ELSEVIER

Physica A 212 (1994) 277–298

PHYSICA A

3D Ising model with Swendsen-Wang dynamics: a parallel approach

M. Bauernfeind, R. Hackl, H.-G. Matuttis, J. Singer, Th. Husslein, I. Morgenstern

Institut für Physik II – Festkörperphysik, Universität Regensburg, 93040 Regensburg, Germany

Received 25 July 1994; revised 28 July 1994

Abstract

We propose an efficent parallel implementation of the Swendsen-Wang algorithm for a 3D Ising system. A modified relaxation method was used for the parallelization. The simulations were performed on the Intel Paragon. We discuss the implementation in detail.

1. Basics

The Hamiltonian of the spin-$\frac{1}{2}$ Ising model is

$$\mathcal{H} = -J \sum_{\langle ij \rangle} s_i s_j - H \sum_i s_i, \qquad s_i \in \{\pm 1\}. \tag{1}$$

J denotes the coupling constant. s_j represents the spin at the position j. $\langle ij \rangle$ means that the sum is taken over next neighbors only. The magnetic field is indicated by H. In this paper we describe simulations of the 3D Ising Model without magetic field ($H = 0$). Up to now only the two dimensional Ising-Model is exactly solved. The standard approach for treating the three dimensional Ising-Model is the simulation. Especially the critical exponents are of interest. Possible algorithms like single spin flip algorithm [6] or Wolf algorithm [5] for large system habe already been implemented. In this paper we present the first parallelisation of a 3D-Swendsen-Wang algorithm we know of.

The single spin flip algorithm chooses a spin s_i satisfying detailed balance and flips this spin with a temperature dependent probability, e.g. with $P = \exp(2Js_i \sum_j s_j / kT)$. One drawback is the critical slowing down, especially at the critical temperature, so that many configurations have to be discarded, due to high correlations.

SSDI 0378-4371(94)00197-9

In the single spin flip algorithm the individual spins are of interest; in the cluster algorithm bonds between neighbouring spins are of interest. Fortuin and Kasteleyn [1] showed that it is possible to rewrite the traditional partition function of the Ising model in single spin representation to get a partition function in cluster representation. Of the two possible dynamics, we use the Swendsen-Wang cluster algorithm [3]. Adjacent parallel spins are linked with probability $P = 1 - \exp(-J/kT)$, then they are flipped together with probability of 0.5. Via this long range dynamics, the algorithm avoids critcal slowing, thereby saving CPU-time.

Recently Tamayo [2] and Hackl [7] implemented two dimensional Swendsen-Wang algorithms efficiently on parallel computers. Adapting this program for three dimensions, several problems occured. The part of the program handling the communication had to be rewritten from scratch. In this paper, we present the two major parts of the program, the local label generation and the communication models.

2. Local label generation

2.1. Hoshen-Kopelman-Algorithm

The method of local label generation is based on the work of Hackl [7]. The Hoshen-Kopelman-Algorithm [4] was implemented for a 3D spin system. The local code was only a 3D-modification of the 2D-code in [7]. The only difference is that the number of neighboring elements increases from four to six.

Hoshen and Kopelman developed a multi-label algorithm in which it is possible to refer to a cluster with more than one label. In an additional one-dimensional array, the so-called owner list, it is noted that cluster l owns cluster k. That means that the label l is found in the k-th owner list entry. Such an entry is by our definition never negative, since the labels are never negative. But label l can also refer to another cluster. A cluster that refers to no other cluster and, therefore, belongs to itself, is called a root cluster.

After the local cluster process has come to an end, it is necessary to decide which cluster should be flipped. Bit 30 is used as a flag in the owner list, since bit 31 represents the minus sign (bits are numbered from 0 to 31). This flag 30 is cleared in all root clusters with a probability of 50%, since entries of the root clusters are negative. This information is transferred to all dominated clusters. This makes is possible to quickly determine whether an individual spin should be flipped, since every spin must only be checked in one owner list entry and not checked iteratively.

With this information it is easy to decide whether a certain spin should be flipped. First, the corresponding cluster label must be extracted from the label array. Second, the flipbit in the corresponding entry of the owner list must be checked.

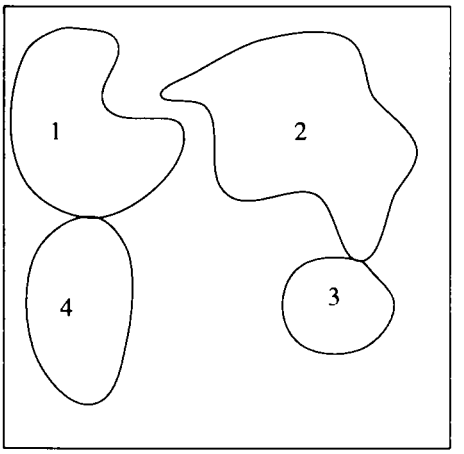

Fig. 1. System with four clusters.

2.2. Program realization

The 9 Byte Algorithm developed by Hackl in [7], page 22, is used. In this algorithm, 9 bytes of memory per site are needed. These 9 bytes are:

- 1 byte (`char`) represents the spin ('1' or '−1'). It is not useful to store the spins in bit-coded form, beause the memory saved is insignificant compared to the effort to compute the spin-coding.
- 4 bytes (`long`) represent the cluster labels for every element.
- 4 bytes (`long`) for the owner list, if the worst case scenario is assumed, so that every spin is its own cluster. [1] Owner list entries that are greater than or equal to the lowest local label indicate that the corresponding cluster belongs to the cluster with this label. This condition can be used to refer to the cluster sizes of the root clusters with negative numbers ($(-1)\cdot$cluster size).
- The first two data fields are as 3-dimensional arrays, whereas the owner list implemented is a one-dimensional array.
- The use of a sentinel (0-spins to mark the boundaries) was avoided to save storage.
- The program was implemented in C. Code written in C on the Intel Paragon that doesn't contain much floating-point operations is marginally faster than a comparable Fortran-program.
- We use periodic boundary conditions.

The sites are always processed in a pre-determined order. One starts with $(0,0,0)$ and goes layer by layer (z-axis) through the system (for the definition of the coordinate axis see Fig. 1). The layers are processed line by line (y-axis). It is therefore better to define the array in the form

[1] At $T < \infty$, the necessary length of the list can be found empirically by running the program. As the largest runs of the code were not troubled by storage shortage, but by lack of CPU time, this is not absolutely necessary

variablename[z][y][x].

If the size of the local system is lx, ly, lz, then an array in the form of `char spin [lz][ly][lx]` is used. No sentinel is used: As the number of dimensions increases, e.g. zero-spins at the boundaries would take an inordinate amount of memory compared to the other spins.

The choice of a label is made according to whether a bond exists to a neighboring element. A bond exists when the neighboring spins are parallel and $p \leq 1 - e^{-2J/kT}$, p is created by the random number generator. In general, there are 4 possiblities.

- no bond
- bond in one direction
- bond in two directions
- bond in three directions

If no bond exists, a new cluster is created. The label counter is then raised by 1. The new cluster refers to this number. The corresponding owner list entry is set to -1, since every new cluster belongs to itself and contains one spin.

If a bond exists in one direction, then the cluster label of the neighboring element is taken. The cluster labels are always the root labels. The entry in the owner list for this cluster is also lowered by 1, since the cluster has grown in size by one spin.

