International Journal of Modern Physics C, Vol. 4, No. 6 (1993) 1117–1130

PARALLELIZATION OF THE 2D SWENDSEN–WANG ALGORITHM

R.HACKL[1], H.-G. MATUTTIS[1,2], J.M.SINGER[1],
TH.HUSSLEIN[1], I.MORGENSTERN[1]

[1] *Institut für Physik II, Universität Regensburg, Universitätsstraße 31*
93053 Regensburg, Germany
[2] *HLRZ, c/o KFA Jülich*
Postfach 1913, D-52428 Jülich, Germany

We implemented a parallel Swendsen–Wang algorithm for a 2D Ising system without magnetization in a host–node programming model. The simulations were performed on the Intel Hypercube IPSC/860. Our maximum number of updates/s on 32 nodes ist three times as high as in the implementation by Stauffer and Kertész on the same machine. With 32 processors we reach half the speed of the simulations by Tamayo and Flanigan on 256 nodes of a CM5. We discuss the non–equilibrium relaxation for the energy and the magnetization.

Keywords: Cluster Algorithms, Swendsen Wang, Parallelization, Host–node–modell

1. Basics

In this paper, we use the Swendsen–Wang algorithm [Wan90] in combination with the Ising model. The Ising–model with the Hamiltonian

$$H = -J \sum_{NN} s_i s_j , \quad s_i, s_j \in \{\pm 1\}$$

is analytically solved in 2 dimensions. Therefore it qualifies as a testing ground for numerical algorithms.

Various algorithms have been proposed and implemented. The single spin update chooses a spin s_i satisfying detailed balance and flips this spin with a temperature dependent probability, e.g. with $P = \exp(2Js_i \sum_j s_j/T)$.

This algorithm is very well suited for parallel implementation [Hee91]. It requires only the communication of nearest neighbor spins on boundaries after domain decomposition of the physical spin–system. One drawback is the critical slowing down, especially at the critical temperature, so that many configurations have to be discarded. The reason is that correlations of the size of the system occur within a dynamic that takes into account only the nearest neighbors.

A way to overcome critical slowing down is the use of cluster spin updates: Instead of wasting computer–power by independently updating correlated spins one forms and flips clusters of correlated spins. The Swendsen–Wang algorithm links

adjacent parallel spins with probability $P = 1 - \exp(-2J/T)$ and flips the resulting clusters with probability 0.5.

The advantage of the reduction of critical slowing down is reduced by costly searching strategies for the identification of the clusters. Recently it could be established [Ito93] that the cluster flips are only faster for systems with a linear dimension (in 2 and 3 D) that is well over several hundreds of spins, else an optimized bit–coded implementation of single–spin flips is faster in terms of CPU–time independent of the hardware.

2. "Design" of the Algorithm

The implementation took place on the basis of the following criteria. We do not claim that they are ultimate criteria in any respect.

Connectivity: In the Ising–system one is most interested in the behavior at T_c, where single clusters extend over the whole system. After the domain decompositioning of the physical system, the largest resulting cluster will stretch over many processors. It may be necessary to use the information located on many processors to attain the global information.

Simplicity: Complicated communication patterns, like cluster dependent communication should be avoided because they are hard to implement without dummy communication processes. Moreover this makes preliminary estimations of the necessary amount of communication rather complicated. As much as possible should be done locally on the nodes.

No scalable algorithms: Currently, poor men's parallel computers are connected workstations. The installed parallel machines usually have less than 100 processors. As an excess in processors is normally one's least concern in parallel computing, the algorithm was not meant to run on arbitrarily many processors.

Implementation: We chose Fortran77 with message passing calls, because the developing of the local kernels was possible on workstations. C was found to be as fast.

Statistics: SW-Codes are Monte–Carlo programs and therefore depend strongly on the number of sampled configurations. As a trivial parallelization is possible by running independent configurations on independent processors, we had no ambition to parallelize too tiny systems as a single node with 16 MB storage can handle up to 1500×1500 spins.

Behavior for large systems: The central point in parallelizing the Swendsen–Wang algorithms is that a system of size $L \times L$ is distributed on P processors with system–size $L \times L/P$. If a strip parallelization is chosen, the boundary–lengh L increases with the total system–size. Therefore, in treating the labels on the boundaries, algorithms that scale with L^2 must be avoided. For large P

the local CPU–time in treating the boundaries $\propto L^2$ would be larger than the local time for the local cluster algorithm $\propto L^2/P$. Fortunately, for the integer labels of the Ising–clusters one can find algorithms $\propto L$ without having to resort to devide–and–conquer algorithms, which cost ($\propto L \ln L$).

