Parallelization of the two–dimensional Ising Model on a Cluster of IBM RISC System/6000 Workstations

Peter Altevogt[§], Andreas Linke
IBM Scientific Center
Heidelberg, Germany

IBM Scientific Center
Vangerowstr. 18
D–6900 Heidelberg
Germany
Phone: 06221–59–3000

[§]altevogt@dhdibm1.bitnet

Abstract

Using the PVM programming environment for parallel applications, we have parallelized a simulation of the two–dimensional Ising Model on a cluster of IBM RISC System/6000[1] workstations connected by a Token Ring (16Mb/sec) and by Serial Optical Channels (220 Mb/sec) via a NSC[2] DX Router. The parallelization is done by dividing the lattice into sublattices, each sublattice being associated with one workstation. On each sublattice, a Metropolis algorithm using Multispin Coding techniques is used to generate new configurations. We provide numerical results concerning the number of spin updates per second, speedups, and efficiencies for various numbers of processors and lattice sizes.

Keywords. Statistical Physics; Ising Model; Workstation Cluster; Geometric Parallelization.

1 Introduction

The goal of Theoretical Statistical Physics is a mathematical description of thermodynamic properties (e.g. of magnetism or phase transitions) of macroscopic bodies, commercing with a description of the microscopic components of these bodies (e.g. molecules, atoms and electrons) and their degrees of freedom [1]. Because the number of degrees of freedom being very large (ca. 10^{23}), an approach similiar to the one used in Classical Mechanics (define a Hamiltonian as a function of the degrees of freedom, specify the initial conditions and solve the equations of motion) is not appropriate. Instead, it is necessary to use statistical methods to derive the thermodynamic properties of macroscopic bodies.

The approach taken in Statistical Physics may be divided in three steps:

1. Define a Hamiltonian (a model) as a function of the microscopic components of the system.
2. Determine the probability distribution of these microscopic components in thermal equilibrium.
3. Compute macroscopic properties of the system using this probability distribution.

Ad 1.: Defining the Hamiltonian as a function of *all* microscopic degrees of freedom and their interactions would result in a complexity of the model, which makes the computation of macroscopic properties impossible. Therefore, one

[1] IBM RISC System/6000 is a trademark of the International Business Machines Corporation.

[2] NSC is a trademark of the Network Systems Corporation.

first identifies the degrees of freedom relevant to the problem one is considering and defines an effective Hamiltonian as a function of these degrees of freedom. These Hamiltonians may then depend on a few parameters, which can be fitted from experiments or approximately computed in a microscopic way.

The definition of such models is very difficult: On one hand, the model should be simple enough to facilitate explicit calculations of macroscopic properties, on the other hand, it should be flexible enough to reproduce the essential features of the system under investigation. These models (e.g. the Ising–, Heisenberg–and XY–Models) have played a crucial role in the development of Statistical Physics: They have been the interface between mathematical studies (e.g. about symmetries, the general structure of interactions, etc.) and experimental data.

Ad 2.: We assume our system to be in thermal equilibrium with a heat bath, i.e. we assume our equilibrium probability distribution P to be given by the Boltzmann formula:

$$P(\{s\}) = \frac{e^{-\beta H_{Model}(\{s\})}}{Z},$$

with the normalization constant Z (the partition function)

$$Z = \sum_{\{s\}} e^{-\beta H(\{s\})}.$$

Here s denotes the microscopic degrees of freedom and we sum over all configurations $\{s\}$. β is related to the absolute temperatur T via the Boltzmann coefficient k ($k = 1.38 \cdot 10^{-23}$ J/K):

$$\beta = \frac{1}{kT}.$$

Ad 3.: In spite of the above mentioned simplifications in building models, exact analytical solutions (e.g. the exact evaluation of the partition function) in general exist only in a few special cases. Furthermore, approximation methods like e.g. low–and high–temperature expansions, are often useful only within a limited domain of the parameters of the model. Therefore computer simulations turn out to be an important tool to make contact with experimental data [2] [3] [4] [5].

For large simulations, e.g. for simulations performing very many sweeps on a big lattice, a multiprocessor system of the MIMD–type[3] with distributed memory seems to be the most appropriate computer system, because (ideally):

[3]For an excellent overview on the various models of computation like SISD, SIMD, etc., see Akl [6].

1. the number of processors may be increased without changing the principal behaviour of the system ("scaling behaviour")

2. the resources of the processors (including memory) add up.

Furthermore, for many computer simulations a natural way of parallelization exists by dividing the domain of the simulation in subdomains and mapping these subdomains on the processors of the computer system (geometrical parallelization) [5]. In this paper we will apply this method to the Ising Model.

The paper is organized as follows: In section 2 we give an introduction to the Ising Model, its computer simulation using the Metropolis Algorithm [7] and discuss a few general aspects of its implementation on multiprocessor systems with distributed memory. In section 3 we describe our hardware and system software, including the programming environment PVM. Section 4 contains the algorithm of our simulation and in section 5 we present our results.

