New SIMD Algorithms for Cluster Labeling on Parallel Computers

John Apostolakis
Paul Coddington
Enzo Marinari

CRPC-TR92246
September 1992

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

SCCS-279

New SIMD Algorithms for Cluster Labeling on Parallel Computers

John Apostolakis, Paul Coddington
and Enzo Marinari(*)

Physics Department,
Syracuse University,
Syracuse, NY 13244, U.S.A.

September 14, 1992

Abstract

Cluster algorithms are non-local Monte Carlo update schemes which can greatly increase the efficiency of computer simulations of spin models of magnets. The major computational task in these algorithms is connected component labeling, to identify clusters of connected sites on a lattice. We have devised some new SIMD component labeling algorithms, and implemented them on the Connection Machine. We investigate their performance when applied to the cluster update of the two dimensional Ising spin model. These algorithms could also be applied to other problems which use connected component labeling, such as percolation and image analysis.

(*): and Dipartimento di Fisica, Università di Roma *Tor Vergata*, Via E. Carnevale, 00173 Roma, Italy.

1 Introduction

Monte Carlo simulation is a very important numerical technique for studying a wide range of problems in the physical sciences, and in particular, the statistical mechanics of spin models of magnets[1, 2]. Unfortunately, traditional Monte Carlo algorithms for these models, such as the commonly used Metropolis algorithm[3], suffer from *critical slowing down* near the regions of interest – the critical points separating different phases of the system. This means that the autocorrelation time (the number of iterations needed to generate a new, statistically independent, data point) increases as L^z at the critical point, where L is the linear size of the system, and z is the dynamic critical exponent[4, 5]. z is at least 2 for most local algorithms, such as Metropolis, so the efficiency of these methods decreases rapidly as the size of the system is increased.

The reason for this poor performance is that standard Monte Carlo algorithms are local. In the lattice of spins which represents, for example, a magnetic material, only a single spin at a time is changed, and this change is influenced only by the spins on neighboring sites. Information undergoes a random walk on the lattice, and thus takes a time of order L^2 to propagate throughout the lattice. In the last few years, algorithms have been invented for certain types of spin models which make large-scale, non-local changes, and greatly reduce critical slowing down[6, 7]. In these so-called *cluster* algorithms, clusters of spins (rather than single spins) are changed at each step of the Monte Carlo procedure. The clusters are formed by generating bonds connecting neighboring sites, using a probabilistic procedure which varies between different models and algorithms (for reviews of cluster algorithms, see Refs. [4, 5, 8, 9]).

The major computational task of these cluster algorithms is the identification and labeling of the clusters of connected sites, given the configuration of bonds. This is an instance of a connected component labeling problem for an undirected graph[10, 11], where the vertices are the lattice sites and the edges are the bonds between connected sites. The goal of the component labeling algorithm is to end up with the same label on all connected sites, and different labels for all disconnected clusters.

Sequentially, this can be done in time of order V (the number of vertices, which in

our case is the volume, or number of sites, in the lattice), and consequently the cluster algorithms run about as fast as the local algorithms (see Refs. [10, 12, 13]).

for a discussion of sequential labeling algorithms). However this may not be the case if our computer is a distributed memory parallel machine. Local algorithms perform very efficiently on parallel machines, whereas efficient component labeling on a parallel machine is a very difficult problem[12]. Here the information concerning the connectivity of a given physical part of the lattice is only contained in a single processor, and obtaining information from distant regions of the lattice (and hence also of the computer) can be very slow if the clusters are large, and thus contain sites which are distributed over many processors.

Let us assume that the time taken to label the clusters scales asymptotically as L^{d+y} for a lattice of L^d sites, where y is an exponent indicating *computational slowing down*, in analogy with the dynamical exponent z expressing the critical slowing down of a Monte Carlo simulation. This means that the overall computational cost of a Monte Carlo cluster algorithm simulation at the critical point will scale as L^{d+y+z}. If we cannot find a parallel labeling algorithm for which y is zero, the advantages of cluster update algorithms over traditional local algorithms may be eliminated on a parallel machine by the computational complexity of labeling the clusters.

Our aim is to find a parallel component labeling algorithm with no computational slowing down (i.e. $y = 0$). We will consider here the case where the parallel computer is a Single Instruction Multiple Data (SIMD) machine, although the ideas described here could also be applied to Multiple Instruction Multiple Data (MIMD) machines. We have implemented all the algorithms on the CM-2 Connection Machine, which is a typical massively parallel SIMD computer[14].

