

A Parallel Algorithm for Perturbation Theory Calculations of Lattice Models including both Interactions and Disorder

Benjamin Buhrow
Physics Department, University of Northern Iowa

September 29, 2000

Abstract

We present a parallel algorithm for numerical calculations of electronic lattice models containing interactions and disorder. Self-consistent perturbation theory provides the framework for calculating the effect of interactions in the model. We demonstrate this method by examining second-order corrections to the specific heat for the attractive Hubbard model on an 8x8 lattice.

1 Introduction

Interest in superconductivity is widespread, due to enormous avenues of application such as powerful magnets for use in medicine and industry and in lossless power transfer. Much can be learned from a study of superconductivity at its fundamental level - that of a small group of atomic sites and their electrons. However, the theoretical understanding of superconductivity is still incomplete. For example, a realistic model of such a system must take into account the effects of interactions between electrons and of disorder in the system. The method in this paper is a step in this direction.

Perturbation theory is one promising approach to this problem. It allows for the study of systems much larger than those available to exact methods. It allows for the study of systems at lower temperatures compared with quantum Monte Carlo calculations, a method exact to within statistical precision. Also, it allows for the easy calculation of thermodynamic properties of the system and of electronic excitations.

For these reasons, we have developed an algorithm for self-consistent second order perturbation theory calculations of lattice models containing interactions and disorder. First order treatments in perturbation theory have been done[1], so we examine the effects of second order corrections to various thermodynamic quantities for these models. We do so with an algorithm that is extendable to orders higher than second.

We concentrate our discussion on the algorithm itself, including discussion of a unique method for obtaining greater accuracy, as well its memory requirements and scalability.

2 Method

We first must define the governing equations of a system of electrons in a lattice. The Hamiltonian operator describes the evolution of the quantum mechanical states of electrons in the lattice. It is defined as:

$$H = K - U \sum_i n_{i,\uparrow} n_{i,\downarrow} + \sum_i (V_i - \mu - h_o) n_{i,\sigma} \quad (1)$$

Here $K = -t \sum_{\langle i,j \rangle} (c_{i,\sigma}^\dagger c_{j,\sigma} + H.c.)$ is the kinetic energy like term representing possible jumps of electrons from site to site in the lattice. This is done via $c_{i,\sigma}^\dagger$ and $c_{i,\sigma}$, the construction and destruction operators

for a site i on the lattice with spin σ , and t , the near neighbor hopping energy. $n_{i,\sigma} = c_{i,\sigma}^\dagger c_{i,\sigma}$ represents the number of electrons at a site with a certain spin σ , and thus takes the value of 1 or 0 for fermions. U is the interaction strength parameter, μ the chemical potential and V_i is a random potential determined independently at each site from a uniform distribution $[-V, V]$. This random potential simulates the disorder by making it harder or easier for an electron to leave a site depending on the size and sign of V_i at each site i .

The start of our calculation is the consideration of the non-interacting part of the Hamiltonian[2]. For such a Hamiltonian, eigenfunctions and eigenvalues can be determined simply and these can be used to evaluate other quantities. For example, the non-interacting Green's function is obtained from:

$$G_o(\tau, x, x') = - \sum_s \psi_s^*(i') \psi_s(i) e^{-\xi_s \tau} (1 - n_s) \quad (2)$$

Where ψ_s are the $2 * N_{sites}$ eigenvectors obtained from the diagonalization, ξ_s are the eigenvalues of the diagonalization, and: $n_s = 1/(e^{\beta \xi_s} + 1)$. This function describes the evolution of an electron added at site x' and removed at site x a time τ later. Looking at how the electron's quantum mechanical states evolve all different combinations of initial and final sites and times can tell much about thermodynamic quantities of the system.

This Green's function can be used to estimate a correction to itself, approximate to some order, yielding a function referred to as the self energy and denoted Σ ... not to be confused with the summation symbol. The algorithm then is an iterative process, with continual incorporation of this self-energy correction back into G , where hopefully the correction gets smaller and smaller until some convergence criteria is met. The resulting Green's function G then contains information about the electrons and their interactions to whatever order we have calculated the self-energy. Theoretically, we can calculate some corrections to infinite order in this way, but this paper discusses the second order case in detail.

