

5. OTHER ALGORITHMS

Currently the only other MIMD parallel cluster algorithm proposed for spin models is the Hoshen and R. Kopelman, Phys. Rev. Lett. **51** (1978) 3438.

parallel extension of the Hoshen and Kopelman algorithm due to Burkitt and Heermann, Phys. Rev. Lett. **58** (1987) 86.

algorithm is more complicated and less efficient than the Hoshen and Kopelman algorithm. It requires self-labeling, giving speedup factors of approximately 11.5 and 11.0 on 16 and 32 processors respectively.

Recently, Brower *et al.*, *Solving Problems on Connected Components* (Prentice-Hall, Englewood Cliffs, New Jersey, 1988).

They have implemented a self-labeling algorithm (which they call "local distributed") and a non-local, multi-grid style algorithm on the Connection Machine.

The non-local method takes fewer iterations, and performs much better than the local algorithm, although it is still very inefficient.

A number of component labeling algorithms have been proposed for image analysis applications for both SIMD and MIMD machines.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

Further investigation is needed to see if these algorithms might be applied to the difficult problem of producing an efficient parallel cluster algorithm for models on large numbers of processors.

ACKNOWLEDGEMENTS

The parallel algorithms were developed on a 512 node Ncube-1, a 192 node Symultis, and a 32 node Meiko Computing Surface.

I would like to thank the Caltech Concurrent Supercomputer Facility for the use of these machines.

This work was sponsored in part by DOE grants DE-FG03-85ER25009 and DE-AC03-81ER40050.

REFERENCES

1. N. Metropolis *et al.*, J. Chem. Phys. **21** (1953) 1087.
2. U. Wolff, Proc. of the Int. Conf. 'Lattice 89', Nucl. Phys. B (Proc. Suppl.) **17** (1990) 93.
3. A. Sokal, these proceedings.
4. C. F. Baillie, Int. J. of Mod. Phys. C (1990) 91.
5. R. H. Swendsen and J. - S. Wang, Phys. Lett. **58** (1987) 86.
6. R. B. Potts, Proc. Camb. Phil. Soc. **24** (1952) 106; F. Y. Wu, Rev. Mod. Phys. **54** (1982) 235.
7. U. Wolff, Phys. Rev. Lett. **62** (1989) 361.
8. D. Stauffer, Phys. Rep. **54** (1978) 1.
9. J. Hoshen and R. Kopelman, Phys. Rev. Lett. **51** (1978) 3438.
10. C. F. Baillie and P. D. Coddington, Identification Algorithms for Spin Systems, "Sequential and Parallel", Caltech preprint C3P-945 (June 1990).
11. U. Wolff, Phys. Lett. **B228** (1989) 379.
12. G. D. Cavoret *et al.*, *Solving Problems on Connected Components* (Prentice-Hall, Englewood Cliffs, New Jersey, 1988).
13. R. G. Bradow, P. Tamayo and B. York, Parallel Multigrid Algorithm for Percolation, Boston University preprint BOSTON-89-11, Phys. Statist. Solidi B **100** (1989) 1.
14. H. Embrechts *et al.*, "Component Labeling in Distributed Memory Multiprocessors", First European Workshop on Hypercube and Distributed Computers, F. Andre and J. P. V. Aerts eds., (North-Holland, Amsterdam, 1988).
15. B. A. Galler and M. J. Fisher, Comm. ACM **7** (1964) 301; W. H. Press *et al.*, *Numerical Recipes in C: The Art of Scientific Programming*, Cambridge University Press, Cambridge, 1988.
16. A. N. Burkitt and D. W. Heermann, Comm. ACM **54** (1989) 210.
17. R. D. Hundt, "Connected component labeling in single processing with MIMD architecture", *Intermediate-Level Image Processing*, M. J. B. Duff ed., (Academic Press, New York, 1989) 18.
18. R. Cypher, J. L. C. Sanz and L. Snyder, *Thin Client Computing*, Prentice-Hall, Englewood Cliffs, NJ, 1989, p. 140.
19. J. Woo and S. Sahni, J. of Supercomputing **10** (1989) 209.
20. W. Lim, A. Agrawal, L. Nekludov, "Fast Parallel Algorithm for Labeling Components in Image Arrays", Thin Client Computing Corporation Technical Report, 1990.
21. M. Manohar, Computer Vision, Graphics and Image Processing **45** (1989) 133.
22. P. D. Coddington and C. F. Baillie, Int. J. of Mod. Phys. C (1990) 305.
23. C. F. Baillie and P. D. Coddington, "Cluster Algorithms for 2-D Potts Models", Caltech preprint C3P-945 (October 1990).

Figure 1: Speedups on the Symult 2010 labeling.

problem, a lattice of size L^2 can be simulated on a single processor, or by doing independent runs on different processors, so the speedup for this small problem are of no great concern. The lattice sizes for which we actually need large numbers of processors are of the order of L^2 and we can see that running on 64 nodes (or running multiple simulations of 64 nodes each) gives us quite acceptable efficiencies of about 70% and 80% for $L^2=1024$. Using all 192 nodes of tech's Symult 2010 in this way gives a performance of approximately one million spin updates per second, which is about six times that of a CRAY X-MP, and about twice that of the best algorithm on a full sized Connection Machine. A machine with much faster processors than an Ncube-2 or Intel iPSC/860, could greatly improve on this figure. We have used this algorithm to measure autocorrelation times for 2-D Potts model on lattices up to 512^2 .

