Parallel Cluster Algorithms

P. D. Coddington
C. F. Baillie

CRPC-TR90158
October, 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

Nuclear Physics B (Proc. Suppl.) 20 (1991) 76–79
North-Holland

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 spin models, but unlike the standard Metropolis algorithm, it is not obvious how to implement these algorithms efficiently on parallel or vector computers. Here we present two different parallel implementations of the Swendsen-Wang algorithm which give reasonable efficiencies on various MIMD parallel computers.

1. INTRODUCTION

Monte Carlo simulations of spin models have traditionally used local algorithms such as that of Metropolis *et al.* [1] These algorithms have the major drawback that the number of iterations needed to generate a statistically independent configuration (the "autocorrelation time") increases approximately as the square of the correlation length, which is of order L (the linear size of the lattice) near a second order phase transition. New algorithms have been developed which dramatically reduce this "critical slowing down" by using a non-local update scheme which changes clusters of spins at a time. These cluster algorithms can greatly increase the computational efficiency of the simulation, in some cases by many orders of magnitude, compared to local algorithms which update single spins, and allow simulations to be performed on much greater lattice sizes (for reviews of cluster algorithms and critical slowing down, see Refs. 2,3,4).

The original idea, due to Swendsen and Wang [5] (S-W), was for the q-state Potts model [6]. They introduce bonds between neighboring spins in the same state, with probability $1 - e^{-K}$ (where K is the interaction strength divided by the temperature), and thus create clusters of bonded spins. The S-W algorithm consists of generating all such clusters, then choosing a random spin value for each cluster and assigning it to all the sites in that cluster. Wolff has proposed a variant of this algorithm, in which a site is chosen at random and a single cluster constructed around it, and then all its spins are flipped [7].

2. CLUSTER LABELING ALGORITHMS

Cluster algorithms have in common the problem of identifying and labeling the connected clusters of spins, which is the most computationally intensive part of these algorithms. Cluster labeling is relevant to the study of percolation models [8], and is also very similar to an important problem in image processing, that of identifying and labeling the connected components in a binary or multi-colored image composed of an array of pixels. First we mention some sequential methods for labeling clusters. A commonly used algorithm in percolation studies is the algorithm of Hoshen and Kopelman [9], which has the desirable feature that it labels the clusters in a time proportional to the number of sites in the lattice. We have found that in practice another algorithm is slightly faster [10] — this is just the obvious method for identifying a single cluster of connected sites, by starting at any site in the cluster, and then continuing to expand outwards to all the connected, unlabeled sites until the entire cluster is labeled.

Whereas Wolff's algorithm is undoubtedly the best method on a sequential computer [2], the S-W algorithm seems to be better suited for parallelization, since each iteration involves the entire lattice rather than just a single cluster. We have therefore concentrated initially on implementing the latter algorithm in parallel. Near criticality, which in most cases is where we want to perform simulations, the S-W clusters are present at all length scales, from a single site to the order of the lattice volume. The highly irregular and non-local nature of the clusters means that

0920-5632/91/$3.50

cluster algorithms do not vectorize, and hence give poor performance on vector machines. One processor of a CRAY X-MP is only about ten times faster than a Sun4 workstation (the CRAY time is taken from Ref. 11). We therefore also expect to get poor performance on SIMD (Single Instruction, Multiple Data) computers such as the Connection Machine or the AMT DAP, and we have thus concentrated our efforts on algorithms for MIMD (Multiple Instruction, Multiple Data) computers, such as the Ncube or Intel hypercubes.

On MIMD machines, we can use the trivial parallelization technique of running independent Monte Carlo simulations on each processor. This method works well until the lattice size gets too big to fit into the memory of each processor, and we have used it to measure autocorrelation times for the Potts model [23]. However in order to simulate large lattices, we need a parallel algorithm where the lattice can be distributed over many processors of a parallel machine. This can be easily and efficiently done for local update algorithms [12], however for cluster algorithms it is a much more difficult problem. The non-locality which makes cluster algorithms so useful also makes them very difficult to parallelize efficiently, since this involves a large amount of non-local communication. Also the extreme irregularity in the size and shape of the clusters means that load balancing is potentially a severe problem.

The parallel cluster algorithms we have implemented involve distributing the lattice onto an array of processors using the usual domain decomposition [12], so that each processor deals with a subdomain of the original lattice. On a MIMD machine, a sequential algorithm can be used to label the sub-clusters 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 tackling this problem, "self-labeling" and "global equivalencing", as well as some other algorithms which have been proposed for this problem.