3 The Algorithm

3.1 Self-Consistent Loop

Referring to Figure 3.1, we first form the initial, non-interacting Green's function: $G_o(\tau, x, x')$ from the non-interacting part of the Hamiltonian matrix, as in equation 2. Here, the G_o refers to the fact that we have not yet incorporated an interaction term. It is defined to be a function of imaginary time τ , original site location x , and final site location x' and thus contains info about the probability of jumps from x to x' taking a time τ , for all combinations of x, x' and τ .

This function is transformed to frequency space using a parallelized, discrete, FFT to yield $G_o(\epsilon_n, x, x')$. This anticipates the next step: incorporating a guess at the interaction effects into G_o to obtain the full (interactions included) Green's function $G(\epsilon_n, x, x')$. This is done using Dyson's equation:

$$G(\epsilon_n, x, x') = [G_o(\epsilon_n, x, x')^{-1} - \Sigma(\epsilon_n, x, x')]^{-1} \quad (3)$$

where $\Sigma(\epsilon_n, x, x')$ is our guess at the interaction self-energy.

Next, the Green's function is transformed back to a time representation using another FFT to yield $G(\tau, x, x')$. We are now in a position to calculate the next guess to the interaction self-energy. With $G(\tau, x, x')$, we can define a function $\chi(\tau, x, x')$ as follows:

$$\chi(\tau, x, x') = G(\tau, x_\uparrow, x'_\uparrow)G(\tau, x_\downarrow, x'_\downarrow) + G(\tau, x_\uparrow, x'_\downarrow)G(\tau, x_\downarrow, x'_\uparrow) \quad (4)$$

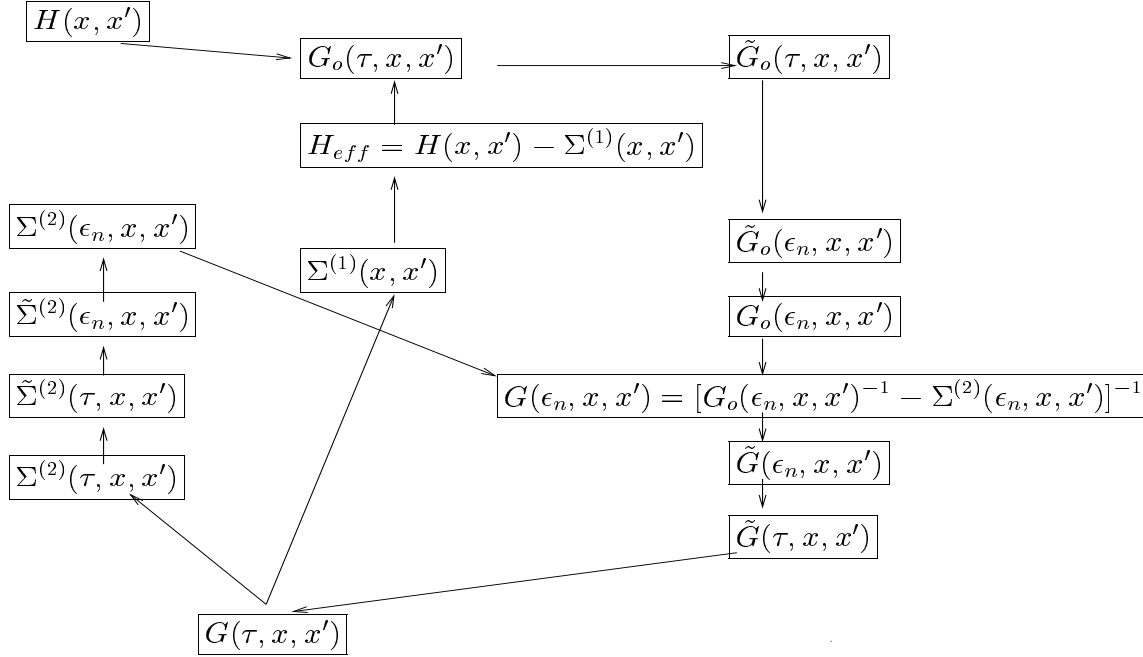


Figure 1: Schematic of self-consistent loop for evaluation of the Greens function, G , and self-energy, Σ

The correction to order U^2 is then defined as:

$$\Sigma_{\uparrow\uparrow}^{(2)}(\tau, x, x') = -U^2 \chi(\tau, x, x') G_{\downarrow\downarrow}(-\tau, x', x) \quad (5)$$

$$\Sigma_{\downarrow\downarrow}^{(2)}(\tau, x, x') = -U^2 \chi(\tau, x, x') G_{\uparrow\uparrow}(-\tau, x', x) \quad (6)$$

$$\Sigma_{\uparrow\downarrow}^{(2)}(\tau, x, x') = -U^2 \chi(\tau, x, x') G_{\uparrow\downarrow}(-\tau, x', x) \quad (7)$$

$$\Sigma_{\downarrow\uparrow}^{(2)}(\tau, x, x') = -U^2 \chi(\tau, x, x') G_{\downarrow\uparrow}(-\tau, x', x) \quad (8)$$

This second order interaction is incorporated back into the loop after being transformed to frequency space, completing the self-consistent loop.

