Parallel Calculation of Properties of Magnetic Impurities in Metals

Eloiza Sonoda[1] Gonzalo Travieso[1]
[1] Departamento de Física e Informática - IFSC, Universidade de São Paulo
Av. do Trabalhador Sãocarlense 400, São Carlos - SP, Brasil
{elo,gonzalo}@if.sc.usp.br

Abstract—
The computation of physical properties of dilute magnetic alloys is a computationally intensive task. In principle, it consists on the diagonalization of a huge matrix. Through the use of a suitable base and of the renormalization group method, the diagonalization can be simplified to the iterative diagonalization of a set of smaller matrices (of the order of some hundreds of lines and columns). The resulting problem is nevertheless still too large if enough precision is desired. We describe here the parallel implementation of an algorithm for this problem in a cluster. The results show that the implementation achieves good reduction of execution times, but lacks speedup and scalability due to load balancing problems. We analyze these problems and suggest paths to their resolution.

Keywords—**parallel processing, clusters, message passing, Anderson model, renormalization group.**

I. Introduction

Anderson [And61] has proposed a model for dilute magnetic alloys with one impurity that was later extended to two impurities [AA64].

The numeric renormalization group method [Wil75], [KMWW80] is often used for computations with this model, due to many advantages discussed elsewhere [Oli92]. The method is numerically efficient, but for the two impurities problem intensive in both computing time and storage space.

The process consists of the iterative diagonalization of a real, symmetric matrix H, the quantum Hamiltonian of the model. In a typical case, the dimension of the matrix to be diagonalized in each iteration is of order 10^4. Using a suitable basis, the matrix is block diagonal and the process can then be simplified to the diagonalization of about hundred smaller matrices (in each iteration), the larger with dimension of the order of eight hundred.

To compute some physical property, many program runs are needed, to tune some parameters and to variate others in order to reduce typical oscillations of the method.

This short description shows clearly that a high performance computer system is needed to tackle this problem. With the constant reduction of feature size of the chips the use of parallel architectures in various levels of implementation has become common practice [CSG99]. In special, the high performance of personal computers and commercial network hardware has made the use of networks of workstations for high performance computing attractive [ACP96].

This work describes the implementation of a parallel version of a renormalization group program in a Beowulf-like [Ste95] cluster.

A. Anderson Model

Initially proposed [And61] for the description of localized magnetic moments, the Anderson model can also be applied to dilute magnetic alloys. Localized magnetic moments are found in non-magnetic metals containing ionic impurities with incomplete valence orbitals (orbital d or f).

The host non-magnetic metal is represented by a conduction band with momenta $\vec{k}$ and energies $\varepsilon_{\vec{k}}$. The impurity lies on a different energy level, separated from the conduction band. The level added by the impurity is characterized by the energy of the d (or f) orbital ε_d, with spin degeneracy σ. According to the Pauli exclusion principle, there are four configurations for the impurity orbital: empty, one electron with spin up, one electron with spin down or two electrons, one up and one down. The coupling of the impurity with the conduction band is represented through the hibridization V. The transition rate between the impurity and the conduction band is computed by Fermi's golden rule. The Coulombian repulsion between electrons in the same orbital U reduces the probability of double occupation of the impurity orbital and, if ε_d is negative, increases the probability of single occupation. Figure 1 gives a schematic representation of the model. The Hamiltonian of the model is given by:

$$\begin{aligned} H &= \sum_{\vec{k}\sigma} \varepsilon_{\vec{k}} c^\dagger_{\vec{k}\sigma} c_{\vec{k}\sigma} + \varepsilon_d \sum_\sigma c^\dagger_{d\sigma} c_{d\sigma} + \\ &\quad V \sum_{\vec{k}\sigma} (c^\dagger_{\vec{k}\sigma} c_{d\sigma} + \text{h.c.}) + U c^\dagger_{d\uparrow} c_{d\uparrow} c^\dagger_{d\downarrow} c_{d\downarrow} \end{aligned} \quad (1)$$

where $c^\dagger_{\vec{k}\sigma}$ creates an electron with momentum $\vec{k}$ and spin $\sigma = \pm 1/2$ in the conduction band while $c^\dagger_{d\sigma}$ creates an electron in the impurity orbital. The abbreviation h.c. indicates that the hermitian conjugate of the preceding term should be also included.