If bonds exist in 2 directions, it is necessary to test whether the bonds indicate two different clusters (root clusters). If both cluster labels are the same, then the processing is identical to the situation in which only one bond exists. The second bond brings no additional information. The dominant cluster label is chosen. The owner list entry of the dominated cluster is added to the owner list entry of the dominant cluster. The the owner list entry of the dominant cluster is lowered by 1, the entry in the owner list of the dominated cluster indicates the new root cluster.

The same is true for bonds in three directions. Instead of one dominated cluster, two dominated clusters exist. The process is otherwise anologous to the previous situation in which bonds in two directions exist.

The label that was generated after this fashion is stored in the label array.

The boundary spins do not check all directions. Since no zero-spins exist in the array as sentinels, it is necessary that the program is able recognize the boundaries via the loop indizes. So superfluous `if` tests can be avoided.

After the local cluster process has come to an end, it is necessary to decide which cluster should be flipped. Bit 30 is used as a flag in the owner list, since bit 31 represents the minus sign. To realize the cluster flips, flag 30 is cleared in all root clusters with a probability of 50%, since entries of the root clusters are negative. This information is transferred to all dominated clusters. This makes is possible to quickly determine whether an individual spin should be flipped, since every spin must only be checked in one owner list entry and not checked iteratively.

With this information it is easy to decide whether a certain spin should be flipped. First, the corresponding cluster label must be extracted from the label array. Second, the flipbit in the corresponding entry of the owner list must be checked.

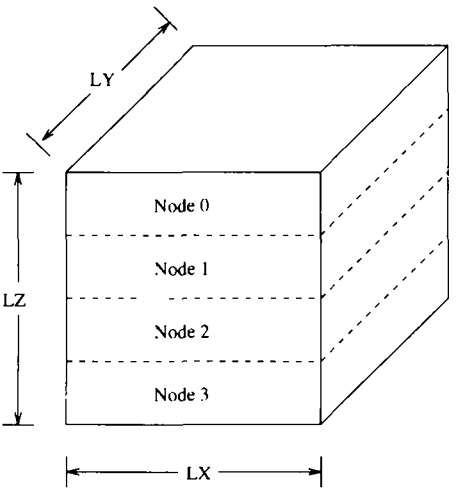

Fig. 2. Layer distribution of the nodes.

3. Communication models

Our parallelisation is realized by geometric parallelisation, the spin arrays are distributed among the nodes. This would be a geometric parallelisation. One problem that arose in programming the parallel subroutines was that the growing number of dimensions caused the number of boundary spins to grow disproportionately to the number of nodes. It became apparent that it was only sensible to use host models with 16 nodes, since a number of nodes greater than 16 results in the host arrays becoming larger that the local arrays. With more than 16 nodes, the speed of the program slowed dramatically. For the sake of completeness, this model is presented in spite of this problem.

3.1. Host models

The host model (Fig. 3) is basically a globalisation of the Hoshen-Kopelman-Algorithm (see section2.1). Global data are sent to an individual node, the host node, and processed there. This node uses a host owner list. The simplest example of this model is the strip model (Fig. 2).

The diagramms are presented in two dimensional form for the sake of simplicity. The flip bit is ignored in the owner list.

3.1.1. Strip model

For this model, the spin array is divided into layers (see Fig. 2); the individual layers are assigned to different nodes. Every node processes only one part of the system.

Every node is also responsible for the processing of the boundary spins in x and y directions. Each node also shares two boundary areas with the neighboring nodes.

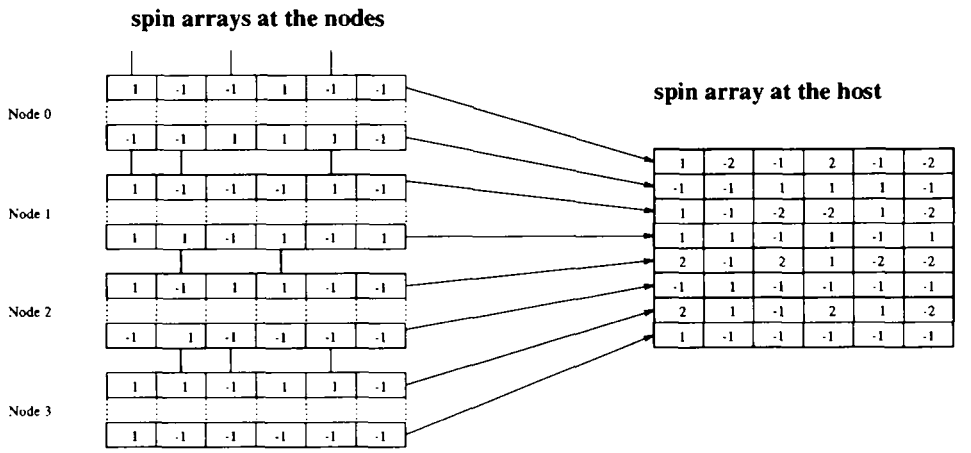

Fig. 3. Transfer of the boundary spins to the host.

Node 0 also acts as the host. Node 0 must connect the clusters that extend beyond the local boundaries. For this purpose, the following information is needed:

(i) where bonds exist beyond the boundaries of the nodes,

(ii) to which clusters these spins belong.

One simple way to do this would be to send all boundary spins to the host, which would store them in an array of the size $lx \times ly \times 2 \cdot$ (number of nodes), since $2 \times lx \times ly$ is the size of the boundary area. The presence of bonds would not yet be known. The host could proceed as in the local section of the program; for all host spins, the probability of a bond would be calculated. That would take too much time and would force the other nodes to wait.

It is better to mark the boundary spins before sending them to the host. Every node then evaluates its upper boundary spins for the existence of bonds. If no bonds exist, then the corresponding spin is multiplied by two before it is sent. The host must then test for equality, since the boundary spins take the values -2, -1, 1 and 2, across the node boundaries(see Fig. 3).

The handling of cluster label is more difficult. If the local labels were taken, then there would be several clusters with the same label. This would make the determination of the dominating cluster impossible.

The simplest method is to choose different labels from the start. Assuming that there were k spins at each node, then the cluster labels at node 0 would be $1 \ldots k$, and at node 1 $k+1 \ldots 2k$, etc. The problem would be that the label array and the owner list would be connected to each other. The complete owner list would have to be transferred to the host, whereby the memory limits on node 0 would soon be reached.

The better method is to create individual boundary labels. If l is the number of spins on a boundary area, then all local clusters start at $2l+1$. The first $2l$ numbers are left free for the boundary labels. The local root clusters on the boundary area are transferred to these boundary clusters. Only the first $2l$ elements of the owner list must be sent to

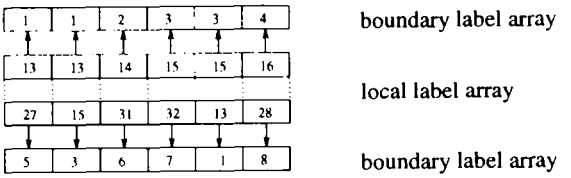

Fig. 4. Preparation of the boundary labels.

