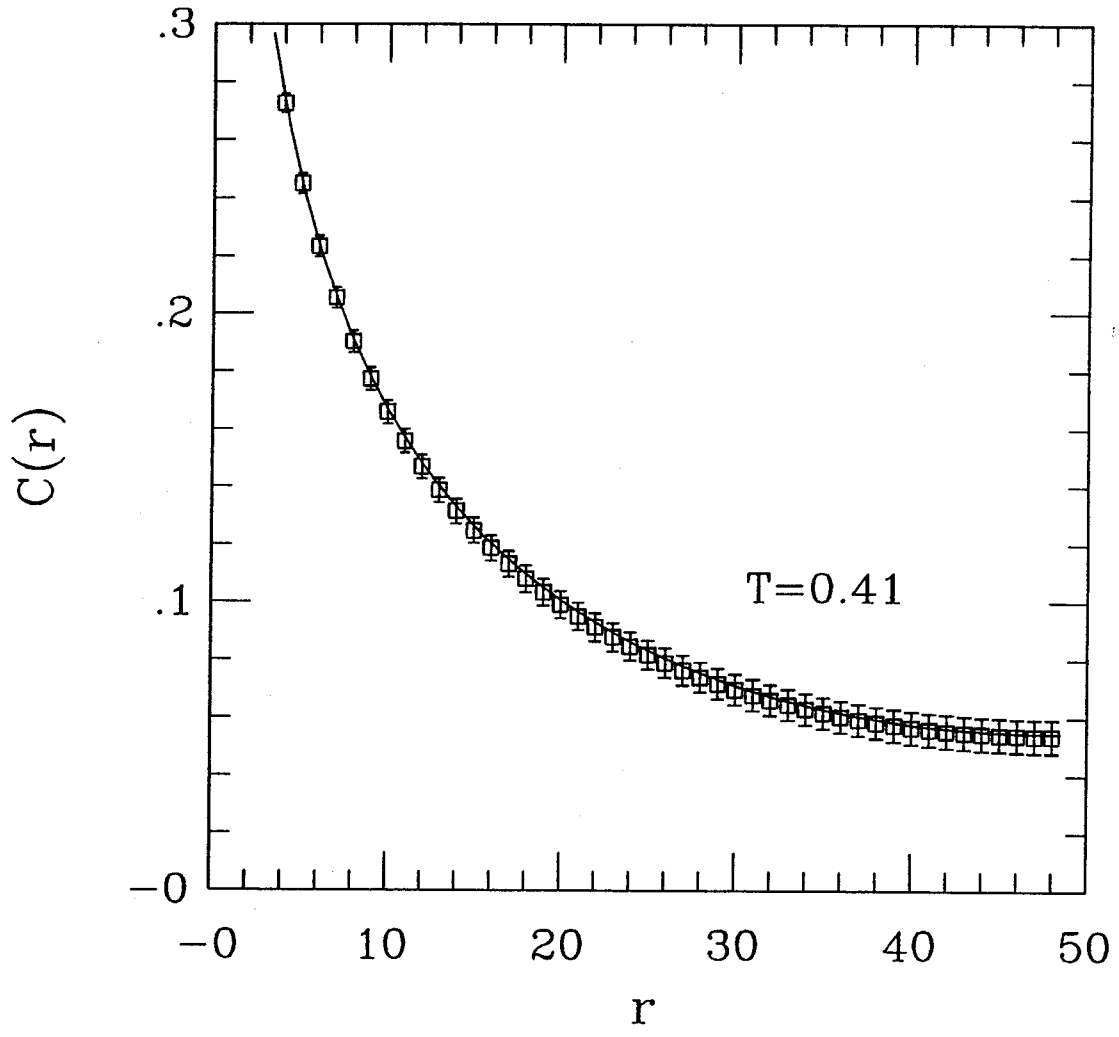


Fig. 27b

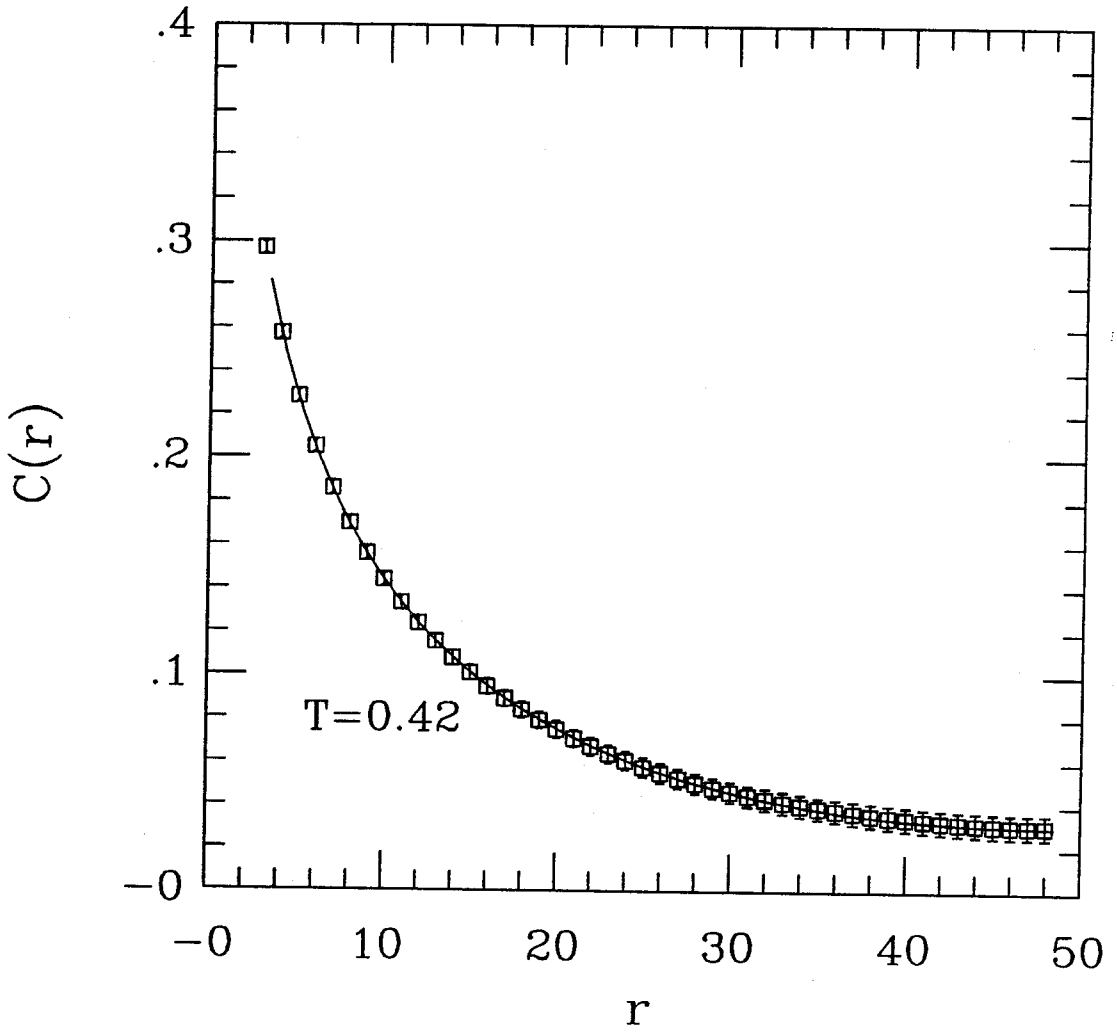


Fig. 28a

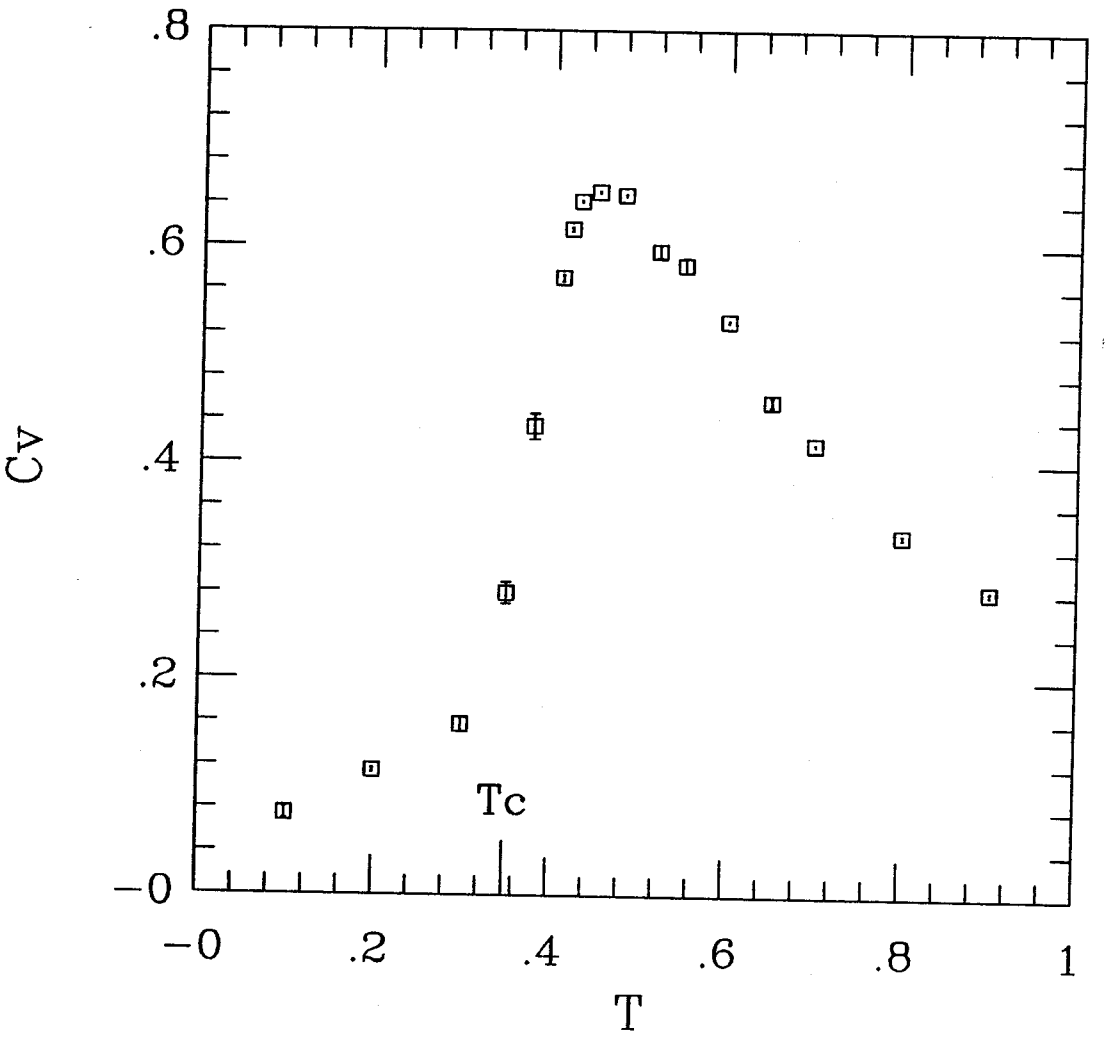


Fig. 28b

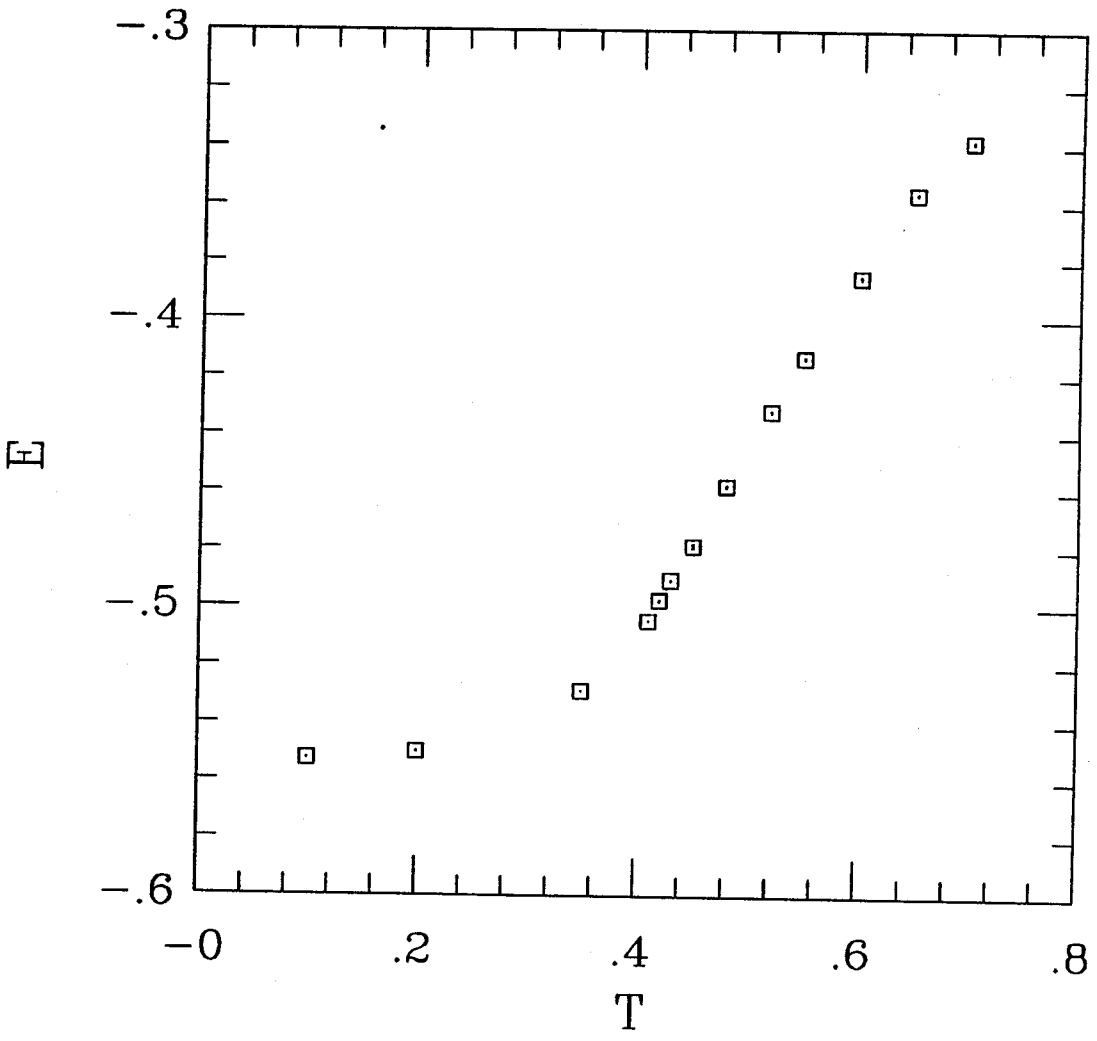


Fig. 29

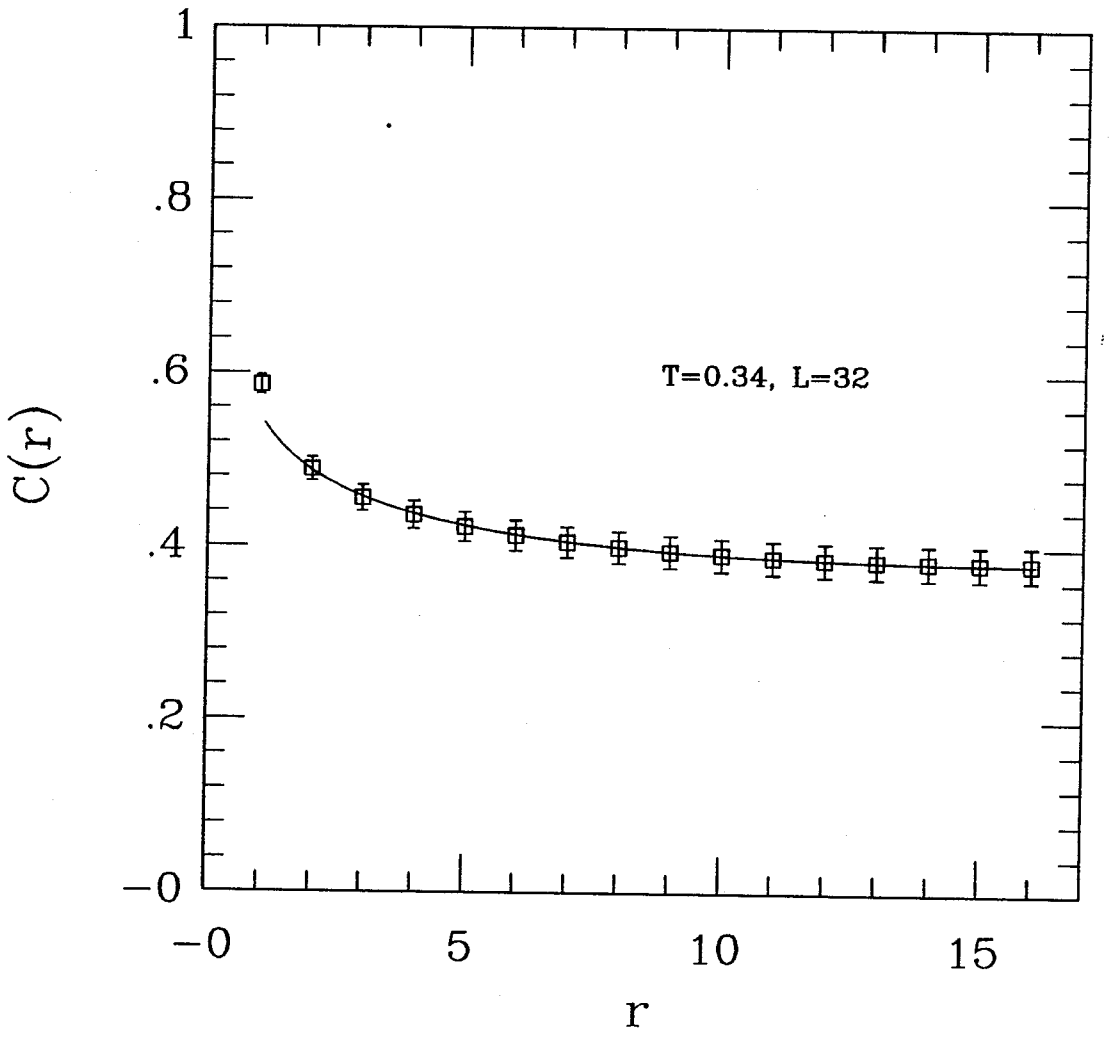


Fig. 30