The Anderson Hamiltonian was at first analyzed by perturbative methods, but the first treatment able to describe the whole temperature range was the numerical renormalization group method (see subsection I-B).

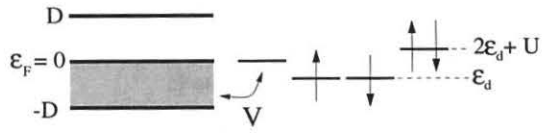

Fig. 1. Schematic representation of the one impurity Anderson model. The host non-magnetic metal is represented by a half-filled conduction band with width $2D$ and states with energy $\varepsilon_{\vec{k}}$. The magnetic impurity is characterized by states with energy ε_d. U is the Coulombian repulsion between electron occupying the same impurity orbital. Interaction between impurity and conduction band is through hibridization V.

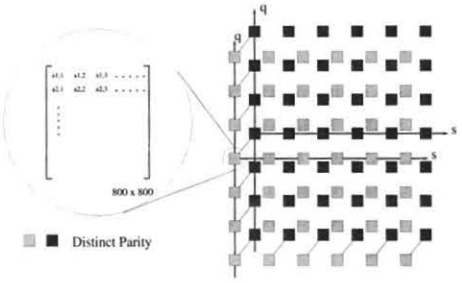

Fig. 2. Block organization of the Hamiltonian in charge q, spin s and parity p. Each block is called a *sector* and labeled qsp

As the single impurity case was already thoroughly studied, research now focus on impurity agglomerates. The two impurities case considered in this work is the simplest one that takes the interaction of many impurities into account. Its Hamiltonian is an extension of the Hamiltonian 1, see [AA64].

The two impurities problem has inversion symmetry: the Hamiltonian does not change under the exchange of the two impurities. As a consequence, the states of the conduction band can be classified according to their parity: the ones which change signal under an exchange (the odd states) and the ones that does not change signal (the even ones). The states can then be classified through three characteristics: their electric charge q, their spin s and their parity p. The diagonalization process can then be done in sub-matrices where all states have the same values for q, s and p. The whole Hamiltonian may be visualized as a set of these sub-matrices, as shown in figure 2

B. Numerical Renormalization Group Method

In the numerical renormalization group method the continuum of energy levels of the conduction band is subjected to a logarithmic discretization, proposed by Wilson [Wil75].

In this work, we use an alternate discretization procedure that introduces another parameter, usually called z. The procedure is described in [YWO90], [CPLO97]. Energies are chosen so that the Fermi energy is zero $\varepsilon_F = 0$ and energies less than $-D$ or greater than D are not considered. The interval between $-D$ and D is divided in pieces as shown in figure 3.

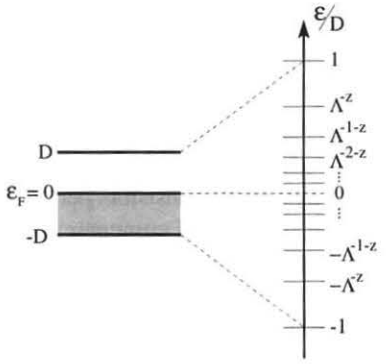

Fig. 3. Logarithmic discretization of the conduction band. Energies from $-D$ to D, measured from the Fermi energy $\varepsilon_F = 0$, are divided in a intervals at rates Λ^{-1}, where Λ is a discretization parameter.

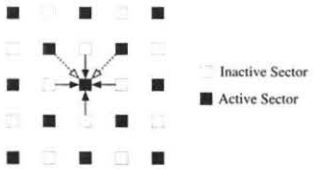

Fig. 4. Active and inactive sectors in one phase. The arrows shows data dependencies. (To simplify the figure, parities are not considered.)

