Ground–State Properties of Two–Dimensional Frustrated Quantum–Spin Models

Inaugural – Dissertation

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Andreas Sindermann

aus Köln

Köln 2004

Institut für Theoretische Physik der Universität zu Köln
Zülpicher Straße 77, D–50937 Köln

Contents

1. Introduction

The examination of quantum–spin models in one and higher dimensions has been subject of interest for more than 70 years [1]. Experiments and theoretical studies have shown that interaction between magnetic ions can be well represented by a model Hamiltonian describing a set of interacting spins $\mathbf{S}_i$. One important class of such quantum–spin models consists of spins coupled to their nearest neighbors on a finite lattice. The Hamiltonian for such a system reads

$$H = \sum_{<i,j>} (J_x S_i^x S_j^x + J_y S_i^y S_j^y + J_z S_i^z S_j^z) \tag{1.1}$$

where $< i, j >$ denotes all nearest neighbor pairs on a lattice and S_i^α is the α component of the spin operator on site i. This generic Hamiltonian describes various types of well known quantum–spin models. For $J_x = J_y = J_z \equiv J > 0$ we find the antiferromagnetic *Heisenberg* model, for $J < 0$ the ferromagnetic Heisenberg model, for $J_z = 0$ the XY model and for $J_x = J_y$ the XXZ model.

In one dimension the XXZ model has been solved analytically by Bethe and Hulthèn about 70 years ago [2, 3]. Unfortunately the Bethe ansatz is limited to a small set of quantum–spin systems, especially to one–dimensional systems. For that reason other methods were needed to study the properties of quantum spin systems. One of the first numerical methods applied is the exact complete diagonalisation which Bonner and Fisher used in 1964 to study the one–dimensional XXZ model [4]. They investigated the dependence on the finite size and anisotropy of different ground state and thermodynamic properties of the model. 14 years later Oitmaa and Betts [5] using also standard diagonalisation routines up to 16 sites and the iterative power method [6] for 18 sites investigated the ground state energies and pair correlations for the two–dimensional antiferromagnetic Heisenberg model. Also this method is only applicable up to a certain system size as it fast comes to technical limits of even today's computers.

Often, only the eigenvalues at the upper and lower end of the spectrum are of interest so that other methods like the Lanczos method come into focus. With this method later many investigations with exact diagonalisation have been performed. Already in 1994 Schulz et al. [7] were able to calculate the ground state energy of a 6×6 Heisenberg model with frustration containing about 12 million states. To our knowledge the largest quantum spin system studied up to now is a spin–$1/2$ XY model on a square lattice with about 32 million states in the subsector of the Hilbert space containing the ground state.

Since it is that difficult to calculate the spectrum and the ground state energies of quantum spin models, those with an exactly known ground state are of special interest. One of these very few quantum spin models has been introduced in 1981 by Shastry and Sutherland. It consists of the two–dimensional Heisenberg model

with additional frustrating interactions between next neighbors which starting with a certain critical value of the frustration establishes the exactly known ground state. This model became especially of interest when Miyahara and Ueda [8] in 1999 pointed out the magnetic properties of the substance $SrCu_2(BO_3)_2$ that was synthesised already in 1991.

In general there is no specific reason to restrict on bilinear interactions like in the Shastry–Sutherland model. Currently, four–spin interactions have been introduced by which magnetic properties of certain substances could be better explained.

A model with a simple structured four–spin interaction is the frustrated *plaquette* model which for large frustration exhibits an exactly known ground state similar to the Shastry–Sutherland model.

By means of different methods, experimentally and theoretically, for many years the exact critical phase boundary between the antiferromagnetic and the so called *dimer* phase is studied where the exactly known eigenvalue is the ground state. Also an intermediate phase is predicted whose nature is still in discussion.

In this work two different approaches are followed to find precise data for the transition point. On the one hand a variational approach published already in 1951 by P. W. Anderson [9] is applied which gives a strict lower bound for the critical value of the frustration. On the other hand by calculating systems with periodic boundary conditions infinite systems can be simulated.

Applying symmetry operators that commute with the model Hamilton operator subsectors can be created whose lowest eigenvalues can be calculated with the Lanczos method. Although these methods are widely used it is still very difficult to treat a system size of $N \geq 32$.

Using subtle parallel programming techniques we are able to investigate lattices containing up to 36 sites.

The outline of this thesis is as follows.

In chapter 2 the two models are introduced and their phase diagrams and their physical properties are discussed. Also, if applicable, the realisation in substances is mentioned and in that case an overview of experimental results on the specific model is given.

In chapter 3 we introduce methods applied to find ground state properties. Symmetry operators commuting with the Hamilton operator of the model investigated are introduced by which the Hilbert space can be diagonalised in blocks leading to subsectors with smaller numbers of states needed to be considered in the calculations. Also a short introduction to the numerical library ARPACK and its parallel version P_ARPACK is given which offers a fast and stable implementation of the Lanczos method which also is described in detail. To finish this chapter we summarise the main features of two different parallelisation techniques of which one in the end is used in a program that calculates the ground state energies of the models described in chapter 2.

Chapters 4 and 5 give a detailed overview of the results of the calculations of the ground state energies that mainly have been performed at the local computing center and at the IBM supercomputing facilities at the Forschungszentrum Jülich on which especially the largest system considered has been calculated.

A summary of the results and an outlook close this thesis in chapter 6. An abstract

in english and german can be found in the appendices.

2. Two–Dimensional Frustrated Quantum–Spin Models

In this thesis we study two–dimensional quantum–spin models with antiferromagnetically coupled spins of size $S = 1/2$. The generic Hamilton operator for such systems reads

$$H = \sum_{ij} J_{ij} \mathbf{S}_i \mathbf{S}_j \, , \qquad (2.1)$$

with $J_{ij} > 0$ representing the exchange coupling between next and next–nearest neighbor sites i and j. Eq. 2.1 thus includes as special case the two–dimensional Heisenberg model which is of interest as a minimal magnetic model of the undoped copper–oxide planes in high T_c superconductors. The Hamiltonian (eq. 2.1) includes also the Shastry–Sutherland model which in this work we examine in detail. In eq. 2.1 only interactions between two spins are assumed. Indeed, most spin models discussed in the literature are of this type. However, both from the phenomenological and from the theoretical point of view there is no restriction to this bilinear type of interaction, and only recently the implications of a four–spin interaction, the so called *ring exchange*, has been widely discussed. In this context it is argued that a careful derivation of spin Hamiltonians from the three band Hubbard model must include as next to leading order four–spin terms of a certain type (the so called *ring exchange*). Also for spin–ladders the four–spin interactions are thought to be essential, both for the ground state properties and for the thermodynamics.

In this thesis besides the Shastry–Sutherland model with its two–spin interaction we also consider a model with a four–spin interaction of a simpler type than the phenomenologically motivated models just mentioned. This model, suggested by J. Zittartz, and the Shastry–Sutherland model both have the interesting and seldomly found property of an exactly known eigenstate which for frustrated couplings is also the ground state.

2.1. Shastry–Sutherland Model

In this section we study a model introduced in 1981 by Shastry and Sutherland [10] as a two–dimensional generalisation of the one–dimensional Majumdar–Ghosh [11] model. The Hamilton operator for this model is given by

$$H = J_2 \sum_{<i,j>} \mathbf{S}_i \mathbf{S}_j + J_1 \sum_{\substack{<i,j> \\ \text{dimer}}} \mathbf{S}_i \mathbf{S}_j \, , \qquad (2.2)$$

where $\mathbf{S}_i$ denotes a spin operator for spin S at site i. The sums are running over nearest neighbors with coupling J_2 and over next–nearest neighbors with coupling

J_1. In the limit $J_1 = 0$ this model is simply the antiferromagnetic Heisenberg model. The Shastry–Sutherland model is of special interest as it has an exactly known eigenstate (the dimer singlet state) as was shown by Shastry and Sutherland [10].
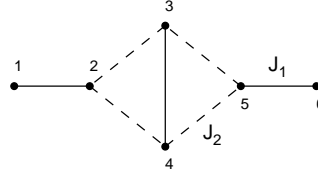

Figure 2.1.: Illustration of a dimer–solid. Spins are denoted with circles. Two spins coupled with J_1 form a dimer. The dimers interact via J_2.

To show that the dimer state is the ground state it is sufficient to consider the unit cell depicted in fig. 2.1. The Hamiltonian can be explicitly written as $H = J_1(S_1 S_2 + S_3 S_4 + S_5 S_6) + J_2(S_2(S_3 + S_4) + S_5(S_3 + S_4))$.
If the system is in the dimer singlet ground state we have the total spin $(S_i + S_j) = 0$ if S_i and S_j belong to the same dimer so that only the first part of H is left over for which we find an eigenvalue $E_0 = -\frac{1}{2}J_1 S(S+1)$ (in case of $S = 1/2$: $E_0/J_1 = -3/8$) per dimer.
With $x = J_2/J_1$ we denote the *inverse frustration*. In case of x becoming very large we find the standard Heisenberg model and for $x = 0$ we find uncoupled dimers on the diagonal bonds. Considering a simple four–spin system for $x < x_c = \frac{1}{2S+1}$ the dimer state is also the ground state. Better estimates for x_c of the infinite system can be obtained by calculating the ground state energies of larger systems with open boundaries as described below.
The model shows a rich zero temperature phase diagram as a function of J_1 and J_2 (fig. 2.3) [12, 10]: In the classical case ($S \to \infty$) one finds two long range helical phases for $0 < |J_2| < J_1$ separated by an antiferromagnetic dimer phase for $J_2 = 0 < J_1$. Analogously one finds a ferromagnetically ordered dimer phase for $J_2 = 0 > J_1$. In the regime $|J_2| > J_1$ one finds an antiferromagnetically ordered ground state for $J_2 > 0$ and ferromagnetically ordered for $J_2 < 0$.
For the quantum mechanical case ($S < \infty$) the scenario changes as follows [12]: The classical antiferromagnetic dimer phase changes by quantum effects to a singlet dimer state which is the exact ground state for certain values of the inverse frustration x.
This singlet dimer phase extends to a larger area than its classical counterpart (see right hand side of fig. 2.3).
For $|J_2| > J_1$ and $J_2 < 0$ the quantum mechanical ferromagnetic region persists. For $S = 1/2$ one finds a first order phase transition from the singlet dimer to the ferromagnetic phase at exactly $x = -1$ without an intermediate regime. For $1/2 < S < \infty$ the system enters an intervening phase between the dimer phase and the ferromagnetic phase at a value $x_c^{fm}(S)$. On the antiferromagnetic side of the phase diagram $J_2 > 0$ for *all* finite values of S an intermediate phase between the antiferromagnetic and the dimer phase exists whose nature is currently not
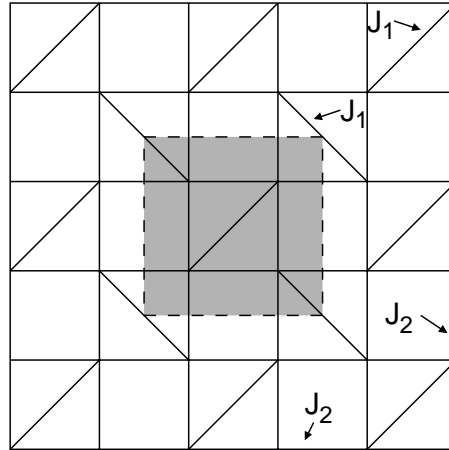
Figure 2.2.: Shastry–Sutherland model with spins situated on the vertices, couplings J_1 on the diagonal bonds (dimers) and J_2 on vertical and horizontal bonds. The grey hatched region in the middle represents the unit cell of the system.
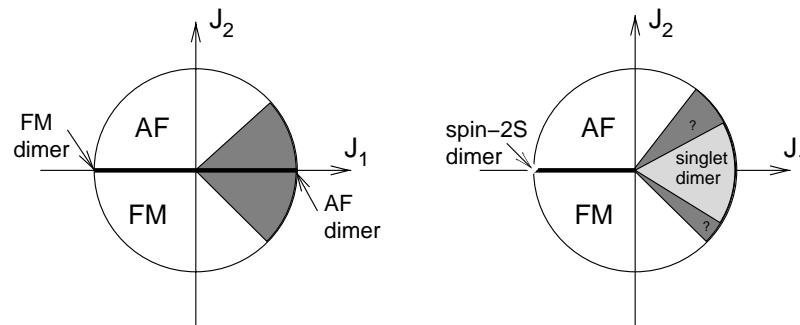

Figure 2.3.: Left: Exact $T = 0$ phase diagram in the classical case of the Shastry Sutherland model. Right: Schematic phase diagram of the model for $S > \frac{1}{2}$ (right). For $S > \frac{1}{2}$ adjacent to the singlet dimer phase two phases occur whose nature is still discussed controversially.

clearly understood. At a certain critical value $x_c^{\mathrm{af}}(S) > 0$ a phase transition of first order [13] from the dimer phase to the intermediate phase takes place.

For $J_2 = 0$ and $J_1 < 0$ the ferromagnetic and antiferromagnetic regimes are separated by a phase of independent spin–$2S$ dimers.

In the following we present a short overview of information about the intermediate regimes between the singlet dimer phase and both the (anti–)ferromagnetic regimes:

Löw and Müller–Hartmann [12] find rigorous lower and upper bounds on the phase boundaries of the singlet dimer phase by using various versions of a variational ansatz for finite clusters in combination with exact diagonalisation (Lanczos).

To find a lower rigorous bound they decompose the Hamilton operator H (eq. 2.2) into cluster terms H_i^N in such a way that the clusters cover the whole lattice without overlapping bonds. Following P. W. Anderson's arguments from 1951 [9] the lowest eigenvalues E_0^N of the clusters are always lower than or equal to the ground state energy of the infinite system E_0^∞ if one takes the ground state of H as variational state for the finite clusters.

Using this argument for an elementary system consisting of four sites they find $x_c^{\text{af}} = \frac{1}{2S+1}$ as a first exact rigorous lower bound which in the case of $S > 1/2$ is already a better estimate than that of Albrecht and Mila [13] who suggest $x_c^{\text{af}} < \frac{1}{2(S+1)}$ using Schwinger boson mean field theory. Even better limits can be obtained by calculating the ground state energies of clusters with larger system size (up to $N = 31$) using the Lanczos method. For a spin–1/2 system of size $N = 31$ they find a best rigorous lower bound for the phase boundary of $x_c = 0.5914$.

For the bound to the ferromagnetic regime they find $x_c^{\text{fm}}(S) \leq -\frac{1}{2S}$ which for $S = 1/2$ coincides exactly with the $x_c = -1$ boundary. This means that for $S = 1/2$ the intermediate regime on the ferromagnetic side of the phase diagram ($J_2 < 0$) is vanishing. For $S > 1/2$ they again find an improvement of this result by calculating ground states of clusters of size $N > 4$ with the Lanczos method. The best value found so far is for a spin–1 system of size $N = 17$ $x_{c,17}^{\text{fm}}(S = 1) = -0.5490$.

Using a variational argument Löw and Müller–Hartmann also find upper bounds by calculating ground state energies using the Lanczos method for different clusters up to system size $N = 32$. Extrapolating these results for $N \to \infty$ they find for $S = 1/2$ a best upper bound between 0.7126 and 0.7127. For ($S = 1$) they find a best upper bound of 0.618 and for the limit $S \to \infty$ a value of $x_c(S > 1) < \frac{1}{\sqrt{2S}}$ as a criterion for the instability of the dimer phase using a helical product state as a variational state.

The nature of the intermediate phases adjacent to the singlet dimer phase is a matter of current investigations:

One might consider a quantum mechanical analogue of the classical helical phase as described above. Albrecht and Mila [13] indeed find such a phase using Schwinger boson mean field theory for finite values of the spin S vanishing in a second order phase transition in favor of the Néel phase at $x \approx 0.91$. They also find the system undergoing a first order transition from the intermediate phase to the dimer phase at $x \approx 0.606$ which is in clear contradiction with our results.

Using field theoretical arguments Chung et al. [14] find an intervening regime with two helical and collinear phases that within this theory appear as Bose–condensates, whereas Carpentier and Balents [15] find an intermediate regime characterised as a weakly incommensurate spin density wave. Also, they argue that there *has* to be an intermediate regime on the antiferromagnetic side ($J_2 > 0$) of the phase diagram, i.e. a direct dimer to Néel phase transition cannot occur.

Focussing on the $S = 1/2$ case a number of observations on the phase boundary of the dimer phase have been made:

By use of exact diagonalisation and fourth order perturbation theory Miyahara and Ueda [8] find a direct dimer to Néel transition of first order at $x \approx 0.7$. Although this value is widely accepted one could put this result into question as it was

obtained by extrapolating three systems of *different* shapes. On the other hand we find that the shape of the clusters taken into account has a strong influence on the ground state energy.

Läuchli et al. [16] performed large scale exact diagonalisation calculations up to system size $N = 32$ which result in an upper critical value of $x_c = 0.67$ for the dimer phase. They suggest that an intermediate plaquette phase might be found in the regime $0.67 < x < 0.7$ and exclude the possibility of an intermediate columnar phase. Further numerical studies based on an operator variational method introduced by Munehisa and Munehisa [17] support a helical intermediate phase again.

Examinations applying perturbational approaches do not seem to help finding the exact critical value for the transition boundary:

Koga and Kawakami [18] suggest a plaquette phase in the interval $0.677 < x < 0.861$ using studies based on different starting points for their perturbation theory: isolated dimers, isolated plaquettes and Ising–limit.