3. SELF-LABELING

We refer to this algorithm as "self-labeling", since each site figures out which cluster it is in by itself from local information. We begin by assigning each site i a unique cluster label S_i. At each step of the algorithm, in parallel, every site looks in turn at each of its neighbors in the positive directions. If it is bonded to a neighboring site n which has a different cluster label S_n, then both S_i and S_n are set to the minimum of the two. This is continued until nothing changes, by which time all the clusters will have been labeled with the minimum initial label of all the sites in the cluster. This is a purely SIMD algorithm, and when implemented in this way on a SIMD machine (such as the Connection Machine or the AMT DAP) it is very inefficient, due to the large amount of communication required, and the poor load balance [10, 13]. Since the largest cluster is of the order of the lattice size, it takes order L iterations to propagate the cluster label through the largest cluster, with communication required at each step. Hence there is a very large amount of communication, and many of the processors are lying idle while the label is propagated through to the final sites in the largest cluster.

On a MIMD machine we can improve this method greatly by using a fast sequential algorithm to label the sub-clusters in the sub-lattice on each processor, and then just use self-labeling on the sites at the edges of each processor, to match up the clusters which go across processor boundaries, and eventually arrive at the global cluster labels. The number of self-labeling steps will now only be of the order of the maximum distance between processors, rather than lattice sites, which for a square array of P processors is just $2\sqrt{P}$. Hence the time to do the self-labeling, which is proportional to the number of iterations times the perimeter of the sub-lattice, goes like L for an LxL lattice, whereas the time taken on each processor to do the local cluster labeling goes like the area of the sub-lattice, which is L^2/P, so as long as L is substantially greater than the number of processors we can expect to obtain a reasonable speedup.

For our test case of the 2-D Potts model, the speedups obtained for the self-labeling algorithm on the Symult 2010 for a variety of lattice sizes are shown in Fig. 1. The dashed line indicates perfect speedup (i.e. 100% efficiency). We obtained similar results on the Ncube-1 hypercube. For this

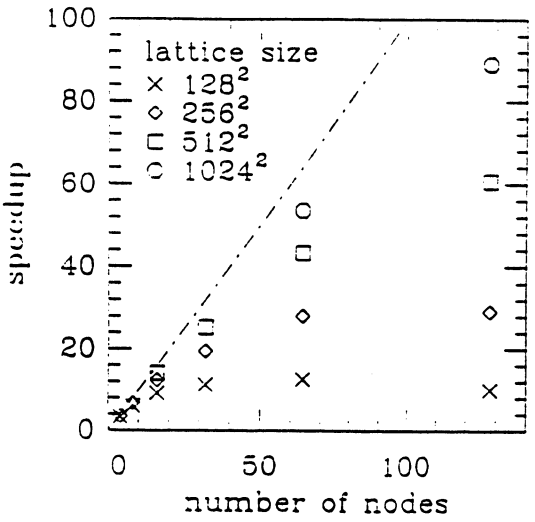

Figure 1: Speedups on the Symult 2010 for self-labeling.

problem, a lattice of size 128^2 can easily be simulated on a single processor, or by doing independent runs on different processors, so the poor speedups 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 512^2 or greater, 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% for 512^2 and 80% for 1024^2. Using all 192 nodes of Caltech's Symult 2010 in this way gives a performance of approximately one million spin updates per second, which is about six times that of one head of a CRAY X-MP, and about twice that of the current best algorithm on a full sized Connection Machine [13]. A machine with much faster processors, such as an Ncube-2 or Intel iPSC/860, could be expected to greatly improve on this figure. We have used this algorithm to measure autocorrelation times for the 2-D Potts model on lattices up to 512^2 [24].

4. GLOBAL EQUIVALENCING

In this method we again use the fastest sequential algorithm to identify the clusters in the sublattice on every node. Each node then checks to see which of the edge sites of its sub-lattice are connected to edge sites on the neighboring nodes in the positive directions, and should be given the same cluster label. These lists of "equivalences" between local cluster labels are all passed to one of the nodes, which uses an algorithm due to Galler and Fisher [15] to sort them into equivalence classes (which in this case are the global cluster labels) and then broadcasts the results to all the other nodes.