C. Iterative Process

The number of states in the base of the Hamiltonian (and therefore the dimension of this Hamiltonian) grows very fast. For example, already in the second iteration there are 2^{16} states. It is thus mandatory to place some limit on the size of the matrices, what is achieved by neglecting states with energies greater than a limit, because these have lower probability and are not thermodynamically relevant.

Each iteration is done in two phases, called *high* and *low*, with only half of the sectors (distributed as in a chess-board) alternately active in each phase (fig. 4). This means that in a given moment only half of the sectors will be computing.

The operations in each phase can be summarized as follows:

- execution parameters are adjusted;
- neighboring sectors to the north, south, east and west (fig. 4, continuous arrows) are consulted to compute the new states;
- the Hamiltonian is computed using values from the other iteration (eigenvalues and invariant matrix elements, the latter stored in structures that will be called here *pages*);
- the Hamiltonian is diagonalized (computation of the eigenvalues and eigenvectors);
- new values for the pages are computed using eigenvalues and eigenvectors of this sector and of two neighbors at northeast and northwest (fig. 4, interrupted arrows);
- thermodynamic values of interest are computed and saved in a file after all sectors are computed as above.

The first iteration, called iteration -1, is handled separately and given as input to the program.

II. Tools

The continuous reduction of feature size in VLSI technologies, with the consequent possibilities to increase the clock rates and the number of transistors per chip has resulted in a convergence in the parallel computer architecture field [CSG99], in the possibility to attack new problems [Fos95] and in the use of commercial, low cost systems to do parallel processing, as in network of workstations [ACP96] and the Beowulf project [Ste95].

In this work, the system used is a cluster of personal computers with 450 MHz K6-III processors, 256 Mbytes of memory, each with its own system, swap and temporary disk space. The computers are connected by a Ethernet hub and a Fast Ethernet switch. The Ethernet connection is used to mount remote user file system and for administration. The Fast Ethernet connection is dedicated to communications of the parallel applications. The systems under Linux kernel 2.2 [Ker], and the program development is done using the GNU tools [Fou] and an MPI [Mes95] implementation from the Argonne National Laboratory (MPICH) [GL96].

III. Implementation

The language chosen for the implementation was C++ [Str97] and for the communication between the parallel processes an MPI library [GL96] was used.

The first step was to rewrite an existing sequential program in an object oriented style. It followed an analysis of data dependencies to assess the parallelization possibilities and the corresponding communication costs.

As already explained (see section I) the program must be run for many different values of some parameters. This fact was the starting point for a first parallel version, based on the variation of input parameters.

To avoid some limitations of this parallelization method, a second parallelization was made, based on the parallel execution of the calculations of each sector.

The following subsections describe these implementations.

A. Sequential Version

The development of a new sequential version of the program had three objectives:

1. to get acquainted with the technical details of the program;
2. to organize the program in such a way as to facilitate the parallelization process, specially through the use of object oriented techniques;
3. to reduce the amount of memory copying in some parts of the program.

TABLE I
Execution times (in hours) for the original and new sequential versions

	Original	*New*	*Reduction*
Small	0.70	0.24	66%
Medium	20	6.7	67%
Large	65	21	68%

The reduction in memory copying together with the simplification of some functions have resulted in a significant reduction in execution time. Times of the new version are about a third of the times of the old version, as exemplified by the typical result of table I for three problem sizes (small, medium and large).

B. Parameter Variation

As stated above (section I) the program must be run with different parameters during a process of tuning. But it also must be run with different values of a parameter (called z above) to reduce oscillations due to the discretization of the energy levels. The process is described in detail, for example, in [dP98].

An initial parallelization method used consists then in running the program for the different values of z simultaneously. A master process reads a files that describes the values of z to be used and send one at a time to each slave. The slaves receive one value, executes what is in essence the sequential program and then ask the master for another job. When all z were already sent for computation to some slave, the next slave to request a job will receive a finalization message.