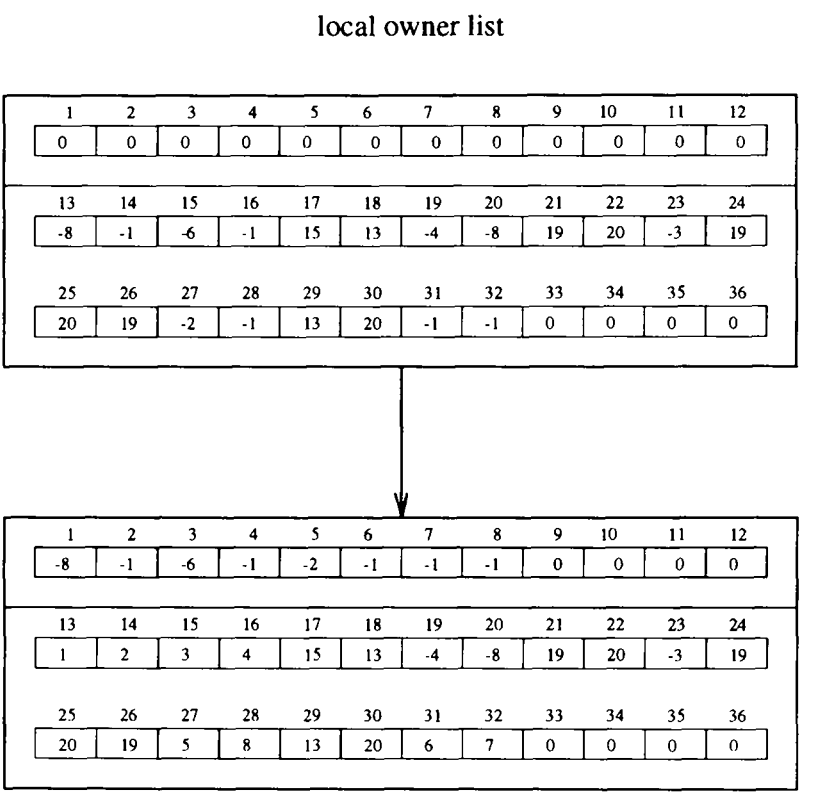

Fig. 5. Preparation of the local owner list.

the host (see Fig. 4).

Additionally, the owner list must be broadened (see Fig. 5), whereby the old local clusters indicate the new boundary clusters. In the area of the boundary labels, the owner list contains only root clusters.

The boundary labels (see Fig. 6) and the corresponding section of the owner list (see Fig. 7) are now sent to the host and given a $2l \cdot$ (node number) offset, since $2l \cdot$ (node number) is the size of the total boundary area. Through this process, the labels are made unambiguous.

The clusters must be connected to each other at the host using the same rules that are used for the local routines. Finally, the current owner lists must be sent back to the

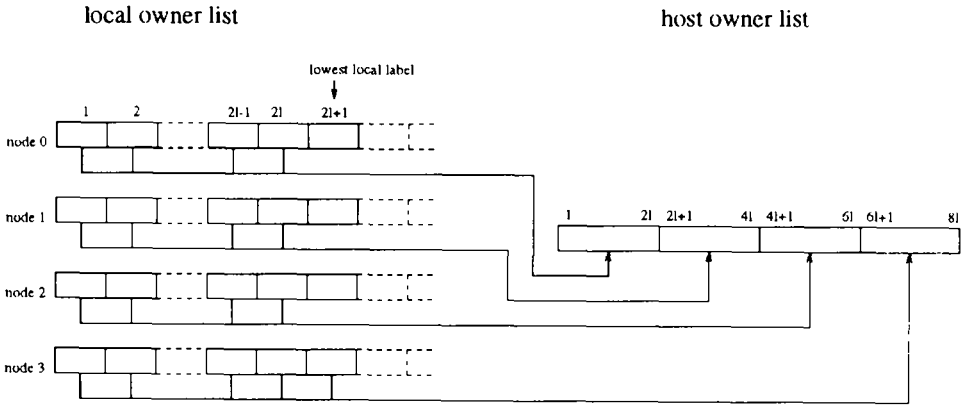

Fig. 6. Transfer of the boundary labels to the host.

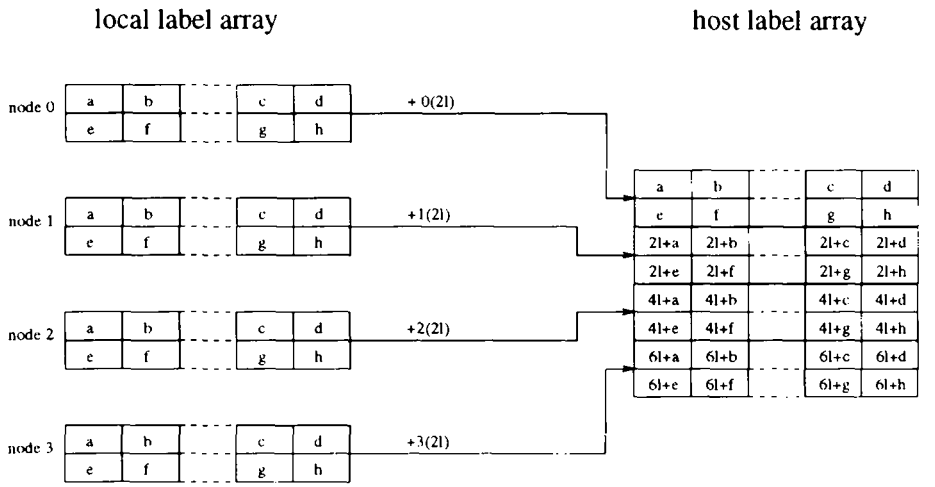

Fig. 7. Transfer of the owner list to the host.

individual nodes, where the flip bits are also transferred.

In keeping with this method, the host must reserve memory space for $lx \times ly \times 2 \cdot$ (number of nodes) spins. The host must also reserve memory for the same number of labels and owners. All other nodes do not make use of this reserved memory space,since the same code is used on all nodes.

The memory space for the host labels can be saved by transferring the information to the owner list. This method corresponds to Hackl's 5 byte algorithm [7], page 24. This reduces the amount of memory needed by a little less than 50%. Every spin position is represented by an owner list entry. The upper boundary labels are transferred to the elements $1, \ldots, l$ and the lower boundary labels are transferred to the elements $l+1, \ldots, 2l$ of the owner list (see Fig. 8).

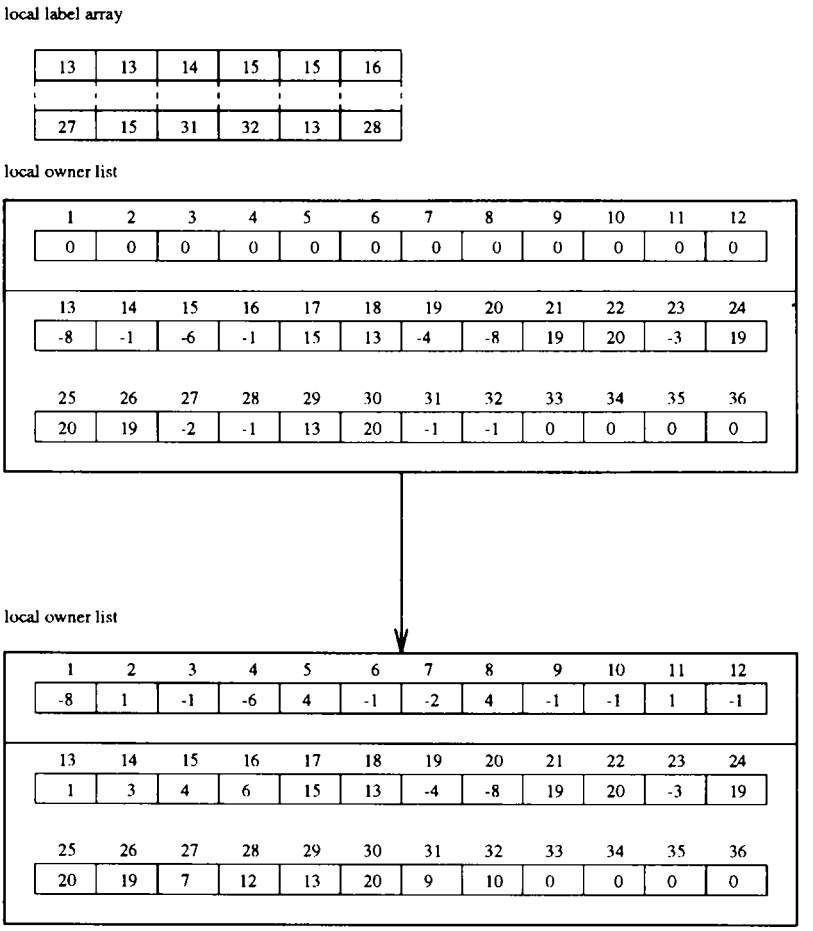

Fig. 8. Merging of the local boundary label array and the owner list.