3.2 Electron Density

The self-consistency loop is buried inside another loop, thankfully one that only requires a few iterations before convergence. This outermost loop of the algorithm adjusts parameters in the Hamiltonian so that the final answer will converge to the specified density of electrons in the lattice.

The density of electrons in the lattice depends on the chemical potential, μ , of the system, incorporated in the Hamiltonian. After obtaining a converged $G(\tau, x, x')$ from an initial guess of μ , the electron density is calculated using properties of the Green's function:

$$G_{\uparrow\uparrow}(\tau \rightarrow 0^+, x, x) = n_x^e - 1 \quad (9)$$

$$G_{\downarrow\downarrow}(\tau \rightarrow 0^+, x, x) = -n_x^e \quad (10)$$

By summing over x and dividing by the number of sites, we obtain the density of electrons in the lattice. A new value of μ is calculated using linear extrapolation on the two points we have in density-chemical

potential space. This converges quite rapidly, typically taking only 3 or 4 iterations with a reasonable initial guess at μ . After the density has converged, we are essentially done. All that remains is to use the Green's function just calculated to compute useful things. These calculations will be discussed later.

3.3 High frequency conditioning

Another issue to confront is that of a hard frequency cutoff. Imaginary time, in this formalism, is a continuous variable with a range of $[0, 1/T]$, where T is the temperature. Frequency, while discrete, has an infinite number of terms. However, we must implement both as being discrete and finite due to the nature of physical storage on a computer. Errors arise in transforming coordinates due to this finite sampling of a continuous function. The discrete Fourier Transforms we use are an approximation to the exact definition:

$$G(\epsilon_n) = \int_0^\beta e^{i\epsilon_n\tau} G(\tau, x, x') \approx \frac{\beta}{N_\tau} \sum_{i=1}^{N_\tau} e^{i\epsilon_n\tau_i} G(\tau_i, x, x') \quad (11)$$

We deal with this loss of information by defining an analytic function that we can Fourier transform exactly, and that will, when subtracted from the Green's function, remove the rapidly changing structure in $G(\tau_i)$ [3][4]. The byproduct of these analytic subtractions are marked with a tilde in Figure 3.1.

From $G_o(\tau, x, x')$ on Figure 3.1, we first subtract off an analytic function, yielding a much nicer looking Green's function with a continuous derivative at $\tau = 0$. We then perform our numerically approximate FFT on this new set of data that is less susceptible to numerical error. Finally, we add back the analytic part that has been transformed exactly into the target domain. This high frequency conditioning costs very little computationally because the operations are algebraic and thus are performed perfectly in parallel, yet benefits greatly in improved accuracy in the FFT's.

4 Computational Issues

Much of the initial work on this project was simply determining if modeling realistic system sizes would be feasible. The algorithm involves multiple inversions, Fast Fourier Transforms, diagonalizations, and other linear algebra operations performed on three dimensional matrices of high precision complex numbers. Each of these steps is performed perhaps hundreds of times before self-consistency is achieved. This enormous calculational burden, along with the storage requirement of the matrices, places restrictions on the size of the problems we can do. The computations are done in parallel and storage is distributed among processors which greatly increases the scope of application, but there seem to be limits even to this.

4.1 Linear Algebra Operations

General matrix inversions scale as N^3 , where N is the number of diagonal elements. Techniques exist to do better than this for special forms of matrices. However, our definitions of the Green's function and the self-energy do not allow the use of these specialized inversions. N^3 is not very good, but it gets worse. Matrix inversions must be done on each time or frequency value of the Green's function. So the problem actually scales as $M * N^3$ where M is the number of time or frequency points to be computed. General matrix diagonalizations also go as N^3 . Matrix diagonalization does not affect the algorithm much though, because it is only performed on one $N \times N$ matrix, the Hamiltonian. The Hamiltonian is diagonalized to produce the eigenvectors and eigenvalues that are then used to create $G_o(\tau, x, x')$.