In contrast to them Weihong, Hamer and Oitmaa [19, 20] defer such an intermediate plaquette phase or helical phase. Applying high order perturbation theory and comparing the ground states of different phases they find that only a columnar phase might occur in the range $0.67 < x < 0.83$. Investigating the behavior of the gap above the singlet dimer ground state they find $x = 0.691$ as an upper bound for the dimer phase.

Knetter et al. [21, 22] suggest $x = 0.63$ as a value for the breakdown of the dimer phase by investigating the behavior of the gap as a function of the inverse frustration x using the perturbative unitary transformation method.

After the synthesis of $SrCu_2(BO_3)_2$ by Smith and Keszler [23] in 1991 the Shastry–Sutherland model came back into focus of new studies. The orthoborate $SrCu_2(BO_3)_2$ has a layered structure composed of $Cu(BO_3)$ –planes which are slightly buckled for temperatures below $T_S = 395 K$ as shown on the left hand side of fig. 2.4[1].

At T_S a second order phase transition occurs [25] where the layers become completely flat. The planes are separated by the Sr–atoms in the crystallographic c–axis. The top–view on a representative $Cu(BO_3)$ plane (shown on the right hand side of fig. 2.4) helps to visualise the properties of the $Cu(BO_3)$ compounds. The magnetism is determined by the the $S = 1/2$ spins located on the Cu^{2+} ions which form dimers by pairs (connected by lines) with interaction strength J_1. Assuming an exchange path over the borate groups to the next–nearest Cu^{2+} ions with couplings J_2 one can map this structure onto the Shastry–Sutherland model (fig. 2.2) as was seen first by Miyahara and Ueda in 1999 [8], nearly 20 years after Shastry and Sutherland had published their observations.

Every second $Cu(BO_3)$ plane is rotated by $\pi/2$ in such a way that each dimer has a rotated dimer above and below. The tetrahedal inter–plane interaction geometry is fully frustrated. *Both* the dimers, above *and* below, must be excited out of the singlet ground state for this interaction to become relevant so that [26] the spin–gap Δ and thermodynamic properties at low temperatures can well be described by the two–dimensional Shastry–Sutherland model.

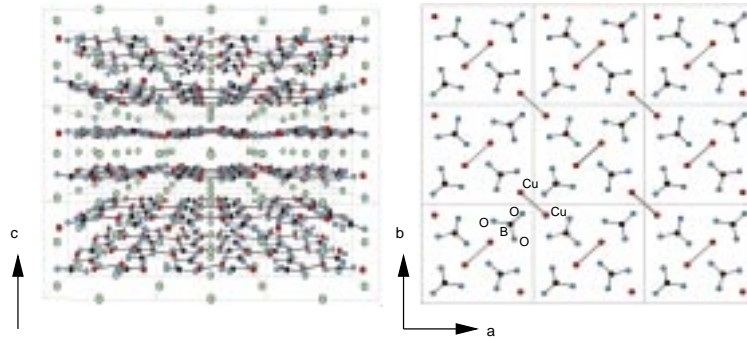[1]Courtesy to A. Bühler [24] for kindly providing the pictures.

Figure 2.4.: Crystal structure of $SrCu_2(BO_3)_2$: A layered compound of slightly buckled $Cu(BO_3)$ planes (for temperatures $T < 395\,K$) separated by Sr–atoms on the left hand side. On the right hand side a top–view on a part of a single $Cu(BO_3)$ plane is shown with nine unit cells.

The first experimental measurements in form of the magnetic response have been performed in 1999 by Kageyama et al [27]. The magnetic susceptibility measured on powder displays a maximum at $\approx$ 20K and a rapid drop towards zero with decreasing temperature that indicates an energy gap in the magnetic spectrum. Applying a simple exponential fit in the low temperature regime they derive a singlet–triplet gap of $\Delta \approx$ 19K and confirm the existence of a singlet ground state. Another exponential fit to the spin–lattice relaxation time gives a magnetic gap of $\Delta \approx$ 30K = 2.6meV to the first excitation. They conclude that $SrCu_2(BO_3)_2$ is a realisation of the Shastry–Sutherland model in the dimer phase.

A year later Kageyama et al. [28] publishes inelastic neutron scattering data obtained from a large crystal confirming a small gap $\Delta \approx$ 34K. The value of the singlet–triplet gap was confirmed by other experiments like electron spin resonance [29], far infrared spectroscopy studies [30], nuclear magnetic resonance [31] or Raman experiments [32] with $\Delta =$ 34K. These experiments also support the singlet nature of the groundstate. Only for the ESR experiments a residual interaction, like a Dzyaloshinsky–Moria interaction, needs to be taken into account to explain the excitation from a singlet to a triplet ground state [29, 33].

Concluding the experimental aspects of $SrCu_2(BO_3)_2$ the exchange couplings are positive so that the crystal is an antiferromagnet. The ratio $x = J_2/J_1$ is sufficiently small so that the system is in the dimer phase. One can find a number of fits of the model parameters to experimental data that in general are based on a simultaneous fit of the gap and the magnetic susceptibility [8, 19, 34, 26, 17]. The range of given x values for the inverse frustration is quite close to the critical value $x_c \approx$ 0.69 but a direct observation of a real substance at or close to the quantum critical point has not been accomplished so far.

A review of the theoretical results of the Shastry–Sutherland model applied to $SrCu_2(BO_3)_2$ can be found in [35].

2.2. Plaquette Model with Four–Spin Interaction

In this section we introduce a two–dimensional Heisenberg model suggested by J. Zittartz which similarly to the Shastry–Sutherland model (eq. 2.2) is constructed in such a way that it has an exactly known twofold degenerate eigenvalue which in the dimer phase regime of the phase diagram is the ground state. The Hamilton operator for this model reads

$$H = J_2 \sum_{<i,j>} \mathbf{S}_i \mathbf{S}_j + J_1 \sum_{\substack{k \\ \text{plaq.}}} \left(\frac{3}{4} + \mathbf{S}_1^k \mathbf{S}_3^k \right) \left(\frac{3}{4} + \mathbf{S}_2^k \mathbf{S}_4^k \right) \ . \tag{2.3}$$

The first sum corresponds to a standard Heisenberg model with exchange couplings J_2 between all nearest neighbors $<i,j>$. The second term represents the two–spin and four–spin interactions with coupling J_1 between spins on an individual plaquette k as depicted in fig. 2.5. The diagonal two–spin interactions on the plaquettes correspond to the twofold degenerate (dimer) ground state.
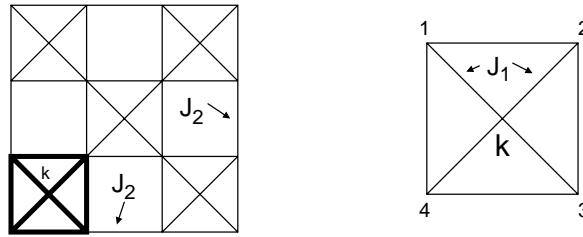

Figure 2.5.: Plaquette model proposed by J. Zittartz. Left: A Heisenberg model with exchange couplings J_2 between spins of size 1/2 located on the vertices and additional plaquettes k. Right: A representative plaquette (marked fat on the left hand side) with interactions between two next–nearest neighbored spins located on sites 1 and 3, and 2 and 4 interacting with coupling J_1 and a four–spin interaction of size J_1 between all four spins belonging to the plaquette k.

In fig. 2.6 the phase diagram of this model is shown for the elementary four–site lattice. For $J_2 < -|J_1| < 0$ one finds the ferromagnetic region with transition to the antiferromagnetic region for $J_2 > |J_1| > 0$. In the regime $J_1 > J_2 > 0$ the dimer phase is expected with the twofold degenerate ground state described above.

Similarly to the Shastry–Sutherland model an intermediate regime of a nature to be discussed between the antiferromagnetic phase and the dimer phase must be assumed. The phase boundaries on both sides of the dimer phase are not known exactly. On the antiferromagnetic side of the phase diagram the boundary of the dimer phase is assumed to be located in the region between $J_2 = J_1/2$ and $J_2 = J_1$ [36].

From a phenomenological point of view this plaquette model might be of interest as it features four–spin interactions of a simple character which currently are of
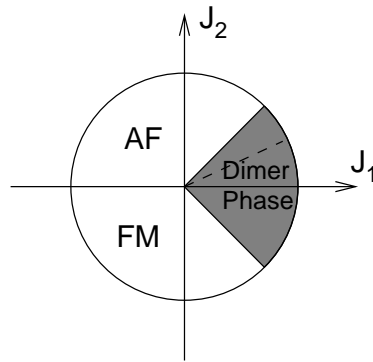
Figure 2.6.: Phase diagram of the plaquette model proposed by J. Zittartz on a four–site lattice. The phase boundaries of the dimer phase are not known exactly for the infinite size system. On the antiferromagnetic side of the diagram the boundary is located between $J_2 = J_1/2$ and $J_2 = J_1$.

special interest in related topics of physics like the spin–ladder with cyclic exchange [24] or the parent compound of high–T_c superconductors [37]: In addition to the bilinear exchange also biquadratic exchange terms, so called ‚cyclic exchange‘ terms, are important for the minimal model describing the magnetic part of cuprate systems [38, 39, 40, 41]. Also in other parts of condensed matter physics multiple spin interactions are of relevance like the nuclear magnetism of ^{3}He [42] or the spin structure of a Wigner crystal [43].

3. Method

In this chapter we describe the methods and algorithms used to cope with the problem to diagonalise a Hamiltonian matrix of very large size.

The most simple way to study a Hamiltonian like those described in Chapter 2 for the Shastry–Sutherland model

$$H = J_2 \sum_{<i,j>} S_i S_j + J_1 \sum_{<i,j>\text{dimer}} S_i S_j \tag{3.1}$$

is to explicitly write down the matrix elements of H in a basis of $\left\{ S_i^z; i = 1, 2, \ldots, n \right\}$, where we choose the z–axis as the quantisation direction, and then diagonalise H with standard eigenvalue routines like `dspev()` of the LAPACK/BLAS library on a computer.

Unfortunately this approach is quite limited as for a spin–S system of size n the number of degrees of freedom is $(2S + 1)^n$. That means for a spin–$\frac{1}{2}$ system with system size $n = 15$ one has to diagonalise a matrix with $2^{15} \times 2^{15} = 32768 \times 32768$ elements which is about the upper limit of what today's computers can solve in reasonable time periods.

To study much larger system sizes we use the *Lanczos method* [44] which gives numerically exact eigenvalues on either end of the spectrum. As this method is well known there are implementations available, such as the numerical library *ARPACK*.

In this thesis we apply different symmetry operators to reduce the size of H significantly, namely the conservation of total S_z magnetisation, spin inversion, and for periodic systems additionally translational symmetry. Other symmetries like rotational invariance and reflexion have not been implemented as for the largest system considered (6×6 Shastry–Sutherland model) these symmetries don't apply.

For the such reduced Hamiltonian H we use the so called *hashing technique* as well as the *Lin–algorithm* [45] to save the states of the subsector of the Hilbert space in a fast accessible and memory saving way which is crucial to calculate the eigenvalues in acceptable time frames.

Additionally, we apply parallel programming techniques like MPI and OpenMP (see 3.7) to distribute calculations on several processors. This is feasible as the number of states grows that strongly with increasing system size that most of the time the program uses is spent on looping through the subsector to calculate the Lanczos–vector for the next Lanczos iteration. Thus, the program part with strictly serial parts (like initialisation/checkpointing and the ARPACK subroutine) is becoming less and less important.

3.1. Symmetry–Operators

Applying symmetry operators on the model one separates the Hilbert space of the Hamilton operator H into different sectors of smaller size than the original Hilbert space.

In this thesis the total S^z conservation, spin inversion and — in case of periodic boundary conditions and depending on the specific geometric properties of the lattice investigated — the translation operator will be used. All these operators commute with the Hamilton operator (e.g. $[H, S^z] = 0$) and with each other so the resulting eigenvalues are conserved quantum numbers.

3.2. Lanczos Method

The *Lanczos–* or *Recursion–Method* is a standard method for diagonalising linear systems of equations and calculating eigenvalues and eigenvectors of sparse matrices in general [46, 47, 48]. For dense matrices one would consider using the Householder or (for the non–symmetric case) the Hessenberg method which take $\mathcal{O}(N^3)$ steps, two magnitudes more than the $\mathcal{O}(N)$ steps needed by the Lanczos method.

Especially the Hamiltonians examined in this thesis are hermitian and sparsely occupied and thus can be worked on well with the Lanczos method.

The main idea is to tridiagonalise a hermitian matrix H of size $n \times n$ with a unitary transformation.

$$X^\dagger H X = T \qquad \text{with } X^\dagger X = 1 \tag{3.2}$$

where T is tridiagonal, real and symmetric:

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \alpha_3 & & & \\ & & & \ddots & & \\ & & & & \alpha_{n-1} & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix} \tag{3.3}$$

The column vectors $X = (x_1, x_2, \ldots, x_n)$ of X are the so called *Lanczos vectors* which are orthonormal:

$$x_i^\dagger x_j = \delta_{ij}. \tag{3.4}$$

That is:

$$HX = XT \Leftrightarrow \begin{aligned} Hx_1 &= \alpha_1 x_1 + \beta_1 x_2, \\ Hx_i &= \beta_{i-1} x_{i-1} + \alpha_i x_i + \beta_i x_{i+1} \quad 2 \leq i \leq n-1, \\ Hx_n &= \beta_{n-1} x_{n-1} + \alpha_n x_n. \end{aligned} \tag{3.5}$$

Starting with an arbitrarily selected unit vector x_1 these equations will give all the α_i, β_i and x_i:

$$\alpha_1 = x_1^\dagger H x_1 \tag{3.6}$$

α_1 will be real since H is hermitian. In the next step calculate

$$\beta_1 x_2 = Hx_1 - \alpha_1 x_1 \tag{3.7}$$

and use $x_1^\dagger x_1 = 1$ to find β_1 and x_2. Similarly iterate through all the equations:

$$\alpha_i = x_i^\dagger H x_i, \tag{3.8}$$

$$\beta_i x_{i+1} = Hx_i - \beta_{x-1} x_{i-1} - \alpha_i x_i. \tag{3.9}$$

H being hermitian ensures that all the Lanczos vectors are orthonormal, e.g.:

$$x_1^\dagger x_2 = \frac{1}{\beta_1} x_1^\dagger (Hx_1 - \alpha_1 x_1) = \frac{1}{\beta_1}(\alpha_1 - \alpha_1) = 0. \tag{3.10}$$

The iteration ends when calculating

$$\alpha_n = x_n^\dagger H x_n \tag{3.11}$$

since we can show that

$$u = Hx_n - \beta_{n-1} x_{n-1} - \alpha_n x_n \tag{3.12}$$

coming out of the last equation is orthogonal to all the Lanczos vectors x_i and must therefore be zero.
A good test for the accuracy of the algorithm is to check

$$\beta_n = |u| = 0. \tag{3.13}$$

In practical applications like computer simulations we have to keep in mind that *rounding errors* will result in that the Lanczos vectors will not be perfectly orthogonal and thus $\beta_n \neq 0$. One needs to re–orthogonalise with a projection when calculating a new vector x_i to make it orthogonal to a previously calculated vector x_j:

$$x_i \rightarrow x_i - x_j(x_j^\dagger x_i). \tag{3.14}$$

Depending on how close the eigenvalues are to each other in extreme cases one needs to apply this correction after each iteration step.
A second problem that potentially can occur is that one of the β_i might be 0 so that the division will fail. This is due to the initially arbitrarily chosen Lanczos vector x_1 to be orthogonal to one of the eigenvectors of H. To overcome this, one simply needs to choose the next Lanczos vector x_i in such a way that it will be a unit vector orthogonal to all previous ones and to continue with the calculation. In practice this problem should occur quite seldomly.
The advantage of the Lanczos method is that one needs to keep only three vectors of size n in memory within one iteration step in contrast to explicitly saving the complete matrice H of size $n \times n$. Additionally, a subroutine calculating the matrix–vector product Hx is needed. Only if one needs the eigenvectors all the Lanczos vectors have to be saved.

3.3. ARPACK (Arnoldi Package)

The Lanczos method described above is a member of a class of methods called Krylov subspace projection methods that allow solving large scale eigenvalue problems. The Arnoldi method generalises the Lanczos method to the non-symmetric case for which an efficient algorithmic variant has been developed [49] that is called Implicitly Restarted Arnoldi Method. This method has been implemented in the numerical library ARPACK (ARnoldiPACKage)[1] which is designed to solve large scale hermitian, non–hermitian, standard or generalised eigenvalue problems. One can focus on specific parts of the spectrum of a matrix A, e.g. search for lowest real k eigenvalues or, as a second example, complex eigenvalues with the largest real part. Eigenvectors can also be calculated on user's request (of course with additional demand on memory). The matrix A does not need to be provided explicitly but instead the action of the matrix on a vector $w \leftarrow Av$ is all that is needed. This product is feeded into the so called *reverse communication interface* provided by the library. Eigenvalues and eigenvectors are calculated to the level of machine precision. This can be changed to arbitrary precision on user's request. One of the most important features of the library is the reverse communication interface mentioned above. As it avoids using a fixed subroutine interface it allows the user to express the matrix–vector multiplication in a convenient data structure to meet his needs. Moreover, if the matrix A is not available explicitly, the user is free to provide the matrix–vector product $w = Av$ through a subroutine call or a simple code segment as shown in the following pseudocode example:

```
while (ok==1) {
    dsaupd(ido, bmat, n, which, ..., workd, ..., info);
    ok=((ido==1)||(ido==-1));
    if (ok) {
        /* user provides matrix-vector-multiplication */
        w = workd+(ipntr[0]);
        v = workd+(ipntr[1]);
        matvecmult(v,w);
    } else {
        /* algorithm sufficiently converged */
        /* extract eigenvalues etc            */
        dseupd(...);
    }
}
```

In this example the user provides a subroutine `matvecmult()` which calculates the action of the matrix A on the vector v and saves the result in the vector w. Both vectors are part of the ARPACK specific array `workd[]` and the location within this array is saved in the second ARPACK specific array `ipntr[]`. In general the user can use any available mechanism to create the vector w. If `dsaupd()` indicates that convergence has achieved the user needs to call a subsequent postprocessing subroutine (in this special case `dseupd()`) to recover the results in a useful form.