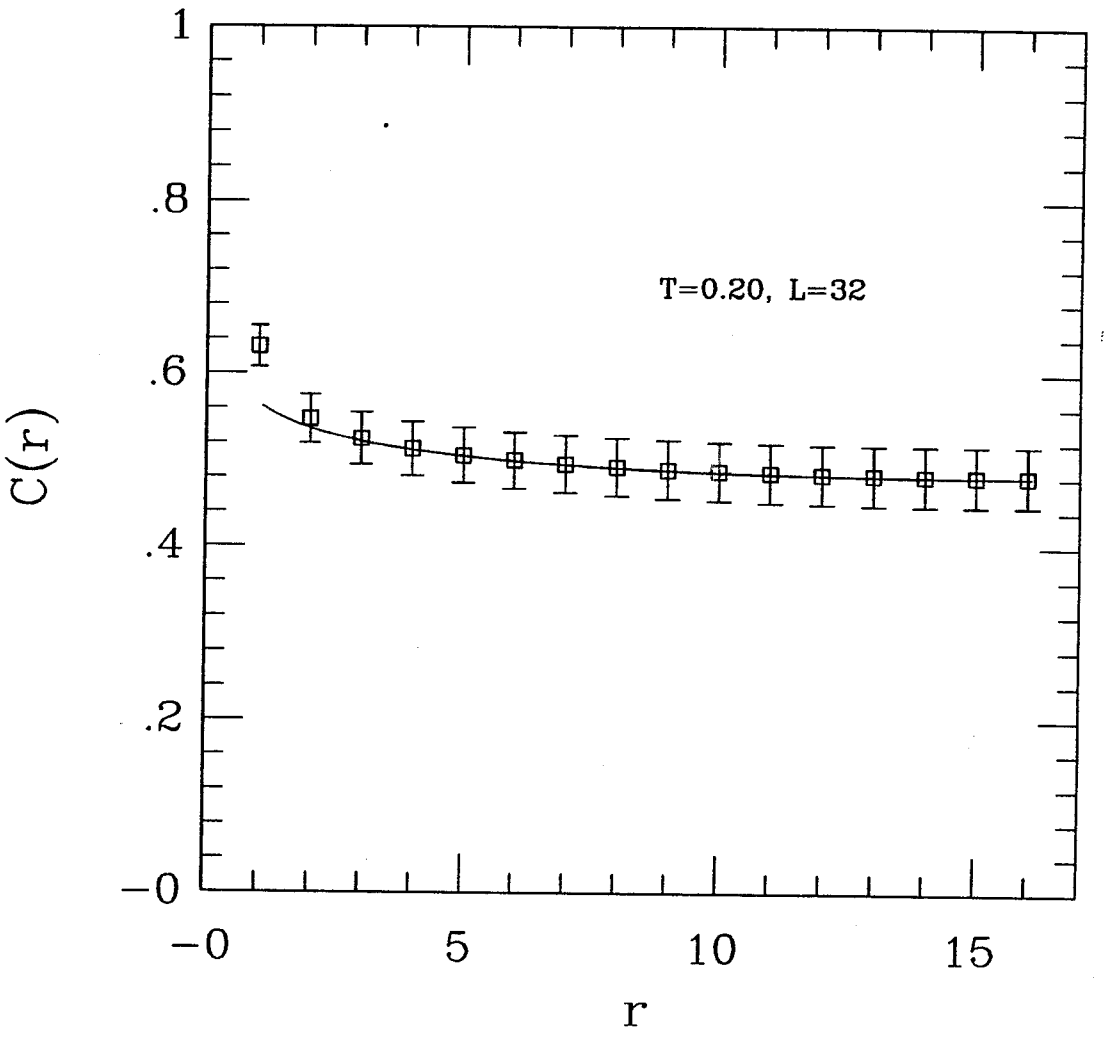


Fig. 31

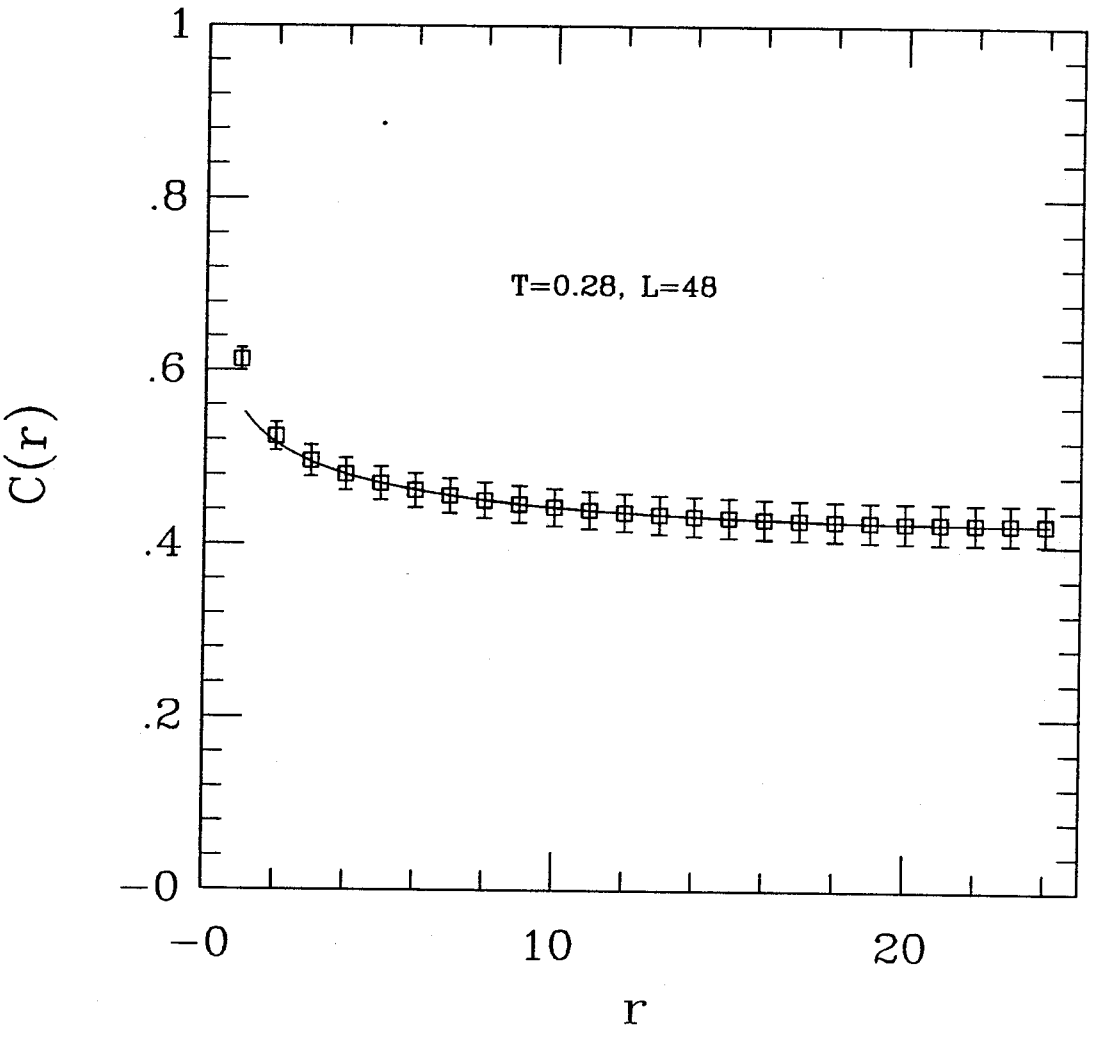


Fig. 32

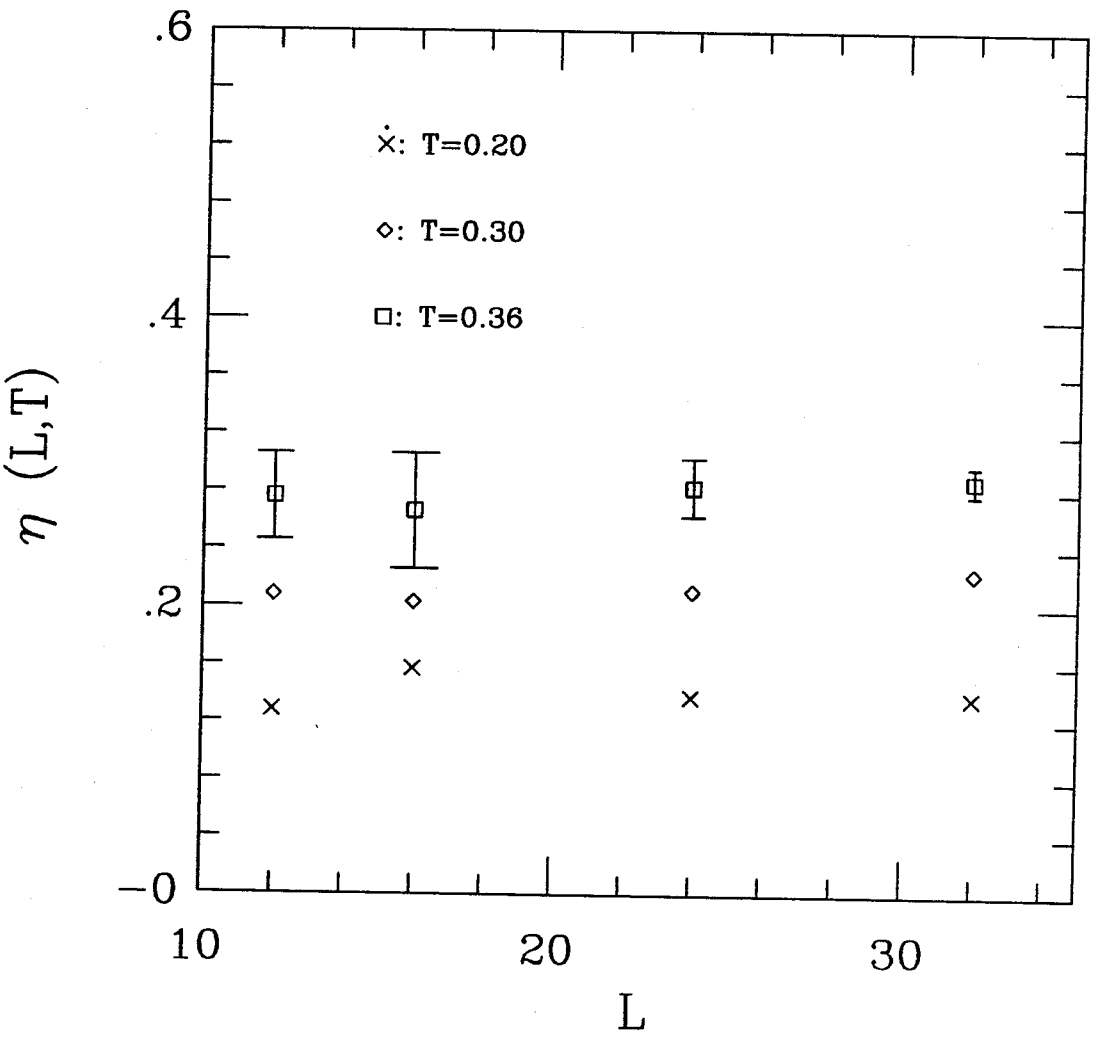


Fig. 33

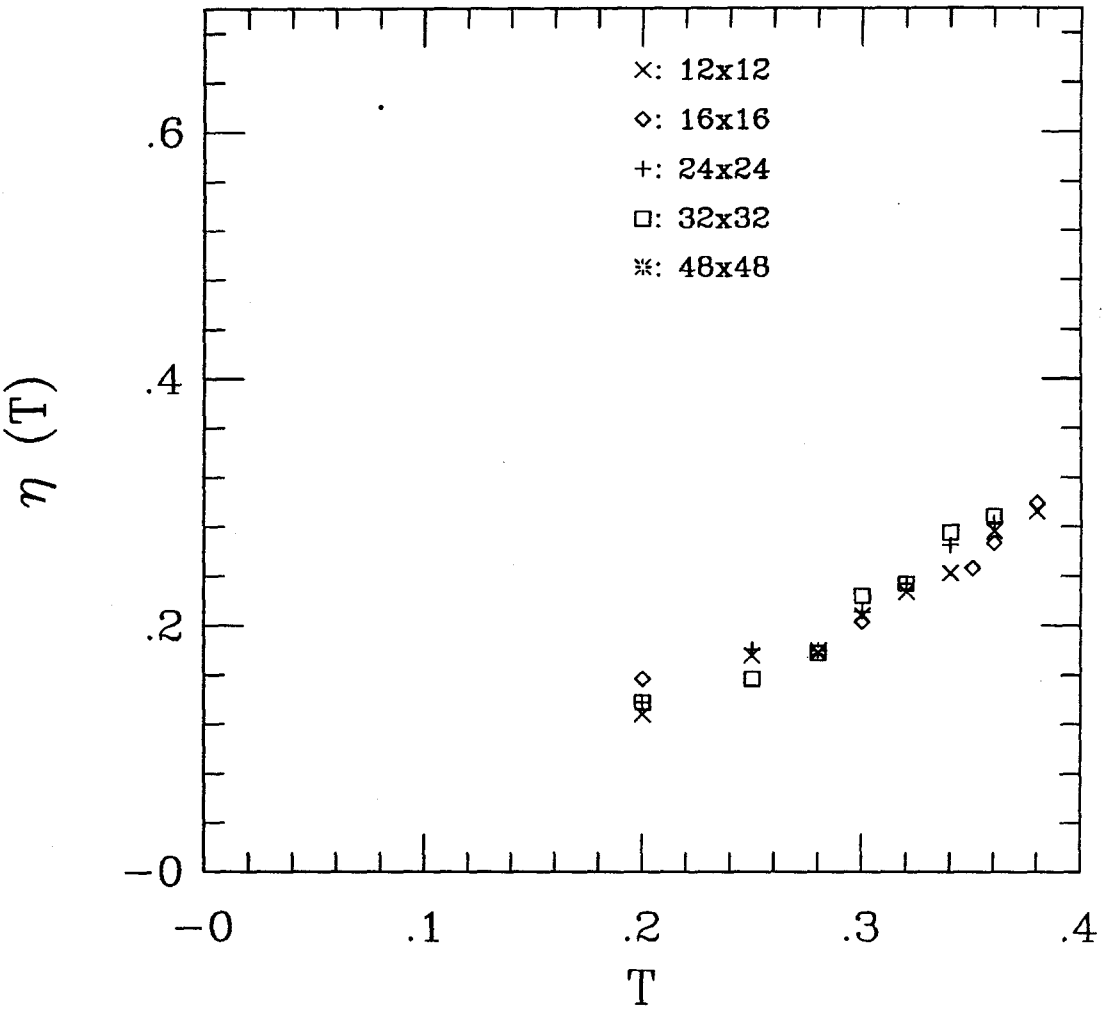


Fig. 34a

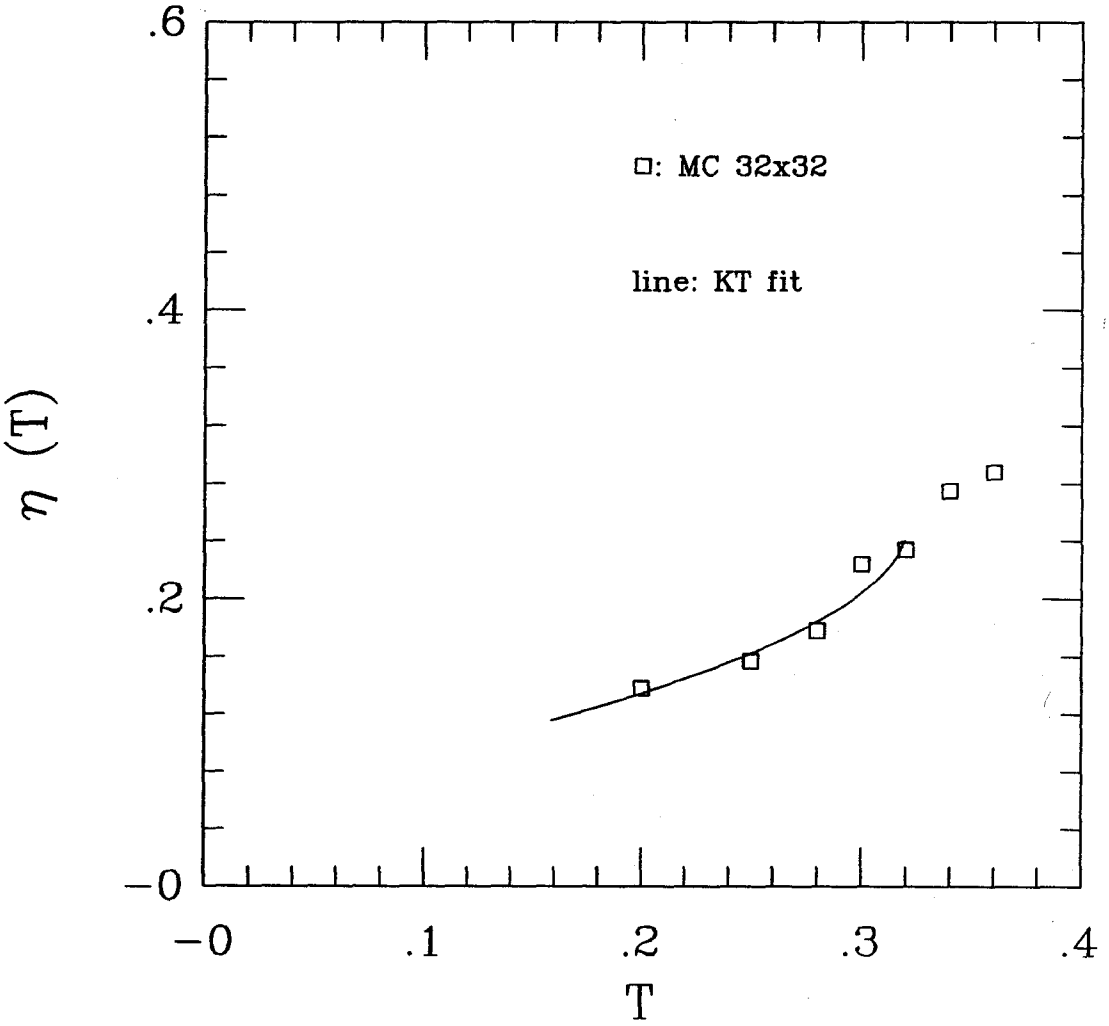


Fig. 34b

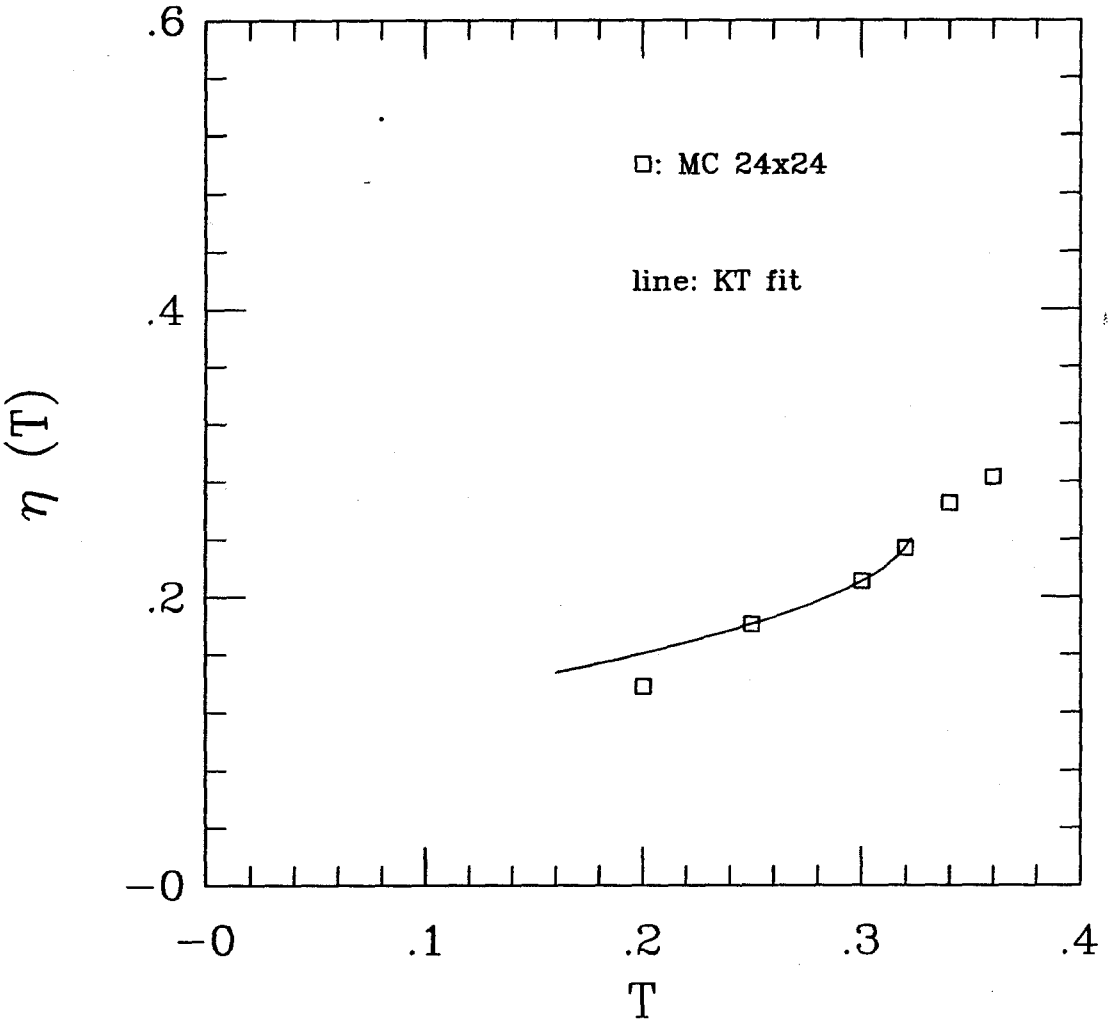