3.1.2. Cube model

In the cube model, the spin array is cut into cubes and distributed among the nodes (see Fig. 9).

The result is a model with six instead of two boundary areas. It can be shown that beyond 8 nodes, the number of boundary spins is smaller than in the strip model.

If the spin array is cubic with boundary length L and n is the number of the computer nodes, then the number of boundary spins which must be dealt with at the host is

$$\text{Strip: } L^2 \cdot 2 \cdot n. \tag{2}$$

For an even cubic distribution ($n = x^3$ with $x \in \{2,3,4,\ldots\}$), the number of boundary spins is

$$\text{Cube: } \left(\frac{L}{n^{1/3}}\right)^2 \cdot 2 \cdot n \cdot 3 = 6 \cdot L^2 \cdot n^{1/3} \tag{3}$$

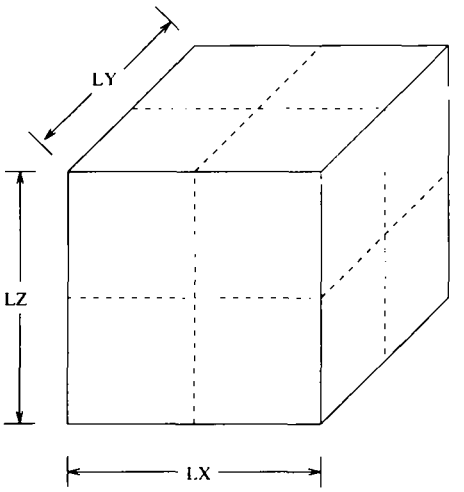

Fig. 9. Cubic node distribution.

It can be shown, that when 8 or more nodes are used, the number of boundary spins in the cube model is smaller than the number needed in the strip model, independent from the system size L. Eight nodes represent the smallest truly three dimensional parallelization. Their distribution in the $x : y : z$ directions is $2 : 2 : 2$.

For a $(z/2)$ cubic distribution ($2n = x^3$ with $x \in \{2,3,4,\ldots\}$), the number of boundary spins is

$$\text{Cube}(z/2)\text{:}\ \left\{\left(\frac{L}{(2n)^{1/3}}\right)^2 + 2\cdot\frac{L}{(2n)^{1/3}}\cdot\frac{L}{\frac{(2n)^{1/3}}{2}}\right\}\cdot 2\cdot n = 5\cdot(2n)^{1/3}\cdot L^2. \quad (4)$$

Here it can be shown that beyond 32 nodes, the number of boundary spins becomes smaller than in the strip model. A system with 32 nodes represents the smallest three dimensional cube $(z/2)$-distribution: $4 : 4 : 2$.

For a $(x/2, y/2)$ cubic distribution ($4n = x^3$ with $x \in \{1,2,3,\ldots\}$), the number of boundary spins is

$$\text{Cube}(x/2, y/2)\text{:}\ \left\{\left(\frac{L}{\frac{(4n)^{1/3}}{2}}\right) + 2\cdot\frac{L}{(4n)^{1/3}}\,\frac{L}{\frac{(4n)^{1/3}}{2}}\right\}\cdot 2\cdot n = 4\cdot(4n)^{1/3}\cdot L^2. \quad (5)$$

In this case the minimal node number is 16, which represents a $2 : 2 : 4$ distribution.

3.2. Relaxation model

Flanigan and Tamayo's [2] relaxation model offers an alternative solution to this problem. In this model, no host is used. Communication is only carried on with the logical neighbor. These often differ greatly from physical neighbors which are connected through communication lines. (see Fig. 10). The cubic model is used to minimize the necessary communication.

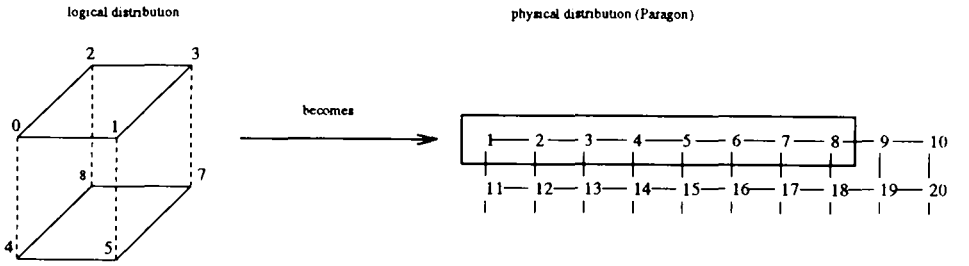

Fig. 10. Map from logical to physical distribution.

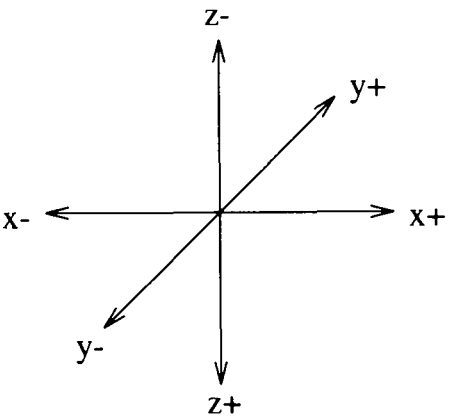

Fig. 11. Terminology of the three-dimensional directions.

Since it is important in this communication that the relative position of the neighboring elements is clear, a special terminology of directions (see Fig. 11) is introduced.

3.2.1. Basic principle

The relaxation is divided in three steps: the preparation phase, the relaxation phase, and the evaluation phase.

Preparation phase: In this phase, the system is prepared for the relaxation. Two tasks must be completed:

(i) Every node must know where bonds exist to its neighbors.

(ii) The local boundary label must be prepared.

The bonds are merged into the boundary spins, analogously to Section 3.1.1. In each dimension, the bonds must be determined in just one direction, since the bonds from x to y must be the same as from y to x (see Fig. 12).

These spins are then sent to the logical neighbor in all six direction (see Fig. 13). Every node then knows where its clusters have bonds to clusters of other nodes. These do not change during the relaxation phase.

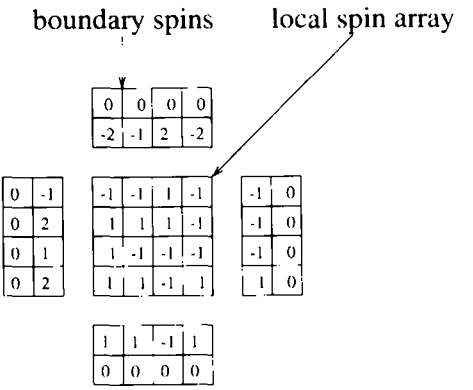

Fig. 12. Preparation of the boundary spins for relaxation.