In order to test the algorithms we have studied the clusters formed in the physically interesting case of the Swendsen-Wang cluster algorithm applied to the Ising spin model at its critical point[6]. These clusters are very similar to those created by the simple procedure of randomly connecting neighboring sites on a two dimensional lattice with probability $\frac{1}{2}$. Clusters created in this way are very difficult to label efficiently on a parallel machine, since the clusters in a particular configuration of the connections come in many different sizes, have extremely irregular shapes, with small clusters embedded in larger ones, and

typically including a very large cluster which will span the lattice. This type of problem is consequently an excellent test of parallel component labeling algorithms.

We also note that the worst case behavior of the labeling algorithms is not relevant for this problem – what we are really interested in is the *average* time to label physically realistic configurations of clusters which occur in the cluster update of the spin model. We have therefore obtained all our data by averaging over a large number (typically 400) of different realizations of the site connections, taken from different Swendsen-Wang bond configurations for the Ising model at its critical point. This is in order to get statistically significant results from which we can obtain the scaling behavior of our algorithms, and timings for our implementations of these algorithms on the CM-2.

2 Simple Parallel Algorithms

The simplest and most obvious SIMD component labeling algorithm is local label propagation[12, 15]. We start with a different label on each site, and with a list of nearest neighbor connections (these will be Boolean variables in the following: *off* means no connection is present and *on* means that there is a connection). Each site then looks to each of its neighbors in turn. If it is connected to this neighbor, and if its neighbor's label is smaller than its own label, then it replaces its label with that of its neighbor. This procedure is repeated until there is no change to the labels, at which time each cluster will be labeled by the minimum initial label of all the sites in that cluster.

This local algorithm suffers from computational slowing down, and for many problems of interest (such as spin models at their critical point) its performance degrades very fast with increasing volume. This is because there is typically a large cluster whose graph-theoretic diameter or *chemical distance*, defined as the maximum value of the shortest path length between two points in the cluster, scales as L^f, where the exponent f is approximately 1 for the two dimensional Ising model[16], i.e. the diameter of the largest cluster scales approximately linearly with L. For any local labeling algorithm, the minimum label has to diffuse across this large cluster, so we expect that $y = f \approx 1$.

This algorithm can be improved by making the propagation step non-local. One way

of doing this is, instead of propagating the labels only to neighboring connected sites, to propagate them as far as we can along a given direction, until we come to a site with no connection in that direction. On the CM-2 this can be done very quickly by using the intrinsic **scan_with_minimum** function[17]. This routine operates on a row of numbers (labels in our case), each of which has an associated Boolean flag (the connections). It runs along each connected section of the row and deposits at each site the minimum of the numbers in the section up to that point (this is done in a distance doubling way, taking $\log_2 L$ steps). One step of this labeling method consists of a *scan* in each of the forward and backward directions of every axis. If periodic boundary conditions are used, this must be supplemented by a local label propagation step, since the *scan* routine does not wrap around the lattice.

Another way of improving the above algorithms is the notion of *connection improvement.* So far we have considered the bonds between sites to be static, in other words they are set up at the beginning and remain unchanged throughout the labeling procedure. However it is actually very useful to change, or improve, the connections as the labeling procedure progresses, and we learn more about the connectivity of the sites. It will often happen that neighboring sites will not have a bond between them, but will still be part of the same cluster, as shown in Fig. 1. If we compare the labels of neighboring sites at each step of the labeling algorithm, then at some point we will find that these two neighboring sites have the same label. We could then place a connection between these sites, since we now know that they are in the same cluster. Improving the bonds in this way means that new labeling information can now flow directly between these two sites, rather than by an indirect route via the original bonds.

Connection improvement is especially useful when applied to the *scan* algorithm, since in that case the addition of extra connections means that labels may be propagated much further in a single *scan* operation. This can be seen in Fig. 2, which shows a log-log plot of the average number of iterations required to complete the cluster labeling for the local algorithm and the *scan* algorithm, both with and without connection improvement. In Table 1 we show the exponent y for computational slowing down for each of these algorithms, which are obtained from the straight line fits shown in Fig. 2. As expected, the expo-

nents are all near 1, except for *scan* with connection improvement, which is substantially smaller, although still far from zero. However we should note that these results are very dependent on the type of bond configurations used. Note for example that configurations for which the clusters are fully connected, smooth, regular shapes, such as may occur in labeling objects in image processing applications, would be labeled in a very small number of *scan* operations. We might expect that y would be zero for the *scan* algorithm for those particular types of configurations.