Fig. 35

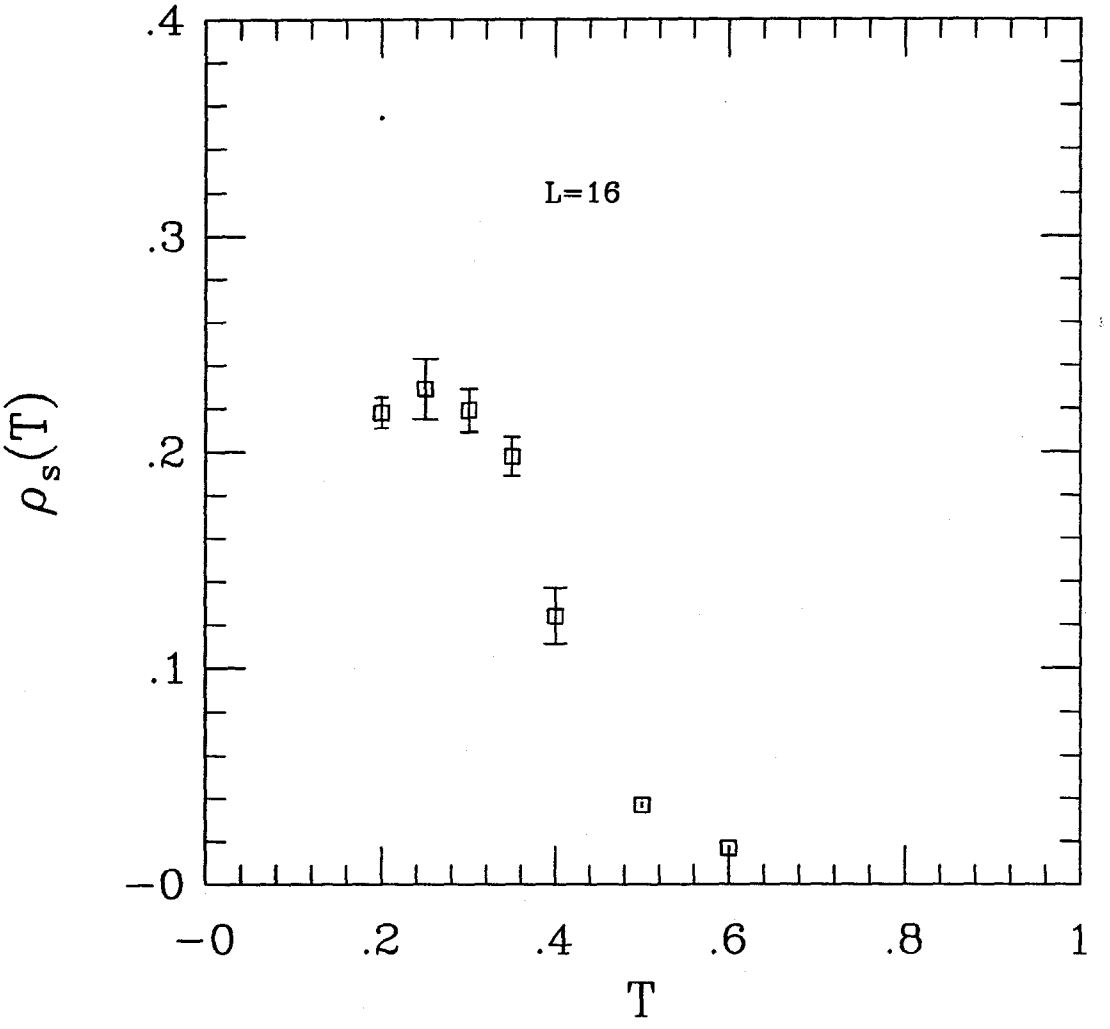


Fig. 36

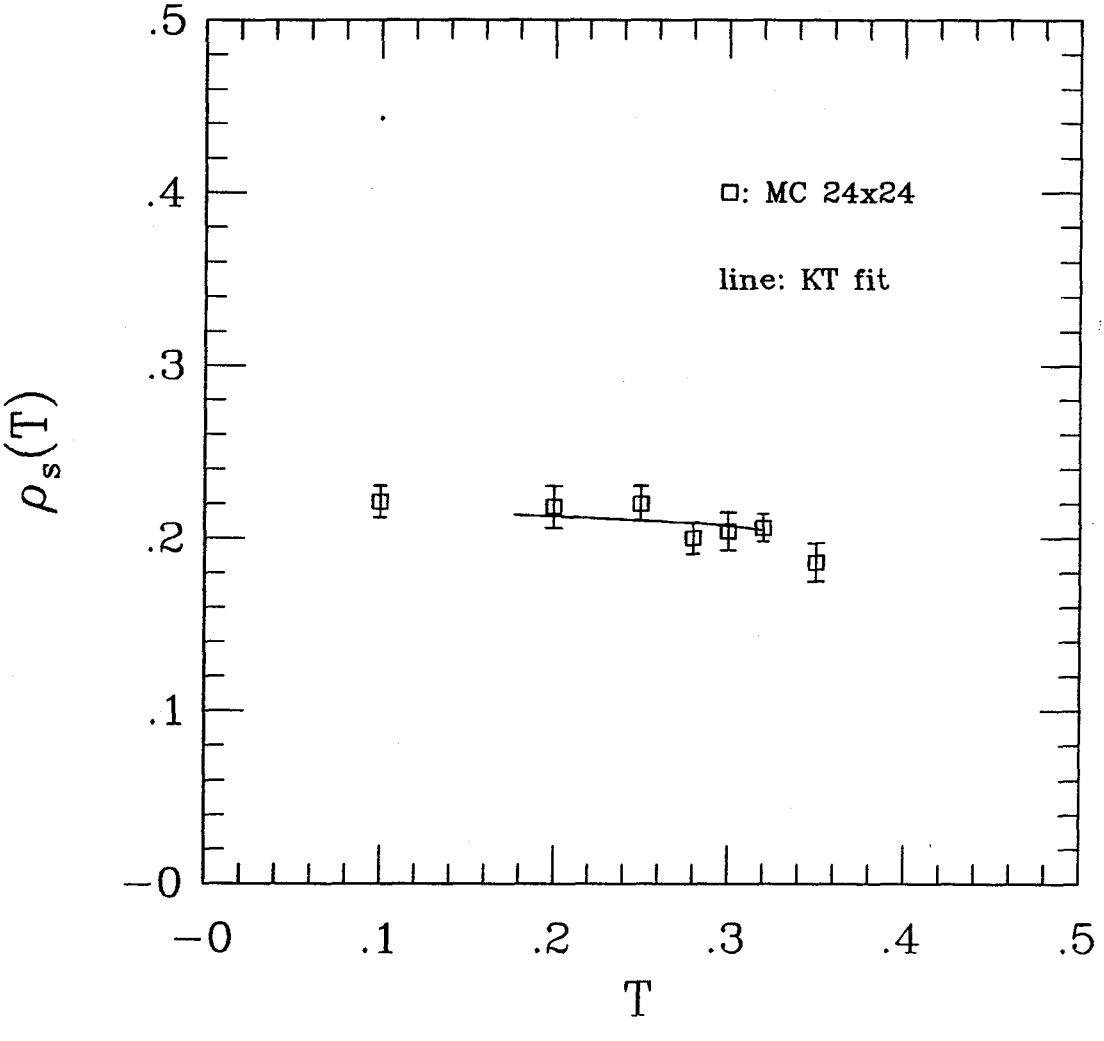


Fig. 37a

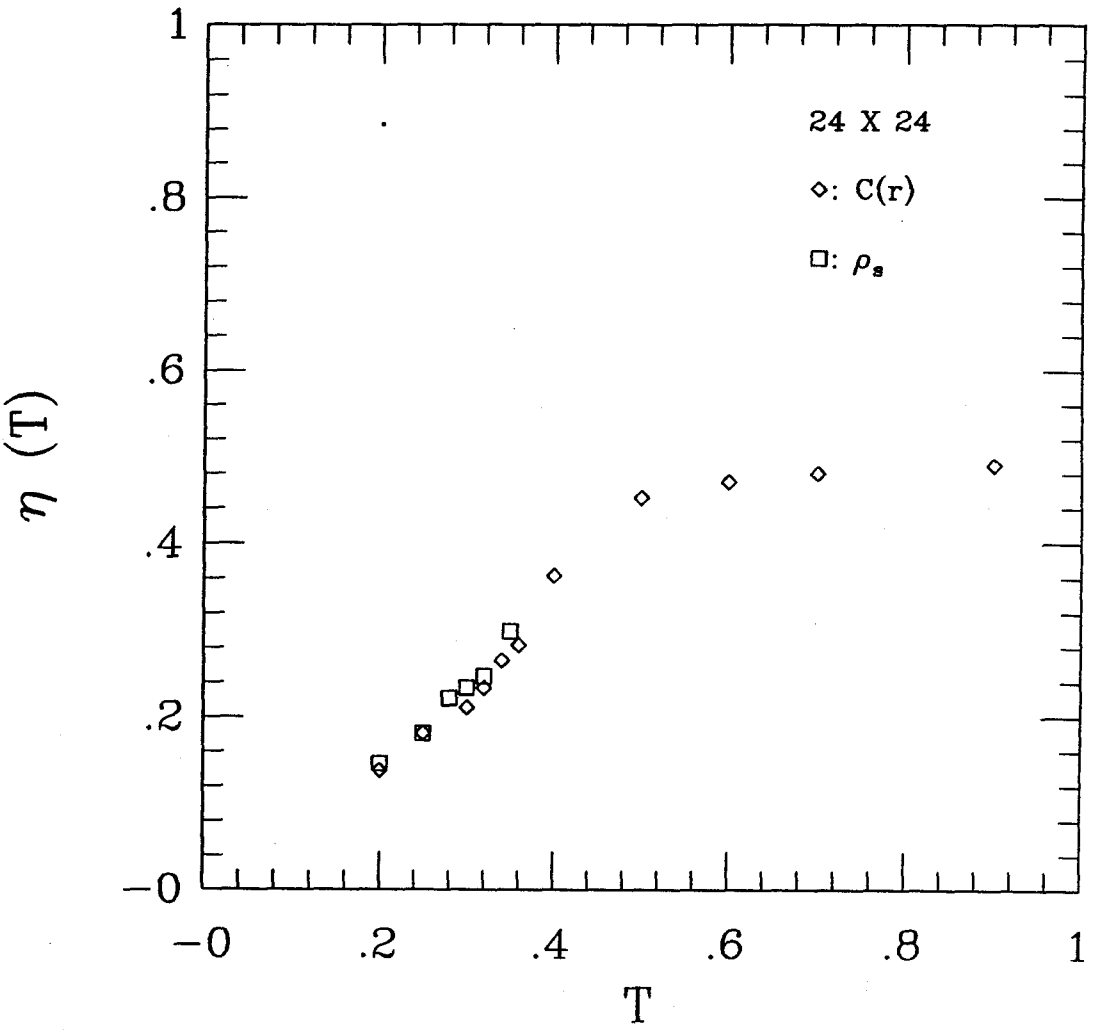


Fig. 37b

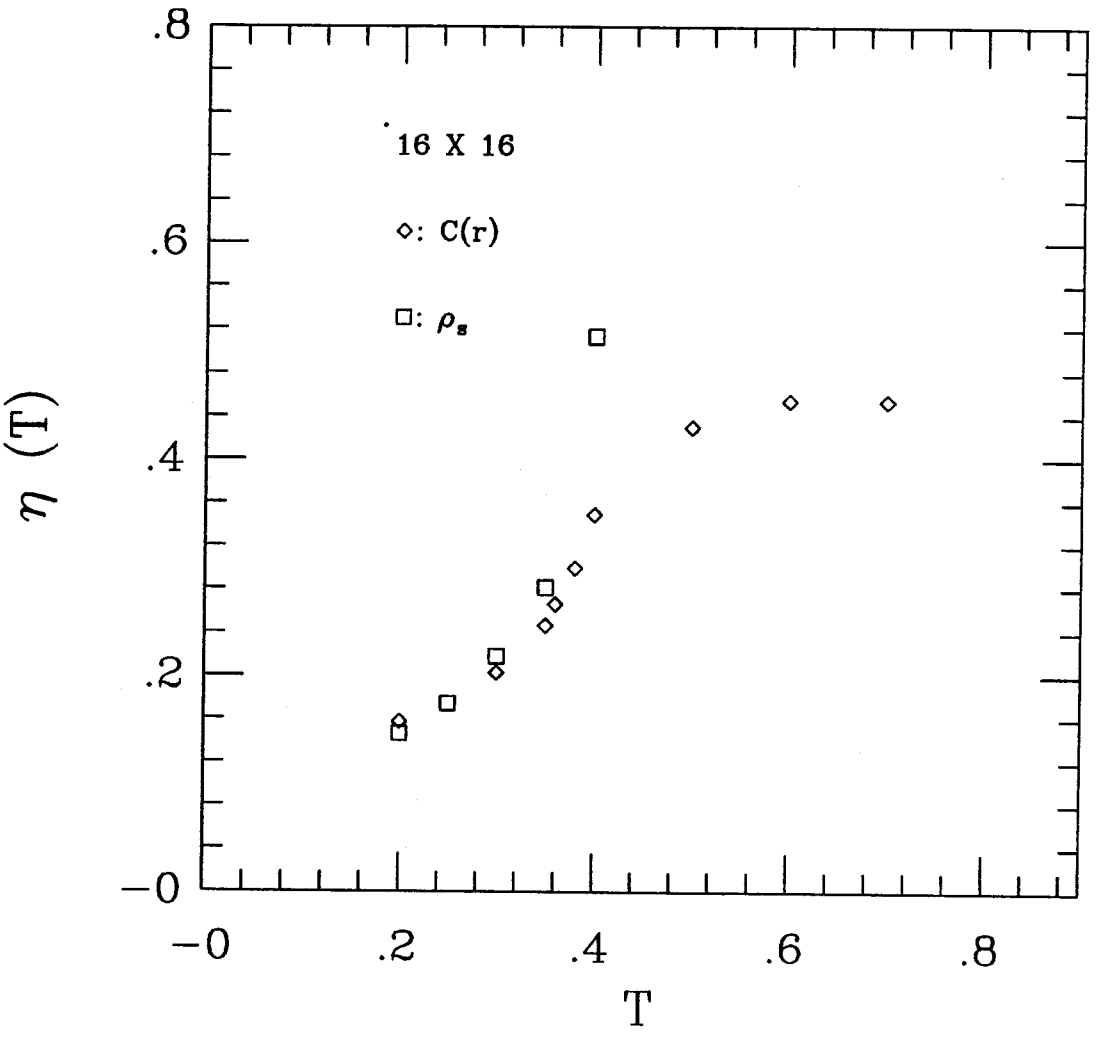


Fig. 38

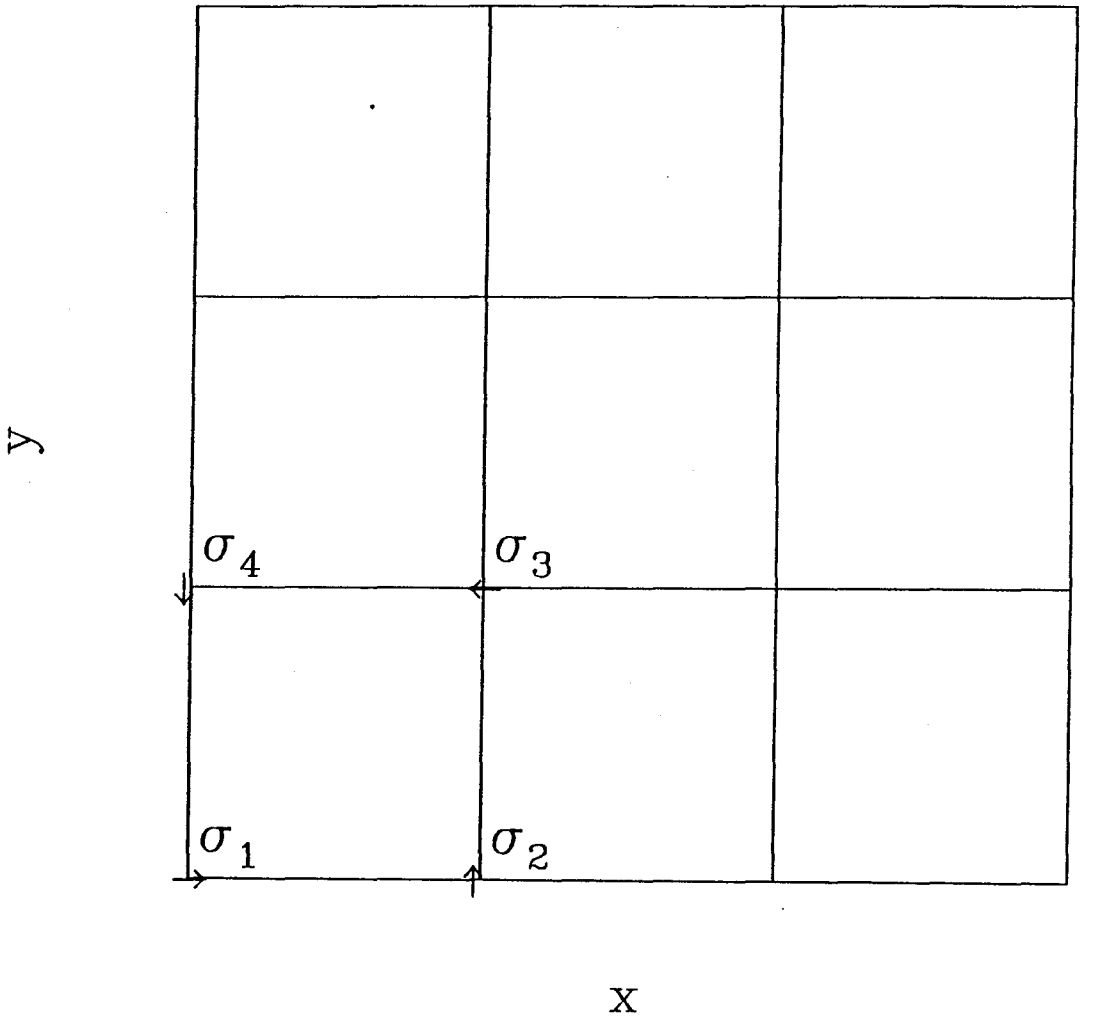