3. Performance Analysis

Speedup and efficiency depend very much on the implementation characteristics. We will give only the speed in updates/s, but no explicit efficiencies or speedups for the following reasons:

- In our case, with the speed of the algorithm still increasing for increasing P and the use of only up to 32 processors, we were not able to reliably devise the behavior of the efficiency in terms of a power law as there were not enough points to fit.
- The best speedup in otherwise equal setups can be attained with worst (= slowest) nodes or worst (= slowest) local algorithm. If one adds an empty do–loop that slows down the local node, one can enlarge the efficiency by slowing down the program, which is contradictory to the primary purpose of improving the efficiency.
- Separation of CPU–time–use of local algorithm and parallel overhang is often not so easy, as communication tasks have to run in the background. The profiling facilities very often fail to eliminate parallel overhead due to the operating system.
- For large systems only a "virtual" speedup can be measured, as the storage of a single node could not handle the system as a whole. Therefore, one has to run the systems on many nodes even if the efficiency is low.
- Different algorithms in different programming languages are implemented with different networks over different processors.

We compare the speed as one would do in the comparison of scalar machines via a benchmark which also doesn't take into account the "efficiency" in the use of functional units of a processor. In the following comparisons we treat the parallel machine and the algorithm as a unit, as at least the nodes of the CM5 and the Intel IPSC/860 are comparable in performance.

4. Our Implementation in Detail

The total system of $L_x \times L_y$ spins is layed out on P processors, so the local system contains $(L_x/P) \times L_y = l_x \times l_y$ spins.

As local algorithm we use a Hoshen–Kopelman type of algorithm. The global communication pattern is a host–node type of algorithm to limit the total communication to P sends and receives. The information about the size of the clusters

is already an inherent part of the local algorithms. Steps 1 to 4 and 8 to 9 are executed in parallel, whereas steps 5 to 7 take place only on the host.

1. The local labeling starts with label $2l_x + 1$.
2. Connected labels are joined by creating an owner–list, the lower of two labels becomes the owner. A positive entry n at the i-th array element indicates that the cluster with the label i is attached to the cluster labeled n, its "owner". A negative entry n' means that the cluster labeled i' has no owner and the number of spins belonging to the cluster and its attached clusters is $|n'|$.
3. Bit 30 of a label is set with probability 0.5. This indicates, whether the spins of the labels will be flipped or not. So the flipping of the clusters is determined before the labels are set, as opposed to the usual scalar algorithm, where this takes place after the labeling. To create the random number for this decision locally has turned out to be slightly more efficient than the determination on the host.
4. Labels on the boundaries are reduced to numbers from 1 to $2l_x$ with a unique offset on every node, so that the global algorithms use much smaller numbers than the maximal INTEGER*4.

---

5. Spins, labels and the owner–list of the node–boundaries are sent to the host (= node 0).
6. The clusters on the boundaries are joined with a kind of modified Hoshen–Kopelman.
7. The relabeled owner–list is sent back from the host to the nodes, the offset has to be taken into account properly so that local clusters get the right owner labels.

---

8. The flip information of the local owner–lists are modified on the nodes corresponding to the label changes of the global clusters.
9. The spins are flipped depending on bit 30 of their labels.