---

[1]http://www.caam.rice.edu/software/ARPACK/

When starting the iteration process the user might want to provide a certain starting vector but he doesn't need to. In this case a randomly created unit vector is used as the starting vector. This feature is selected by setting the variable `info` to 1 or 0 when calling `dsaupd()` the first time.

As ARPACK relies on the well known standard numerical BLAS and LAPACK routines which in general are provided by the computer manufacturers in highly optimised versions, this library also performs very well on many different types of computers. Secondly, as this library is written in standard Fortran 77 it is easily portable to all architectures providing a `f77`–compiler.

As a contributed addition to the ARPACK library a *checkpointing* variant of certain subroutines of the library is available which is of essential help in situations where the user can run jobs only for a given period of (wall clock) time which definitely is not long enough to calculate a certain task completely. I.e. after a certain number of iterations the library hands over a special value of `ido` (see the pseudocode example above) so that the user can save all needed data on disk.

Another advantage of this library is that it additionally provides parallelised frontends either for MPI or BLACS. For that reason P_ARPACK is running on many different parallel systems.

During the calculations for this thesis, the checkpointing ability mentioned above has also been ported to the MPI version of the P_PARPACK library which wasn't available before.

3.4. Hashing Technique

When applying one or more of the symmetry operations described in 3.1 the original Hilbert space will be reduced to a number of representative states (usually the ,minimum' state) [45]: Consider a spin–S system of size N with total $S_z^{\text{total}} = 0$ conservation. The Hilbert space is composed of all those spin configurations $\left\{ S_i^z; i = 1, \ldots, N \right\}$ that have total magnetisation $\sum_{i=1}^{N} S_i^z = S_z^{\text{total}} = 0$.

Its dimension will be M, where M is much smaller than the dimension of the original Hilbert space $(2S + 1)^N$. Labelling of the sites $i = 1, \ldots, N$ is arbitrary (but should be chosen well depending on the geometry of the system investigated and of the symmetry operators applied). But once chosen these basis states of the reduced Hilbert space must be treated consistently.

To implement numerical calculations usually a given spin configuration $\left\{ S_1^z, \ldots, S_N^z \right\}$ is defined as a representative integer I according to

$$ I = \sum_{i=1}^{N} s(i)(2S + 1)^{i-1}, \tag{3.15} $$

with $s(i) = S_i^z + S = 0, \ldots, 2S$ resulting in a one–to–one correspondence between a single integer and a spin configuration.

In the original Hilbert space all possible integers I from 1 to $(2S+1)^N$ would then be needed.

In the example (total $S_z = 0$ conservation) only M of all the integers I will occur so that one has to introduce some kind of lookup table labelling the allowed I–states in an arbitrary way $\left| 1 \right\rangle, \left| 2 \right\rangle, \ldots, \left| M \right\rangle$.

When applying the Hamiltionian H on such a spin configuration many other spin configurations will be generated for which one has to lookup the table $|1\rangle$, $|2\rangle$, ..., $|M\rangle$ generated before.

The problem is to find the locations of these spin configurations in the storage table.

A naive way to find the right location is to simply introduce a vector $J(I) =$ position of the spin configuration represented by I in the table. However, it is obvious that one has to allocate up to $(2S + 1)$ places for $J(I)$ although most of the places will be useless null entries:

In case of $N = 4$, $S_z^{\text{total}} = 0$, $S = \frac{1}{2}$ one will have the basis of $M = 6$ representatives $|0011\rangle, |0101\rangle, |0110\rangle, |1001\rangle, |1010\rangle, |1100\rangle$. But with the naive method described above one needs to allocate $1100_2 = 12_{10}$ places (the value of the numerically largest spin configuration of the subsector investigated) of which only 6 actually would be used. The relation of unneeded places to allocated places is much worse for larger system sizes N so this method is unfeasible.

A somewhat more sophisticated way to create a storage table is the so called *hashing technique*:

A hashing function $h(I)$ is constructed that gives a correspondence between the M representatives $I \in \{|1\rangle, |2\rangle, \ldots, |M\rangle\}$ and a position vector $h(I)$. This function often is defined as [50]

$$h(I) = [I(\text{mod}\,K)] + 1. \tag{3.16}$$

The size of memory used by this function is about K which is of the order of M. Usually K is chosen as the smallest prime number larger than M. For some representatives $I_1, I_2, \ldots$ it happens that they have the same remainder, i.e. ‚collisions‘ will occur.

When choosing $K = 3$ in the example described above (one wouldn't really choose 3, but here just for demonstration) the representatives $|0011\rangle, |0110\rangle, |1001\rangle, |1100\rangle$ will have the same remainder and thus will ‚collide‘.

For that reason one either has to chose the prime number K that large that no collisions do occur or otherwise create a two–dimensional array in which for each remainder the list of corresponding representatives is kept.

A sample piece of code in the programming language C for this algorithm is provided in A.1.

We actually use this code in case of translational invariance. That means that in case of the 6×6 Shastry–Sutherland model (consisting of 504.174.594 representatives) the hashing technique is applied with a sufficiently large prime number (PRIME= 5.915.587.277) so that no collisions actually do occur. That means that the hashing algorithm used in the simulations should not be slower than the more sophisticated *Lin–Algorithm* described in the following section. By applying the parallel programming techniques described below the program is efficiently using a whole cluster node containing 32 processors and 112GB of available main memory of which about 52GB (PRIME$\times$8 bytes $+$ number of representatives $\times$8 bytes) are left over for the hash array without any problems.

3.5. Lin–Algorithm

Although the hashing technique described in 3.4 is quite simple to implement it nevertheless is really memory consuming as it allocates much more places (depending on the prime number used) than needed for the number of representative states contained in the Hilbert subspace being investigated.

In case of total S_z conservation a more sophisticated way to create a lookup table is the algorithm developed by Lin [45]. In contrast to the hashing technique which in effect is a one dimensional sequential search through a lookup table he suggests a two dimensional storage table (that in principal can be extended to even higher dimensions):

Divide the lattice investigated in two parts A and B and define two integers

$$I_a = \sum_{i=1}^{[N/2]} s(i)(2S+1)^{i-1} \tag{3.17}$$

$$I_b = \sum_{i=1}^{[N+1/2]} s(i+[N/2])(2S+1)^{i-1}, \tag{3.18}$$

with $[X]$ as the integer part of number X. Correspondingly define two vectors $J_a(I_a)$ and $J_b(I_b)$ so that the position of the spin configuration represented by the integer I results as the sum

$$J = J_a(I_a) + J_b(I_b). \tag{3.19}$$

One immediately sees

$$I = (2S+1)^{[N/2]} I_a + I_b \tag{3.20}$$

and that (J_a, J_b) behaves just like a two–dimensional coordinate (x, y). The advantage of this approach is that the maximum length of J_a and J_b is $(2S+1)^{[(N+1)/2]}$ which is about the square root of that of $J(I)$. This surely is a considerable improvement compared to the hashing technique described above.

When a spin configuration represented by I is changed to another configuration represented by $\tilde{I}$ due to the application of the Hamiltonian H on can easily find $\tilde{I}_a$ and $\tilde{I}_b$ and then immediately the resulting $\tilde{J}$.

Thus, this method gives a one–to–one correspondence between different spin configurations and their position in the lookup table, and it uses very little memory compared to the hashing technique described before.

We have applied this algorithm also in the case of total S_z conservation and additional spin inversion which just halves the memory needed for this kind of lookup table. See A.2 for a sample code implementation in C.

In case of other symmetries applied (like translational invariance in periodic systems) this method is rather difficult to implement and depends extremely on the lattice being investigated. But in case of just total $S_z = 0$ conservation and spin inversion this method is quite simple to implement.

Here an example for a spin–$\frac{1}{2}$ system of size 4 with total $S_z = 0$ conservation (table 3.1):

Configuration	a	b	I_a	$J_a(I_a)$	I_b	$J_b(I_b)$	$J = J_a + J_b$
1100	11	00	3	0	0	0	0
0101	01	01	1	0	1	1	1
1001	10	01	2	1	1	1	2
0110	01	10	1	0	2	3	3
1010	10	10	2	1	2	3	4
0011	00	11	0	0	3	5	5

Table 3.1.: Spin configurations, their representations I_a and I_b, their ‚coordinates'
J_a and J_b and their positions in the storage table J for a spin$-\frac{1}{2}$ system
of size 4 with total $S_z = 0$ conservation.

3.6. Serial Programming Approach

As a very first step in this project we created a program to calculate eigenvalues
that is structered as follows:

```
init();
hash_create();     OR      lin_create();
rcl() {
  while (ok) {
    dsaupd() /* ARPACK */
    checkpoint();  /* test whether to checkpoint and quit the program */
    if (ok) {
      /* now the user has to provide the action of matrix A on vector x  */
      for all possible spin configurations <i> in the reduced Hilbert space do {
        /* apply interactions with symmetries corresponding
           to the model investigated */
        model() {
          newstate=exchangebits()
          /* apply translational symm. and find representative in the
             reduced Hilbert space */
          findtransmin(newstate)
          /* additionally apply spin inversion and find representative in the
             reduced Hilbert space */
          findtransmin(inversionmask^newstate)
        }
        for all interactions <i> defined in model
          y[statecount]=coeff(i) * x[representative(i)]
      }
    } else {
      /* ARPACK converged successfully; postprocessing to find eigenvalues */
      dseupd()
    }
  }
}
output(); /* results to be written on disk */
```

After an initialisation phase `init()` where global arrays and variables are defined
and allocated the storage/lookup table in which the positions of the spin config-
urations in the reduced Hilbert space are saved will be created either via hashing

technique or Lin algorithm (in case of total S_z magnetisation and/or spin inversion, only) by using the `hash_create()` or `lin_create()` subroutine.

Later the program will enter ARPACK's reverse communication interface in `rcl()`. After an initial call to the ARPACK routine `dsaupd()` which sets up the internal ARPACK infrastructure and giving out a first (randomly created) unit vector x the program will provide the user's individually implemented matrix–vector multiplication, in this case it is done serially by means of a loop over all allowed spin configurations.

The subroutine `model()` is called for each indiviual spin configuration which applies the interactions between the distinct spins of the model investigated.

The program has been written in such a way that many different models can be integrated into the program easily (e.g. it is no problem to examine one–dimensional spin–chains or three–dimensional cubic lattices), as all the interactions beetween the spins need to be written down explicitly with calls to the `exchangebits()` subroutine.

During this stage of the program also the model specific symmetry operations are applied (via `findtransmin()` for translational invariance and `findtransmin(inversionmask^newstate)` for translational invariance combined with spin inversion). Of course, in this part of the program very specific properties (boundary conditions, geometry of the model etc) of each model investigated need to be implemented. After applying the symmetry operations the representative spin configuration has to be found (also being part of `findtransmin()`) which usually is the ,minimum' state of all possible configurations belonging to each other.

The matrix–vector multiplication is saved in a vector y which itself is feeded back into the ARPACK routine `dsaupd()` in the next loop step of the Lanczos algorithm. Either a new vector x is offered to the user's matrix–vector multiplication, or the reverse communication process will be stopped via the variable `ok` because the convergence criterion has reached or because checkpointing via `checkpoint()` should be performed to temporarily interrupt the program as the maximum CPU time has been reached (or in terms of loop steps: the maximum number of allowed `dsaupd()` iterations has taken place). In that case the program needs to be set up again with a certain flag so that it will read in the checkpointing information saved before.

In case the convergence criterion is fulfilled the program enters the postprocessing subroutine `dseupd()` described in 3.3 which provides the specified eigenvalues and/or eigenvectors the user is interested in. The results are exported to harddisk in the finishing routine `output()`.

Using this serial program we were able to produce results for the Shastry–Sutherland model with up to a system size of $N = 32$ (equivalent to 37.582.307 representative spin configurations). To calculate a single value close to the critical point on a Sun Fire 15K machine with UltraSPARC III 900MHz processors this program needed about 14 days of computing time.

For the Plaquette model with double the number of diagonal bonds and additional four–spin interaction the program needed nearly three weeks (19 days) for a value next to the critical point. Having this experiences in mind it was clear that addi-

tional techniques had to be applied to calculate the eigenvalues for larger system sizes.

3.7. Parallel Programming Environments

In this section we will give a short overview of the two most commonly used parallel computing environments. On the one hand *MPI* (Message Passing Interface) [51] which is available especially on so called *distributed memory* machines as well as on symmetric multiprocessor machines and on the other hand *OpenMP* (Open Multi Processing) which is available exclusively on *symmetric multiprocessor* (SMP) machines.

On SMP architectures a number of processors share system resources like memory and I/O subsystems that can be accessed equally from all the processors. They are typically controlled by a single operating system (kernel) which schedules processes containg a single or more threads on processors in such a way that the load is balanced evenly on the system. The processors usually are connected by a bus or a crossbar switch.

In contrast to this architecture, distributed memory machines consist of nodes that are connected by a network that is typically high–speed (HiPPI, Myrinet, standard Gigabit Ethernet to name a few). Each node has its own processor, memory, and I/O subsystem and is running an individual instance of the operating system, i.e. each node can be considered a workstation.

The main difference between the two models is how data is shared between different processors. On a SMP machine in an OpenMP process with several threads the corresponding processors have direct access to shared data in memory. On the other hand an MPI process on distributed memory architecture has to explicitely send data to and receive data from other processors over the network mentioned above.

An important aspect one has to keep in mind is what is called *Amdahl's law*. This law gives an idea of the *speedup* of a program running in parallel compared to the same program running on a single processor (i.e. the serial program version):

$$S(n) = \frac{n}{n * b + (1 - b)} \qquad (3.21)$$

where n is the number of processors being used and $0 \leq b \leq 1$ is the strictly serial part of the program that cannot be parallelised.

It turns out that the speedup behaves logarithmic, i.e. depending on the serial part of the program it doesn't make sense to allocate more than a few processors. In principle this law is valid for both, MPI and OpenMP (see figure 3.1).

3.7.1. MPI

MPI is an implementation of a standard wich usually is delivered in an optimised version by the computer manufacturer in form of a library that needs to be linked to the user's program:

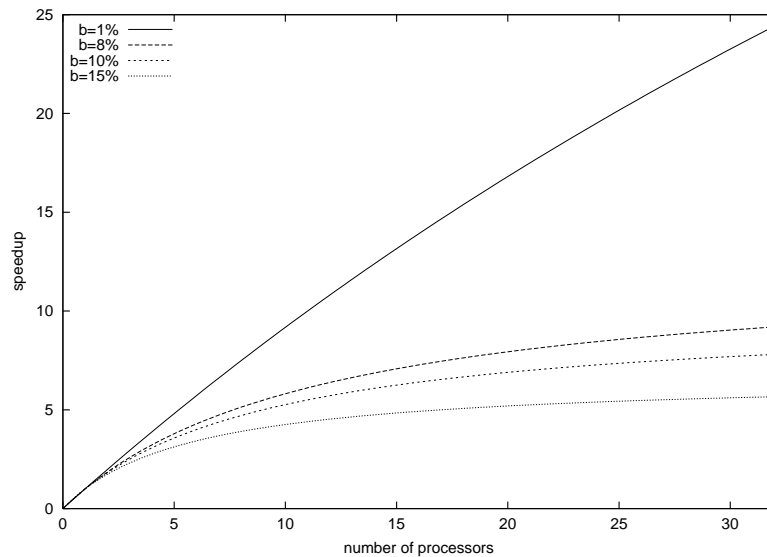```
cc -o a.out prog.c -lmpi
```

Figure 3.1.: Only for very small serial parts of a program the CPU time consumption of the program scales well with the number of the processors allocated.

When starting a program the user has to provide the number of processors on which the program should run, i.e. the same program runs in $n$ instances *identically* on all the processors managing and controlling all their own memory and I/O resources. The user has to take care what data each processor is working on (*data distribution*) and that data is exchanged in an explicit and controlled way between the processors (*communication*) so that no deadlock situation and other race conditions can occur. This is achieved generally by different types of communication: blocking and non–blocking point–to–point or broadcasting communication (fig. 3.2) using so called *tags* which are nothing but simple numbering labels for each sending and receiving action of a certain amount and type of data.

For instance, to transfer an array $x$ of length $n$ of data type `double` from processor 'src' to processor 'dest' with non–blocking point–to–point communication one would code (in C) the following lines:

```
/* Using non--blocking communication ensures that deadlocks don't occur, */
/* i.e. a process can send and reveive data at the same time.   */
/* procsendtag of sending process and procrecvtag on receiving
   process must be equal. */
MPI_Isend(x,n,MPI_DOUBLE,dest,procsendtag,MPI_COMM_WORLD,&procsendireq);
MPI_Irecv(xneu,n,MPI_DOUBLE,src,procrecvtag,MPI_COMM_WORLD,&procrecvireq);

/* checking whether data transfer has finished */
MPI_Test(&procsendireq,&procsendmpiflag,MPI_STATUS_IGNORE);
MPI_Test(&procrecvireq,&procrecvmpiflag,MPI_STATUS_IGNORE);

/* continue checking end of sending transfer indicated by flag==1 */
while (1 != procsendmpiflag) {
```
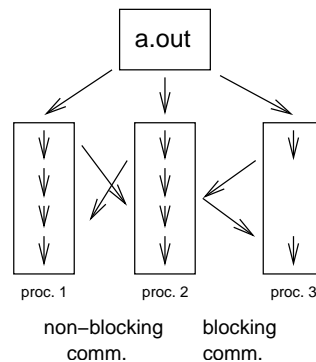
Figure 3.2.: General example for the MPI concept: A process a.out runs on three processors with non–blocking communication between processors 1 and 2 and blocking communication between processors 2 and 3.