3 A Multi-Scale Algorithm

In this section we describe a regular, synchronous, multi-scale algorithm for cluster labeling[18]. We present numerical evidence that the average number of iterations and the average time taken do not undergo any power-law computational slowing down (i.e. $y = 0$) for our application of labeling Ising model clusters.

The algorithm is effective on a general SIMD machine provided that the switching network has some very basic non-local connections. In the following we will assume that the machine allows very fast communication between sites which are a distance of 2^m sites away in any direction of the physical lattice. These are the only non-local connections we need in order to build an algorithm which is not affected by power-law slowing down. Such connections would be provided, for example, by a machine with a hypercube topology.

On the CM-2 the mapping of the physical structure of the lattice to the (almost) hypercube processor communication network provides specific communication to nodes of the lattice that are at a distance of any power of two away, known as **power_of_two** operations. This involves the transfer of information over not more than two links of the hypercube, and should thus be executed at not less than half the speed of local communications; however the relative timing compared to a local communication depends on the virtual processor to physical processor (VP) ratio (i.e. the number of lattice sites per processor), as we will see later.

In common with the method proposed by Brower, Tamayo and York[15], this method uses a *multi-scale* approach in propagating cluster labels, in order to overcome the slowing

down inherent in local labeling algorithms. However this algorithm is much simpler, and seems to have better scaling properties.

Our algorithm works for a lattice of any dimensionality d, but for ease of description we will consider a two dimensional problem. In this case the key variables used are Boolean connections that are set up in the x and the y axis at a distance 2^m, for $m = 1, \ldots, l-1$ (where the lattice size $L \equiv 2^l$), by a logical AND of connections at level $m-1$. For example, the distance 2 connection between sites i and $i + 2\hat{x}$ is set (i.e turned *on*) if both the connections between sites i and $i + \hat{x}$ and sites $i + \hat{x}$ and $i + 2\hat{x}$ are already *on*. These connections are rebuilt in this way at each iteration. In addition to building up the long distance connections in this manner, at each iteration we also use improve the connections, thus a connection between two sites at a generic distance M which was originally *off* can be set (i.e. declared to be *on*) if the two sites are found to have the same label. Using connection improvement greatly reduces the number of iterations needed to converge to the final values of the labels.

Thus, during one multi-grid label updating cycle each site will look in turn at each of its $2d$ neighbors at each level m of the multi-scale connections. It will update, when possible, its label and also update its connection by merging the level $m-1$ connections and by using connection improvement. A full cycle of the algorithm sweeps all l connection levels, and a single such cycle solves the trivial case where all connections are *on*. As the labeling progresses, what happens is that an increasing fraction of ever longer distance connections are set as sites are recognized as belonging to the same cluster, and these connections become fast long distance communication channels.

In Fig. 3 we show the average number of iterations needed to label the Ising clusters as a function of $\log_2 L$. The logarithmic slowing down is very clear. We do not see any sign of power-law behavior, or of a higher power of the logarithm. Each iteration of the algorithm involves a multi-grid cycle of $\log_2 L$ steps, with each step taking approximately the same amount of time, which is proportional to L^d/N, where N is the number of processors ($N \leq L^d$). Thus the total CPU time goes as $L^d(\log L)^2/N$, or $(\log L)^2$ for a machine with L^d processors. Hence this algorithm adds only a $(\log L)^2$ term to the overall slowing down of a spin model cluster algorithm. The average labeling time per site as a function of $\log_2 L$

is shown in Fig. 4.

The effect of different VP ratios means that the times for the multi-scale algorithm on a fixed number of processors for different lattice sizes do not scale simply as $L^d(\log L)^2/N$. Firstly, we note that local operations are more efficient at higher VP ratios, since a greater proportion of the neighboring sites will be on the same processor, so less inter-processor communication is required. This effect decreases as the depth of the multi-scale procedure is increased, since more sites at a distance 2^m are going to be on different processors as m increases. Eventually the communication distance will be greater than the the size of the subdomain on each processor, so that all data must be communicated between processors. The communication time will therefore be roughly constant at this level and higher, and at the highest levels it is roughly independent of the VP ratio. (The situation is actually slightly more complicated than this, since on the CM-2 there are 16 processors per chip, and it is inter-chip, rather than inter-processor, communication which is costly.) Thus the ratio of the time taken to do a step at the largest depth to the time to do a local labeling step increases from about 2 at $L = 128$ to about 6 at $L = 2048$ on a $16K$ CM-2.