2 The Ising Model

One of the most important models of Statistical Physics is the Ising Model, introduced by E. Ising in 1925 [8]. It describes ferromagnetic (resp. antiferromagnetic) materials with a very strong uniaxial anisotropy, so the spins can take on only values along this axis. The Ising Model is defined by a Hamiltonian on a simple cubic lattice:

$$H_{Ising}(\{s\}) = -J \sum_{<i,j>} s_i s_j - h \sum_i s_i,$$

where $< i, j >$ denotes the summation over all possible nearest neighbour[4] pairs of the lattice, the spins s_i can take only the values ± 1 and h denotes an external magnetic field. If the exchange parameter J is positive, H_{Ising} describes a ferromagnetic system; if J is negative, the system is antiferromagnetic. We are considering a ferromagnetic system and fix the exchange parameter J to be $+1$.

Expectation values of physical observables (e.g. the magnetization), correlation functions and the free energy can be derived from the partition function:

$$Z_{Ising} = \sum_{\{s\}} e^{-\beta H_{Ising}(\{s\})},$$

where $\{s\}$ denotes the sum over all spin configurations. E.g., the expectation value of the magnetization $< M >$ is given by

[4] In one dimension a lattice point has two neighbouring lattice points, in two dimensions four, in three dimensions six, etc..

$$< M >= \frac{\sum_{\{s\}} \sum_i s_i e^{-\beta H_{Ising}(\{s\})}}{Z_{Ising}},$$

the spin–spin correlation functions $< s_i s_j >$ are obtained as follows:

$$< s_i s_j >= \frac{\sum_{\{s\}} s_i s_j e^{-\beta H_{Ising}(\{s\})}}{Z_{Ising}},$$

and for the free energy F_{Ising} we have:

$$F_{Ising} = -\frac{1}{\beta} ln(Z_{Ising}).$$

Exact solutions of Ising Models (i.e. evaluation of the free energy and calculation of the correlation functions) exist only in one dimension and in two dimensions with the external magnetic field h being zero. Therefore in many cases computer simulations provide the only tool for the evaluation of physical observables (e.g. critical exponents) of Ising Models [11] [12].

The computer simulation of the Ising Model consists of two parts:

1. The Metropolis Algorithm [7] to generate a Markov chain of spin configurations such that the probability $P(\{s\})$ for the spin configurations $\{s\}$ tends to $P(\{s\}) \sim e^{-\beta H_{Ising}(\{s\})}$:

   - Initialize the spin configuration (e.g. all spins "up" or a random spin configuration)
   - For each spin s_i of the lattice[5]:
     - Generate a new spin configuration by flipping the spin s_i.
     - Compare $e^{-\beta \Delta H_{Ising}}$ with a random number[6] r, where

       $$\Delta H_{Ising} := H_{Ising}(\{s\}_{new}) - H_{Ising}(\{s\}_{old})$$

       denotes the energy difference between the new and the old spin configuration and is evaluated by

       $$\Delta H_{Ising} = 2 s_i (\sum_{<j>} s_j + h),$$

       where $< j >$ stands for the summation over the next neighbours of s_i. The random numbers are uniformly distributed in $(0, 1)$,

[5] We have chosen to pass through the lattice in a typewriter like fashion, starting at the "upper left corner". Any other way of passing through the lattice and updating the spins is allowed, as long as only dynamically independent spins are updated in parallel.

[6] Because of the use of random numbers, these simulations are frequently called Monte Carlo simulations.

- Accept the new configuration only if $e^{-\beta \Delta H_{Ising}} > r$.

2. The calculation of expectation values $< O >$ of physical observables O:

$$< O >= \lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} O(\{s\}_i),$$

where $O(\{s\}_i)$ denotes the value of O for the spin configuration $\{s\}_i$[7].

The spins being binary variables, using bits for the representation of spins[8] instead of integers or floating point numbers not only saves a lot of computer memory, but also allows various operations (e.g. addition, logical operations, etc.) to be performed simultaneously on all spins (bits) of one computer word. Therefore techniques based on this idea, Multispin Coding techniques [13] [14] [15] [18], have become an important tool for simulating spin systems.

For calculating the energy difference ΔH_{Ising} between new and old spin configurations, we have to add up all neighbouring spins of the spin under consideration. This results in a number between 0 and 4. Therefore, to apply Multispin Coding techniques in two–dimensional Ising Models, *three* bits are necessary to represent one spin[9]. Using an integer variable on a IBM RISC System/6000 consisting of 32 bits, this sum may be computed in parallel for ten spins.

A remarkable feature of the Ising Model is its phase transition at the Curie temperature[10] to the ferromagnetic phase, i.e. the Ising Model exhibits spontanous magnetization[11].

Because of their importance, a lot of work has been done on algorithms to simulate Ising Models on various computer systems, like e.g. on single processor computer systems (SISD–computers) [13] [14] [15] [16], on vector computers (SIMD–computers) [17] [18] [19] and on multiprocessor systems (MIMD–computers) [20] [21].