```
  MPI_Test(&procsendireq,&procsendmpiflag,MPI_STATUS_IGNORE);
  MPI_Test(&procrecvireq,&procrecvmpiflag,MPI_STATUS_IGNORE);
}

/* continue checking end of receiving transfer indicated by flag==1 */
while (1 != procrecvmpiflag)
  MPI_Test(&procrecvireq,&procrecvmpiflag,MPI_STATUS_IGNORE);
```

3.7.2. ARPACK and MPI

As the ARPACK library used in this project provides support for the MPI parallel programming interface in form of the P_ARPACK program package we at first chose this way to implement a parallel version of a program to calculate the lowest eigenvalues.

As explained in section 3.3 the ARPACK library has a so called *reverse communication interface* for which the user has to provide the matrix–vector multiplication in any convenient programming technique (here: MPI) which fits best the user's needs.

Both vectors used in the Lanczos iterations, x and y (with $y = Ax$), are equally distributed on all the processors the user has allocated when starting the program. Vector x is accessed read–only in contrast to vector y that needs to be calculated. With a number of different approaches we tried to enhance the serial program in such a way that is was able to reliably access the data on the other processors.

A first naive approach was to implement communication in such a way that each time processor i needed a single element of the (read–only) vector x from processor j it immediately started a communication process. This process generally consists of two parts. Processor i has to transmit the index number of the element to processor j which at the same time has to listen to processor i to be able to receive the index number.

In the second step processor j starts sending back the element of the array x to processor i which has to listen for processor j in that moment. Afterwards

processor i can continue its calculation of vector $y = Ax$. This process has been implemented in non–blocking communication so that processor i doesn't have to wait for the answer of processor j and also to avoid dead–lock conditions since of course not only processor i tries to calculate components of y but also all other processors $\{0, \ldots, i-1, i+1, \ldots, numproc-1\}$ where $numproc$ is the number of processors allocated by the user.

This leads to an extremely complex communication pattern for two reasons: a) Each processor needs to send data to all other processors and has to listen to all other processors to be able to receive data and b) within each communication event only a single element of x is transmitted.

The implementation lead to a complex program code that at the same time became about 300 times slower than the serial program version.

In a second approach we tried to overcome problem b) by introducing structures so that processor i was collecting a whole group of index numbers needed from processor j so that many individual communication requests could be bundled to a single one. This approach on the one hand lead to an even more complex code as new structures and the limit of the maximum number of elements to be transferred needed to be introduced. On the other hand we gained a factor of 100 in performance compared to the first version. But still this parallel program version was slower by about a factor of three compared to the serial version.

We then turned to eliminate problem a) as still all processors were communicating with all processors.

Starting over we designed the communication process in a different way.

Due to the complex nature of the interactions of the models studied in this thesis the read–only vector x cannot be restructured in such a way that all elements of interest for a certain processor can be saved locally. For that reason we ordered the communication in a closed ring so that each processor received information from its predecessor and sends information to its successor exclusively. This structure also has the advantage that each processor doesn't have to maintain lists for *all* other processors but only for its direct neighbors.

Additionally with each single communication process the largest possible number of elements is transferred: the complete processor's portion of the vector x is transferred at once to the successor.

Implementing this communication pattern lead to the following scenario for calculating $y = Ax$:

The first step of the matrix–vector multiplication was to use all elements of the vector x that initially were located on the local processor.

Then the circular communication started on all processors by sending away their portion of x to their successors and at the same time the new elements of x receiving from their predecessors. After the communication event each processor was looping again over all spin configurations of the subspace the same way as during the first step with the initially local elements.

This is in principle the structure used in the serial program version with an additional circular communication structure leading to a further performance gain so that the parallel program was about as fast as the serial version.

This MPI approach revealed an essential structural problem arising from the large

size of the matrix A. As we cannot explicitly save the matrix elements we have to calculate them ‚on the fly‘ leading to a loop in the program that iterates over all possible spin configurations to apply the lattice specific interactions of the model investigated. In fact, each time processor i received a new portion x_j from its predecessor $i - 1$ the loop running over all representative spin configurations has to be started again in order to find the elements of x_j needed.

The result is that this version of a parallel programming approach didn't evolve a reasonable scaling behavior. Independent of the number of processors allocated it needed about the same wall clock time as the serial program.

An additional problem arises. Using MPI we are working with a distributed memory architecture. For that reason the storage/lookup table introduced in section 3.5 has to be kept on each processor. As this array is one of the most memory consuming parts of the serial program memory management for the parallel program becomes practically impossible for the large sizes we want to study.

For these reasons a different programming approach needed to be found.

3.7.3. OpenMP

In contrast to MPI the OpenMP standard is implemented in the compiler. The user has to mark those parts of the program to be auto–parallelised with so called OpenMP *compiler directives* that look similar to preprocessor statements (here the C compiler is used):

```
#pragma omp parallel
```

This technique is available also for Fortran and C++ Compilers but one should keep in mind that the C++ version always will be behind the development of the features for the other programming languages.

When compiling the sources the user needs to explicetly switch on the compiler's OpenMP features (here the Sun version is shown):

```
cc -xopenmp -o a.out prog.c
```

When not using the OpenMP interface of the compiler (i.e. leaving away the `-xopenmp` switch the `#pragma` line shown above is treated like a comment (thus being simply ignored).

When starting a program the user has to provide the number of threads the process is started with by setting an environment variable. In the following example a program is started running with 5 threads (syntax assuming one of the bourne shell derivatives):

```
$ export OMP_NUM_THREADS=5
```

The program then will start corresponding to the so called *Fork–and–Join model* [52], i.e. it first will run like a normal serial program on a single processor (using a single thread, called *master thread*) and subsequently allocating then additional processors for the other threads specified by the environment variable mentioned above (fig. 3.3) when entering the first parallel region that has been specified by the user:

During the serial parts of the program the user principally can decide whether to keep the unused processors or to free them so other users have access to them. In general the user will keep them for own purposes (because the time not using
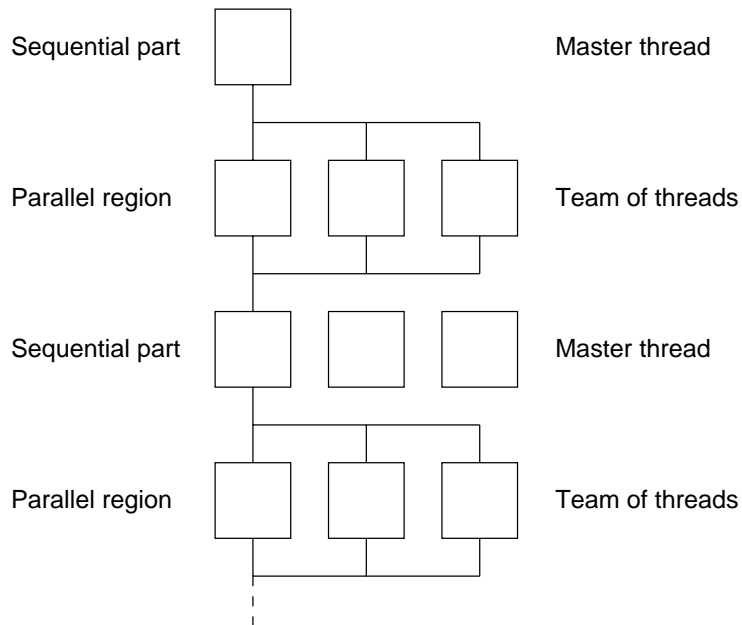
Sequential part   Master thread

Parallel region   Team of threads

Sequential part   Master thread

Parallel region   Team of threads

Figure 3.3.: Fork–and–Join model used by OpenMP programs with several threads

these processors should be as low as possible).

The essential task the user has to solve when programming with OpenMP is to care about variables and fields that will be changed during parallel processing. These variables need to be marked for the auto–parallelisation mechanism in such a way that the compiler knows that it has to create local copies for each thread while working in the parallel region.

3.7.4. ARPACK and OpenMP

For several reasons the MPI standard could not help us to parallelise the serial program: First the ARPACK specific data distribution of the Lanczos vector x, second the fact that each processor needed it's own copy of the storage/lookup table (Lin or Hash) which is a major memory problem and third resulting of the data distribution the communication overhead for sending around the needed information.

For other models where extremely sparse matrices are used the P_ARPACK/MPI alternative is a good method to calculate eigenvectors and other data of interest. E.g. in quantum chromodynamics this parallel package is used [53] to calculate eigenmodes of the Wilson–Dirac matrix and the Hermitian Wilson–Dirac matrix.

In our case the dimers on the two–dimensional lattice lead to a more complex matrix which doesn't seem to allow this technique to be applied without problems. Knowing that the ARPACK subroutine is consuming only a small part of the program (in terms of CPU time, i.e. for a 6×6 system we find for `dsaupd()` 50 seconds of CPU time and for the user supplied matrix–vector multiplication

about 1400 seconds per iteration) we changed the serial programming approach
described in 3.6 in such a way that only that the ARPACK part is left running
serially and that the user supplied matrix–vector multiplication is parallelised auto-
matically by the compiler using the provided OpenMP directives.

This solves all the problems described above and has additional advantages in our
case:

As we use this approach on a shared memory architecture the Lanczos vector x
is available for all the processors the user has ordered. I.e. this information does
not need to be handed round with time consuming MPI communication calls.
Also filtering out the needed parts and ignoring the unneeded parts of x is not
necessary (as described in section 3.7.2).

Especially the storage/lookup table needs to be kept in memory only once for all
the processors working on the problem which is — using the hashing technique
— the most memory consuming part of the program.

Third, the Lanczos vectors don't need to be distributed evenly on the processors,
instead the loop running over all spin configurations will be automatically distrib-
uted as even as prossible on the processors (i.e. a loop running from 0 to 9
distributed on 3 processors will result in one processor counting from 0 to 3 ($= 4$
iterations) and the both other processors each iterating 3 steps) which is also
important for an evenly distributed load average.

Fortunately the loop over all spin configurations calculates individual elements of
the subsequent Lanczos vector y, i.e. the iterations don't depend on each other
so that we don't have to care about data integrity.

```
#include <omp.h>      /* Mandatory header file to be included */
/* Here some globally defined variables and arrays: */
long *transfeld,*b;
double *invfaktor;
int invcounter;
int *indexliste;
long i_lin;
/* With a compiler directive mark globally defined variables as
   private for each thread ordered by the user (via the
   environment variable OMP_NUM_THREADS), i.e. the compiler will
   introduce internal copies for these variable for each thread
   to keep them safe and avoid collisions when two thread at the same
   time want to write on these variables. */
#pragma omp threadprivate(indexliste,i_lin,transfeld,b,invcounter,invfaktor)
init();
hash_create();    OR      lin_create();
rcl() {
  while (ok) {
    dsaupd()  /* serial ARPACK version, in contrast to the MPI versions */
    checkpoint();  /* test whether to checkpoint and quit the program */
    if (ok) {
      /* Here the parallelised region will start with user supplied
         matrix-vector multiplication taking the most time used by the
         program. */
      /* Globally defined arrays (see above) written to during the loop iterations
         need to be allocated within a parallel region so each thread will have
         it's own copy of the array. Additionally the local variable i is marked
         to be a private copy for each thread. */
```