There is a way however to combat the higher cost of deep iterations and significantly reduce the running time of our algorithm with only a simple modification. Clearly at the beginning of the labeling procedure the long distance connections are all *off*, and due to the fractal structure of the connections, it takes several iterations before a significant number of long distance connections are generated. It is thus very useful to tailor the number of multi-grid levels as a function of the cycle number: a lower depth is useful at the beginning, while using longer distance connections is more useful towards the end of the procedure. Fig. 3 shows the average iterations for the simplest case, where the depth is constant, while Fig. 4 gives the timings for both the full depth version and the optimized method.

In order to investigate the effect of varying the depth of the multi-grid procedure, we have measured the number of connections at every multi-grid level after each iteration of the full depth algorithm. We show this for a typical configuration in Fig. 5. It can be seen that the points where the different levels become useful, i.e. where there are a reasonable number of connections (of the order of 10%, for instance), increases roughly linearly with the number of iterations. We therefore chose in our modified algorithm to make the depth

a linear function of the iteration number. Since the usefulness of a connection at a certain level depends on the relative timings of different operations on a specific machine, this relation must be determined empirically.

Although Fig. 5 shows that a large number of connections at higher levels exist, we found that steps at the highest levels cost too much and were thus of comparatively little use, so a maximum depth of N_{max} steps was used. We thus parametrize the depth at each iteration by

$$depth = \min\{slope * iteration, N_{max}\}. \tag{1}$$

and seek to find the optimum value for the *slope* and the maximum depth N_{max}.

The behavior of the average labeling time versus the slope for some different values of L and N_{max} can be seen in Fig. 6. A minimum exists between 0.3 and 0.5 in all cases. The minimum is fairly broad and its breadth tends to increase as L increases. Fig. 4 shows the average labeling time per site for the optimized algorithm as well as the full depth multi-scale procedure. Note that the time for the optimized procedure is significantly smaller. The optimal value of N_{max} is $\log_2 L - 3$ (i.e. the 3 highest levels are not used) for most lattice sizes, although for large L it is slightly more efficient to exclude the fourth highest level as well, since the ratio of the time taken at the higher levels to the time for a local iteration is greater at large L.

We also tried a very simple telescoping scheme in order to determine whether any benefit could be derived from reducing the size of the problem in such a way. For this we used only the top level and one lower level, consisting of the even-even coordinate sites, and attempted to solve this partial problem by iterating until the labels of this sublattice did not change. Neither this nor an attempt to do a fixed number of multi-scale iterations on the lower level managed to reduce the amount of time required for the full labeling procedure. We note that only a single reduction and expansion was tried. This poor result was not due to the overhead of communicating between the different lattices, which cost very little time, but must have been due to the lack of information at the lower level about enough of the important connections.

4 Get/Send : An algorithm using general communications

We now present a different component labeling algorithm[19] that was inspired by the efficiency of the SIMD algorithm of Hillis and Steele[20] for finding the end of a linked list. We treat the labels as pointers in a dynamic tree-like structure, making our method similar to the sequential algorithms of Galler and Fisher[21] and of Hoshen and Kopelman[22] and the paralell algorithm of Shiloach and Vishkin[23].

All the cluster labeling algorithms discussed to this point start with each site being given a unique label. It is convenient to set the original label to be the site number. We will again label a cluster by the smallest starting label of all the sites in that cluster. For a two dimensional lattice we could assign the original cluster label C of the site (x, y) to be $(y * L) + x$, for example. During the cluster labeling procedure we can consider the current label as a pointer to the site where it originated (i.e. $x_{orig} = C \bmod L$, $y_{orig} = C/L$). The site (x_{orig},y_{orig}) started with the current label of the site (x, y), and as we shall see, at any later time in the labeling procedure it must have a label which is less than or equal to this value. Thus at any time each site can get the labél of the origin (x_{orig},y_{orig}) of its current label and use it as its new label. On the Connection Machine this is done using the general communication routine **get**. To ensure that the algorithm eventually gives the correct result, at each stage a local label propagation step must be performed as well.