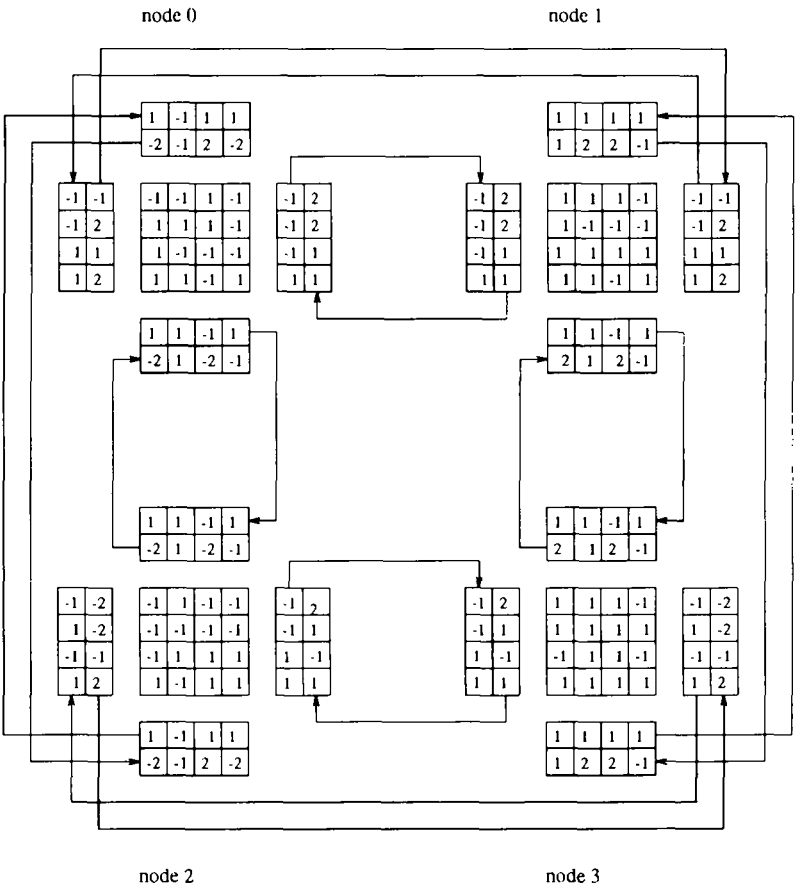

Fig. 13. Communication of the boundary spins with the neighbours (the third dimension is not shown).

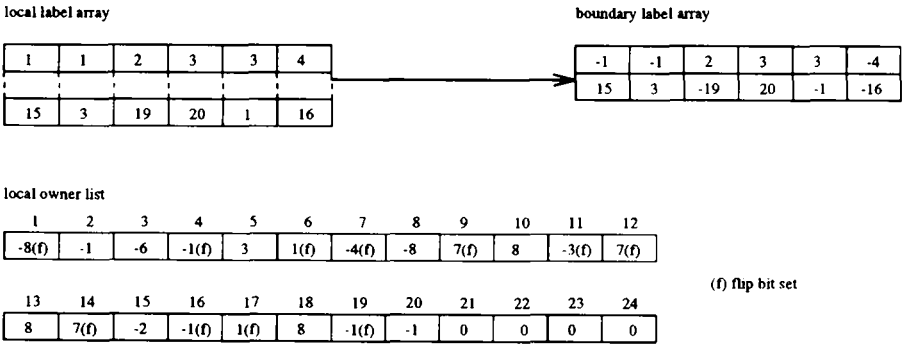

Fig. 14. Preparation of the boundary spins.

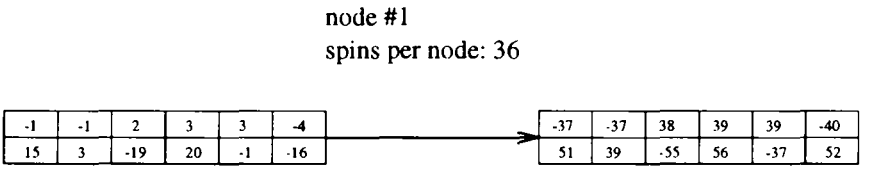

Fig. 15. Simple scaling of the boundary spins.

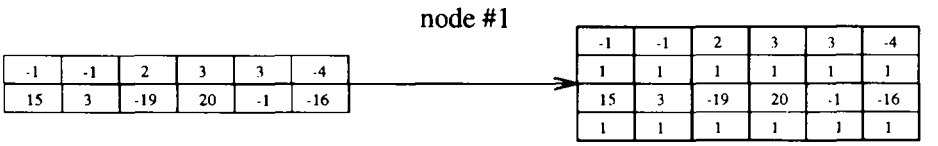

Fig. 16. Scaling of the boundary spins with `struct`.

The boundary labels consist only of root cluster labels. Only the flip information is added. Since the number of boundary clusters is by definition positive, the flip bit can be represented by a minus sign (see Fig. 14). The smallest possible local label is 1, since -0 does not exist as an integer.

As in the host model, the boundary labels must now be made unambiguous. The simplest method to achieve this goal is to add an offset of the number of the local spins to the absolute value of the boundary label. The sign must then be restored (see Fig. 15).

This method works up to an boundary length of the entire system of 1024 spins. For larger systems, the boundary label must be expanded. The most appropriate way to do this is to mark the boundary label so that its home node is apparent. To minimize the communication needed, both data are summarized in a `struct new={long label; short node;}` command (see Fig. 16).

Since no global owner list exists in this model, it is not possible to determine the size of the clusters spanning over more than one node.

Relaxation phase: The relaxation phase consists of at least one relaxation cycle. This cycle is repeated until nothing has changed in the system since the last cycle. This is necessary since the information in each node can only be passed in every cycle to one further node.

The relaxation cycle consists of six relaxation units; each unit represents one direction. The order of these units within a cycle can be arbitrarily chosen. It is only important that the order be consistent for all nodes.

A relaxation unit always follows the same series of steps; this series of steps runs simultaneously on all nodes. A relaxation unit processes only the boundary label. The local labels are not altered at this time.

Suppose that the relaxation takes place on the $(x-)$ boundary. The following steps would take place:

(i) The local $(x+)$ boundary label would be sent to the neigboring node, which lies opposite to the $(x+)$ boundary.

(ii) The boundary label would be received from the neigboring node, which lies opposite to the $(x-)$ boundary.

(iii) The spin array of the $(x-)$ boundary is processed element by element.
  (a) The spin is tested for the existence of a bond (spin equality) with its neighbor.
  (b) If a bond exists, then the label received from the neighbor is tested for dominance. The element with the smallest absolute value, since the minus sign indicates the flip bit, dominates.
  (c) If the label of the neighbor is dominant, then the dominated label is replaced by the dominant label throughout the entire node. Not only the labels on the $(x-)$ boundary can be changed, but also all labels on all other boundaries!

Example: For the sake of simplicity, the process of finding the root labels over several processors for a two dimensional array is explained in one dimension.

First, all nodes send their boundary labels in one direction. In this diagram, all boundary labels are sent upwards. It is important to note that node three is above node 0 (see Fig. 17).