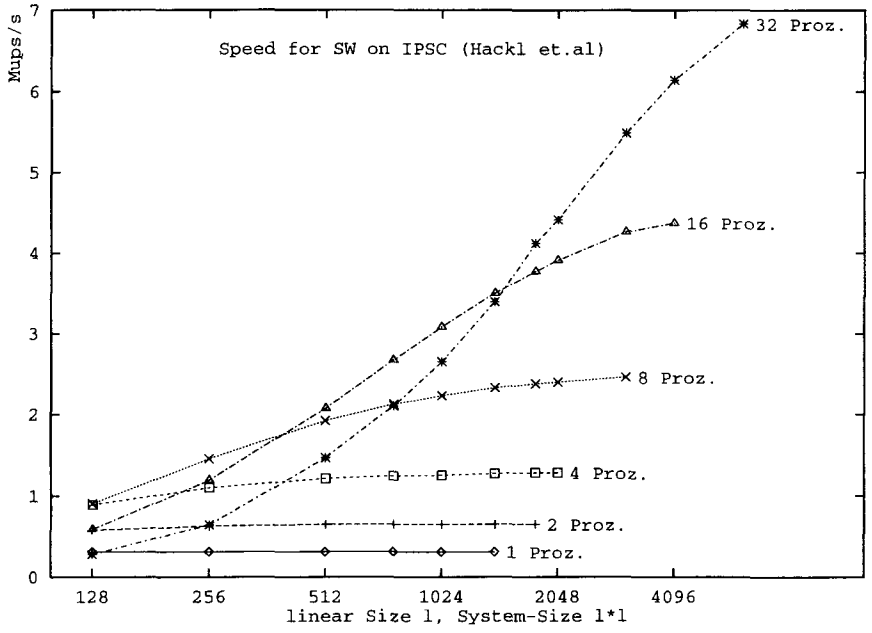
5. Some Implementations of SW in 2D

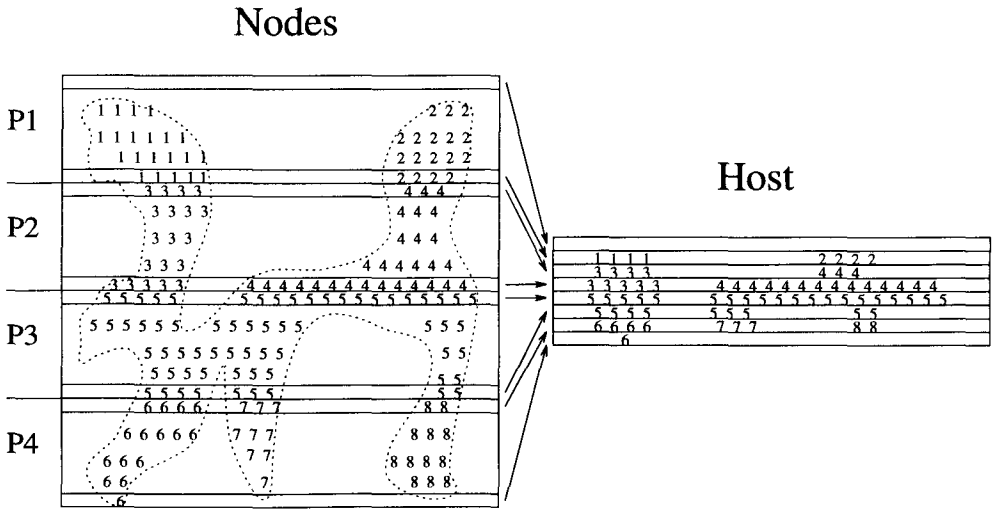
Advantages of the algorithms are indicated by a "+" in the following description, disadvantages with a "−". Of course, for serious implementations speed is a rather more important issue than the convenience of implementation.
The speed is always indicated for the simulation of the system at T_c in thermal equilibrium. Different average cluster sizes at different temperatures lead to different search depths in joining the clusters, so the speed of the algorithm is temperature–dependent.

5.1. *Hackl et al.*

Our algorithm was implemented on the Intel IPSC/860 via F77 with message–passing on a configuration with up to 32 nodes. The host-node programming model

used node Nr 0 as host, not the "front-end" processor SRM, an Intel PC386. Our implementation uses 9 byte/spin, 1 byte for the spin (up/down), 1 integer for the label and 1 integer for the owner. At T_c the actually used owner-list is only 0.24 the length of the label-list.





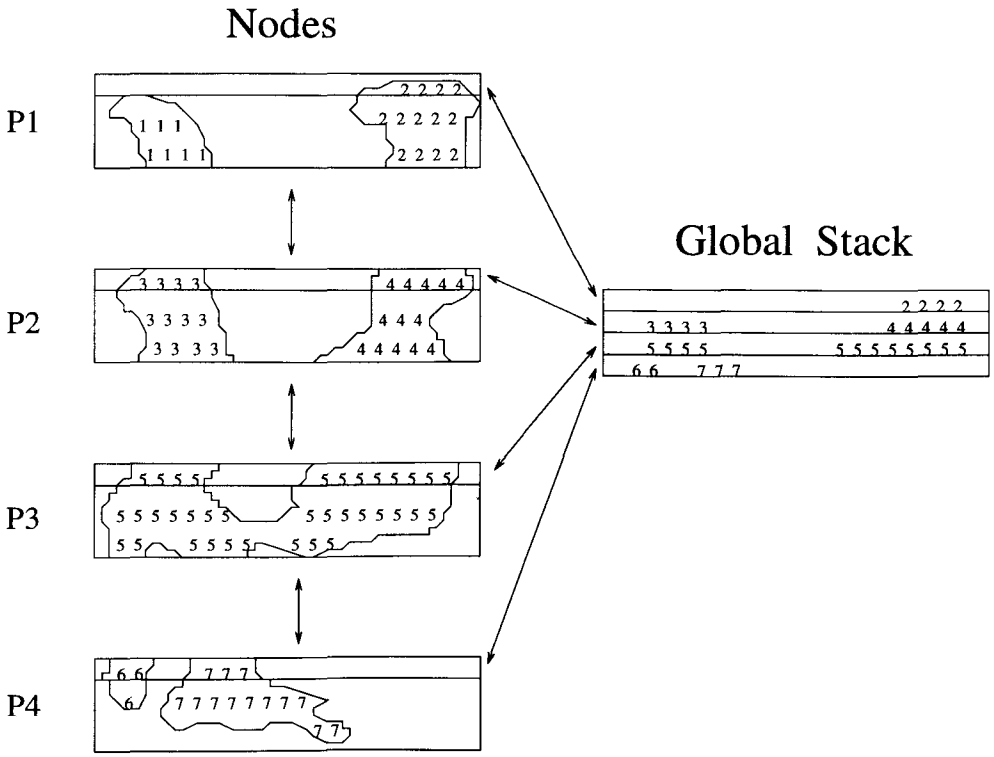
+ We use a very "simple" communication pattern.
+ The algorithm is still fast on many processors.
+ The information about global clusters is generated in each update step and causes nearly no overhang.