By itself this method will correctly label any lattice but performs very badly, because in many places labels propagate only with the local step, on paths that can be very long. To overcome this problem we have added an important supplement – an inverse step which propagates information large distances in the opposite direction and proceeds as follows. Each site saves its label, and then performs a local iteration. It then compares its current label with that old label and, if they are different, sends the current label to the originating site of its old label. For this step we use the Connection Machine routine **send_with_minimum**, for which any site that is sent more than one value keeps only the smallest. Each site then takes the minimum of the labels it is sent, if that is smaller than

its current label.

Doing a *send* step before each *get* means that if the label of any site is changed, the new label is then propagated immediately (by the *get*) to all sites with the old label. Thus we can wholly relabel a large area, or subcluster, in one step as soon as it contacts another large subcluster with a smaller label. In Fig. 3 we show the average number of iterations required to label the Ising model clusters. The data fit perfectly to a logarithmic increase with the lattice size.

The costliest parts of this algorithm are the *get* and *send* steps, which require general communication routines which take about ten times longer than local grid communications on the CM-2. This is compensated of course by the value of the information which is passed over large distances. However a way of doing some of the work by a less costly method will reduce the total running time. Since in the first few iterations the *get/send* step is able to do very little useful work, at a large cost, it is always faster to wait for i initial iterations before using it. Also, depending on the machine specifics and the efficiency with which *get* and *send* are implemented, it may be more efficient to do the *get/send* step only at every pth iteration of the algorithm, with just local label propagation for the other iterations. The parameters i and p can be tuned to optimize the algorithm for any particular application, system size, and parallel computer. We found that for our application on the CM-2, $p = 1$ or 2 and i between 4 and 8 generally gave the best results.

We also note that for large lattices it is not necessary to *get* the label of the originating site for every point since, in any sizable cluster, most neighboring points belong to the same cluster. We have experimented with having only a portion (we use a quarter) of the sites perform a *get*. The new labels received are then transferred to neighboring sites by the subsequent local iteration. Whether this modification proves to be the fastest option depends of course on the communications hardware and software of the particular machine. For the CM-2, the time for each *get* is roughly halved, although the number of iterations required is increased slightly, so there is a trade-off. For an earlier implementation of the *get* routine on the CM-2 this method was substantially faster; however, an improved *get* now means that it is about 10% faster to do the *get* step at every site.

Another way of improving the performance of this algorithm is to combine it with the

scan operation. Using *scan* in some of the initial iterations pushes the label more efficiently over moderate distances. On the CM-2, doing a *get* and a *send* is about as expensive as doing the 4 *scans* (one in each direction) of a *scan* step, however it provides a much better way to propagate labels by making large changes at ever increasing length scales, and of handling the large, irregular and labyrinthine clusters for which the *scan* algorithm fares poorly. Including a few initial *scan* steps makes the algorithm slightly more efficient at large VP ratios (i.e. larger lattice sizes), but again this will be highly dependent on the specifics of the problem and the machine.

Of course we continue to use connection improvement for the local steps, and see a benefit for all the variations of the algorithm. Fig. 7 shows the average time per site to label lattices of different sizes. It is evident that between sizes of 128 and 512 the efficiency of large VP ratios reduces the average time, while for larger lattices this behavior subsides and it is dominated by the increase in the number of iterations needed to converge.

The average labeling time per site for the basic *get/send* algorithm ($i = 0$, $p = 0$, and no *scans*), as well as the optimized algorithm, is shown in Fig. 4. We can see that this method is substantially faster than the multi-scale algorithm.

5 Discussion and Conclusions

Our labeling algorithms are very general, and can be applied to any application where component labeling is necessary, such as percolation[24], image analysis[11], and for the various cluster Monte Carlo algorithms which have been proposed for many different spin models[4, 5, 9].

We have presented numerical evidence that the average number of iterations required by our algorithms to label percolation-like Swendsen-Wang clusters in the Ising model, which are highly irregular in both shape and size, scales with the logarithm of the lattice size. Up to corrections caused by differing VP rations on the Connection Machine, the times required for the labeling using these algorithms scale as $(\log_2 L)^2$ per lattice site.

There is only a subtle difference between the multi-scale and *scan* algorithms: both methods look at connections at distances 1, 2, 4, etc., but for multi-scale we also do a com-

parison at each distance and set the connection accordingly. This connection improvement is enough to give the multi-scale algorithm significantly better scaling behavior.