All nodes now assume the dominant label. This occurs in Fig. 18 in node 3. All boundary labels of 31 and 32 are replaced with the label 2.

The updated boundary labels are now sent downwards and processed in the same way (see Fig. 19).

At the end of this unit, clusters 1 and 3 have migrated from node 0 to node 1 and cluster 11 has migrated from node 1 to node 2.

This ends the first relaxation cycle. After three further cycles (see Figs. 20–22), the information has been distributed to the entire system. In order to simplify the diagram, the boundary labels replace the local labels.

After the second relaxation cycle has ended (see Fig. 20), the entire picture is dominated by clusters with a home node of 0.

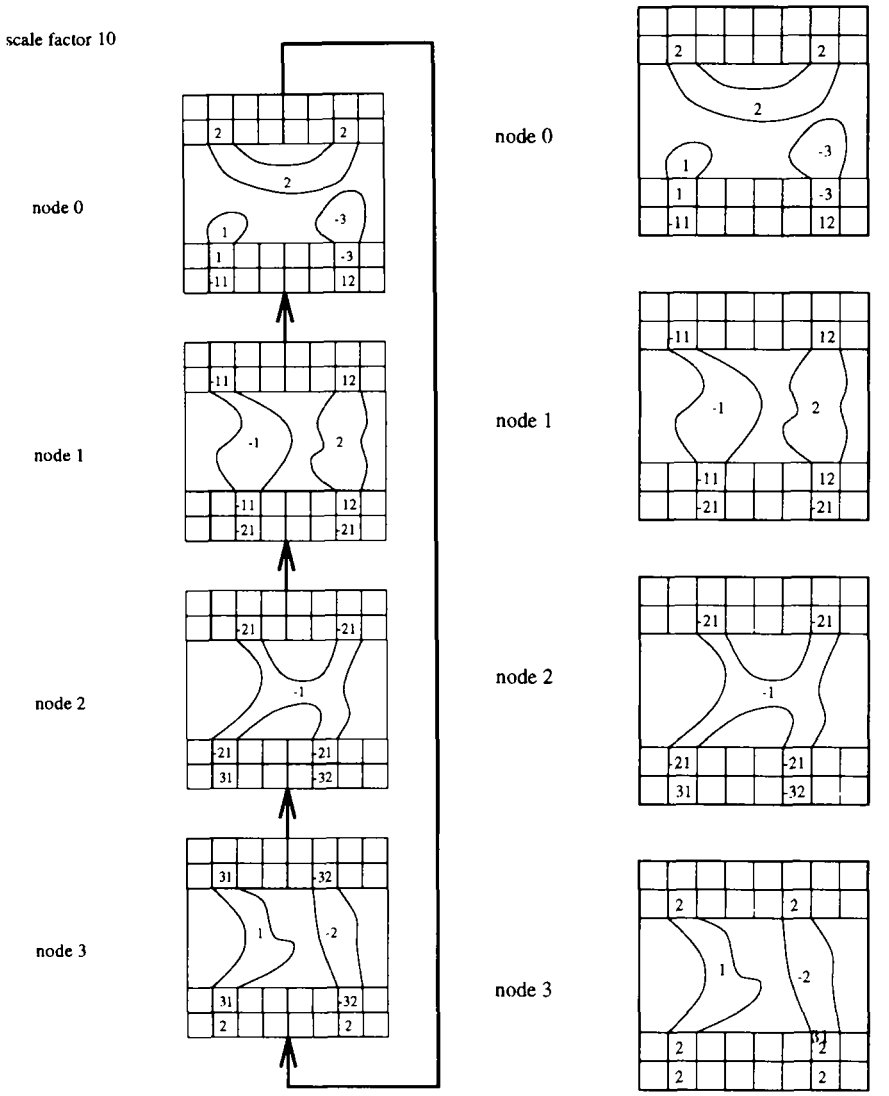

Fig. 17 Fig. 18

Fig. 17. The first relaxation cycle: sending the labels upward.

Fig. 18. The first relaxation cycle: processing of the labels.

The cluster has now distributed itself over the entire system. Inspite of this, another relaxation cycle must be started, since the system does not yet know that the relaxation phase is finished. It is only clear at the end of this cycle that no new changes have occurred.

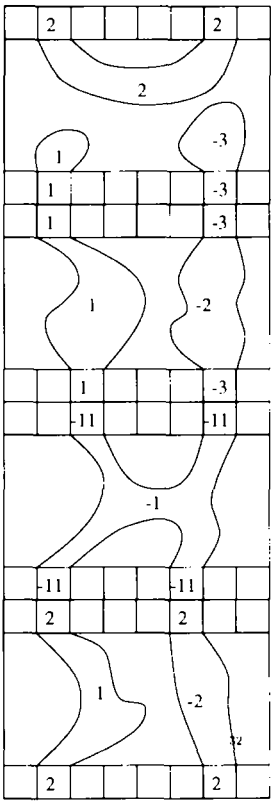

Fig. 19. First relaxation cycle: direction of relaxation is downward.

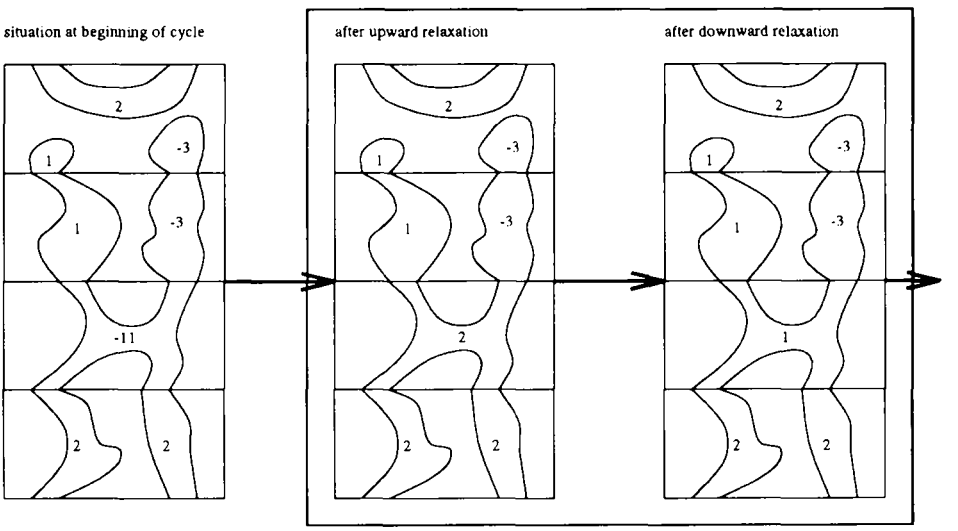

Fig. 20. Second relaxation cycle.

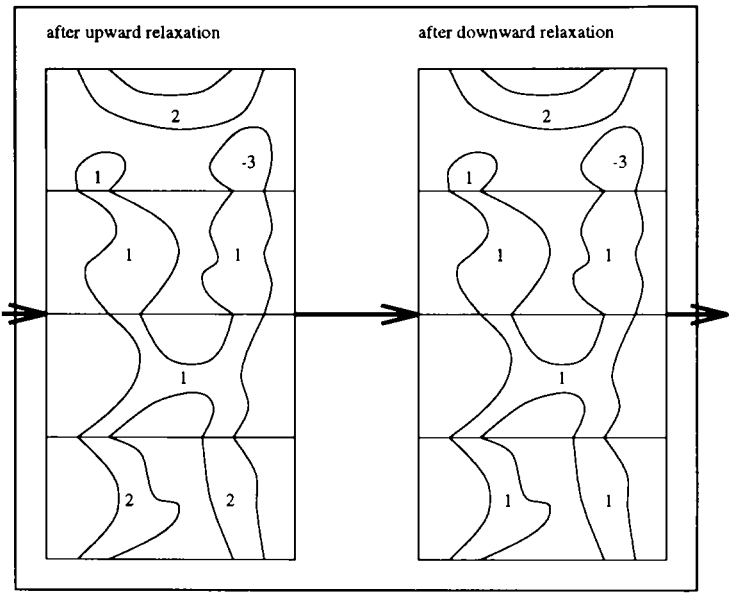

Fig. 21. Third relaxation cycle.