This implementation is not considered in detail here. It has a good speedup (only somewhat restricted by the fact that different zs have different computing times), but lack scalability both on the number of processors and on the problem size. It does not scale with the number of processors because the number of different zs is limited (about eight are used). Also, to have more precision in the computations, needed for some problems, it is necessary to grow the size of the matrices, and this leads to a huge use of memory resources. As all matrices from one z need to be stored on the memory of the same node, the amount of memory in one node is a limitation to the precision achieved.

It is important then that the computations of each value of z be done in parallel. This is the goal of the second parallel implementation.

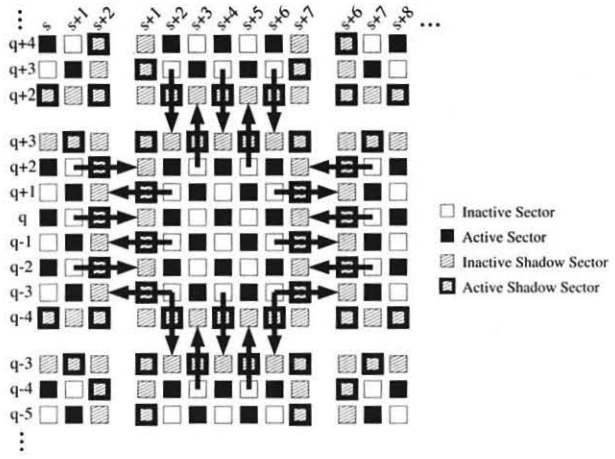

Fig. 5. Communication of parent data between processes (parity is not shown). Shadow sectors does not do any computations, they only store data.

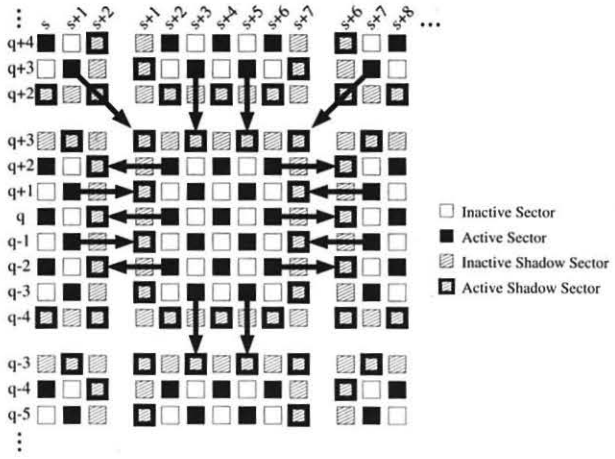

Fig. 6. Communication of neighbors between processes (parity is not shown).

C. Parallelizing Sector Computations

The computations done in each sector qsp are to an extent independent of the computations in another sector in the same phase. So it is possible to parallelize the program by distributing the computation of the sectors.

But the computations are not totally independent, due to the data dependencies depicted in figure 4:

- for the construction of the Hamiltonian, data from the neighbors in the horizontal and vertical, here called *parents*, directions of figure 4 are needed;
- for the computation of the new pages, data from the neighbors in the diagonal in figure 4, here called just *neighbors*, are needed.

We note here that, when a sector is being computed, its parents are not active (they will be active in the next phase). It should also be noted that these two communications are needed in different moments during the computations (see section I-C).

The parallelization is based on a division of the sectors (figure 2) through the directions q and s, but not p. The division in direction p will introduce much more communications and synchronizations between processes for large blocks. Each process has then a certain amount of sectors to compute and additionally some *shadow* sectors to store data from sectors that it needs from its neighbor processes (these are the hatched sectors in figures 5 and 6).

Figure 5 shows the communication of parent data to its shadow (only one process and its neighbors is shown).

After the diagonalization of the Hamiltonian, sectors on the border send some of its data to the neighboring process, where it will be put in the corresponding shadow, as in figure 6. Only neighbors up are needed, and so not all border sector must be sent.

TABLE II
EXECUTION TIMES OF SOME SECTORS.

Sector (q,s,p)	*Time (s)*
0, 2, 1	1539
-1, 1, 0	1117
-2, 1, 0	452
3, 4, 0	1.77
-4, 5, 1	0.05

Sectors that are not in the border can do its computations without waiting for these communications.