Other operations also affect the time scaling of the algorithm. The analytic subtractions and additions can take a significant amount of time. They scale as $M * N^2$. The biggest bottleneck of all involves the generation of $G_o(\tau, x, x')$ from the eigenvectors and eigenvalues of the diagonalized Hamiltonian. Referring to equation (2), for every time/frequency point M , N original site locations x , and N final site locations x' ,

we must do a sum over all the N states of the diagonalized Hamiltonian. It then scales as $M * N^3$, similar to other operations in the algorithm. It is the biggest bottleneck because each pass through the innermost loop requires 3 memory lookups and assignments, 2 multiplications, and an add operation. This, coupled with the fact that our current compiler doesn't seem to optimize nearly as well as the LAPACK and BLAS implementations of the inversions and diagonalizations, cause these calculations to be very time consuming in our implementation of the algorithm. Newer compilers are being looked at, which will hopefully clear up this problem.

4.2 Memory Allocation

Overall, it is the storage of these huge matrices that restricts the problem size domain. To be able to get physically valid results, lattice sizes of 8x8 or larger are required. For a typical problem run with a 12x12 lattice and 128 time/frequency points, the Green's function matrix $G(\tau, x, x')$ requires 288x288x128 complex numbers to be stored. It is 288x288 because information about spin up and spin down electrons at each site must be stored. Each complex number is made up of two floating point numbers, which on our system require 4 bytes each to store. So a 12x12 lattice requires approximately 288x288x128x8 or 85x10⁶ bytes.

The algorithm requires not only that we store $G(\tau, x, x')$, but two copies of $\Sigma(\epsilon_n, x, x')$ and $\chi(\tau, x, x')$ as well. Two copies of the self-energy are required because in order to check for its convergence, we must keep a copy of its value on the current and previous iteration. The whole program then needs 288x288x128x8x4 or approximately 340x10⁶ bytes. One can see now why parallization is so valuable. By splitting up this matrix among 8 processors, modest workstations having only 128Mb can be used to tackle problems of this size or larger.

4.3 Parallization

Being able to do computations like this in parallel is a major motivation for doing them in the first place. Problem scaling would quickly overwhelm a workstation, forcing the use of some kind of supercomputer to get anything done. Beowulf clusters have proven to be highly effective because they are relatively cheap, portable, and powerful. Our Beowulf consists of 16 nodes connected with a 100Mb Smart Switch via fast ethernet.

The Green's function and the 2nd order self-energy correction, $\Sigma^{(2)}$, are split among parallel processors along the time/frequency axis. Linear algebra operations are done with respect to the space indices x and x' , with ϵ_n or τ fixed. Thus, these calculations can be done independently, with each computer assigned a range of τ or ϵ_n values. This is the obvious choice in parallization. Each computing node stores its own copy of G , making the operations entirely local, resulting in a $1/(N_{proc})$ reduction in storage and computation time.

The success of the cluster is made obvious with the following table. It shows timings for different cluster sizes all doing two inversions on a three-dimensional matrix.

Matrix Size	# of Proc.	Time(sec)
8x8x128	2	13.441
8x8x128	4	6.782
8x8x128	8	3.395
8x8x128	16	1.697

The actual reduction in time is very close to this theoretical limit, showing the efficiency of this method. We expected this result, because there is actually very little communication going on here. Each section of the matrix is generated locally at the start of the process, the inversion is done locally, then the process stops. The only communication is the initial command telling the process to start, and each process telling the master it is done.

Things aren't so perfect when actual communication must occur. This happens in our algorithm with the FFT's. Basically, we must do a parallel transpose operation on a three-dimensional matrix to get the right axis split among processors. This requires sending large chunks of the matrix around all at once. The following table summarizes the results for different numbers of nodes computing three FFT's:

Matrix Size	# of Proc.	Time (sec)
8x8x128	2	8.359
8x8x128	4	6.479
8x8x128	8	5.553
8x8x128	16	9.309

Initially, as the number of processors used is increased, significant performance increase is seen. However, if we continue to increase the number of processors the performance increase becomes less significant, or even starts to decrease. This is simply due to network traffic. As more and more nodes start cramming large amounts of data through the switch essentially at the same time, we see slower and slower transfer times.