```
 #pragma omp parallel private(i)
         {     /* this paranthesis needed as a whole region will be parallelised */
           indexliste = (int *) calloc((size_t) tauschint, (size_t) sizeof(int));
           transfeld = (long *) calloc((size_t) maxmult, (size_t) sizeof(long));
           b = (long *) calloc((size_t) maxmult, (size_t) sizeof(long));
           invfaktor = (double *) calloc((size_t) tauschint, (size_t) sizeof(double));
           /* The following for-loop will be auto-parallelised with OpenMP */
           /* The loop variable <statecount> will be marked private for
              all threads automatically */
 #pragma omp for
           /* now the user has to provide the action of matrix A on vector x  */
           for all possible spin configurations in the reduced Hilbert space <statecount> do {
             /* apply interactions with symmetries corresponding
                 to the model investigated */
             model() {
               newstate=exchangebits()
               /* apply translational symm. and find representative in the
                  reduced Hilbert space */
               findtransmin(newstate)
               /* additionally apply spin inversion and find representative in the
                  reduced Hilbert space */
               findtransmin(inversionmask^newstate)
             }
             for all interactions <i> defined in model
               y[statecount]=coeff(i) * x[representative(i)]
           } /* end of 'for' loop
           /* clean up global arrays for which OpenMP instances have
              been created before: */
           free(indexliste);free(transfeld);free(b);free(invfaktor);
         } /* end of parallel region */
       } else {
         /* ARPACK converged successfully; postprocessing to find eigenvalues */
         dseupd()
       }
     }
   }
 output(); /* results to be written on disk */
```

Compared to the serial code presented in 3.6 the OpenMP code looks similar. With several changes to the serial code the auto–parallelisation via OpenMP can be implemented:
At the beginning of the program the mandatory C header inclusion line
`#include <omp.h>`
needs to be inserted so that OpenMP specific subroutines are available to the user similar to the MPI interface, e.g. `omp_get_num_threads()` to find out within the program how many processors have been allocated by the user setting the environment variable `OMP_NUM_THREADS` in his shell. `omp_get_thread_num()` usually is used to find out the thread number (important for data or load distribution like in MPI).
In our program we are working with a number of global variables. Such global variables need to be marked by `threadprivate()` as they are going to be modified in an OpenMP parallel region (so that the program needs local instances within the distinct threads to conserve data integrity).

For the scalar variable `invcounter` nothing special needs to be considered later in the parallel region.

Additionally several arrays like `transfeld` etc are defined. For such global arrays one has to know that memory allocation via `malloc()` or `calloc()` needs to be performed *within* the parallel region they will be modified. The reason is that only when allocating memory *within* the parallel region each distinct thread will have it's own instance of the array to work on.

When allocating *outside* the parallel region the distinct threads each indeed would see the allocated memory under private names but it would be the same memory location being worked on uncontrolled und thus most surely leading to program abortion or inconsistent data.

The `threadprivate()` instruction is used only for *globally* defined variables.

Local (also called *stack*) variables and arrays that are worked on by each thread don't have to be marked with the `threadprivate()` statement but instead have to be marked when starting the parallel region with

`#pragma omp parallel private(var1,var2,field1,field2,...).`

as shown in the code example.

Additionally, all variables and arrays defined in subroutines called from within a parallel region are automatically defined as `private` (in our coding example above this would be all variables defined e.g. in `model()`, `exchangebits()`, `findtransmin()`, etc.). This is called *orphaning* in OpenMP terms.

Arrays and variables used read–only in parallel regions don't need to be marked as private, instead they can be marked as `shared` (which is the default with Sun Compilers so it's not necessarily needed but explicitly recommended to have control over all variables used).

In the programming language C the `#pragma omp parallel` statement is acting only on the directly following statement in the next line. In order to auto–parallise a whole region, one needs to add additional curly brackets { ... } around the region as shown in the code example.

Now having cared on all variables used we succeed and let the compiler auto–parallelise the loop over all spin configurations with the very simple statement:

`#pragma omp for`

The reason why we have decided that especially the loop shown in the code example should be parallelised is that for largest system size $N = 6 \times 6 = 36$ the loop iterates over 500 million steps which is a high enough number to be parallelised even on a quite large number of processors (up to 32 processors in our case). Second, in the called subroutines only scalar variables (no large private arrays) are used so no difficult (and thus time consuming) memory management (in form of array allocating/copying/deleting) needs to be performed by the system. Also, this loop is located quite high–level in the program part consuming the most CPU time so this is a good choice to parallelise with only a few OpenMP directives in an effective way.

4. Numerical Results for the Shastry–Sutherland Model

By applying the methods described in chapter 3 we have calculated the ground state energies for a number of systems on two–dimensional lattices up to system size $N = 36$ spins.

To verify that our program was calculating the correct energies we reproduced data for systems with open boundary conditions calculated before by U. Löw and E. Müller–Hartmann [12]. Their results represent strict lower bounds on the critical value of the inverse frustration with $x = J_2/J_1$ as depicted before in fig. 2.2. They show a $1/\sqrt{N}$ plot to find a rough estimate of $x_c \to 0.65$ for the lower bounds for systems of size $N \to \infty$. In fig. 4.1 we present an overview of the critical values of the inverse frustration where the system undergoes a phase transition of first order.
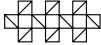
	Shastry–Sutherland model	
N=4	⊡	0.5
N=12		0.5658
N=17		0.5840
N=20		0.5789
N=24		0.5910
N=28		0.5847
N=31		0.5914

Figure 4.1.: Critical values x_c for the phase transition of the dimer phase to the intermediate phase for $J_2 > 0$ in lattices of the Shastry–Sutherland model with open boundary conditions.

Having crosschecked that the program yields correct results we calculated the ground state energies for lattices up to $N = 36$ spins with periodic boundary conditions. The smaller systems up to size $N = 32$ were calculated at the local computing center of the University of Cologne (RRZK) on a Sun Fire15K with 72 UltraSPARC III Cu processors running with 900Mhz and 144GB amount of physical main memory. When using the hashing technique for the lookup table for the mapping of the states between the complete Hilbert space and the subsector depending on the symmetries applied to the individual model we chose the prime number in such a way that the maximum free memory of the shared memory machine was allocated. Typically this was in the range of $50 - 80$GB fortunately leading to a quite low collision number (at most 10 collisions to account for) in the hash table which is an important condition for good performance numbers. On this machine we used the serial code to calculate data for both, the Shastry–Sutherland model and the plaquette model, up to system size $N = 8 \times 4 = 32$ with periodic boundary conditions equivalent to a subsector of the Hilbert space containing 37.582.307 states.

First we present a summary of the results obtained for the periodic systems of the Shastry–Sutherland model shown in fig. 4.2 and next we discuss the specific details for the different systems.

	Shastry–Sutherland model	
N=8		0.56414
N=16		0.6665
N=18		0.6260
N=24		0.66609
N=32		0.66509 extrap.
N=32		0.67753 extrap.
N=36		0.67501 extrap.

Figure 4.2.: Critical values for the crossover of the dimer phase to the non–dimer phase in lattices of the Shastry–Sutherland model with periodic boundary conditions.

In case of the $N = 32$ system we have two possible lattices as shown in fig. 4.2.

We first focus on the 8×4 lattice which we partially have calculated on the Sun Fire15K system described above and partially on the IBM p690 eServer Cluster 1600 supercomputer facility at the Forschungszentrum Jülich. This computer system contains 1312 processors on 41 individual nodes, each constisting of 32 Power 4+ processors running at 1.7GHz with 128GB of physical main memory of which in the end 112GB are available for the user.

The 8×4 lattice is calculated using the following symmetries: conservation of the total S^z magnetisation, spin inversion and translational invariance in both directions leading to the size of 37.582.307 states in the subsector of the Hilbert space that contains the ground state.
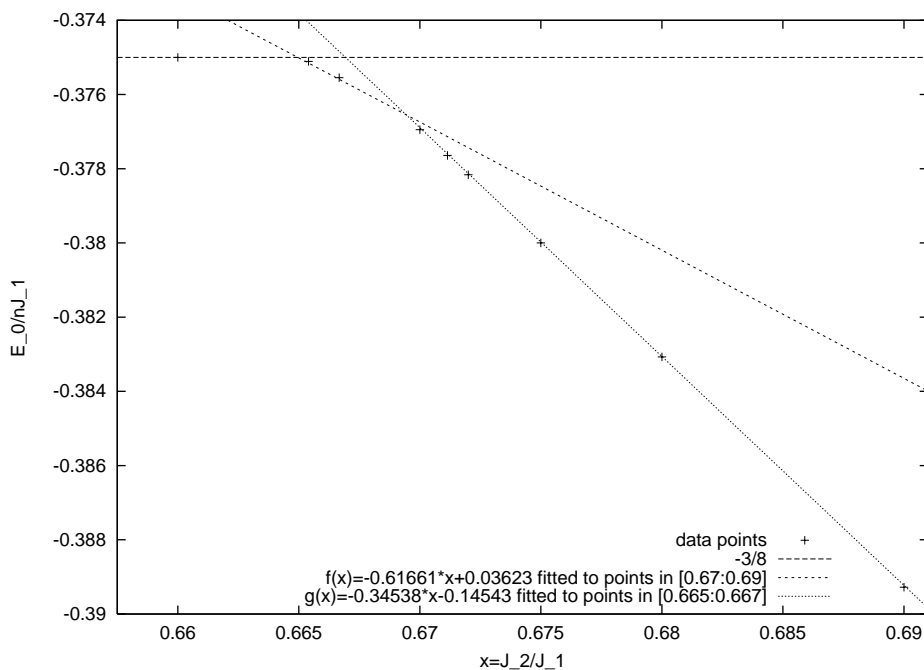

Figure 4.3.: Ground state energies per dimer obtained for a $N = 8 \times 4$ Shastry–Sutherland model with periodic boundary conditions. Although at first sight the data points in the non–dimerised regime $x > 0.67$ seem to lie on a straight line for large x this indeed is not the case as indicated by the two linear fits $f(x)$ and $g(x)$. The extrapolated value for the crossover through the two data points next to it is $x_c \approx 0.66509$.

A performance gain can be seen in table 4.1: Numbers have been calculated on both, the Sun Fire15K serially and the IBM in parallel. We came down from calculation times of nearly two weeks real time (also called *wall clock time*) to one to two days. Focussing on the data point $J_2/J_1 = 0.6554$ and keeping in mind that a single Power 4+ processor of the IBM machine is about a factor of 3 faster than a single UltraSPARC III Cu processor of the Sun system one would expect a calculation time of 278h/3≈93h serially. On the other hand the point is much closer to the critical point x_c which might lead to a doubling of the CPU

J_2/J_1	$E_0/J_1 n$	wall clock time	accum. CPU time
0.50000000	-3.7499999999989E-001	126h $\approx$ 5.25d	serial
0.66540000	-3.7510622099930E-001	35h	1105h
0.66666667	-3.755437096962E-001	278h $\approx$ 11.5d	serial
0.67000000	-3.769459588433E-001	19h	620h
0.67114094	-3.776395522653E-001	295h $\approx$ 12.3d	serial

Table 4.1.: Data points calculated for the $N = 8 \times 4$ Shastry–Sutherland model. The dimension of the subsector of the Hilbert space containing the ground state was 37.582.307

time: $\approx$180h serially which for 32 processors used in parallel would ideally mean an expected calculation time of about 6h wall clock time. In contrast to this the really used wall clock time was 35h.

This can be explained by a bad relation of the amount of serial code compared to the amount of parallelised code in the program.

When checking the logfiles we indeed find typical entries like this:

```
...
real time used for initialisation: 205 seconds
...
real time used for dsaupd(): 10 seconds
real time used for this lanczos step: 66 seconds
...
```

This means that compared to typical runtimes for a single job of 2h20m the (serial) initialisation phase takes about 2.5% and each (serial) `dsaupd()` ARPACK call takes $\approx$17%. So in fact only 80% of the code is in fact parallelised. From Amdahl's law shown in fig. 3.1 we can see that only for very few processors (5-10) the scaling behavior would be good so in this case we did in fact allocate too many processors.

Using the UNICORE[1] client which enables the user to organise all tasks concerning the queueing system of the supercomputer facilities in Jülich the user gets a bonus of 30% of the used CPU time, i.e. in total for this specific system we were charged only with 5.6% of the total available CPU time.

Extrapolation via a linear function through those two data points located next to the critical point x_c leads to an expected value of $x_c \approx 0.66469$.

The other system investigated with $N = 32$ spins has a quadratic geometry (cf. fig. 4.2). Compared to the $N = 8 \times 4$ system discussed above we have less translational symmetry due to the quadratic character of the lattice. Using conservation of the total S^z magnetisation, spin inversion and translational symmetry leads to a subspace containing the ground state which is about twice as large as that of the 8×4 system: 75.164.451 states. The graph shows the data points calculated in fig. 4.4.

Data obtained for this system are listed in table 4.2. As the subspace is twice as large as that of the stripe lattice the run times correspondingly grew higher as can be read off table 4.2. To our surprise for the data point $x = 0.6778$ the
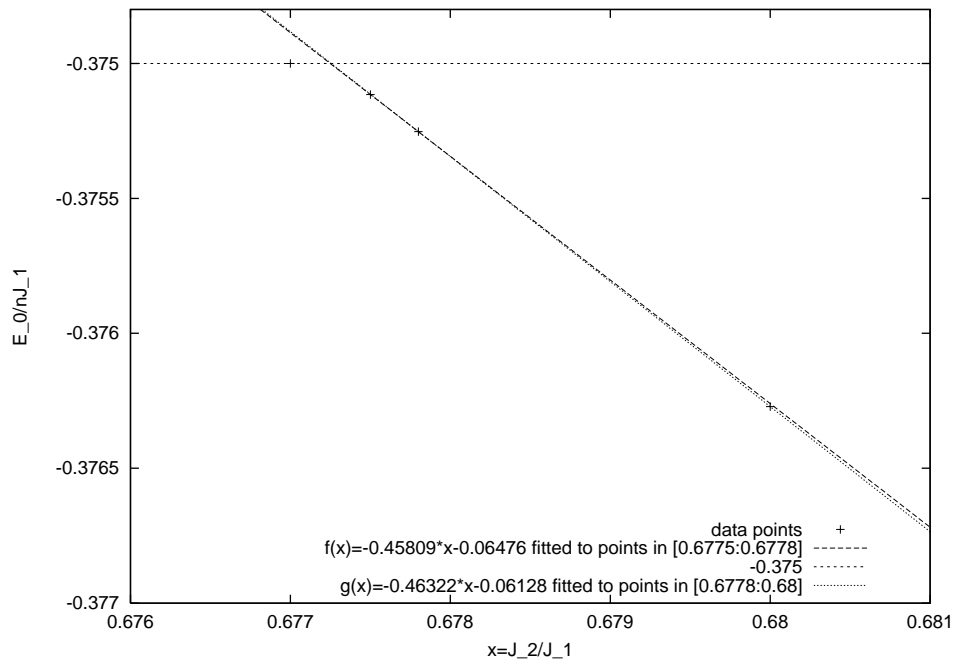
---

[1] http://www.fz-juelich.de/unicore/

Figure 4.4.: Ground state energies per dimer obtained for a square $N = 32$ Shastry–Sutherland model with periodic boundary conditions. The extrapolated value through the two data points is $x_c \approx 0.67753$.

J_2/J_1	$E_0/J_1 n$	wall clock time	accum. CPU time
0.30000000	-3.75000000000000e-01	42h	1340h
0.67700000	-3.74999999998987e-01	provided by U. Löw	
0.67750000	-3.75115180019318e-01	provided by U. Löw	
0.67780000	-3.75252606784488e-01	17h	545h
0.68000000	-3.76271688712272e-01	68h	2184h
0.70000000	-3.86720856118170e-01	32.5h	1040h

Table 4.2.: Data points calculated for the square $N = 32$ Shastry–Sutherland model. The dimension of the subspace containing the ground state is 75.164.451.

program needed only 17h wall clock time which is the lowest amount of CPU time used by any of the points of this specific system, even the data point at $x = 0.3$ which for reliability purposes was calculated already needed 42h wall clock time. Currently we don't have any explanation for this interesting behavior. A linear fit to both data pairs, $x_{1_a} = 0.6775, x_{2_a} = 0.6778$ and $x_{1_b} = 0.6778, x_{2_b} = 0.68$ leads to the rounded critical inverse frustration $x_c \approx 0.67725$. The data points were calculated on the FZ Jülich IBM supercomputer facilities. Two values were kindly provided by U. Löw. Accumulated for this lattice we used about 15% of the total available CPU time quota.

The largest system investigated is the Shastry–Sutherland model consisting of

$N = 6 \times 6 = 36$ spins placed on a square lattice as shown in fig. 4.2. This lattice is expected to be quite interesting similar to the square $N = 32$ system. The graph for the ground state energy behavior obtained so far is presented in fig. 4.5 and the individual data points are listed in table 4.3.
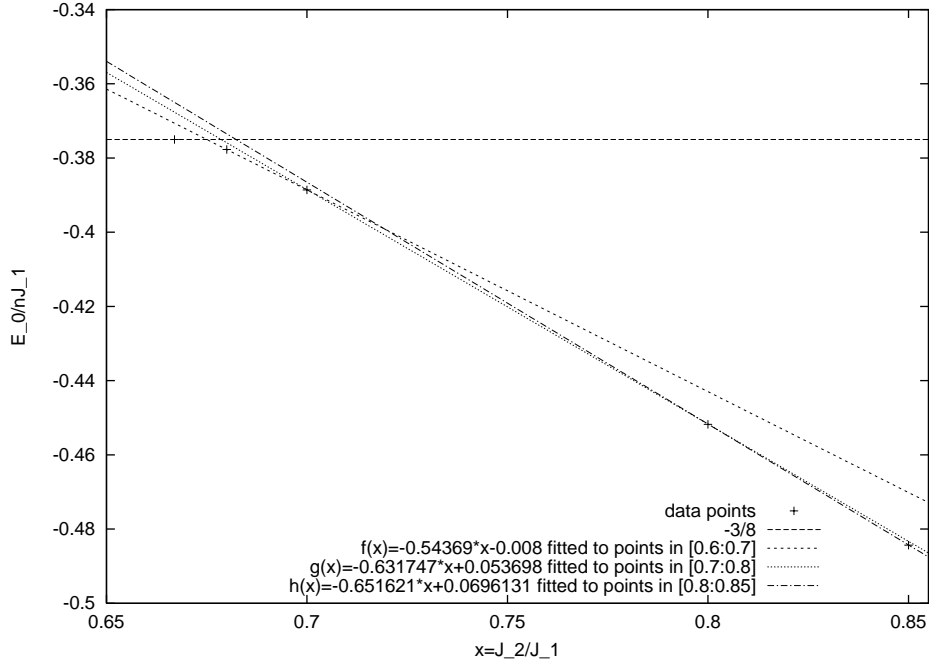

Figure 4.5.: Ground state energies per dimer obtained for a square $N = 6 \times 6 = 36$ Shastry–Sutherland model with periodic boundary conditions. The best estimate for the crossover is $x_c \approx 0.67501$ via a linear fit through the points at $x = 0.68$ and $x = 0.7$. The dimension of the subspace containing the ground state is 504.174.594.

Not applying any symmetries at all would lead to a size of the Hilbert space of $2^{36} = 6.87 \times 10^{10}$ states. Using conservation of the total S^z magnetisation would lead to a size of the corresponding subsector of $\binom{36}{18} = 9.075.135.300$ states. Spin inversion and a triple translational symmetry for each of both possible orientations of the square lattice lead to a reduction of this still very large Hilbert space by a factor of $1/18$ which in the end results in 504.174.594 states to be considered in the calculations performed which is about a factor of 15 larger than the sectors investigated so far [54] to our knowledge and of about the size of the 8×4 lattice studied above.

As a crosscheck for the program and the states file with the representative states of the subsector containing the ground state we first recalculated data that originally had been calculated already in 1994 by Schulz et al. [7]. They studied the frustrated Heisenberg antiferromagnet for which they also published the ground state energy per site that we could reproduce up to the last digit in the special case of vanishing frustration: $(E_0^{\text{Heisenberg}}(S = 1/2, N = 36)/n = 0.678872$ with n rep-

J_2/J_1	$E_0/J_1 n$	wall clock time	accum. CPU time
0.50000000	-0.374999999999425	36h	1157h
0.66697792	-0.374999999995975	171h ≈ 7d	5490h
0.68000000	-0.377704740424690	294h ≈ 12d	9400h
0.70000000	-0.388578449404367	216h ≈ 9d	6920h
0.80000000	-0.451753105004501	61h	1968h
0.85000000	-0.484334145942023	51h	1635h

Table 4.3.: Data points obtained for the $N = 6 \times 6 = 36$ Shastry–Sutherland model. The dimension of the subspace containing the ground state is 504.174.594.

resenting the number of sites). This fortunately (and to some extent surprisingly) took only quite a short period of about 25h wall clock time on 32 processors which gave us the hope that we would be able to calculate the ground state energies of the 6×6 Shastry–Sutherland model with its additional two–spin interactions also within a reasonable time frame. Of course Schulz et al. could apply much more symmetry properties to the Heisenberg model resulting in a subsector containing ‚only‘ 15.804.956 states. But keeping in mind that in 1994 1GB of physical main memory did cost in fact 100.000 DM ($\approx$ 51.000 Euro, the computer around the memory *not* included) this number of states surely is very impressive.

Coming to the individual data points as usual we first calculated one point ($x = 0.5$) in the dimer phase for which we exactly know the ground state energy and afterwards started quite conservatively with points at $x = 0.85$ and $x = 0.8$ leading to a rather inaccurate estimate of the critical inverse frustration of $x_c \approx 0.68221$. On the other hand these points needed only a moderate amount of CPU time so that this choice was suitable to get a first impression.

In order to calculate as few data points as possible to find a reasonable estimate for the phase transition away from the dimer phase we investigated a corresponding smaller periodic system (of size $N = 4 \times 4$) to find out whether it is possible to find a feasible function to fit at a few points far away from the transition point.

To verify whether we can find an anomaly in the behavior of the slope of the ground state energy per dimer depending on the inverse frustration x we calculated ca. 1000 data points in the range $[x_c : 0.72]$ for a square $N = 4 \times 4 = 16$ Shastry-Sutherland model as depicted in fig. 4.6. At first sight at the upper graph one would expect a simple linear behavior. But when choosing two data points, e.g. $x_1 = 0.68$ and $x_2 = 0.685$, for a linear fit one finds a different behavior as shown in the second and third graph.

Close to the transition point as well as far away from the transition point the linear fit doesn't give satisfactory results. We also tried fit–functions of second order $f(x) = a * x^2 + b * x * c$ and a power law $f(x) = a * x^b + c$ but without success. In the end we arbitrarily chose the frustration values for the next data point to calculate using the extrapolated x_c estimates resulting out of the linear fits as upper bounds.

Coming back to the 6×6 lattice this means that we should calculate another data point around $x \approx 0.68221$. To get a better linear fit next to the transition point we indeed decided to calculate points at $x = 0.68$ and $x = 0.7$. Using these

as a new basis for an estimate we currently find $x_c \approx 0.675$ as a best estimate for the critical value for the inverse frustration where the dimer phase is expected to vanish.

Concerning the technical aspects with the $N = 6 \times 6$ system we see that the ratio between the serial and parallel parts of the program becomes much better for several reasons leading to a much more efficient usage of the processors participating. The size of the subsector considered is more than a magnitude larger than that of the 8×4 or square $N = 32$ system described above leading alone to a significant increase of the parallel part of the program in which the matrix–vector multiplication takes place. Second, the higher number of symmetries taken advantage from also leads to a higher portion of the parallel sections because it becomes more expensive to find the representative state belonging to the specific subesector of the Hilbert space. And as a third point we should mention that also the (serial) initialisation phase of the program takes more time as the checkpointing and states files to be read in and the arrays to initialise of course are also a magnitude larger than with the $N = 32$ systems. On the other hand we gratefully were enabled to use job queues that allowed us to run jobs up to 24h wall clock time (instead of 4h as before) which in the end leads to a further improvement of the serial/parallel ratio.

Going from the smaller systems to the larger one we in the end find a ratio of serial to parallel parts of the program of less than 5% which allows an efficient usage of the supercomputer facilities of the FZ Jülich.

To summarise the results for the periodic systems we present a $1/\sqrt{N^3}$ plot in fig. 4.7 which seems plausible, e.g. following Sandviks arguments [55] for two–dimensional spin–1/2 systems based on chiral perturbation theory that are in agreement with results obtained from renormalisation–group calculations for the nonlinear σ model [56, 57].

Connecting the values for the square periodic systems of size $N = 16, 36$ with a straigth line we find a tendency $x_c(N \to \infty) \to 0.6786$ which seems quite reasonable following the theoretical and experimental arguments elaborated on in section 2.1. Taking also the square $N = 32$ system into account we find an estimate for the critical frustration $x_c(N \to \infty) \to 0.6808$. On the other hand the data points for the square systems $N = 32, 36$ are not located exactly on the line. The value for the largest system considered is lower than that of the square $N = 32$ system, so the estimate even becomes more inaccurate.

For the ‚stripe‘ type lattices of size $N = 16, 24, 32$ we find a lower value $x_c(N \to \infty) \to 0.6647$ of which we think that it is not well suited to find an estimate for the lattice of inifite size since this family of lattices tends to become quasi one–dimensional in the limit $N \to \infty$.

Although these estimates should not be taken too seriously they nevertheless all are reasonably lower than the the rigorous upper bound $x_{c_{upper}} \approx 0.6955$ for the spin–1/2 case found by Löw and Müller–Hartmann [12].
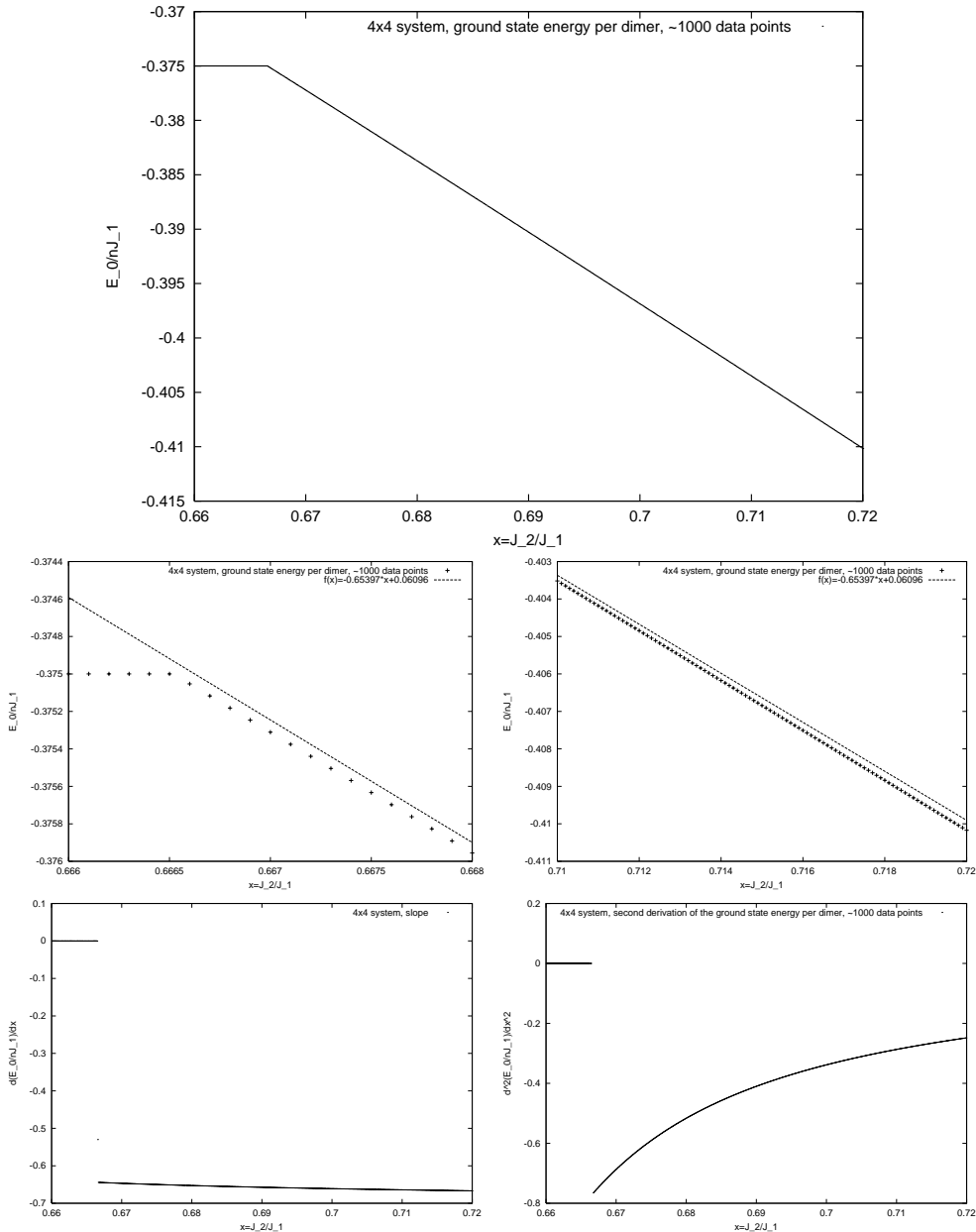
Figure 4.6.: First graph: Ground state energies per dimer obtained for a square $N = 4 \times 4$ Shastry–Sutherland model with periodic boundary conditions (ca. 1000 data points). Second graph: Zoom to the region close to the transition point. Third graph: Zoom to the region far from the transition point. Fourth graph: Slope of the ground state energy depending on $x = J_2/J_1$. Fifth graph: 2nd derivative of the ground state energy depending on the inverse frustration. No unusual behavior of the slope or second derivative of the ground state energy can be observed that might implicate a first order transition in the regime $x_c < x < 0.72$. From the slope and 2nd derivative one can immediately see that a linear fit cannot give good estimates for the critical frustration.
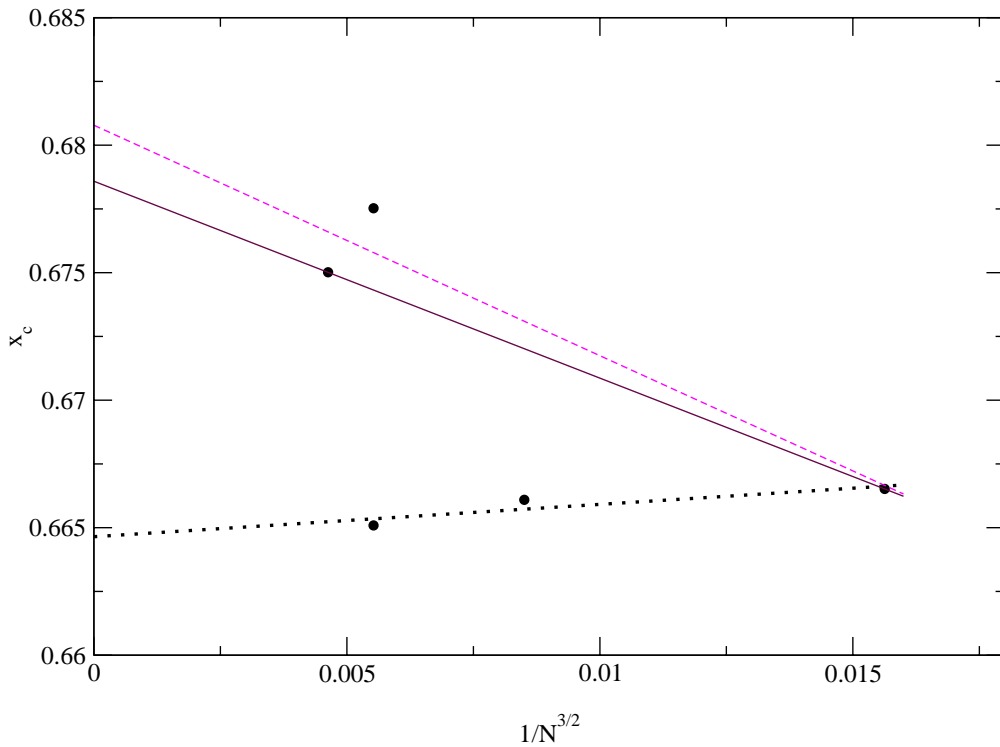
Figure 4.7.: Possible extrapolations of the critical inverse frustration of the infin-
ite lattice considering values of finite systems with periodic boundar-
ies and similar geometry. The solid line connects the systems with
the most pronounced two–dimensional character of size 16 and 36
(cf. fig. 4.2). The dashed line additionally takes the square system
of size 32 into account that has a slightly different orientation. The
dotted line connects the ,stripe' type systems $N = 16, 24, 32$. Of
course these lines represent only a tendency, they cannot be taken as
a serious extrapolation.

5. Numerical Results for the Plaquette Model

Similar to the Shastry–Sutherland model for the plaquette model we searched for lower bounds following P. W. Anderson's arguments [9] by calculating clusters of different system sizes with open boundary conditions. As we cannot apply translational symmetry to these lattices the calculations for such systems need more memory compared to the corresponding systems with periodic boundary conditions.

Most of the smaller systems have been calculated with the serial program version at the computing center of the RWTH Aachen on a Sun Fire 15K computer system rather similar to that of the University of Cologne technically already described in chapter 4.

We calculated the ground state energies for systems with open boundary conditions up to system size $N = 28$ as presented in figure 5.1.
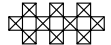
	Plaquette model	
N=4	⊠	0.5
N=12		0.52519
N=17		0.53458
N=20		0.53453
N=24		0.53405
N=28		0.53782

Figure 5.1.: Critical values x_c for the phase transition of the dimer phase to an intermediate phase for $J_2 > 0$ in lattices of the plaquette model with open boundary conditions.

Up to the size $N = 24$ we could calculate the lowest eigenvalues to arbitrary

precision applying conservation of the total S_z magnetisation and spin inversion (for systems with an even number of sites) so that these results indeed are exact to the fourth digit.

For the open $N = 28$ system we have calculated the data points shown in fig 5.2, also applying the same symmetries as for the $N = 24$ lattice. This leads to a size of the subsector of the Hilbert space containing the ground state of 40.116.600 states.
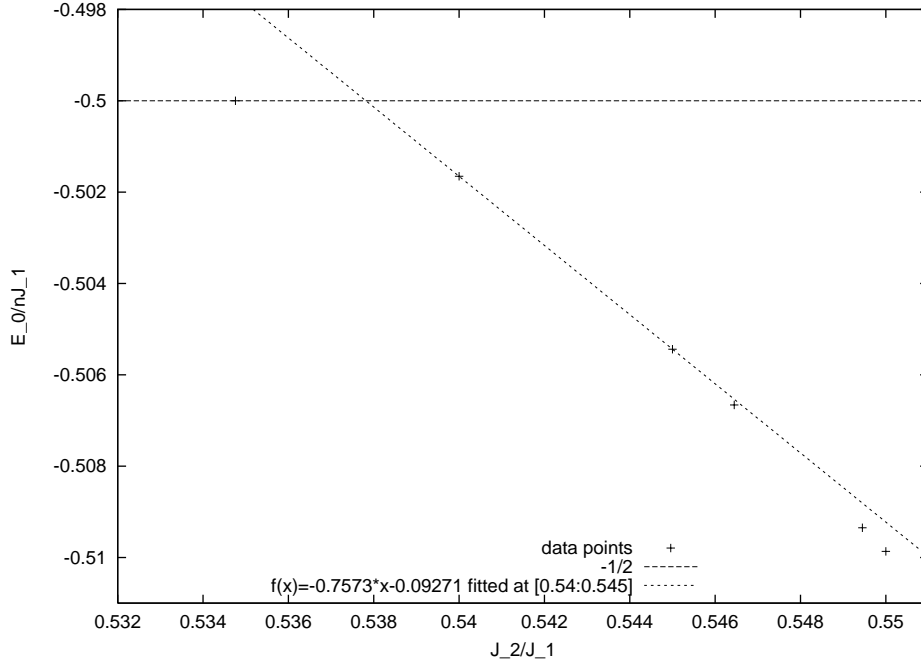

Figure 5.2.: Ground state energies per plaquette for an open $N = 28$ system with a geometry as shown in fig. 5.1. With a linear fit we extrapolate a critical value $x_c \approx 0.53782$ where the dimer phase vanishes.

Presenting all data collected for the lattices with open boundaries in a $1/\sqrt{N}$ plot with system size N we find the scenario as depicted in fig. 5.3.

The lines connect the results for several possible series of lattices with a related geometry. The values for $x(N \to \infty)$ where the lines meet the ordinate are $x(N = 17, 24) \approx 0.531$ which we think is an unusual behavior as the value for the $N = 24$ lattice also is unexpectedly low as already mentioned above. For this case the $N = 31$ lattice (cf. fig. 4.1) would be of interest as it is the continuation for this family of models. $x_c(N \to \infty) \to 0.563$ is a value for systems that for large system sizes develop a more or less one–dimensional structure. Nevertheless the values calculated for these systems are strict lower bounds for the critical frustration.

The third value obtained for the square systems of size 12 and 24 $x_c(N \to \infty) \to 0.555$ could be a reasonable estimate for the infinity lattice as it results from systems with the most pronounced two–dimensional character.
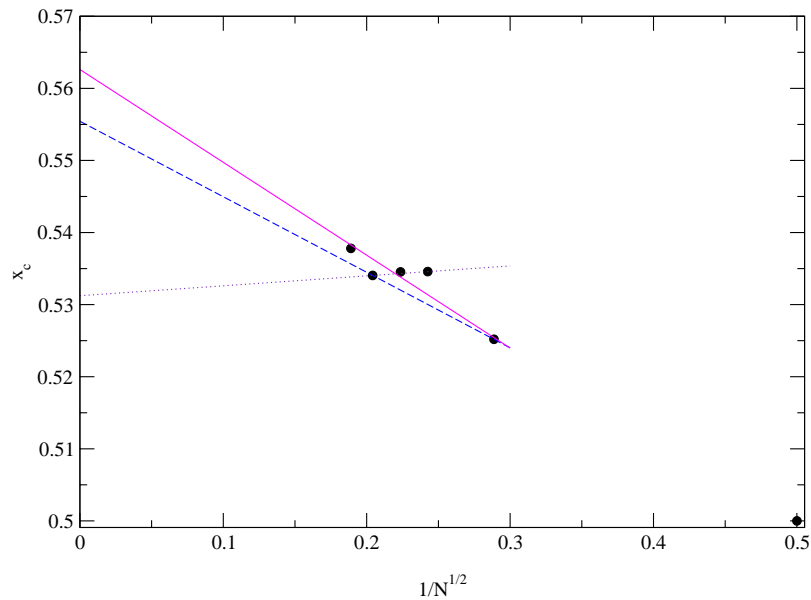
Figure 5.3.: Possible extrapolations of the critical inverse frustration of the infinite lattice considering values of finite systems with open boundaries and similar geometry. The solid line connects the systems of size 12, 20 and 28 (cf. fig. 5.1). The dotted line connect the systems of size 17 and 24. And the dashed line connects the systems of size 12 and 24. Of course these lines represent only rough tendencies, they cannot be taken as serious extrapolations.

We then calculated the ground state energies of the plaquette model for systems with periodic boundary conditions up to system size $N = 8 \times 4 = 32$. The systems with 24 and less sites are estimated exactly to arbitrary precision as at most 2.704.156 states needed to be taken into account.

The largest system studied for this model is a lattice with 8×4 sites. Using conservation of the total S^z magnetisation, spin inversion and translational invariance like in the case of the identically structured Shastry–Sutherland model we extrapolate the critical inverse frustration $x_c = 0.57237$.

For the square $N = 32$ lattice with periodic b. c. we currently find $x_c > 0.58$ as a first estimate.

Presenting all data collected for the lattices with periodic boundaries in a $1/\sqrt{N^3}$ plot we find the scenario as depicted in fig. 5.5.

Connecting the values of the systems $N = 16, 24, 32$ with a line we find a tendency $x_c(N \to \infty) \to 0.5795$. As these systems develop a quasi one–dimensional character towards larger system sizes we don't believe that this is a reasonable value for the two–dimensional lattice of infinite size. On the other hand we find the square $N = 32$ lattice for $x = 0.58$ (empty square symbol in fig. 5.5) still in the dimer phase. Taking this value as a lower bound for the critical inverse frustration and also taking the square lattice with system size 18 into consider-
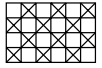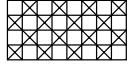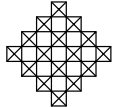
	Plaquette model	
N=8		0.5
N=16		0.56198
N=18		0.55555
N=24		0.57179
N=32		0.57237 extrap.
N=32		> 0.58

Figure 5.4.: Critical values x_c for the phase transition of the dimer phase to the intermediate phase for $J_2 > 0$ for the plaquette model on lattices with periodic boundary conditions.

ation we find a tendency $x_c(N \to \infty) \to 0.5978$ which in our opinion might be the most reasonable lowest value for the critical inverse frustration. Additionally taking the smallest system of this series with size 8 into account we find $x_c(N \to \infty) \to 0.5868$. But we think that this specific system is too small to be taken into consideration.

All these estimates are lying below the upper bound calculated by J. Zittartz [36]. Using a variational approach he finds the algebraic value $x_{c_{upper}} = \frac{31}{48} \approx 0.64583$ as a strict upper bound for the frustration where the dimer phase of the plaquette model vanishes.
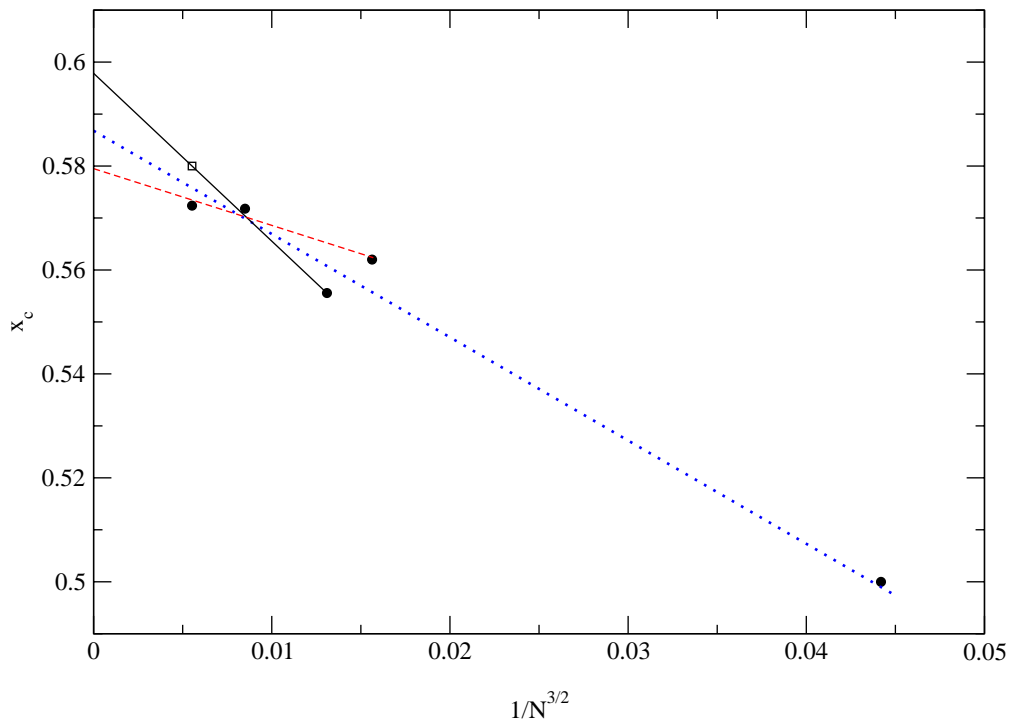
Figure 5.5.: Possible extrapolations of the critical inverse frustration of the infinite lattice considering values of finite systems with periodic boundaries and similar geometry. The estimate for the square $N = 32$ lattice is marked with an empty square. The dashed line connects the stripe systems of size $N = 16, 24, 32$ (cf. fig. 5.4). The dotted line could be a lower estimate for the systems with the most pronounced two–dimensional character of size $N = 8, 18, 32$. The solid line connects the square $N = 32$ system and the $N = 18$ system. Of course these lines represent only a rough tendency, they cannot be taken as a serious extrapolation.

6. Summary and Outlook

The present thesis deals with the ground state properties of two–dimensional antiferromagnetic quantum spin models.

On the one hand the Shastry–Sutherland model with nearest neighbor interactions and an additional frustrating two–spin interaction between next–nearest neighbors is considered (Shastry and Sutherland, 1981). It is of special interest as it is one of the few models featuring an exactly known ground state, the dimer singlet state. The model shows a rich zero temperature phase diagram both in the classical and in the quantum mechanical case.

In this work we focus on the quantum mechanical case where the phase diagram on the antiferromagnetic side shows a dimer phase with an adjacent intermediate phase of a nature which is still discussed. As a function of the frustration at a certain critical point that still is not determined exactly the phase transition between the dimer phase and the intermediate phase occurs.

The Shastry–Sutherland model describes the magnetic properties of the orthoborate substance $SrCu_2(BO_3)_2$ which has been synthesised in 1991 by Smith and Keszler.

$SrCu_2(BO_3)_2$ is a layered compund of $Cu(BO_3)$ planed separated by the Sr atoms. In 1999 Miyahara and Ueda mapped this structure on the two–dimensional Shastry–Sutherland model which allows us to present a detailed overview of experimental and theoretical results.

Another two–dimensional frustrated antiferromagnetic model is studied that has been suggested by J. Zittartz. It shows a number of similarities to the Shastry–Sutherland model. It has nearest neighbor interactions and next-nearest two–spin interactions, but it also features additional four–spin interactions why it is called plaquette model. Similar four–spin interactions are currently discussed in other magnetic systems like the spin–ladder with cyclic exchange or the parent compounds of high–T_c superconductors. The model investigated in this work is also constructed in such a way that it shows an exactly known ground state in the dimer phase and similar to the Shastry–Sutherland model a transition to an intermediate phase is expected where the transition point is not clearly determined either.

To study the ground state energy of the models we follow two approaches. On the one hand we concentrate on finite lattices with periodic boundary conditions and on the the other hand we use a variational ansatz proposed by P. W. Anderson by decomposing the Hamilton operator into clusters of finite size covering the lattice without overlapping bonds. Taking the ground state of the Hamilton operator as a variational state for the finite cluster (with open boundary conditions) it follows that the ground state of the finite cluster is always smaller than or equal to the ground state of the original Hamiltonian. Thus, we find the ground state energy of the finite cluster as a strict lower bound.

Since we are interested only in the lowest lying eigenvalues we can use the Lanczos method which we explain in detail. We apply different symmetry operators commuting with the Hamilton operators leading to subsectors smaller than the original Hilbert space. As we encounter still very large subsectors with up to 500 million representative states we use the numerical library ARPACK which provides a special parallel programming frontend for the Lanczos method based on the parallel programming standard MPI (message passing interface) which we give a short introduction to. Unfortunately we come to the conclusion that due to several technical reasons the MPI standard doesn't fit our needs: The data distribution implemented by the ARPACK library is such that with our models with next–nearest neighbor spin interactions a high amount of communication over all processors is created. A second reason is that MPI is designed for a distributed architecture which means that certain arrays with e.g. read–only information, like the storage/lookup table for the representative states belonging to the subsector of the Hamilton operator studied, have to be kept in memory separately for each processor. As the size of the subspace is the limiting factor of the calculations concerning the memory usage of course this is a major drawback for our investigations.

An alternative resolving these problems is a second parallel programming technique called OpenMP which (still in the process of being developed) is implemented in the compiler, in contrast to the MPI library which is compiler independent and where the user has to program the data management and communication himself.

By so called OpenMP *compiler directives* the user adds to his serial program, certain regions in the program are marked for the computer system to work on in parallel. Also the user can mark global variables in such a way that the compiler keeps a local copy for each processor allocated. This is necessary as in principle all processors share the same memory which immediately solves our problem with the array keeping the storage/lookup table of the representative states mentioned above because all processors can access this single copy in contrast to the MPI version.

Combining the well known methods like application of symmetry operators, Lanczos method, Lin–algorithm and hashing technique to create the storage/lookup table mentioned above with the parallel programming technique OpenMP we calculate ground state energies for both, lattices with open boundary conditions and up to 28 sites with 40.116.600 representative states in the subsector as well as for the lattices with periodic boundary conditions and up to 36 sites with 504.174.594 representative states.

For the Shastry–Sutherland model Löw and Müller–Hartmann find a best strict lower bound $x_c = 0.5914$ for the inverse frustration where the phase transition between the dimer phase and the intermediate phase might occur at lowest. For the periodic systems with 32-36 sites we have calculated individual data points from which we have extrapolated the critical values for the inverse frustration x_c. Smaller systems have been calculated exactly up to the fourth digit. We find that the results strongly depend on the individual geometry of the lattices considered so we collect corresponding lattices to perform a $1/\sqrt{N^3}$ extrapolation, e.g. following Sandviks arguments based on a chiral perturbation theory that are in agreement

with results obtained from renormalisation–group calculations for the nonlinear σ model to possibly find an estimate of the critical value of the inverse frustration for the infinite system.

Unfortunately the values do not follow a simple law so we cannot perform such an estimate. Nevertheless we can state that the results of the periodic systems show an increasing tendency in general. For the square lattice with 32 sites we find an extrapolated value $x_c = 0.6775$ and for the 6×6 lattice an also extrapolated value $x_c = 0.6750$.

We discuss the literature concerning the nature of the intermediate phase. From the theoretical point of view many different estimates for the critical frustration where the dimer phase vanishes were presented using various methods. With some of these estimates our results are in agreement. For instance, using exact diagonalisation and fourth order perturbation theory Miyahara and Ueda find a direct dimer to Néel transition at $x \approx 0.7$. Weihong, Hamer and Oitmaa find $x = 0.691$ as an upper bound of the dimer phase investigating the behavior of the gap above the singlet dimer ground state to name a few. On the other hand a number of investigations show a different scenario. Using the perturbative continuous unitary transformation method Knetter et al. find $x = 0.63$ as a value of the breakdown of the dimer phase. This value is in clear contradiction with our results. Also Albrecht and Mila find a first order transition of the dimer phase to an intermediate phase at $x \approx 0.606$ as an uncontrolled estimate which is much smaller than the results presented in this thesis.

From the experimental point of view $SrCu_2(BO_3)_2$ has been investigated by a number of different methods. The existence of the singlet ground state has been confirmed by Kageyama et al. in 1999 measuring the magnetic response and the magnetic susceptibility. They conclude that $SrCu_2(BO_3)_2$ is a realisation of the Shastry–Sutherland model. Experiments with inelastic neutron scattering, electron resonance spectroscopy, far infrared spectroscopy, nuclear magnetic resonance or Raman experiments all confirm the existance of a gap $\Delta \approx 34K$. The range of given values x for the inverse frustration is quite close to the value $x_c \approx 0.69$ but a direct observation of real substance at or close to the quantum critical point has not been accomplished so far.

Turning to the plaquette model proposed by J. Zittartz we present the results for lattices with open boundary conditions up to system size $N = 28$ equivalent to $40.116.600$ representative states in the subsector of the Hilbert space containing the ground state. As a best strict lower bound for the inverse frustration where the plaquette model undergoes a phase transition from the dimer phase to the adjacent phase we find an extrapolated value $x_c = 0.5378$. We observe an untypical behavior of the square $N = 24$ system which yields $x_c = 0.5340$ which is a lower value than we find already at smaller system sizes. In all other cases we find that at least within the same family of lattices of a similar geometry the critical values for the inverse frustration are increasing. For the lattice $N = 24$ we find a lowering when comparing to the $N = 17$ case where we find a value of $x_c = 0.5345$ already. Correspondingly to the Shastry–Sutherland model we show a $1/\sqrt{N}$ plot for the systems with open boundaries. Three series of lattices can be identified of which two in the limit $N \to \infty$ become quasi one–dimensional. The series with lattices of

a square character consisting of the lattices of size $N = 12, 24$ gives an estimate $x_c(N \to \infty) \to 0.563$.

The lattices with periodic boundary conditions have been calculated up to systems of size $N = 32$. The $N = 8 \times 4$ system considered yields an extrapolated value for the inverse frustration $x_c = 0.5723$. For the square version of the $N = 32$ system we find a first estimate $x_c > 0.58$.

For the plaquette lattices with periodic bounday conditions the $1/\sqrt{N^3}$ plot presents two possible series of related lattices. One family that for $N \to \infty$ becomes quasi one–dimensional and the other one consisting of lattices with a square character. The latter series gives a rough estimate $x_c(N \to \infty) \to 0.5868$ for the critical inverse frustation where the dimer phase might vanish. Neglecting the smallest lattice consisting of 8 sites only, we find a most reasonable value of $x_c(N \to \infty) \to 0.5978$.

All estimates are lying below the algebraic upper bound $x_{c_{\text{upper}}} = \frac{31}{48} \approx 0.646$ calculated by Zittartz considering variational arguments.

In general we observe that for the plaquette model the lower bounds predicted by analysing square lattices with open boundary conditions are not as close to the direct approximations via systems with periodic boundary conditions ($x_{c_{\text{open}}} \to 0.555$, $x_{c_{\text{periodic}}} \to 0.5978$) than the corresponding values for the Shastry–Sutherland model ($x_{c_{\text{open}}} \to 0.65$ estimated by Löw and Müller–Hartmann, $x_{c_{\text{periodic}}} \to 0.6786$).

6.1. Outlook

From a technical point of view calculations for quantum spin models like the two frustrated two–dimensional models considered in this thesis are a very challenging task, both in terms of memory and CPU time usage. To our experience enlarging a lattice by only four spins leads to an increase of CPU time and memory usage by a whole order of magnitude. For that reason the method of exact diagonalisation is directly coupled to the technical development of modern computer systems and parallelisation techniques. Although we were able to calculate the ground states of systems containing up to 500 million states in the corresponding subsector of the Hilbert space we did not calculate the first excitations due to the high amount of CPU time used already for the ground states. In fact we overall have used about 40000h CPU time on the IBM supercomputer at the Forschungszentrum Jülich compared to which the CPU hours consumed on the SunFire 15K systems of the Universities in Aachen and Köln can be neglected.

Also calculating other observables like the staggered magnetisation was not possible because for such quantities all Lanczos vectors need to be kept in memory which is impossible as a single Lanczos vector for the $N = 36$ system already allocates about 4GB main memory. Also saving this data on disk as a slow workaround in this case is not feasible.

For that reason we think that calculating at least the ground state energies already is a good achievment.

We hope that our expertise will find further application in the future.

A. Sample Implementations in C

A.1. Hashing Technique