It can be seen from the description above that ample parallel execution possibilities does exist, but also many synchronization points limit the independent execution. The following section shows the results of the implementation of this program on the cluster described in section II.

IV. RESULTS

To complete the parallelization we need to decide the mapping of sectors to processes. But there is a problem: the computational load is very different from one sector to another. Sectors near the origin in figure 2 do more computations than sectors with larger values of q or s. Table II shows total computation times of selected sector on the same program run to demonstrate the large differences present.

The effect of this imbalance on the speedup of the parallel program can be seen by the comparison in table III. Both times are for the execution of the same problem on 15 processors. They differ by the fact that in one run all sectors with the same value of q (sector in the same line of figure 2) are computed by the same processor; the other run distributed

four sectors, (q, s) and $(q,s+1)$ in both parities, to each process (called *fine distribution* in table III), and maps the processes to the processors in a round-robin scheme. This last division is the most aggressive possible without dividing also the parities; its advantage is that the sector with large computational load can be distributed to different processors, as happens with the round-robin scheme used. A further division of (q,s) from $(q,s+1)$ would not help, because when one is active the other is inactive. The advantage can be seen in the results shown in table III: an almost doubling of the speedup.

TABLE III
EXECUTION TIME FOR 15 PROCESSORS, WITH TWO DIFFERENT MAPPING OF SECTORS TO PROCESSORS.

Division	*Time (hours)*
By lines	3.4
Fine	1.8

Using the last division and mapping scheme cited on the previous paragraph, table IV gives execution times and speedups (absolute with respect to sequential time of the new version, table I) for different number of processors, from 3 to 16. Less then 3 processors where not tested because of a limitation in the MPI library implementation used: MPICH (version 1.2.0) has problems starting too many processes on the same processing node.

TABLE IV
EXECUTION TIMES (T) AND ABSOLUTE SPEEDUP (S) OF THE PARALLEL PROGRAM.

P	T *(hours)*	S
03	4.0	1.7
04	7.6	0.9
05	3.5	1.9
06	2.1	3.3
07	2.1	3.2
08	3.1	2.2
09	2.1	3.2
10	1.9	3.5
11	2.1	3.3
12	2.1	3.3
13	2.0	3.4
14	1.8	3.8
15	1.8	3.7

Some problems can be seen by these results, all of them related with the load imbalances described in the discussion of the results of table II:

1. The efficiency is not good, being about 50% for the better results. This is because the large computation time differences among the sectors make a load balanced mapping almost impossible.
2. The speedup does not grow after six processors. This is due to the fact that, after six processors are available, all processes with very high computation times are in different processors and so the total computation time is dependent only on the computation time of the slowest process. If we compare the execution time of the most demanding sector with the total execution time we come to the conclusion that the theoretical maximum speedup achievable is 5.5, disregarding communication times. We see then that the factor that limits the speedup are the load imbalances, with communication costs of second importance.
3. Computation with four processors is slower than with three processors, and with eight processors is slower than with seven. This is an anomaly due to the fact that, with this number of processors, the round-robin mapping scheme puts a process with many calculations together in the same processor with the slowest process, thus resulting in sensibly enlarged total time.

In spite of the low efficiency achieved, the parallel program is very useful, because being able to reduce to a third the execution time of a program that may take some weeks to execute, as is the case in some of the interesting physical problems, is very important to the advancement of the research in the field.

V. CONCLUSIONS AND FURTHER WORK

The numeric renormalization group method applied to the Anderson model with two impurities is a computationally intensive problem. Its execution time limits the ability of the researcher to explore the configuration space and the computation of different physical properties; its memory requirements limit the precision that can be achieved. A parallel implementation of the method can help solve these two problems by distributing the computations and the data set among many processing nodes.

This work shows that a parallelization is possible and achieves significant reductions in execution times, as shown in section IV, although with somewhat limited efficiency in the use of the resources and scalability.