4. GLOBAL EQUIVALENCING

In this method we again use the fast partial algorithm to identify the clusters on every node. Each node then

sees which of the edge sites of its sub-connected to edge sites on the neighboring positive directions, and should be given a cluster label. These lists of "equivalent local cluster labels" are all passed to a program which uses an algorithm due to Gallier (to sort them into equivalence classes - the case are the global cluster labels) and casts the results to all the other nodes.

The problem here is that the equivalent algorithm is purely sequential, and is thus a potential bottleneck. To get around this we adopt a hierarchical divide-and-conquer approach. In this hierarchical equivalencing the problem is divided up into smaller sub-arrays of 2×2 processors. Each sub-array performs the equivalencing algorithm on its section. The results of these partial matchings are combined on each 4×4 sub-array, and this process is continued until finally all the partial results are put together to give the global cluster values. The number of processors performing the equivalencing steps is $P/4$ for the first level of the algorithm, $P/16$ for the second level, and so on, until the final stage is done on a single processor. The major bottleneck has been at least partially alleviated by using a hierarchical algorithm which uses the same procedure as has been implemented on the hypercube. For the image processing code being problem by Embrechts *et al.*

Our results for the hierarchical equivalent algorithm are shown in figure 2. Note that an optimized version of the algorithm on a full sized Connection Machine. Note that all these results are for spin models, the two-dimensional Ising model, although the parallel algorithm would be expected to improve on this figure. We have used this algorithm to measure autocorrelation times for more computationally intensive cases, such as continuous spin models, of calculation involved per processor is much more than the amount of communication. We would like to implement the efficient algorithm on larger numbers of processors.

cluster algorithms do not vectorize and their performance is poor on vector machines. On the other hand, in parallel, every processor of a CRAY X-MP is only about ten times faster than a Sun4 workstation (the CRAY time is bounded to a neighboring site from Ref. 11). We therefore also expect the performance on SIMD (Single Instruction, Multiple Data) computers such as the Connection Machine, and we have thus concentrated our efforts on algorithms for MIMD (Multiple Instruction, Multiple Data) computers, such as the Connection Machine and Intel hypercubes.

On MIMD machines, we can use the technique of parallelization of running independent simulations on each processor. Since the largest works well until the lattice size gets to the order of the lattice size of the memory of each processor, and we have to propagate the cluster to measure autocorrelation times for the Potts model, with communication. However in order to simulate large lattices, we need a parallel algorithm where the label is propagated over many processors of a parallel machine. This can be easily and efficiently done on a SIMD machine. However for cluster algorithms on a MIMD machine we can improve it is a much more difficult problem. The non-locality of the algorithm makes it very difficult to parallelize efficiently. It involves a large amount of non-local communication of each processor, to make the clusters means that load balancing is a severe problem.

The parallel cluster algorithms we have implemented involve distributing the lattice to a number of processors using the usual domain decomposition. The time to do this is proportional to the original lattice. On a MIMD machine, the perimeter of the partial algorithm can be used to label the sub-lattice, whereas the time on each processor, but we need a procedure for converting these labels to their correct global values. We will outline two methods we have used for this problem, "self-labeling" and "global labeling", as well as some other algorithms which have been proposed for this problem.

3. SELF-LABELING

We refer to this algorithm as "self-labeling". The dashed line indicates since each site figures out which cluster it is in (i.e. 100% efficiency). We begin by a series of steps on the Ncube-1 hypercube

For our test case of the 2-D Potts model, the speedups obtained for the self-labeling algorithm are shown in the Symult 2010 for a variety of lattice sizes.

PARALLEL CLUSTER ALGORITHMS

P. D. CODDINGTON

Physics Department, Syracuse University, Syracuse NY 13244, USA

and C. F. BAILLIE

Physics Department, University of Colorado, Boulder CO 80309, USA

Cluster update algorithms dramatically reduce critical slowing down in the Metropolis algorithm, it is not obvious how to implement these algorithms on parallel computers. Here we present two different parallel implementations of the algorithms which give reasonable efficiencies on various MIMD parallel computers.

1. INTRODUCTION

Monte Carlo simulations of spin models using the Metropolis algorithm have in common that they traditionally used local algorithms to update the spins and label the connected clusters. These algorithms have the major drawback that the number of iterations needed to generate a statistically independent configuration of spins (the "autocorrelation time") increases as the square of the correlation length L (the linear size of the lattice) for a second order phase transition. New algorithms have been developed which dramatically reduce critical slowing down by using a non-local update procedure. The Swendsen-Wang algorithm, which changes clusters of spins at a time, has the desirable property that cluster algorithms can greatly increase the efficiency of the simulation, in many orders of magnitude, compared to local algorithms which update single spins, and the operations to be performed on much greater clusters of sites (for reviews of cluster algorithms and their applications, see Refs. 2, 3, 4).

The original idea, due to Swendsen and Wang (S-W), was for the q -state Potts model. Whereas Wolff's algorithm introduces bonds between neighboring spins in a sequential state, with probability $K/(1+K)$ (where K is the interaction strength divided by the temperature), the Swendsen-Wang algorithm consists of generating all such clusters and then choosing a random spin value for each cluster. Near criticality, which is characterized by the divergence of the correlation length, the Swendsen-Wang algorithm performs simulations, proposed a variant of this algorithm which is chosen at random and a single cluster is chosen at random and all its spins are updated. The non-local nature of the cluster

2. CLUSTER LABELING ALGORITHMS