---

[7] Considering effects due to the finiteness of the lattice, it may be necessary to average over the absolute value of $O(\{s\}_i)$, see [3]

[8] We choose a 1-bit to represent "spin up" and a 0-bit to represent "spin down"

[9] In models, where the interaction between neighbouring "spins" may be described by logical operations, e.g. Lattice Gases, one bit is sufficient.

[10] Working in "natural units" (setting the Boltzmann constant k equal to one), the critical temperature of the two–dimensional Ising Model is $T_C = 2.269$ for the exchange parameter $J = 1$.

[11] This phase transition, i.e. especially the formation of spin clusters near the Curie temperature, may be nicely visualized for the two dimensional case on a graphic workstation using e.g. blue pixels for "spin up" and red pixels for "spin down".

Because of their local interactions, Ising Models [9] [10] are well suited for a parallelization on MIMD–systems with distributed memory by dividing the lattice in sublattices and mapping these sublattices on the processors: Using local algorithms, only the spin configurations on the surface of the sublattices have to be communicated to neighbouring processors. Considering furthermore the very small surface (compared to the volume) of sublattices in two dimensions it is quite obvious, that the two–dimensional Ising Model allows for a parallelization with modest communication between the processors.

Therefore a multiprocessor system with moderate bandwidth between the processors but high computational power, like a cluster of high–end workstations connected e.g. with a 16Mb/sec Token Ring, is an attractive hardware platform for the implementation of a simulation of the two–dimensional Ising Model.

3 Hardware und Systemsoftware

Our hardware platform consists of one RS/6000–560 and four RS/6000–550 workstations, all equipped with 128MB main memory and connected by a Token Ring (16 Mb/sec) as well as with Serial Optical Channels (220 Mb/sec) via a NSC DX Router. All machines use AIX Version 3.2 as the operating system.

Although first prototypes of tools for automatic parallelization of programs for distributed memory machines exist, these tools seem to be limited to fine grain parallelizations, e.g. a parallelization of loops. Because of high latency times ($\sim$ 1msec), this kind of fine grain parallelism is not well suited for a workstation cluster. Therefore, a new design and implementation of a more coarse grain parallel algorithm, including the initiation, the communication and synchronisation of the various processes on the processors is necessary.

Several interfaces exist to accomplish this task within Unix systems.

The lowest level interface is provided by the driver interfaces of the network adapters. Programming these interfaces, on the one hand one avoids a lot of overhead originating from various network protocols, e.g. from TCP/IP or UDP/IP, but on the other hand, the functionality of these interfaces is very poor[12] and highly hardware dependent.

The standard interface for programming distributed applications in Unix Systems is the sockets interface [24]. This interface is based on network protocols like TCP/IP or UDP/IP and forms the basis for most distributed applications in

[12]The standard Unix driver interface just consists of the routines open(), read(), write(), ioctl() and close(), see Bach [22] or Leffler et. al. [23].

networks of Unix sytems, e.g. for RPC[13] (Remote Procedure Call) and NCS[14] (Network Computing System). Using this interface, the programmer is still left with the administration of processes and addresses of the distributed application and therefore sockets do not provide a convenient interface for programming parallel applications.

For programming parallel applications on a workstation cluster, the use of a programming environment, e.g. PVM (Parallel Virtual Machine) [25] [26], EXPRESS [27] [28] [29] or PARMACS [30], seems to be the most convenient (but not necessarily the most performant) choice. These programming environments provide as the basic functionality a library of routines to create processes on the cluster's workstations, to communicate between these processes, to synchronize these processes, etc..

In PVM, this functionality is provided with the help of demon processes, running on each of the cluster's worktations. The processes of the parallel application communicate with each other via these demon processes.

4 The Algorithm

Our simulation of the Ising Model consists of one client process and several server processes communicating with each other via the PVM system by exchanging messages.

The parallelism is achieved by dividing the two dimensional lattice into stripes associated with one server process[15]. The server processes are mapped on the cluster's workstations such that there is a one–to–one corespondence of processes and processors[16]. Each stripe is augmented with auxiliary lattices on all sides to implement periodic boundary conditions as well as to store the part of the spin configurations communicated by the neighbouring processors which is necessary to do the local updates.

In the sequel we will describe the two kind of processes (client and server processes) and their implementation[17].

[13] RPC is a trademark of Sun Microsystems, Inc.

[14] NCS is a trademark of the Apollo Systems Division of the Hewlett–Packard Company.

[15] Because of the high latency times for message passing in a workstation cluster, it is more performant to communicate fewer but larger messages [32]. Therefore parallelizing only in one space direction is often appropriate.

[16] Measurements with all five workstations of our cluster have been made with the client process and one of the server processes running on the RS/6000-560. Because the client sleeps most of the time, this should not affect significantly the performance of the server process.