```
void hash_create(void) {
  long i,rest;
  int max=0;

  /* This subroutine is called once during initialisiation phase of the program */

  /* globally defined two-dimensional array 'hash_table' of size 'PRIME' */
  hash_table = (int **) calloc((size_t) PRIME, (size_t) sizeof(int *));

  /* number of collisions for each possible remainder */
  hash_length = (int *) calloc((size_t) PRIME, (size_t) sizeof(int));
  if (NULL == hash_length) exit(13);

  /* first: find out the number of collisions corresponding to a given remainder */
  /* 'n' is the number of states 'sz0_states' in the subspace investigated */
  for (i=0;i<n;i++) {
    hash_length[sz0_states[i] %  PRIME]++;
    /* 'max' is the maximum number of collisions */
    /* 'max' should be as low as possible, preferably 'max=1', but
        for this one needs to choose a large prime number 'PRIME'   */
    if (hash_length[sz0_states[i] % PRIME] > max) max=hash_length[sz0_states[i] % PRIME];
  }

  /* second: allocate the needed memory individually for each remainder */
  for (i=0L;i<PRIME;i++) {
#ifdef _AIX
    if (0 != hash_length[i])
#endif /* _AIX */
    /* hash_table is not an equally distributed array:
        hash_table[0]: ---
        hash_table[1]: -
        ...
        hash_table[PRIME]: --
    */
    hash_table[i]=(int *) calloc((size_t) hash_length[i], (size_t) sizeof(int));
#ifndef _AIX
    if (NULL == hash_table[i]) {
      exit(3);
    }
#endif /* _AIX */
    hash_length[i]=0;
  }

  /* third: finally fill in the representatives corresponding to their remainder */
  for (i=0;i<n;i++) {
    rest = sz0_states[i] % PRIME;
    hash_table[rest][hash_length[rest]] = i;
    hash_length[rest]++;
  }

  /* clean up unneeded memory for later usage */
```

```
  free(hash_length);
}


/* This subroutine returns the place of a representative state in the
   original Hilbert space */
int return_hash(long val) {
  long k;
  int i=0;
  k=val%PRIME;
  while ( sz0_states[hash_table[k][i]] != val ) i++;
  return hash_table[k][i];
}
```

A.2. Lin-Algorithm