- The code is lengthy (11 pages in total, compared to only 3 pages for the scalar algorithm) and the implementation took a lot of fine-tuning to reach the current speed.
- The length of the boundary per processor increases with the number of nodes. The algorithm can not be expected to work efficiently for arbitrarily many processors.

The profilings indicated that the total algorithm was slowed down by the parallel overhead of the host, not by a gridlock due to the nodes sending on the network. As the host is the last processor to execute also its local tasks, we tried to eliminate this overhead by cyclic permutation of the host, i.e. in the first update, the 0-th node acted as a host, in the second update the 1-st node and so on. Although the algorithmic parallelization overhang was then shared by all the nodes, the speed of the algorithm increased only moderately for large systems (6144×6144). Therefore, we conclude that for such systems the bottleneck is the communication from and to the host.

5.2. *Stauffer and Kertész*

Stauffer and Kertész also implemented their algorithm on up to 32 nodes of the Intel IPSC/860 via F77 with message-passing, they also use strip-parallelization. Their algorithm is called "global stack" in [Hee91].

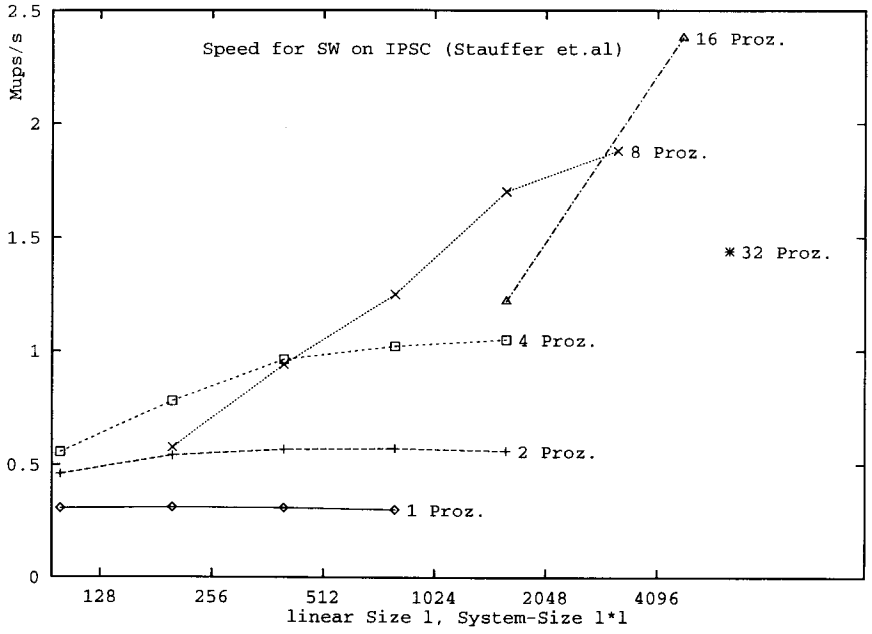


It communicates a list of all the clusters that are located on more than 1 processor ("multinational clusters" in Stauffer's terminology) to all other processors. Their scalar algorithm requires 8 byte/spin.

+ Their source–code is impressively short (only 2 pages).

– They reach their highest speed for 16 nodes, their speed on 8 nodes is larger than that on 32 nodes using the same amount of storage on each node.

– The length of the boundary per processor increases with the number of nodes.