Fig. 39

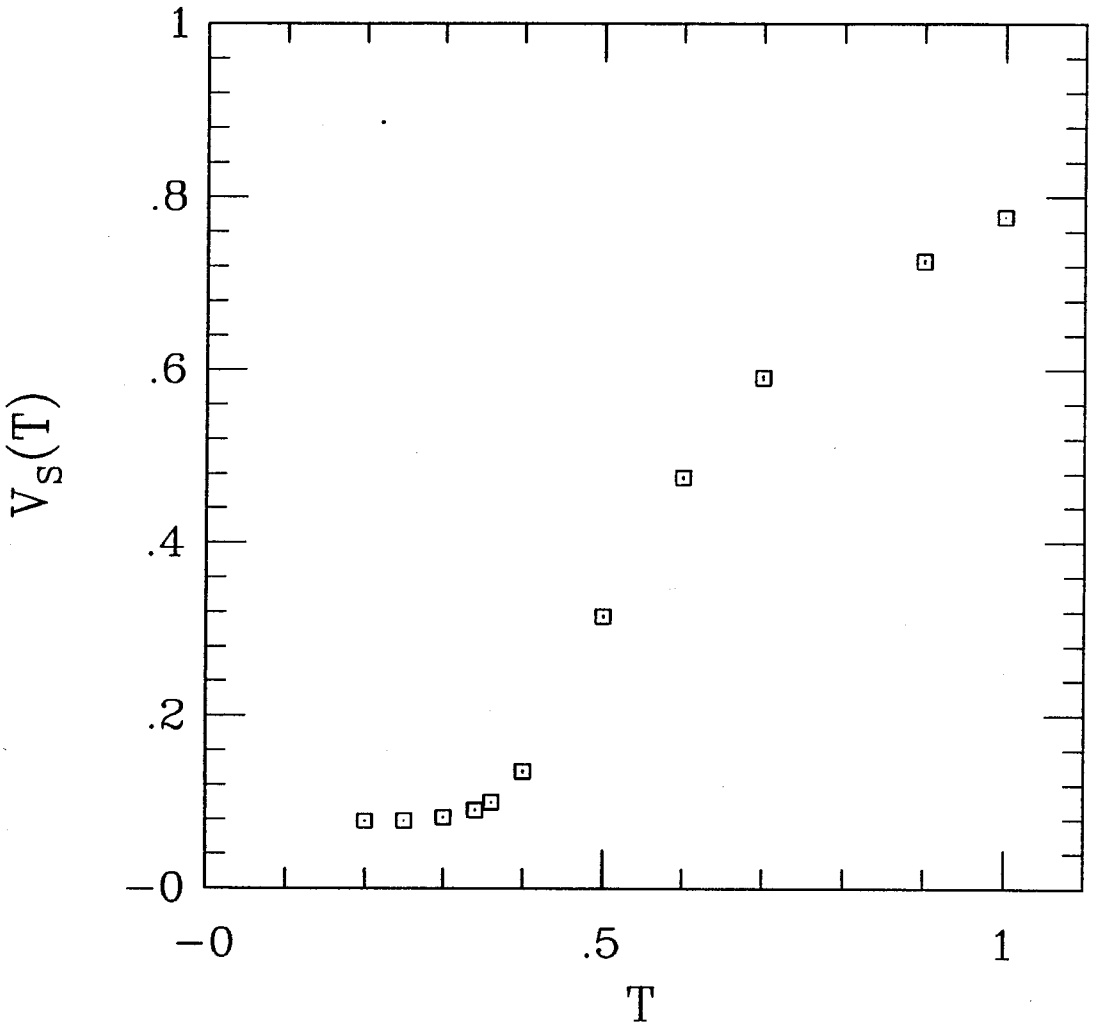


Table 1. Temperature, Trotter number, linear size, energy, specific heat, uniform susceptibility, correlation length and exponent.

T	m	L	-E	Cv	χ	ξ	λ
2.5	16	24	0.1600(2)	0.067(1)	0.201(1)	--	--
2.0	16	24	0.2003(3)	0.102(1)	0.227(2)	0.569(2)	0.5