The problem here is that the equivalencing is purely sequential, and is thus a potentially disastrous bottleneck. To get around this problem, we can adopt a hierarchical divide-and-conquer approach. In this hierarchical equivalencing the processor array is divided up into smaller sub-arrays of, for example, 2x2 processors. Each sub-array performs the global equivalencing algorithm on its section of the lattice. The results of these partial matchings are then combined on each 4x4 sub-array, and this process is continued until finally all the partial results are merged together to give the global cluster values. In this way the number of processors performing the equivalencing step is $P/4$ for the first level of the hierarchy, $P/16$ for the second level, and so on, until the final stage is done on a single processor. However by that time most of the work has been done, so the bottleneck has been at least partially alleviated. A very similar algorithm which uses the same hierarchical procedure has been implemented on the iPSC-2 hypercube for the image processing component labeling problem by Embrechts *et al.* [14]

Our results for the hierarchical equivalencing are slightly worse than for self-labeling, however we are hopeful that an optimized version of the program will do better. Note that all these results are for the simplest of spin models, the two-dimensional Potts model, although the parallel algorithms could clearly be used in more general cases. In fact for higher dimensions or for more computationally intensive models, such as continuous spin models, the amount of calculation involved per processor should increase much more than the amount of communication required for the parallel algorithms, and consequently we would expect to implement them efficiently on larger numbers of processors.

5. OTHER ALGORITHMS

Currently the only other MIMD parallel cluster algorithm proposed for spin models is a parallel extension of the Hoshen and Kopelman algorithm due to Burkitt and Heermann [16]. Their algorithm is more complicated and less efficient than self-labeling, giving speedups for a 512^2 lattice of approximately 11.5 and 11.0 on 16 and 32 processors respectively. Recently, Brower *et al.* have investigated parallel cluster algorithms for SIMD machines [13]. They have implemented a self-labeling algorithm (which they call "local diffusion") and a non-local, multi-grid style algorithm on the Connection Machine. The non-local method takes many 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 [20, 21, 22] and MIMD [17, 18, 19] machines. Further investigation is needed to see if they might be applied to the difficult problem of producing an efficient parallel cluster algorithm for spin models on large numbers of processors.

ACKNOWLEDGEMENTS

The parallel algorithms were developed and run on a 512 node Ncube-1, a 192 node Symult S2010, and a 32 node Meiko Computing Surface. We 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 1 (1990) 91.
5. R.H. Swendsen and J.-S. Wang, Phys. Rev. Lett. **58** (1987) 86.
6. R.B. Potts, Proc. Camb. Phil. Soc. **48** (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. B 14 (1976) 3438.
10. C.F. Baillie and P.D. Coddington, "Cluster Identification Algorithms for Spin Models — Sequential and Parallel", Caltech preprint C3P-855 (June 1990).
11. U. Wolff, Phys. Lett. **B228** (1989) 379.
12. G.C. Fox *et al.*, *Solving Problems on Concurrent Processors* (Prentice-Hall, Englewood Cliffs, New Jersey, 1988).
13. R.C. Brower, P. Tamayo and B. York, "A Parallel Multigrid Algorithm for Percolation Clusters", Boston University preprint, submitted to J. Stat. Phys.
14. H. Embrechts *et al.*, "Component Labeling on a Distributed Memory Multiprocessor", *Proc. First European Workshop on Hypercube and Distributed Computers*, F. Andre and J.P. Verjus eds., (North-Holland, Amsterdam, 1989).
15. B.A. Galler and M.J. Fisher, Commun. 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, Comp. Phys. Comm. **54** (1989) 210.
17. R. Hummel, "Connected component labeling in image processing with MIMD architectures", in *Intermediate-Level Image Processing*, M.J.B. Duff ed., (Academic Press, New York, 1986).
18. R. Cypher, J.L.C. Sanz and L. Snyder, J. Algorithms **10** (1989) 140.
19. J. Woo and S. Sahni, J. of Supercomputing 3 (1989) 209.
20. W. Lim, A. Agrawal, L. Nekludova, "A Fast Parallel Algorithm for Labeling Connected Components in Image Arrays", Thinking Machines Corporation Technical Report NA86-2.
21. D. Nassimi and S. Sahni, SIAM J. Comput. 9 (1980) 744.
22. M. Manohar, Computer Vision, Graphics and Image Processing **45** (1989) 133.
23. P.D. Coddington and C.F. Baillie, Proc. of the Int. Conf. 'Lattice 89', Nucl. Phys. B (Proc. Suppl.) **17** (1990) 305.
24. C.F. Baillie and P.D. Coddington, "Comparison of Cluster Algorithms for 2-D Potts Models", Caltech preprint C3P-945 (October 1990).