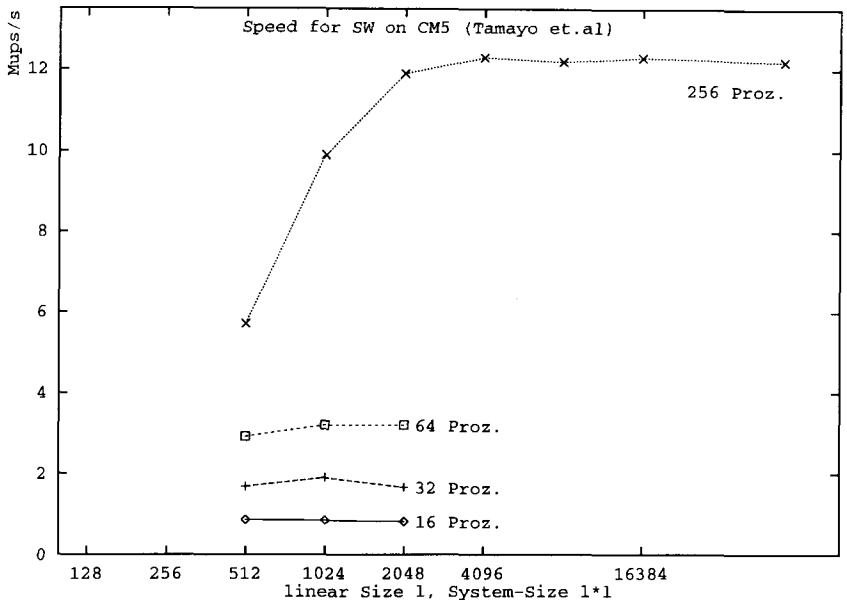


5.3. *Tamayo and Flanigan*

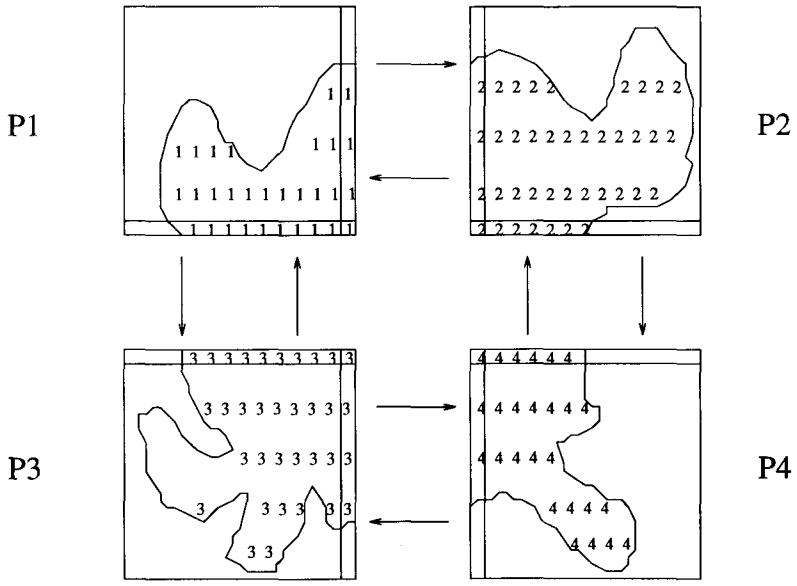
Tamayo and Flanigan implemented their algorithm on a CM5 in C with message–passing. They decompose their physical system in quadratic local systems. The communication of the cluster label information is implemented by relaxation of cluster labels on neighboring processors. The local algorithm uses 5 byte/spin.

+ The algorithm has been proven to be scalable in theory as well as in the implementation up to 256 nodes, the efficiency lying well about 90 %. Apart from the relaxation technique this is due to the fact that for fixed local system sizes the ratio of local boundaries to local systems remains constant.

– The algorithm is not easy to implement, as boundaries in two dimensions have to be taken into account.



Relaxation

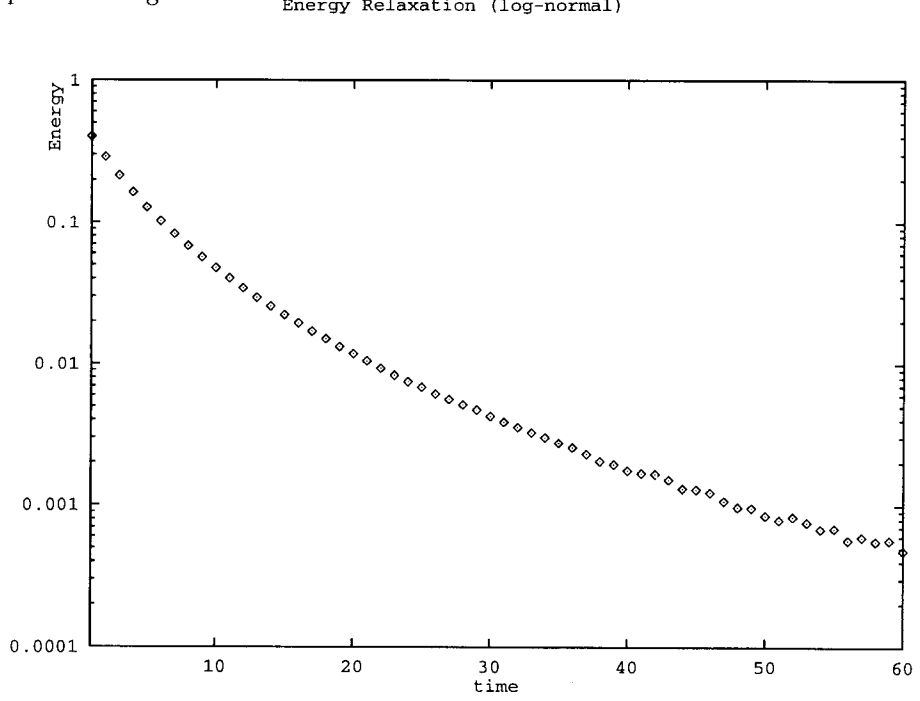


– The speed of the local algorithm is very low. It takes approximately four times as long as the other two algorithms, though the speed of the nodes (Sun Sparc 2 with 33 MHz, Intel i860 with 40 MHz, compilation with the the highest scalar optimisation level) is comparable. We implemented our scalar code

also on SUN–Workstations, and the difference in execution time compared with the i860 was proportional to the ratio of the clock cycles of the two machines, so the cluster algorithms should perform similarly on different RISC architectures. P.Ueberholz indicated to us during this workshop that a CM5–node should behave in scalar C–code more or less as a stand–alone SPARC, so that we can give no reason for this low scalar performance. One explanation for the different performance of the local algorithms may be the use of only 5 bytes per spin. There is no way to use another array for "shortcutting" the relabeling.