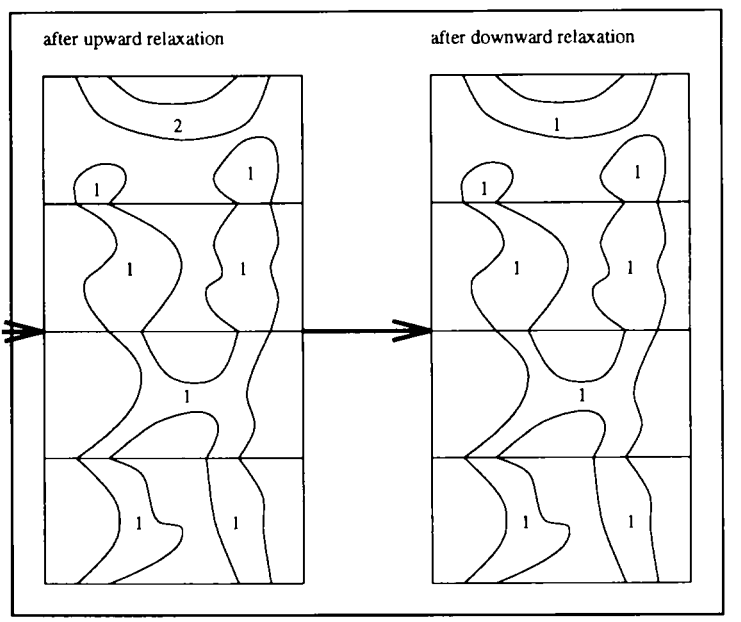

Fig. 22. Fourth relaxation cycle.

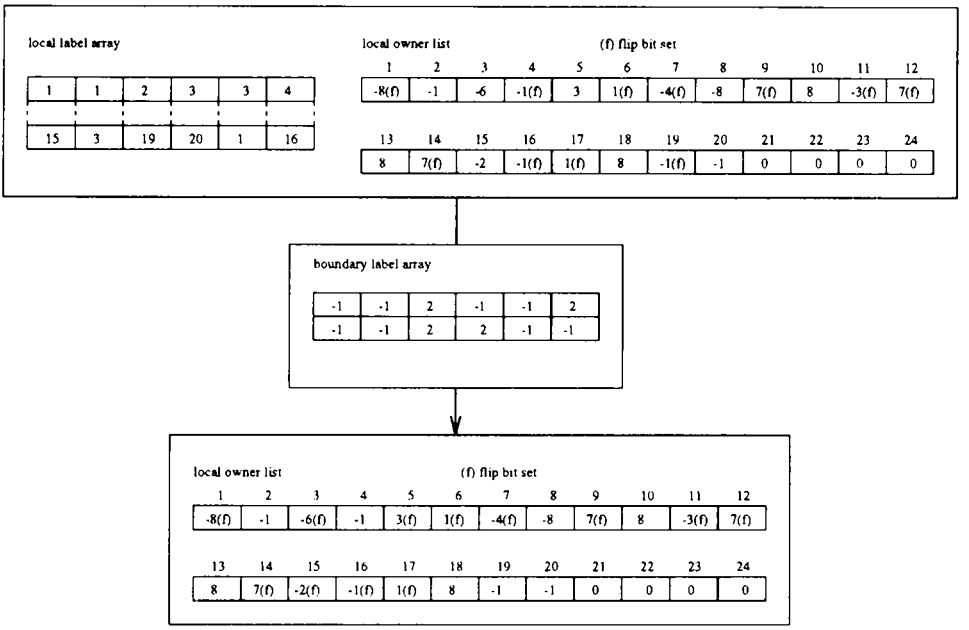

Fig. 23. Evaluation of the boundary spins.

Evaluation phase: At the end of the relaxation, the results must be translated into the local system. Since every boundary label is unambiguously assigned to a certain label, the flip information can be transferred to the local owner list. This information must then be transferred to the local subclusters (see Fig. 23).

3.2.2. *Optimization*

During the relaxation phase, it might be of interest to know:

(i) whether too much information is transferred,

(ii) whether too many boundary labels are updated.

To answer these question, it is necessary to employ channel reduction and a net list.

Channel reduction: It is unimportant for the relaxation algorithm whether the boundary labels consist of clusters with or without a bond to the next node. The boundary labels without a bond are only decoration, since the changes in these boundary labels have no effect on the neighboring labels. For this reason, boundary labels are only generated where a bond to a neighbour exists. The number of transferred boundary labels is reduced through this process. The transferred labels could be compared with information channels between the individual neighbours. The term *channel reduction* is used for this reason. For an boundary label it is not important what exactly its position is in the local system. Since the neighbor behaves in an identical way, both create the same number of boundary labels (see Fig. 24). For example, the labels of the third spin in the first row are no longer tested, but rather the labels of the first bond between the neighboring

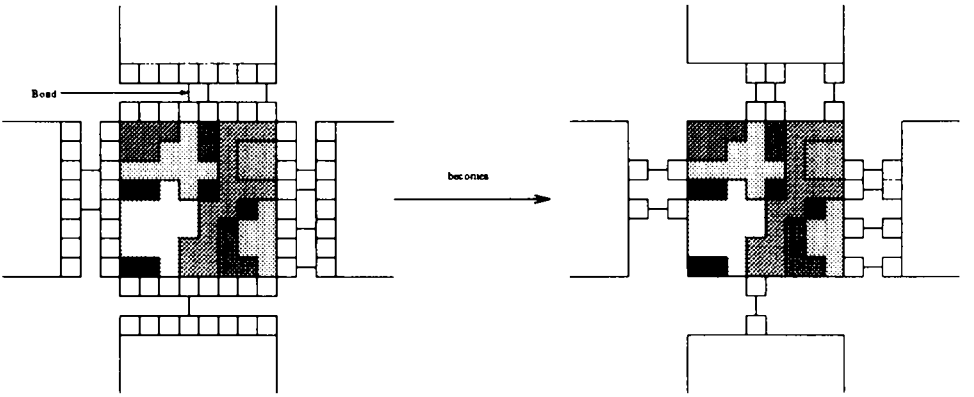

Fig. 24. Channel reduction.

nodes. The evaluation for the existence of a bond is no longer necessary, since only bonds are transferred.

The evaluation works inversely in an analogous process. The spin array is searched for bonds. Where bonds exist, the next boundary label is taken.

In this way it is possible to reduce the number of transferred labels by at least 50% (at $T = 0$). As the temperature rises, at T_{crit} we save 73% of the number of labels, since the probability to form bonds decreases.

Net list: The process of testing all boundary labels before changing a local boundary label is actually a brute force approach. It would be rare that so many boundary labels would be connected by a local cluster. The number of boundary labels would also be reduced through channel reduction. A Hoshen-Kopelman-Algorithm is not appropriate, since it would use up too much memory. A different approach is necessary.