The problem has shown to be amenable to cluster computing, because it is not limited by communication costs, the weaker part of clusters of commercial hardware for parallel computing.

Efficiency and scalability can be enhanced by combining the parallelization described in section III-C, and whose re-

sults are shown in section IV, with that described in section III-B based on parameter variation. This will enable, for example, to achieve speedup with the use of all 16 processors in our cluster by running four parameter sets in parallel. This is a simple extension of the work already done and is being worked out now.

Another form to increase scalability and efficiency is to allow division by parity. This may enable a doubling on the scalability because sectors that differ only in the parity have almost the same computational difficulty, which means that the two heaviest sectors go always to the same processor in the parallelization described above. The problems with this solution are that the communications and synchronization grow significantly and that the parallel algorithm is more complex. An implementation under these lines is being considered.

To reduce the scalability limitations further the solution will be to parallelize the computations done inside a sector. These computations include (section I-C) diagonalization and a form of matrix multiplication (to compute the pages). These computations could be parallelized with the use of ScaLapack [BCC+97], although a careful study is necessary to enable the harmonious integration of the existing parallelization with that done by the ScaLapack routines.

The implementation of these alternatives will increase the importance of communication costs for the performance of the parallel program. The assessment of the usefulness of clusters under these conditions will be of great interest.

ACKNOWLEDGMENTS

We would like to thank Professor Luis Nunes de Oliveira for the useful collaboration at several stages. Financial support from FAPESP under grant number 98/14681-8 and scholarship number 99/04805-4 (ES) is acknowledged.

REFERENCES

[AA64] S. Alexander and P. W. Anderson. Interaction between localized states in metals. *Phys Rev*, 133(6), 1964.

[ACP96] T. E. Anderson, D. E. Culler, and D. A. Patterson. A case for now. *IEEE Micro*, 1(15), 1996.

[And61] P. W. Anderson. Localized magnetic states in metals. *Phys Rev*, 124(1), 1961.

[BCC+97] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLapack User's Guide*. SIAM, 1997.

[CPLO97] S. C. Costa, C. A. Paula, V. L. Libero, and L. N. Oliveira. Numerical renormalization-group computation of specific heats. *Phys. Rev. B*, 55(1), 1997.

[CSG99] David E. Culler, Jaswinder Pal Sing, and Anoop Gupta. *Parallel Computer Architecture. A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[dP98] C. A. de Paula. *Densidade espectral para o modelo de Anderson de duas impurezas*. PhD thesis, IFSC-USP, 1998.

[Fos95] Ian Foster. *Designing and Building Parallel Programs*. Addison Wesley, 1995.

[Fou] Free Software Foundation. The gnu project, `http://www.gnu.org`. visited 2001/05/04.

[GL96] Willian Gropp and Edwing Lusk. *User's Guide for MPICH, A Portable Implementation of MPI*. Argonne National Laboratory, 1996.

[Ker] The Linux Kernel. `http://www.kernel.org`. visited 2001/05/04.

[KMWW80] H. R. Krishna-Murthy, J. W. Wilkins, and K. G. Wilson. Renormalization-group approach to the anderson model of dilute magnetic alloys. *Phys Rev B*, 21(3), 1980.

[Mes95] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 1995.

[Oli92] L. N. Oliveira. The numerical renormalization group and the problem of impurities in metals. *Braz. J. of Phys.*, 22(3), 1992.

[Ste95] T. Sterling. Beowulf: a parallel workstation for scientific computing. In *Proc. of the 1995 Intl. Conf. on Parallel Processing*, 1995.

[Str97] Bjarne Stroustrup. *The C++ Programming Language*. Addison Wesley, 3rd edition edition, 1997.

[Wil75] K. G. Wilson. The renormalization group: critical phenomena and the kondo problem. *Rev of Mod Physics*, 47(4), 1975.

[YWO90] M. Yoshida, M. A. Whitaker, and L. N. Oliveira. Renormalization-group calculation of excitation properties for impurity models. *Phys. Rev. B*, 41(3), 1990.