The most efficient way we found to do the transpose required by the FFT's involves pairs of nodes trading information. This is done with a loop wherein each pair of processors whose summed ID's mod $(N_{proc} - 1)$ equals the loop incrementation variable, trade information. This might be causing problems for larger numbers of nodes because every process is sending information through the switch at the same time. Possible ways to overcome this problem involve simply getting a higher volume switch, such as a Gigabit switch, limiting the size of the packet sent by each computer, or somehow better optimizing the transfer, maybe by staggering the sends and receives so not all nodes are talking to each other at the same time.

Here is a summary of the most time consuming sections of one iteration of the algorithm, for different sized lattices and different sized clusters.

Lattice Size	# of Proc.	Inversions	FFT's	Analytic +/-	Generating G_o
8x8x128	2	13.441	8.359	15.459	16.809
8x8x128	4	6.782	6.479	8.225	9.754
8x8x128	8	3.395	5.553	4.090	4.668
8x8x128	16	1.697	9.309	2.054	2.485
12x12x128	4	95.47	31.263	41.717	98.178
12x12x128	8	47.362	21.348	20.474	52.118
12x12x128	16	23.806	43.79	10.126	28.843

It looks as if there is going to be a point where the communication costs will overwhelm the benefits gained by parallization.

5 Results

One of the major motivations for using perturbation theory is the fact that it makes it relatively easy to calculate thermodynamic properties of the lattice. This section presents the equations necessary to solve for interesting quantities, as well as some of the actual results obtained from the code.

The main quantity of interest is the grand thermodynamic potential Ω , which can be obtained from the self-energy Σ and the Green's function G via the Luttinger and Ward formula[5]:

$$\Omega(T, \mu) = -2\text{Tr}[\Sigma G + \ln(-G_o^{-1} + \Sigma)] + \Phi[G] \quad (12)$$

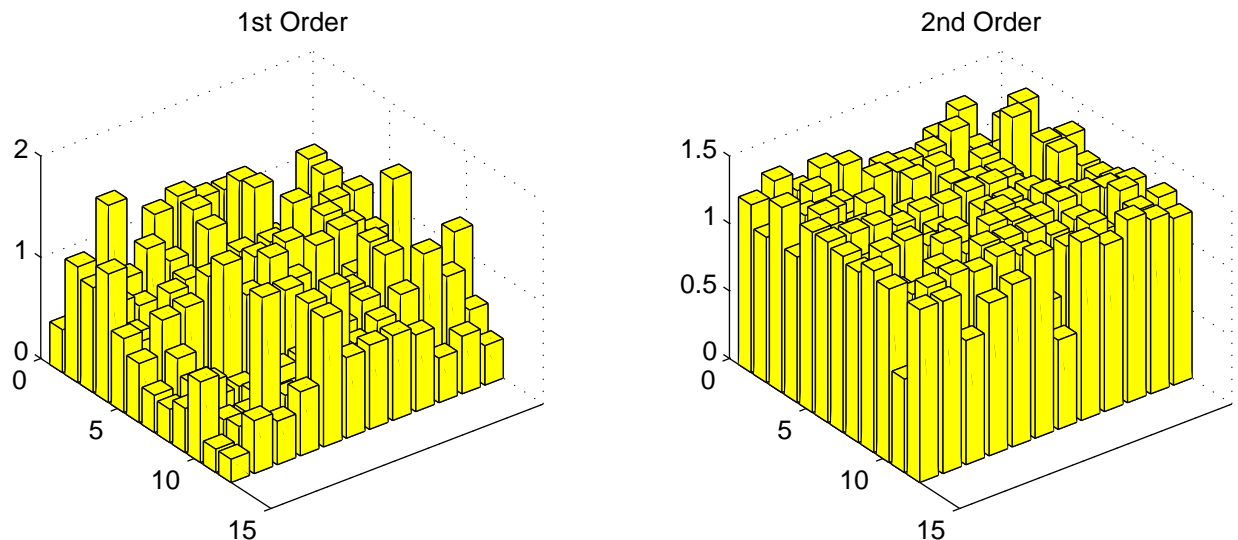


Figure 2: Comparison of 1st- and 2nd-order perturbation theory results for the spatial dependence of the local superconducting order parameter.