Our multi-scale algorithm is simpler than that of Brower *et al.*[15], and appears to scale better with increasing lattice size. For the lattice sizes of interest (of order 1024×1024), our optimized algorithm gives an average labeling time for the Ising problem of 6.5 microseconds per site on a 16K CM-2 running at 7 MHz, which is comparable with the time of 6.0 μs per site obtained by Brower *et al.* for the same size machine. However this time for the optimized *get/send* algorithm is substantially better, at 2.6 μs per site.

These kind of SIMD algorithms work quite well on massively parallel fine grained SIMD machines like the CM-2, as long as the objects to be labeled are fairly small, for example in image processing applications such as analyzing images on a radar screen. However fine grained SIMD parallelism does not usually work well for problems which are very irregular and require a lot of non-local communication. Unfortunately the clusters to be labeled in spin model and percolation applications are very large and irregularly shaped, and we would therefore expect that it would be very hard to get good performance for labeling algorithms on these problems using fine grained SIMD machines. This is reflected in our results, since cluster labeling for the Ising model can be done at a rate of about 5 μs per site on a single IBM RS/6000-550 workstation, compared to 2.6 μs per site with our best algorithm on a 16K CM-2.

However we have previously obtained quite good efficiencies on coarse grained MIMD machines for parallel component labeling algorithms which use only local propagation of labels[12], and thus do not scale well for very large numbers of processors. Incorporating the above multi-scale and general communication (*get/send*) ideas into these MIMD algorithms promises to allow us to greatly improve their efficiency and scalability, and thus exploit the power of large MIMD parallel supercomputers such as the nCUBE, the Intel machines, and the CM-5.

Note added: After this work had been substantially completed, and preliminary results reported at a conference[19], we found that the *get/send* algorithm had been independently proposed by P. Rossi and G.P. Tecchiolli[25].

Acknowledgements

This work was done using Connection Machines at the Northeast Parallel Architecture Center at Syracuse University, Sandia National Laboratory, and Rice University. Work supported in part by the Center for Research on Parallel Computation with NSF cooperative agreement No. CCR-9120008, and a grant from the IBM Corporation.

References

[1] G. Parisi, *Statistical Field Theory,* (Addison Wesley 1988).

[2] *Monte Carlo Methods in Statistical Physics, Topics in Current Physics 7, Second Edition*, Ed. K. Binder (Springer-Verlag, Berlin, 1986); *Applications of the Monte Carlo Method in Statistical Physics, Topics in Current Physics 36, Second Edition*, Ed. K. Binder (Springer-Verlag, Berlin, 1987).

[3] N. Metropolis *et al.*, *J. Chem. Phys.* **21**, 1087 (1953).

[4] A. D. Sokal, in *Computer Simulation Studies in Condensed Matter Physics: Recent Developments*, eds. D. P. Landau *et al.* (Springer-Verlag, Berlin-Heidelberg, 1988).

[5] A. D. Sokal, in Proc. of the International Conference on Lattice Field Theory, Tallahassee, October 1990, *Nucl. Phys. B (Proc. Suppl.)* **20**, 55 (1991).

[6] R. H. Swendsen and J.-S. Wang, *Phys. Rev. Lett.* **58**, 86 (1987).

[7] U. Wolff, *Phys. Rev. Lett.* **62**, 361 (1989).

[8] U. Wolff, in Proc. of the Symposium on Lattice Field Theory, Capri, September 1989, *Nucl. Phys. B (Proc. Suppl.)* **17**, 93 (1990).

[9] J.-S. Wang and R. H. Swendsen, *Physica A* **167**, 565 (1990).

[10] E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice* (Prentice-Hall, Englewood Cliffs, N.J., 1977); E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, (Computer Science Press, Rockville, Maryland, 1978).

[11] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, (Academic Press, New York, 1982).

[12] C. F. Baillie and P. D. Coddington, *Concurrency: Practice and Experience* **3**, 129 (1991).

[13] R. G. Edwards, X.-J. Li and A. D. Sokal, *Sequential and Vectorized Algorithms for Computing the Connected Components of an Undirected Graph*, in preparation.

[14] D. Hillis, *The Connection Machine*, (MIT Press, Cambridge, Mass., 1985).

[15] R. C. Brower, P. Tamayo and B. York, *J. Stat. Phys.* **63**, 73 (1991).

[16] E. N. Miranda, *Physica A* **175**, 229 (1991).