[17] We have chosen C (instead of Fortran) to implement our algorithm, because the binary operations needed for the Multispin Coding technique are an integral part of C. Furthermore,

The client process:

The client process handles most of the administrational tasks (e.g. doing the I/O to obtain various parameters that specify the model and to store the configurations) and carries out the calculations concerning the lattice as a whole, e.g. calculating physical observables like the magnetic densities from the results of the servers obtained for their sublattices:

- Scans the commandline of the process for the parameters of the simulation (lattice size, number of server processes, number of sweeps, temperature and the external magnetic field).
- Introduces itself to the PVM programming environment.
- Starts the server processes on the cluster's workstations.
- Creates a lookup table for $e^{-\beta \Delta H_{Ising}}$.
- Broadcasts the following data to each server:
  - The lookup table for $e^{-\beta \Delta H_{Ising}}$.
  - The external magnetic field.
  - The sizes of the stripe (sublattice) associated with the appropriate server.
  - The virtual process numbers (initialized by PVM) of the neighbour ("upper" and "lower" servers of the appropriate server.
  - The number of sweeps.
- If measurements (e.g of physical observables or time spent for communication) were done, the client process waits for the results from the servers (e.g. the magnetic densities as calculated for the sublattices) and averages over these results.
- Stores the averaged results on disk.
- Leaves the PVM programming environment.

---

the dynamic allocation of memory is much easier in C than in Fortran and the interfaces to system software libraries (e.g. to libraries for visualisation) on Unix systems are often only provided in C.

A server process:

A server process executes the usual Metropolis algorithm (using a Multispin Coding technique [13] [14] [15] [18]) for the simulation of the Ising Model at constant temperature [4] [5] on its sublattice:

- Introduces itself to the PVM programming environment.
- Initializes a seed (unique for each server) for its random number generator.
- Receives the data sent by the client.
- Allocates storage for the spin configuration on its sublattice, including the auxiliary lattices, and for physical observables, if necessary.
- Initializes the configuration (all spins "up")
- Performs the specified number of Metropolis sweeps on the sublattice by using a Multispin Coding technique. The periodic boundary conditions are implemented by updating the appropriate auxiliary lattices if necessary. In detail:
    - Allocates storage (an array) for random numbers (one for each spin of a line).
    - For each line, starting with the upper line:
        - Initializes the array with random numbers between 0 and 1. using the *surand()* subroutine of the ESSL subroutine library [31] and
        - For all spins in a line, starting with the leftmost spin:
            - Calculates the sum of the neighbouring spins (this is done in parallel for ten spins with the help of the Multispin Coding technique).
            - For all spins in a word:
            - Uses the above sum as an index in the "look up table" for $e^{-\beta \Delta H_{Ising}}$. Only if this number is bigger than the random number, will the spin be reversed.
        - If the current line is the first line of the sublattice, the server sends the updated spin values of this line to the "upper" neighbour processor.
        - If the current line is the penultimate line of the sublattice, the server receives the updated spin values from the first line of the "lower" neighbour processor and stores the received spin values in its auxiliary lattice.

  - The server sends the updated spin values of its last line to the "lower" neighbour processor.
  - The server receives the updated spin values for its first line from the last line of the "upper" neighbour processor and stores the received spin values in its auxiliary lattice.
- If measurements were done, the results are send to the client process.
- Leaves the PVM programming environment.

The receive subroutines of the PVM system being blocking, the receive calls within the client and server processes constitute the synchronization points of the processes of the simulation. The send subroutines of the PVM system are nonblocking: they return as soon as possible, i.e. when the local PVM demon process believes it can deliver the message [33].

This behaviour of the receive subroutines ensures that every processor uses the *same unique*[18] spin values assigned to each lattice site for updating the spins on the boundaries of its own sublattice. E.g., a processor is allowed to calculate the updates of the spins of its last line only *after* receiving the updated spin values from the first line of its "lower" neighbour processor.

5 Results

To check the correctness of our algorithm, we have measured e.g. the magnetization of the two–dimensional Ising Model as a function of the temperature for various lattice sizes, see figure 1[19].

We have measured the spin updates per second (flips) and the speedup (efficiency) as a function of the lattice size and the number of processors. A graphical representation of our results can be found in figures 2 to 7.

The speedup S_p and the efficiency E_p for p processors are defined by [34]:

$$S_p = \frac{T^*}{T_p},$$

$$E_p = \frac{S_p}{p}.$$

[18] This uniqness of the spin values is essential for the Hamiltonian of the system to be well defined.

[19] Due to finite size effects, the critical temperature is shifted to higher values for small lattice sizes.

Here T^* denotes the time for the optimal serial algorithm to solve the problem and T_p the time a parallel algorithm using p processors needs.
It is well known [35] that the measurement of the speedups (efficiencies) may be quite misleading for several reasons:

- In general the optimal serial algorithm is unknown and therefore the speedup depends on the serial algorithm chosen instead, e.g. one defines T^* as the time required by a single processor to execute the particular parallel algorithm being analyzed (as we do). Therefore with this choice of T^*, the speedup (efficiency) provides no information about the absolut quality of the algorithm, but only indicates how well this particular algorithm has been parallelized.
- Ideal values for speedup $S_p = p$ and efficiency $E_p = 1$ can easily be surpassed on virtual memory systems due to paging effects: In the "worst" case, the measured speedup converges to the difference in the access times between disk and memory.