Every local boundary label is given two additional pieces of information:

(i) the first boundary label that belongs to the same local cluster,

(ii) the next boundary label that belongs to the same local cluster.

If the boundary label is changed, then it is only necessary to check in the net list [start] (see Fig. 25) to see what the first boundary label of this local cluster is.

If this cluster consists of one boundary label or if it is the first label of a row, then this label indicates itself. The new label is copied to this entry of the boundary label array. If further boundary labels of this cluster exist, they are successively changed. A zero in this list means that the last entry for this cluster has been reached. In order to be able to achieve this optimization, all boundary labels must be placed in a single one dimensional array, since only the bonds are numbered.

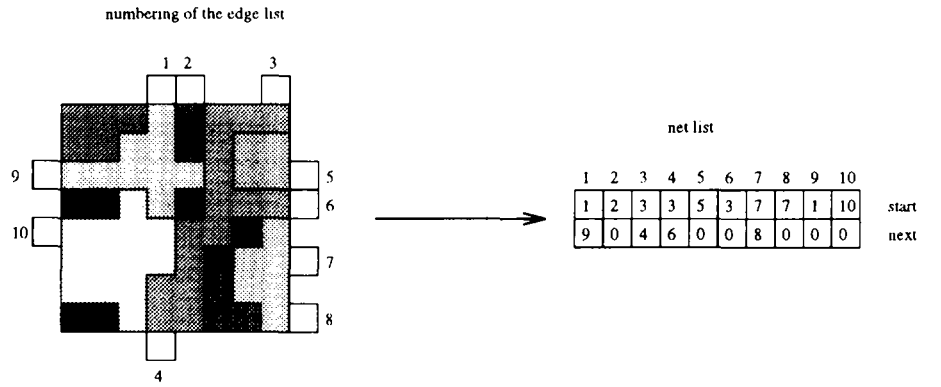

Fig. 25. The net list.

4. Benchmarking

We used the Intel Paragon at the Zentralinstiut für Mathematik at the KFA Jülich with 140 nodes of i860 Processors. The time used by the local part of the program can be described by

$$t_{loc} = a_1 n + a_2 n \ln(n). \tag{6}$$

n denotes the size of the system. The part $a_1 n$ considers the label generation for every individual spin. The iterative search in the owner list is represented by $a_2 n \ln(n)$. The running time of the part of the communication part is more complex. Empirically we could fit the time to

$$t_{com} = a_1 + a_2 l + a_3 n \ln(n) + a_4 l \ln(l) + \ldots, \tag{7}$$

but there may still be some terms left out. l denotes the total boundary length and n the system size. a_1 is something like a setup time. $a_2 l$ expresses that the time needed increases with the amount of boundary labels which have to be send. $a_3 n \ln(n)$ is forced by the owner list and $a_4 l \ln(l)$ indicates the net list.

In table 1, we give the effciencies and update-times for various processor numbers and system sizes. The efficiency η was computed as $t_{local}/(t_{local} + t_{global})$, t_{local} and t_{global} were measured on node 0 via the `mclock`-function.

The efficiencies of the 140-processor systems are also plotted in Fig. 26.

The trouble with table 1 is, that though the efficiency increases, so does the necessary amount of time /spinflip. Several competing effects are at work: On the one hand, to realize periodic boundary conditions, the relative amount of CPU time increases (against all expectations, but consistent with the behaviour of our scalar code on a single node). On the other hand, the optimal System lengh is 280^3, because the message length fits exactly into the communication buffers. The effects could not be properly investigated, because too few system sizes can be run: non-cubic local systems exhibit different runtime-behaviour in the local routines due to different boundary conditions.

Table 1

	N_{proc}	L	$\approx$	η in %	ns/spinflip
	64	64^3	2.6×10^5	18	252
	128	96^3	8.8×10^5	20	115
	128	128^3	2.1×10^6	78	89
*	140	140^3	2.7×10^6	20	109
*	140	280^3	2.2×10^7	26	88
*	140	420^3	7.4×10^7	87	109
*	140	560^3	1.8×10^8	92	125

The system-sizes which are marked with a * were measured at the critical temperature, the other measurements were made over a range of temperature symmetrically to the critical temperature.

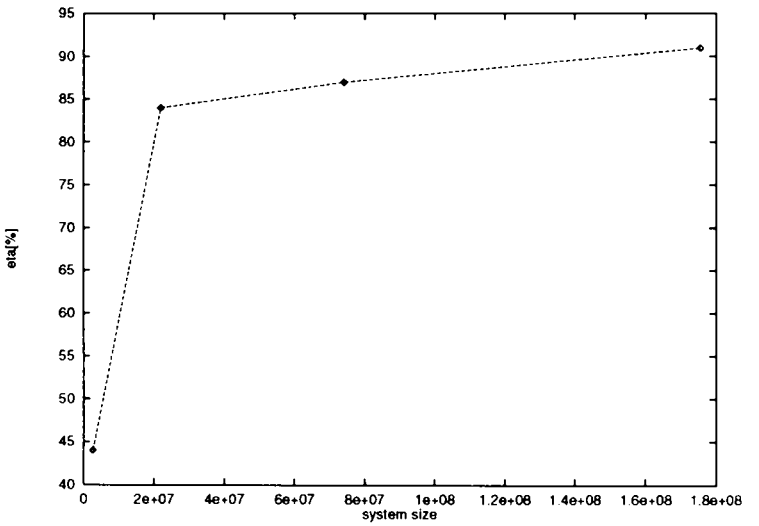

Fig. 26. Efficiency versus system sizes on 140 nodes. The line is guideline for the eye only.

Comparisons to 2D-codes are also very problematic, because the necessary amount for communication is much higher in 3D. Therefore, the communication routines had to be much better optimized to avoid the communication of redundant data (see channel reduction in Section 3).

5. Conclusion

We have shown that it is possible to successfully parallize a three dimensional Swendsen-Wang algorithm. When rising the number of dimensions a more and more sophisticated communication pattern will be needed. This is important because of the increasing amount of communication.

The best single spin update time we achieved was 88ns on 140 nodes on an Intel

Paragon. The largest system we simulated had an edge length of 560. This size represents not the largest possible system, but the size we could process in a reasonable amount of time.

The differences to the 2D-case were, that no efficient host-node model was possible even theoretically due to storage requirements, so the relaxation algorithm was the method of choice. Apart from channel reduction, no further tricks compared to the 2D-algorithms had to be employed to obtain high efficiency.

Acknowledgement

We thank the Zentralinstitut für Mathematik at the Forschungszentrum Jülich for providing the necessary Computer-time on the Intel Paragon and J. Docter for support.

References

[1] C.M. Fortuin and P.W. Kasteleyn, Physica 57 (1972) 536.
[2] M. Flanigan and P. Tamayo, Int. J. Mod. Phys. C 3 (1992) 1235.
[3] R. Swendsen, J.S. Wang, Phys. Rev. Lett. 58 (1987) 86.
[4] J. Hoshen and R. Kopelman, Phys. Rev. B 14 (1976) 3438.
[5] N. Ito, PhD thesis, University of Tokyo, December 1990.
[6] C. Münkel, D.W. Heermann, J. Adler, M. Gofman and D. Stauffer, Physica A 193 (1993) 540.
[7] R. Hackl, Diplomarbeit, Institut Physik II – Festkörperphysik der Universität Regensburg, January 1994.