6. Non–Equilibrium Relaxation

So far, we have only dealt with "computational" problems like execution time and efficiency. But for physicists it is of higher importance to get an answer to real physical questions. To test our algorithm in such a way we have decided to investigate the non-equilibrium relaxation which presently is of considerable interest because it is expected to reveal the dynamical critical exponent z (see also [Stau92], [Ito93] und [Tam93]). If $z > 0$, the magnetization $M(t)$ is expected to decay as $t^{-\frac{\beta}{z\nu}}$ while an expotential relaxation would correspond to the dynamical critical exponent being zero.

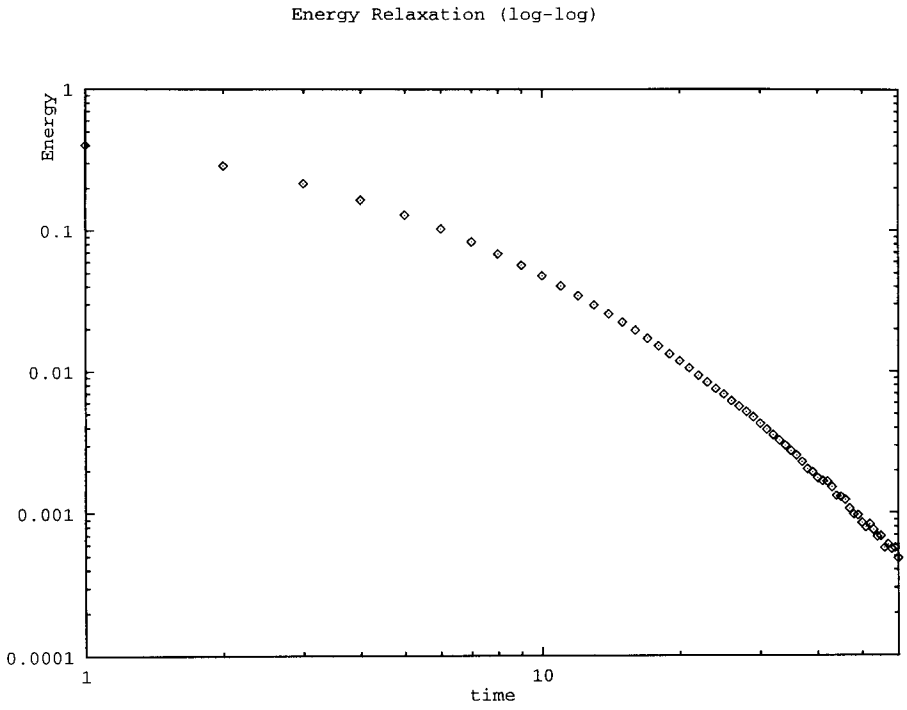


We started with an initial configuration in the ground state, i.e. all spins pointing up, and observed the behavior of the energy and the absolute value of the magnetization for 50 – 60 updates of a 6144×6144 system. The values at every

time which is given in units of system updates, are averaged over 30 independent relaxation processes.

We tried to fit $E_{eq} - E(t)$, with $E_{eq} = -\sqrt{2}$ being the ground state energy, to an exponential decay, but we did not succed. For $t < 40$ a power law ansatz does not work either, but for larger t the relaxation may be described by a power law.

For a reliable confirmation, however, we would have to observe the behavior of the system for significantly longer times.