```
#define GETBIT(VEKTOR,POSITION) (((VEKTOR) >> (POSITION)) & 1L)
/* This subroutine is called once during the initialisation phase of the program */
void lin_create(void) {
  /* a: left half; b: right half; */
  int a,b,ma,mb,counta,countb,bitsupa,bitsupb,i,j,k,l;
  long count;
#ifdef INVERSION
  invmask=(1L<<(N/2))-1;
#endif
  count=0L;
  countb=0;
  for (mb=0;mb<=N/2;mb++) {
    ma=N/2-mb+odd;
    a=(1<<ma)-1; /* first number with magnet. ma */
    b=(1<<mb)-1; /* first number with magnet. mb */
    for (i=b;i<1<<(N/2);i++) {
      bitsupb=0;
      for (j=0;j<N/2;j++)
        if (GETBIT(i,j) == 1) bitsupb++;
      if (bitsupb == mb) {
        countb=(int) count;
        Ib[i]=countb;
        counta=0;
        for (k=a;k<1<<(N/2+odd);k++) {
          bitsupa=0;
          for (l=0;l<N/2+odd;l++)
            if (GETBIT(k,l) == 1) bitsupa++;
          if (bitsupa == ma) {
#ifdef INVERSION
            if ( (invmask^k) > k ) {  /* xor */
#endif
              Ia[k] = counta;
              /* count=counta+countb, bzw. count=Ia[k]+Ib[i] */
              sz0_states[count]=(((long) k)<<(N/2))+(long) i;
              counta++;
              count++;
#ifdef INVERSION
            } /* if invmask^k */
#endif
          } /* if bitsupa */
        } /* for k=a */
      } /* if bitsupb */
    } /* for i=b */
  } /* for mb=0 */
} /* lin_create() */
```

```
/* This subroutine is called to give back the position of the representative
   in the original Hilbert space */
int lin(long val) {
  return Ia[(int) (val>>(N/2))]+Ib[(int) (val&((1<<(N/2))-1))];
}
```

A.3. Calling Fortran Subroutines from a C/C++ Program

In this section we want to demonstrate how to simply call a typical standard numerical eigenvalue/eigenvector subroutine, e.g. of the BLAS/LAPACK[1] libraries, since we have recognised that many people at the beginning of their studies and calculations have experienced problems implementing code (usually in the programming language C or C++) to solve this task. A common difficulty is to call a subroutine which usually is programmed in Fortran from the user's C or C++ program leading to various more or less dubious types of programming solutions. The reason is that Fortran and C use different methods how subroutine calls are handled.

The solution is that the user programming in C or C++ has to pass *all* arguments by reference independent of their type, not only arrays but also scalars like integer or floating point variables. Even scalar input–only variables need to be passed by reference as illustrated in the following C code fragment which returns the largest element of the two arrays ‚a‘, and ‚b‘ containing numbers of type `float`[2]:

```
/* 1. Declare scratch variable 'one' to allow the constant 1 to be
   passed by value */
int one=1, n=10, large_index;
float *a, *b, largest;

/* 2. Append underscore to conform to FORTRAN naming system */
/* 3. Pass all arguments, even scalar input-only, by reference
      to the BLAS subroutine 'isamax' */
/* 4. Subtract one to convert from FORTRAN array indexing conventions */
large_index = isamax_ (&n, a, &one) - 1;
largest = a[large_index];
large_index = isamax_ (&n, b, &one) - 1;
if (b[large_index] > largest)
     largest = b[large_index];
```

---

[1] http://www.netlib.org/

[2] Adapted from an example in *Sun Studio 9: Sun Performance Library User's Guide*, available from the internet address http://docs.sun.com/.

Bibliography

[1] W. Heisenberg, Z. Phys. **49**, 613 (1928).

[2] H. Bethe, Z. Phys. **71**, 205 (1931).

[3] L. Hulthèn, Arkiv, Mat. Astron. Fys. **26A**, No. 11 (1938).

[4] J. Bonner and M. Fisher, Phys. Rev. **135**, A640 (1964).

[5] J. Oitmaa and D. D. Betts, Can. J. Phys. **56**, 897 (1978).

[6] A. Ralston, *A first course in numerical analysis*, McGraw-Hill, New York, NY, 1965.

[7] H. J. Schulz, T. A. L. Ziman, and D. Poilblanc, J. Phys. I **6**, 675 (1996).

[8] S. Miyahara and K. Ueda, Phys. Rev. Lett. **82**, 3701 (1999).

[9] P. W. Anderson, Phys. Rev. **83**, 1260 (1951).

[10] B. S. Shastry and B. Sutherland, Physica B **108**, 1069 (1981).

[11] C. K. Majumdar, J. Phys. C **3**, 911 (1969).

[12] U. Löw and E. Müller-Hartmann, J. Low Temp. Phys. **126**, 1135 (2002), *ibid.* **127**, 290 (2002).

[13] M. Albrecht and F. Mila, Europhys. Lett. **34**, 145 (1996).

[14] C. H. Chung, J. B. Marston, and S. Sachdev, Phys. Rev. B **64**, 134407 (2001).

[15] D. Carpentier and L. Balents, Phys. Rev. B **65**, 024427 (2002).

[16] A. Läuchli, S. Wessel, and M. Sigrist, Phys. Rev. B **66**, 014401 (2002).

[17] T. Munehisa and Y. Munehisa, J. Phys. Soc. Jpn. **72**, 160 (2003).

[18] A. Koga and N. Kawakami, Phys. Rev. Lett. **84**, 4461 (2000).

[19] Z. Weihong, C. Hamer, and J. Oitmaa, Phys. Rev. B **60**, 6608 (1999).

[20] Z. Weihong, J. Oitmaa, and C. Hamer, Phys. Rev. B **65**, 014408 (2002).

[21] C. Knetter, *Perturbative Continuous Unitary Transformations: Spectral Properties of Low Dimensional Spin Systems*, PhD thesis, Universität zu Köln, 2003.

[22] C. Knetter, A. Bühler, E. Müller-Hartmann, and G. S. Uhrig, Phys. Rev. Lett. **85**, 3958 (2000).

[23] R. W. Smith and D. A. Keszler, J. Solid State Chem. **93**, 430 (1991).

[24] A. Bühler, *High Temperature Series Expansions for Spin- and Spin-Phonon-Systems*, PhD thesis, Universität zu Köln, 2003.

[25] K. Sparta et al., Eur. Phys. J. B **19**, 507 (2001).

[26] S. Miyahara and K. Ueda, Supplement B to J. Phys. Soc. Jpn. **69**, 72 (2000).

[27] H. Kageyama et al., Phys. Rev. Lett. **82**, 3168 (1999).

[28] H. Kageyama et al., Phys. Rev. Lett. **84**, 5876 (2000).

[29] H. Nojiri, H. Kageyama, K. Onizuka, Y. Ueda, and M. Motokawa, J. Phys. Soc. Jpn. **68**, 2906 (1999).

[30] T. Rõõm et al., Phys. Rev. B **61**, 14342 (2000).

[31] K. Kodama et al., J. Phys.: Condens. Matter **14**, L319 (2002).

[32] P. Lemmens et al., Phys. Rev. Lett. **85**, 2605 (2000).

[33] H. Nojiri et al., Physica B **294-295**, 14 (2001).

[34] C. Knetter, E. Müller-Hartmann, and G. S. Uhrig, J. Phys.: Condens. Matter **12**, 9069 (2000).

[35] S. Miyahara and K. Ueda, J. Phys.: Condens. Matter **15**, R327 (2003).

[36] J. Zittartz, private communication.

[37] E. Dagotto and T. M. Rice, Science **271**, 618 (1996).

[38] A. Reischl and E. Müller-Hartmann, Eur. Phys. J. B **28**, 173 (2002).

[39] S. Brehmer, H.-J. Mikeska, M. Müller, N. Nagaosa, and S. Uchida, Phys. Rev. B **60**, 329 (1999).

[40] T. Senthil and M. P. A. Fisher, Phys. Rev. Lett. **86**, 292 (2001).

[41] L. Balents, M. P. A. Fisher, and S. M. Girvin, Phys. Rev. B **65**, 224412 (2002).

[42] G. Misguich, B. Bernu, C. Lhuillier, and C. Waldtmann, Phys. Rev. Lett. **81**, 1098 (1998).

[43] K. Voelker and S. Chakravarty, Phys. Rev. B **64**, 235125 (2001).

[44] C. Lanczos, C. Res. Natl. Bur. Stand. **45**, 255 (1950).

[45] H. Q. Lin, Phys. Rev. B **42**, 6561 (1990).

[46] J. M. Thijssen, *Computational Physics*, Cambridge University Press, Cambridge, 1999.

[47] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*, SIAM, Philadelphia, 1998.

[48] I. M. Barbour, N.-E. Behilil, P. E. Gibbs, G. Schierholz, and M. Teper, *The Lanczos Method in Lattice Gauge Theories, in: The Recursion Method and Its Applications, edited by D. G. Pettifor and D. L. Weaire*, Springer Series in Solid-State Sciences Vol. **58**, Springer, Berlin, 1985.

[49] D. C. Sorensen, *Implicitly Restarted Arnoldi/Lanczos Methods for Large Scale Eigenvalue Calculations*, NASA CR-198342 ICASE Report No. 96-40, 1996, http://www.icase.edu/library/reports/rdp/96/96-40RDP.tex.refer.html.

[50] D. Knuth, *The Art of Computer Programming*, Addison-Wesley, Reading, 1973, Vol. 3, Sec. 6.4.

[51] Y. Aoyama and J. Nakano, *RS/6000 SP: Practical MPI Programming*, IBM redbook, available from http://www.redbooks.ibm.com/, 1999.

[52] C. Schmitz and P. Brühne, *Slides of the RRZK course ‚Paralleles Programmieren auf den Compute Servern des ZAIK/RRZK'*, available from the RRZK webpages http://www.uni-koeln.de/rrzk/kurse/unterlagen/parallel/, 2003.

[53] H. Neff, *Low fermionic eigenmode dominance in QCD on the lattice*, PhD thesis, Bergische Universität–Gesamthochschule Wuppertal, 2001.

[54] C. J. Hamer, T. Hövelborn, and M. Bachhuber, J. Phys. A **32**, 51 (1999).

[55] A. W. Sandvik, Phys. Rev. B **56**, 11678 (1997).

[56] H. Neuberger and T. Ziman, Phys. Rev. B **39**, 2608 (1989).

[57] D. S. Fisher, Phys. Rev. B **39**, 11783 (1989).

English Abstract

In this thesis ground state properties of two–dimensional antiferromagnetic quantum–spin systems are investigated. Especially, we study the spin–$1/2$ Shastry–Sutherland model consisting of a Heisenberg model with additional two–spin interactions, and a spin–$1/2$ plaquette model proposed by J. Zittartz with two–spin interactions and additional four–spin interactions. These models are of special interest as they feature an exactly known ground state in the dimer phase. The Shastry–Sutherland model has a realisation in the compound $SrCu_2(BO_3)_2$. The phase boundary of the dimer phase could not yet been determined exactly although various theoretical approaches have been used. By means of a variatonal ansatz suggested by P. W. Anderson in connection with exact diagonalisation via the Lanczos method and additional application of parallel programming techniques (OpenMP) we calculate ground state energies of lattices with open boundary conditions up to 31 sites that give a strict lower bound of the ground state energy of the infinite system.

Also lattices consisting of up to 36 sites with periodic boundary conditions are considered whose ground state energies directly approximate that of the infinite system.

During the calculations subsectors of the Hilbert space up to a size of 500 million states have been examined which to our knowledge extend the subspaces investigated so far by more than an order of magnitude.

We find that the ground state energies strongly depend on the specific geometry of the individual lattice studied. We try to extrapolate the ground state energies of systems with similar geometry to an infinite lattice. Unfortunately the lattices considered still seem to be too small to give a clear determination of the critical inverse frustration where the dimer phase of the two models vanishes.

Deutsche Kurzzusammenfassung

In der vorliegenden Arbeit werden Grundzustandseigenschaften zweidimensionaler antiferromagnetischer Quantenspinsysteme untersucht. Wir betrachten insbesondere das Spin–$1/2$ Shastry–Sutherland Modell, das aus einem Heisenbergmodell mit zusätzlichen Zweispinwechselwirkungen besteht, und ein von J. Zittartz vorgeschlagenes Spin–$1/2$ Plakettenmodell, ebenfalls mit Zweispinwechselwirkungen und mit zusätzlichen Vierspinwechselwirkungen auf Plaketten.

Diese Modelle sind von besonderem Interesse, da sie sich durch einen exakt bekannten Grundzustand in der dimerisierten Phase auszeichnen. Das Shastry–Sutherland Modell ist in der Substanz $SrCu_2(BO_3)_2$ realisiert. Die Phasengrenze der dimerisierten Phase konnte trotz Anwendung verschiedener theoretischer Ansätze bis heute nicht exakt bestimmt werden. Wir können mit Hilfe eines von P. W. Anderson vorgeschlagenen Variationsansatzes in Verbindung mit exakter Diagonalisierung durch das Lanczos Verfahren und zusätzlicher Anwendung von Parallelisierungstechniken (OpenMP) Grundzustände offener Gitter bis 31 Gitterplätze berechnen, die eine strikte untere Schranke des Grundzustands des unendlich großen Systems bilden.

Desweiteren werden Gitter mit bis zu 36 Gitterplätzen mit periodischen Randbedingungen betrachtet, deren Grundzustände den des unendlich großen Systems direkt annähern.

Dabei werden Untersektoren des Hilbertraums mit bis zu 500 Millionen Zuständen betrachtet, was einer Erweiterung der uns bekannten bisher betrachteten Räume um mehr als eine Größenordnung darstellt.

Die Gundzustandsenergien hängen stark von der Geometrie der jeweiligen betrachteten Gitter ab. Es wird versucht, die Grundzustandsenergien von Systemen mit ähnlicher Geometrie mit Hilfe linearer Fits auf ein unendlich großes System zu extrapolieren. Leider scheinen die untersuchten Gitter noch zu klein zu sein, um eine eindeutige Aussage über die genaue kritische inverse Frustration machen zu können, bei der die dimerisierte Phase der beiden Modelle jeweils verschwindet.

Danksagung

- Mein besonderer Dank geht an Frau Priv.–Doz. Ute Löw für die Ausgabe und Betreuung dieser Arbeit. Sie stand jederzeit für Diskussionen und Hilfestellungen zur Verfügung und trug durch ihre Unterstützung und Geduld wesentlich zum Gelingen dieser Arbeit bei.

- Ein Dank auch an Prof. Dr. E. Müller–Hartmann und Prof. Dr. J. Zittartz für ihre stete Bereitschaft zur Diskussion.

- Desweiteren danke ich der Arbeitsgruppe von Herrn Prof. E. Müller–Hartmann für die freundliche und konstruktive Atmosphäre in den Mitarbeiterseminaren.

- Dem Institutsvorstand danke ich für die weitgehende Freistellung von Verpflichtungen, die die Durchführung dieser Arbeit ermöglicht hat.

- Einen herzlichen Dank auch an die Mitarbeiter des John–von–Neumann Instituts und des Zentralinstituts für angewandte Mathematik am Forschungszentrum Jülich, darunter zuvorderst Herrn Dr. G. Arnold und Herrn Dr. N. Attig für die freundliche und kompetente Unterstützung bei der Benutzung des IBM Supercomputers. Eine Reihe von technischen Problemen konnte so gut und sehr schnell gelöst werden. Ein Dank auch an Herrn A. Spiegel vom Rechen– und Kommunikationszentrum der RWTH Aachen für seinen kompetenten Rat in Sachen OpenMP.

- Außerdem möchte ich vielen Kollegen und Freunden für die angenehmen Stunden innerhalb und außerhalb des Instituts danken, die mich durch die vielen Jahre im Institut begleitet haben, darunter Priv.–Doz. Dr. R. Klesse, Dr. W. Kromen, Dr. P. Wurth, Dr. S. Dahm, Dr. B. Strocka, Dr. J. Kierfeld, Dr. R. M. Vetter und vielen anderen.

- Ein besonderer Dank auch an Axel Weinkauf für die freundschaftliche Atmosphäre bei der Arbeit im Institut und auch außerhalb.

- Meiner Familie danke ich für ihren Beistand und ihr Interesse.

- Nicht zuletzt danke ich Johannes Klüser für seine Eselsgeduld.

Erklärung

Ich versichere, dass ich die von mir vorgelegte Dissertation selbständig angefertigt, die benutzten Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit − einschließlich Tabellen, Karten und Abbildungen −, die anderen Werken im Wortlaut oder Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie − abgesehen von unten angegebenen Teilpublikationen − noch nicht veröffentlicht worden ist, dass ich eine solche Veröffentlichung vor Abschluss des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen dieser Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Frau Priv.−Doz. Dr. Ute Löw betreut worden.

Köln, den 13. Dezember 2004

Teilpublikationen

Es wurden keine Teilpublikationen veröffentlicht.

Lebenslauf

Persönliche Daten

Name:	Andreas Sindermann
Geburtsdatum:	20. Mai 1969
Geburtsort:	Köln
Familienstand:	ledig
Staatsangehörigkeit:	deutsch

Schulbildung

1975–1979	Ernst-Moritz-Arndt-Grundschule, Köln
1979–1988	Gymnasium Rodenkirchen, Köln

Grundwehrdienst

1988–1989	Grundwehrdienst

Hochschulstudium

1989–1997	Studium der Physik an der Universität zu Köln
	Diplomarbeit am Institut für Theoretische Physik über *Anwendung von Maximum-Flow Algorithmen auf Grundzustandsprobleme klassischer ungeordneter Ising-Modelle*
seit 1997	Promotion am Institut für Theoretische Physik an der Universität zu Köln

Wissenschaftliche Anstellungen

1992–1997	Studentische Hilfskraft und Studentische Aushilfskraft am Institut für Theoretische Physik und am Regionalen Rechenzentrum der Universität zu Köln
seit 1997	Wissenschaftlicher Mitarbeiter am Institut für Theoretische Physik der Universität zu Köln