From this and the total average energy, we can calculate the Helmholtz free energy, F , and the entropy, S , of the system.

$$F = \Omega(T, \mu) + \mu N_e \quad (13)$$

$$S = \frac{\Omega(T, \mu) - E_{avg} + \mu N_e}{T} \quad (14)$$

The Green's function also contains information on the magnetization of the lattice, the density of states of electrons in the lattice, and on a superconducting 'order parameter' $|U G_{12}(\vec{r} = \vec{r}', \tau = 0)|$. The latter characterizes the strength of superconductivity as a function of position in the lattice.

The algorithm will respond to an external magnetic flux, h_o in equation 1. However, in all our results so far $h_o = 0$, so the magnetization is uninteresting. The superconducting order parameter varies greatly as disorder is changed. Moreover, it reacts differently in our 2nd order calculation as compared to 1st order. As can be seen in Figure 2, the lattice superconductivity is less disordered as well having a higher average magnitude for 2nd order.

Figure 3 shows the specific heat $c(T) = TdS/dT$ as a function of temperature for 1st and 2nd order at 3 different disorder strengths. The peaks are all more pronounced at 2nd order, and the order parameter is bigger. This corresponds to the higher average magnitude for 2nd order in Figure 2.

6 Conclusion

The results presented here are only the beginning of what is possible given the very general nature of the algorithm. Higher disorder strengths for both 1st and 2nd order, disorder averaging, and superfluid density calculations are a few things we hope to look at in the future. The speed of the algorithm also will continue to improve, both from developing new ways of implementing the slower functions as well as improving hardware and software support.

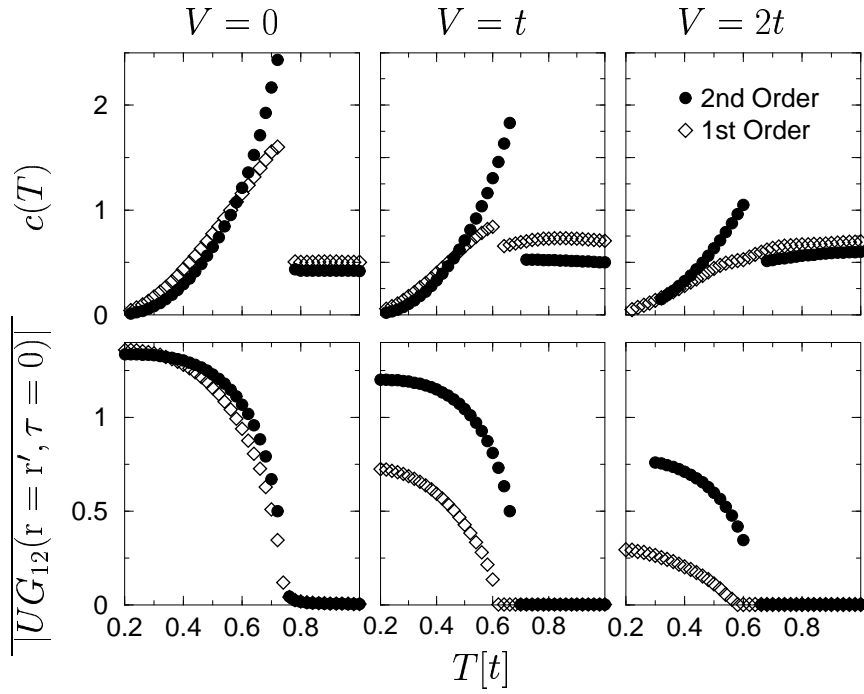


Figure 3: (Top) Specific heat vs. temperature for different disorder strengths. (Bottom) Average magnitude of the local superconducting order parameter.

References

- [1] A. Gosal, M. Randeria, and N. Trivedi, “Role of Spatial Amplitude Fluctuations in Highly Disordered s-Wave Superconductors”, *Phys. Rev. Lett.*, vol. 81, no. 18, 3940 (1998).
- [2] J.W. Serene and D.W. Hess, in *Recent Progress in Many-Body Theories*, vol. 3, edited by T.L. Ainsworth, *et. al.*, (Plenum, New York, 1992).
- [3] J.J. Deisz, D.W. Hess, and J.W. Serene, in *Recent Progress in Many-Body Theories*, vol. 4, edited by E. Schachinger, *et. al.*, (Plenum, New York, 1995).
- [4] B. Buhrow and J.J. Deisz, to be published.
- [5] J.M. Luttinger and J.C. Ward, *Phys. Rev.* **118**, 1417 (1960).