Bearing these warnings in mind, we take a look at our results. As the independent variable of our measurements we take either the number of processors or the lattice size. While the parallelization turns out to be rather destructive for lattices smaller than a "critical lattice size" (approximateley 300×300), for lattices bigger than 1000×1000 lattice points[20], the speedups and efficiencies are very satisfactory. A similiar behaviour is true for the spin updates per second (Flips resp. MFlips).
The results presented in figures 2–6 have been measured by using the Serial Optical Channels and the NSC DX Router, the results presented in figure 7 contain also data obtained by using the Token Ring[21]. As can be seen (especially from the efficiencies shown in Figure 7), the results are qualitatively identical in both cases with clear advantages for the Serial Optical Channels for small and medium sized lattices ($< 1000 \times 1000$). The communication costs being negligible for large lattice sizes ($> 10000 \times 10000$), the measured values tend to the same asymptotic values for the Serial Optical Channels and the Token Ring.

[20] If the lattices' sizes cannot be divided by the number of processors, we round the lattice sizes up to the next appropriate value.

[21] We used the virtual circuit routines *vsnd()* and *vrcv()* of PVM 2.4 for the communication between the processors. All measurements have been done on a dedicated cluster after rebooting all workstations.

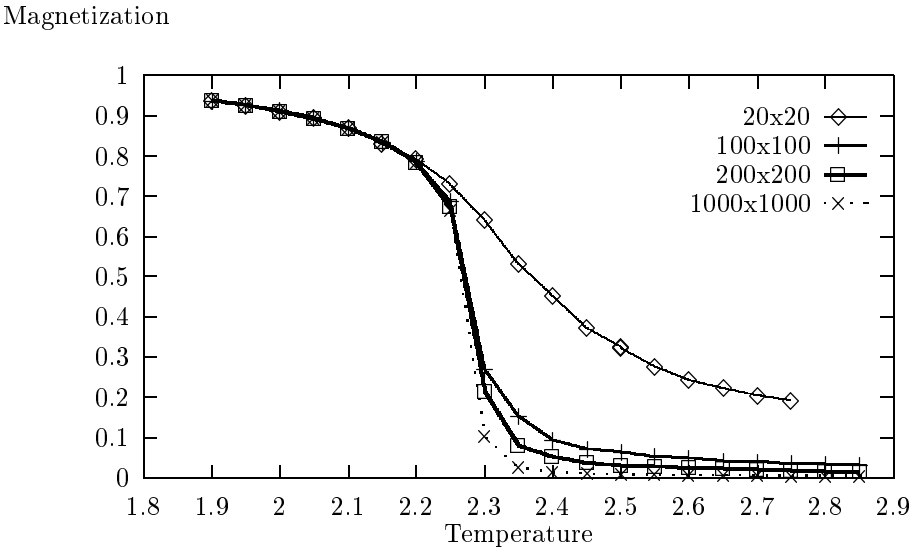

Figure 1: The magnetization as a function of the temperature for various lattice sizes.

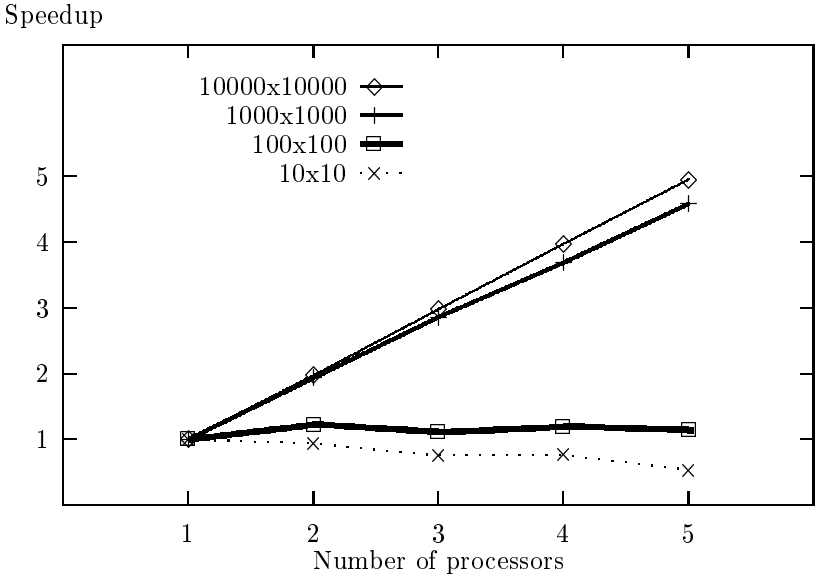

Figure 2: Speedup as a function of the number of processors for various lattice sizes using the Serial Optical Channels.