1.5	16	24	0.2666(6)	0.168(3)	0.256(3)	0.68(3)	0.51(9)
1.2	16	24	0.328(1)	0.248(3)	0.275(1)	0.83(4)	0.47(16)
1.0	16	32	0.3885(8)	0.322(4)	0.281(2)	0.968(2)	0.36(22)
0.85	16	24	0.439(2)	0.393(1)	0.281(2)	1.21(3)	0.42(6)
0.75	24	32	0.4812(9)	0.43(1)	0.274(1)	1.44(2)	0.46(9)
0.60	24	32	0.5498(8)	0.452(5)	0.259(2)	2.20(4)	0.47(3)
0.50	32	32	0.5946(5)	0.425(3)	0.236(2)	3.5(1)	0.51(3)
0.45	16	32	0.6263(2)	0.412(4)	0.219(1)	5.2(2)	0.51(3)
0.45	32	32	0.6208(5)	0.386(5)	0.221(2)	4.6(2)	0.47(5)
0.40	40	64	0.6341(2)	0.353(5)	0.207(2)	6.5(2)	0.47(4)
0.35	24	64	0.6529(1)	0.283(7)	0.191(1)	9.9(4)	0.44(6)
0.35	48	64	0.6507(2)	0.23(1)	0.190(2)	10.1(5)	0.36(5)
0.30	24	96	0.6655(1)	0.21(3)	0.178(1)	18.0(5)	0.38(2)
0.30	48	96	0.6597(7)	0.19(3)	0.182(3)	17.5(5)	0.40(2)
0.27	48	128	0.6642(1)	0.175(3)	0.173(3)	28.0(1.2)	0.39(2)
0.25	48	32	0.6669(1)	0.091(4)	0.155(7)	--	--

Table 2. 4<Sz Sz > at several temperatures.

T	r=0	r=1	r=2	r=3
0.55	1	-0.1145(1)	-0.0016(1)	-0.0002(1)
0.45	1	-0.1353(2)	-0.0030(1)	-0.0007(1)
0.35	1	-0.1516(2)	-0.0051(2)	-0.0012(2)
0.2	1	-0.1647(1)	-0.0073(1)	-0.0027(1)