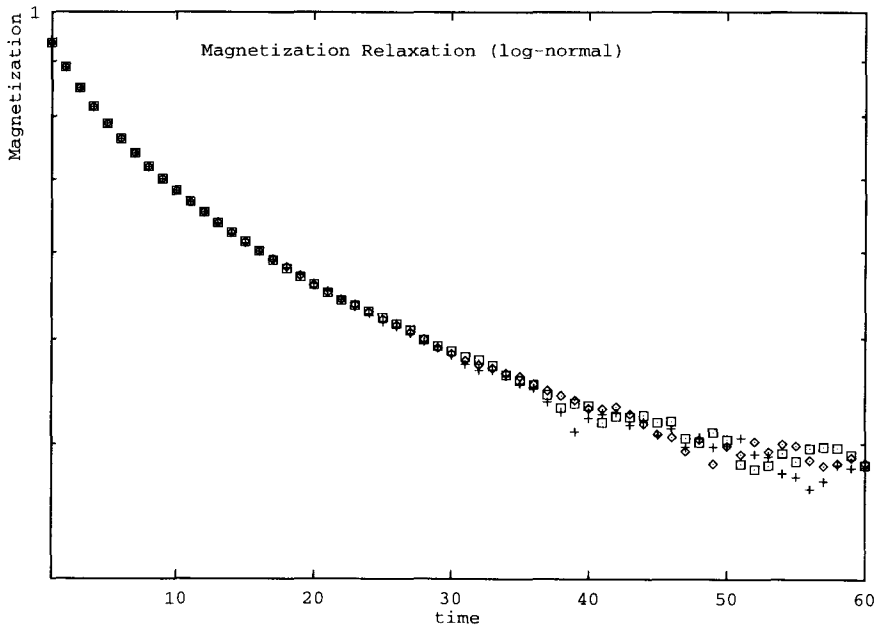


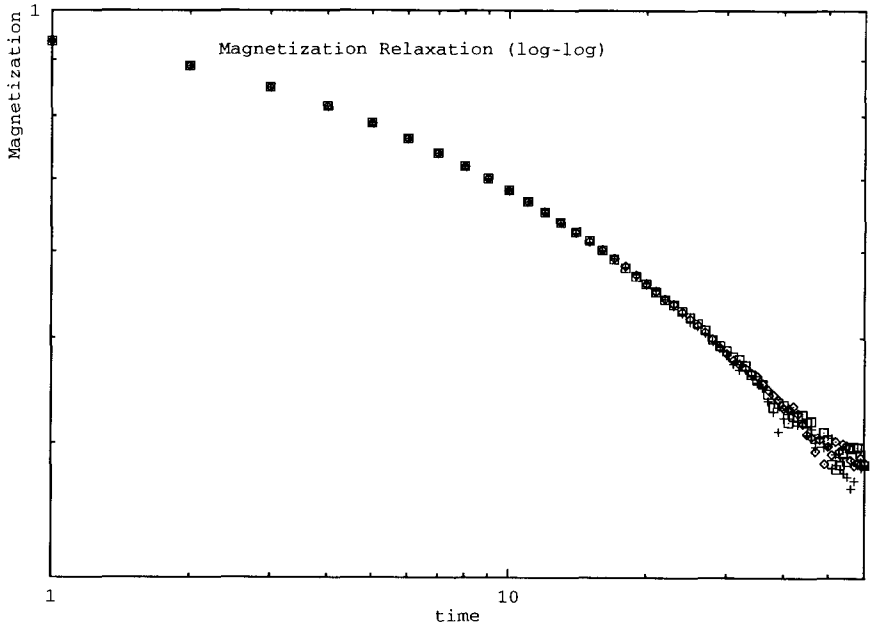
Even more difficult is the interpretation of the magnetization, because for $t > 35$ we get strong fluctuations which do not allow us to favor either an exponential or a power law behavior.

In order to exclude effects by correlated random number sequences, we have also tested several random number generators, and got discouraging different results. So far, we are not sure whether this is due to normal statistical deviations or a built-in effect of the generators.

The plots of the magnetization show the data of three measurements with the random number generator CONG (multiplication by 16807) which is also used by Kertész and Stauffer, each initialized with a different starting seed. Although the single curves seem to be relatively smooth, and simulate a bearable error, one has to keep in mind that values of consecutive time steps are highly correlated. Indeed, for times > 30 we estimate our error to be about 0.02. For $t = 60$ we measure a

magnetization of .38 , which is in accordance with [Tam93] and contradicts [Sta92].





Therefore, we come to the preliminary conclusion that in order to get a reliable result we have to measure for a larger time in the case of energy and magnetization,

and to gain a much better statistic, at least in the case of magnetization. Thus, further ample computations will be necessary for two dimensions and the behavior of 3-D-Systems would be interesting, too.

7. Conclusions

Cluster–Algorithms **can** be parallelized. On moderately large configurations, even a straightforward communication pattern can be employed successfully if the algorithm is properly tuned. The host–node communication scheme has performed very well for the available configuration, also providing global information of the cluster sizes. If global information is not required, and if the simulation is run at a higher temperature with smaller clusters, the use of other schemes than host–nodes will be advisable.

Good local algorithms help a lot to achieve good performance. For few processors the main issue will still be the optimization of the scalar code. In our case, replacing the random number generator of the library with a short inline code improved the speed significantly.

Parallel programs are definitely longer than scalar programs. Our optimized parallel code is three times as long as the scalar kernel, the amount of work was even more. Especially MC–calculations should use trivial parallelization whenever possible.

If you are in doubt on which machine you should implement a code of the structure of a cluster–search–problem, choose a machine with less but more powerful nodes: Less sophisticated algorithms can be fast as well if powered by few but fast nodes. Current trends on the hardware market also tend into that direction: IBM SP1, Fujitsu VPP 500 and Cray T3D are not intended as arbitrarily massive configurations, not only for marketing reasons but because of Amdahl's law.