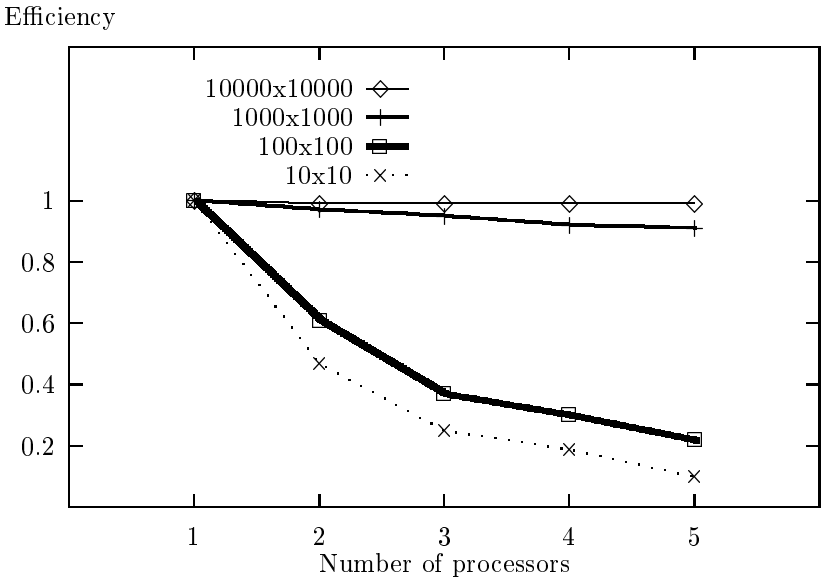

Figure 3: Efficiency as a function of the number of processors for various lattice sizes using the Serial Optical Channels.

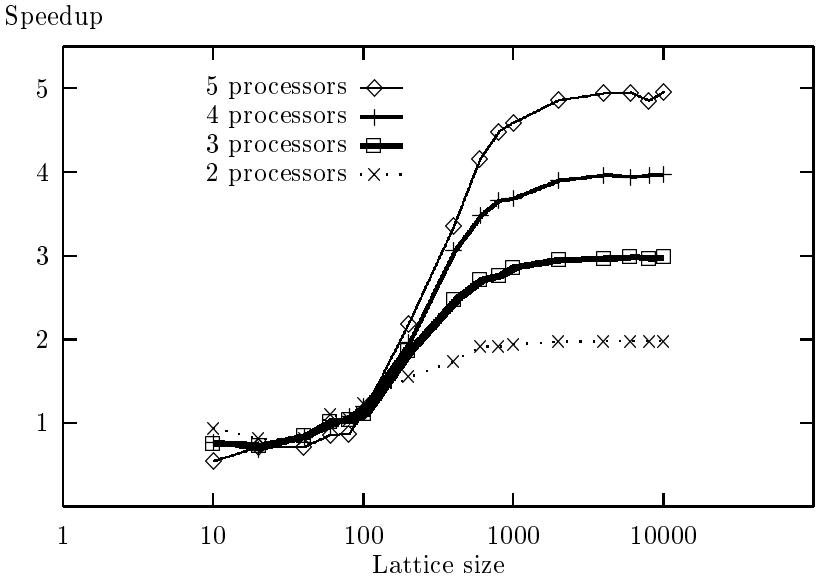

Figure 4: Speedup as a function of the lattice size for various numbers of processors using the Serial Optical Channels.

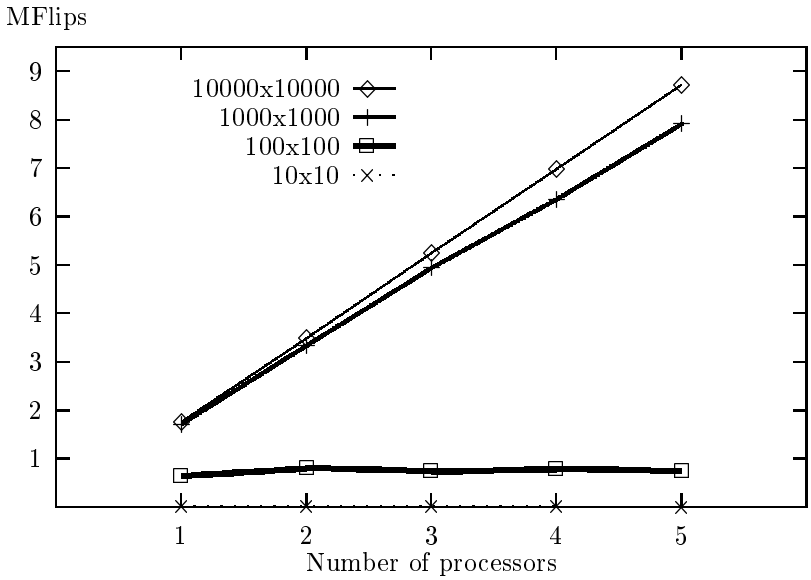

Figure 5: Spin updates (MFlips) as a function of the number of processors for various lattice sizes using the Serial Optical Channels.