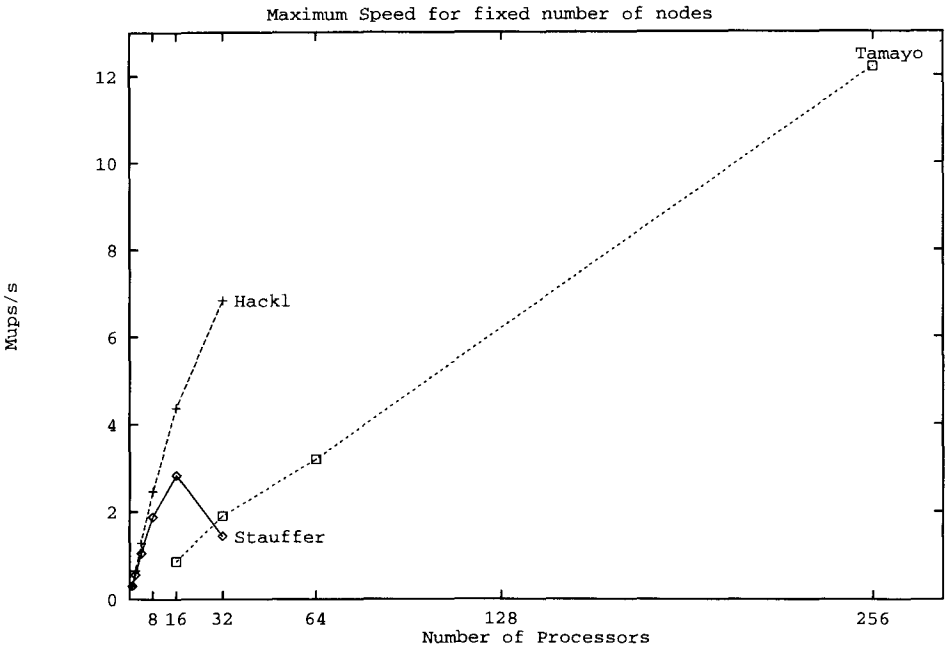
Looking at the "generations" of a parallel–computer doesn't give any information about performance. Our experiences with the Intel Paragon show that, apart from stability problems, the operating system is not yet (July 1993) able to take advantage of the better communication hardware.

Parallelization of cluster–algorithms is very sensitive to fine–tuning: Use of parallel–compilers (Fortran90, C*) instead of message passing may cause problems, if one expects the compiler to handle the communication part of the simulation.

Don't parallelize large system–sizes you can't run anyway because of the CPU–time requirements. Tamayo reaches twice the number of updates/s for a system about ten times as large as our largest system. That means, a single update for a system will last 5 times as long as in our simulations. For university work groups, the necessary amount of computer time will not be attainable.

Several conclusions can be drawn for related algorithms or Hamiltonians and higher dimensions. Our algorithm will achieve still higher speed for larger memories, as fraction of the parallel overhead scales with $1/L$, L being the linear dimension of the system. If the number of spins in a simulation is fixed (e.g. caused by memory limitations), the parallel overhead will increase in higher dimensions, because the

ratio between spins in the interior of a system and spins on the boundaries becomes smaller.



For slightly different problems, such as finding the fermion loops in $SU(X)$–simulations or Worldline Quantum Monte Carlo, even in higher dimensions, better performance can be expected, as not the whole boundaries have to be communicated and processed on the host.

For systems of continuous spins $U(X)$, the efficiency can be expected to increase, as the amount of computations increases. Calculation of the probability for broken bonds requires the exponential of a scalar product of vector–length X. The evaluation of the exponential can either be executed via use of the built–in functions, which need ≈ 50 cycles on a RISC–processor, or the interpolation between values on a stored lookup–table, which will also take dozens of cycles. The storage requirements per spin only rise by $4 \times X$ bytes.

Acknowledgements

We would very much thank the ZAM Jülich for the support during the development of the algorithm, especially Jutta Docter and Inge Gutheil, as well as for providing ample computer time on the Intel IPSC/860.
We thank Nobuyaso Ito for providing us with a faster random number generator as well as for drawing our attention to the non–equilibrium relaxation of the energy.
We are very much indebted to Dietrich Stauffer for fruitful hints and continous advice as well as for providing us with a listing of the source code of his algorithm

and the data produced.
Discussions with the members of the Many Particle Group of the HLRZ Jülich are gratefully acknowledged.

References

[Hee91] D.W.Heermann, A.N.Burkitt, *Parallel Algorithms in Computational Science,* Springer 1991

[Hos76] J.Hoshen, R.Kopelman,*Percolation and cluster distribution,* Phys.Rev. **B14**, p.3438–3445 (1976)

[Jak92] A.Jakobs, R.W.Gerling, *Scaling Results for Parallelilsm on the Intel i860 Hypercube,* Physica A **180**, 407 (1992)

[Sta92] J.Kertész, D.Stauffer, *Swendsen–Wang Dynamics of large 2D Critiacl Ising Models,* Int.Journ.Mod.Phys. **C** Vol. 3, No. 6 (1992) 1275–1279

[Wan90] Jian-Sheng Wang, Robert H.Swendsen, *Cluster Monte Carlo Algorithms,* Physica A 167 p.565-579 (1990)

[Tam92] P.Tamayo, M.Flanigan, *A Parallel Cluster Labeling Method for Monte Carlo Dynamics,* Int.Journ.Mod.Phys. **C** Vol. 3, No. 6 (1992) 1235–1249

[Tam93] P.Tamayo, *Magnetization Relaxation to Equilibrium on Large 2D Swendsen-Wang Ising Models,* preprint subm. to Physica A

[Ito93] N.Ito, G.Kohring, *Non-equilibrium critical relaxation of the cluster dynamics,* preprint subm. to Physica A