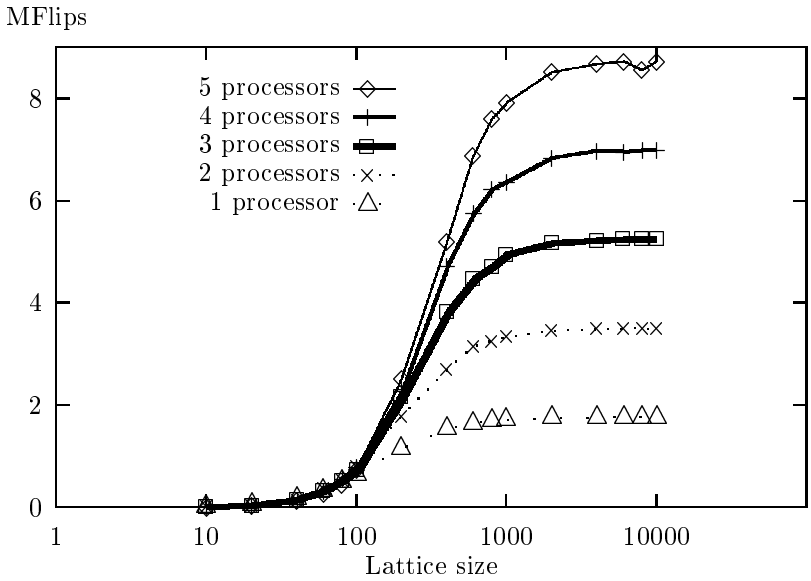

Figure 6: Spin updates (MFlips) as a function of the lattice size for various numbers of processors using the Serial Optical Channels.

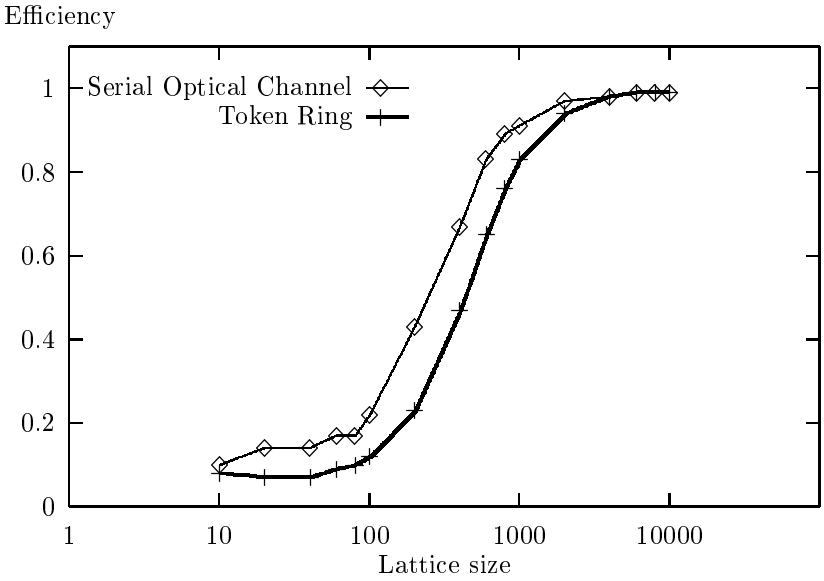

Figure 7: Efficiency as a function of the lattice size for 5 processors using the Serial Optical Channels (220 Mb/sec) and the Token Ring (16 Mb/sec).

6 Conclusions

We have shown in this paper that for the simulation of the two–dimensional Ising Model defined on "large" lattices ($> 1000 \times 1000$), the geometric parallelization of the Metropolis algorithms on a cluster of high–end workstations turns out to be very successful. This indicates, that a workstation cluster is an appropriate multiprocessor system for the simulation of a large class of models from Statistical Physics of a structure similiar to the one of the Ising Model.

Implementing such a synchronous parallel algorithm on a *heterogeneous* cluster of workstations, a loadbalancing between the processors of the system (taking into account the actual resources being available on each node of the system)

turns out to be very helpful, because the processor with the least resources determines the speed of the complete algorithm.

The heterogenity of the MIMD system may not only result because of heterogeneous hardware resources, but also due to a *heterogeneous use* of homogeneous hardware resources (e.g. on a workstation cluster, there may exist several serial tasks running on some of the workstations of the cluster for some time in addition to the parallel application; this results in a temporary heterogenity of the cluster, even if the workstations of the cluster are identical). This kind of heterogenity can in general only be detected during the runtime of the parallel algorithm.

Therefore, our approach of geometric parallelization to parallelize algorithms by a *static* decomposition of a domain into subdomains and associating each subdomain with a processor of the MIMD system seems to be appropriate only for a homogeneous MIMD system. On a heterogeneous system this geometric parallelization should be done *dynamically* [36].

References

[1] G.Parisi, Statistical Field Theory (Addison–Wesley Pub. Comp., Inc. 1988).

[2] K. Binder (ed.), Monte Carlo Methods in Statistical Physics, Topics in Current Physics, Vol. 7, 2nd edition (Springer–Verlag, Berlin, Heidelberg 1986).

[3] K. Binder, D.W.Heermann, Monte Carlo Simulation in Statistical Physics: An Introduction, Springer Series in Solid–State Sciences, Vol. 80 (Springer–Verlag, Berlin, Heidelberg 1988).

[4] D.W.Heermann, Computer Simulation Methods in Theoretical Physics, 2nd edition (Springer–Verlag, Berlin, Heidelberg 1990).

[5] D.W. Heermann and A.N. Burkitt, Parallel Algorithms in Computational Science, Springer Series in Information Sciences (Springer–Verlag, Berlin, Heidelberg 1991).

[6] S.G. Akl, The Design and Analysis of Parallel Algorithms, Prentice–Hall International Editions (Prentice–Hall, Inc. 1989).

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

[8] E. Ising, Z. Physik 31, 253 (1925).

[9] Kadanoff et. al., Rev. Mod. Phys. 39, 395 (1967).

[10] B.M. McCoy and T.T. Wu, The Two–Dimensional Ising Model (Harvard Univ. Press, Cambridge, MA 1978).

[11] G.S. Pawley, R.H. Swendsen, D.J. Wallace and K.G. Wilson: Monte Carlo renormalization–group calculations of critical behavior in the simple–cubic Ising model, Phys. Rev. B 29, 4030 (1984).

[12] A.M. Ferrenberg and D.P. Landau: Critical behavior of the three–dimensional Ising model: A high resolution Monte Carlo study, Phys. Rev. B 44, 5081 (1991).

[13] L. Jacobs and C. Rebbi, Multi–spin Coding: A very efficient Technique for Monte Carlo Simulations of Spin Systems, J. Comp. Phys. 41, 203 (1981).

[14] G.O. Williams and M.H. Kalos, A new Multispin coding Algorithm for Monte Carlo Simulation of the Ising Model, J. Stat. Phys. 57, 283 (1984).

[15] R. Zorn, H.J. Herrmann and C. Rebbi, Tests of the Multi–Spin–Coding Technique in Monte Carlo Simulations of Statistical Systems, Comp. Phys. Commun. 23, 337 (1981).

[16] C. Kalle and V. Winkelmann, Monte Carlo Technique for very large Ising Models, J. Stat. Phys. 28, 639 (1982).

[17] G. Bhanot, D. Duke and R. Salvador, A fast Algorithm for the Cyber 205 to simulate the 3D Ising Model, J. Stat. Phys. 44, 985 (1986).

[18] S. Wansleben, J.G. Zabolitzky and C. Kalle, Monte Carlo Simulation of Ising Models by Multispin Coding on a Vector Computer, J. Stat. Phys. 37, 271 (1984).

[19] A.Desalvo, G.Erbacci and R.Rosa, Vectorized code for the three–dimensional spin–exchange kinetic Ising model on cubic and diamond lattices, Comp. Phys. Commun. 60, 305 (1990).

[20] G. Bhanot and S. Sastry, Solving the Ising Model exactly on a $5 \times 5 \times 4$ Lattice using the Connection Machine, J. Stat. Phys. 60, 333 (1990).

[21] D.W. Heermann and R.C. Desai, Ising Model, Transputer and dynamical Correlations using a Microcanonical Ensemble, Comp. Phys. Commun. 50, 297 (1988).

[22] M.J. Bach, The Design of the UNIX Operating System (Englewood Cliffs, NJ: Prentice Hall, 1987).

[23] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman, The Design and Implementation of the 4.3BSD UNIX Operating System (Addison Wesley, Reading, 1989).

[24] W.R. Stevens, Unix Network Programming (Prentice Hall Inc., Software Series 1990).

[25] A. Beguelin, J.J. Dongarra, G.A. Geist, R.Manchek and V.S. Sunderam, A user's guide to PVM parallel virtual machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[26] V.S. Sunderam, PVM: A Framework for Parallel Distributed Computing, Concurrency: Practice&Experience Vol.2 No.4, Dec. 1990.

[27] Express 3.2, Introductory Guide (Workstations), ParaSoft Corporation, 1988, 1989, 1990, 1991.

[28] Express C, Users Guide (Version 3.0), ParaSoft Corporation, 1988, 1989, 1990.

[29] Express C, Reference Guide (Version 3.0), ParaSoft Corporation, 1988, 1989, 1990.

[30] L. Bomans, R. Hempel and D. Roose: The Argonne/GMD macros in FORTRAN for portable parallel programming and their implementation on the Intel iPSC/2, Parallel Comp. 15, 119 (1990).

[31] Engineering and Scientific Subroutine Library (Version 2), Guide and Reference, IBM Corporation (1992).

[32] P. Altevogt et. al.: To be published.

[33] The PVM Programmer's Manual.

[34] D. P. Bertsekas and J. N. Tsitsiklis: Parallel and Distributed Computation (Prentice Hall Inc., Englewood Cliffs, NJ 1989).

[35] R.Hockney: Performance parameters and benchmarking of supercomputers, Parallel Computing 17 (1991) 1111-1130.

[36] P. Altevogt, A. Linke: To be published.