[17] *Programming in Paris*, (Thinking Machines Corporation, Cambridge, Mass., 1989).

[18] J. Apostolakis, P. Coddington and E. Marinari, *Europhys. Lett.* **17**, 189 (1992).

[19] Talk presented at the conference "Physics Computing '91", San Jose, California, June 1991.

[20] W. D. Hillis and Guy L. Steele Jr., *Comm. of the ACM* **29**, 1170 (1986).

[21] B. A. Galler and M. J. Fisher, *Commun. ACM* **7**, 301 (1964); D. E. Knuth, *Fundamental Algorithms*, vol. 1 of *The Art of Computer Programming* (Addison-Wesley, Reading, 1968).

[22] J. Hoshen and R. Kopelman, *Phys. Rev. B* **14**, 3438 (1976).

[23] Y. Shiloach and U. Vishkin, *J. Algorithms* **3**, 57 (1982).

[24] D. Stauffer, *Introduction to Percolation Theory*, (Taylor and Francis, Philadelphia, 1985).

[25] P. Rossi and G. P. Tecchiolli, *Finding Clusters in a Parallel Environment*, unpublished.

Tables

algorithm	y
local	1.08(2)
scan	1.09(3)
local improved	1.01(2)
scan improved	0.84(3)

Table 1: Exponents y of computational slowing down for some simple component labeling algorithms applied to clusters of Swendsen-Wang bonds for the Ising model.

Figure Captions

Figure 1: An illustration of connection improvement. The original bonds are shown as the thick lines. If at some point in the labeling algorithm it is found that sites i and j (denoted by the filled circles) have the same label, then a new bond (the dashed line) is introduced, so that changes in the labels are now propagated faster between these two points. For the multi-scale algorithm the same idea is used, except that i and j do not have to be neighboring points.

Figure 2: A log-log plot of the average number of iterations versus lattice size for labeling Ising model clusters using the local propagation and *scan* algorithms, with and without connection improvement. The errors are not shown, but are smaller than the points.

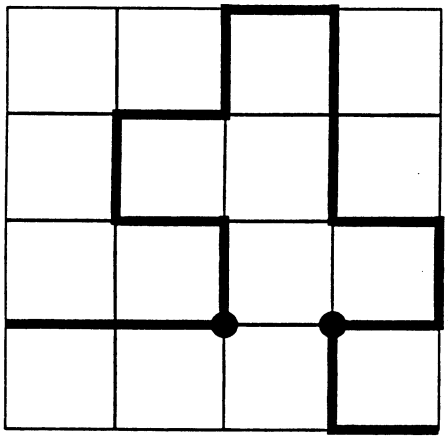
Figure 3: Average number of iterations versus $\log_2 L$ for labeling Ising model clusters using the full depth multi-scale and the *get/send* algorithms.

Figure 4: Average times per iteration per site versus $\log_2 L$ for labeling Ising model clusters on the CM-2 using the *scan* with connection improvement, multi-scale (optimized and full depth) and *get/send* (with and without optimization) algorithms.

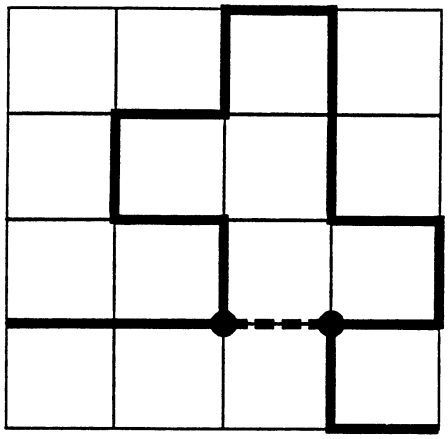
Figure 5: The number of connections at different levels (shown at right) as a function of iteration number for the multi-scale algorithm, for a lattice of size $L = 1024$.

Figure 6: Average times per iteration per site on the CM-2 for labeling Ising model clusters using the multi-scale algorithm as a function of the slope parameter, for different lattice sizes and maximum depths.

Figure 7: Average times per iteration per site as a function of lattice size for labeling Ising model clusters on the CM-2, for different values of the parameters for the *get/send* algorithm. i_{send} is the number of steps without a *get* or *send*. Here the interval between *get*s is $p_{get} = 1$, while $p_{send} = p_{scan} = 2$. l_{scan} denotes the number of steps after which *scan*s are not performed.



(a)



(b)

Figure 1

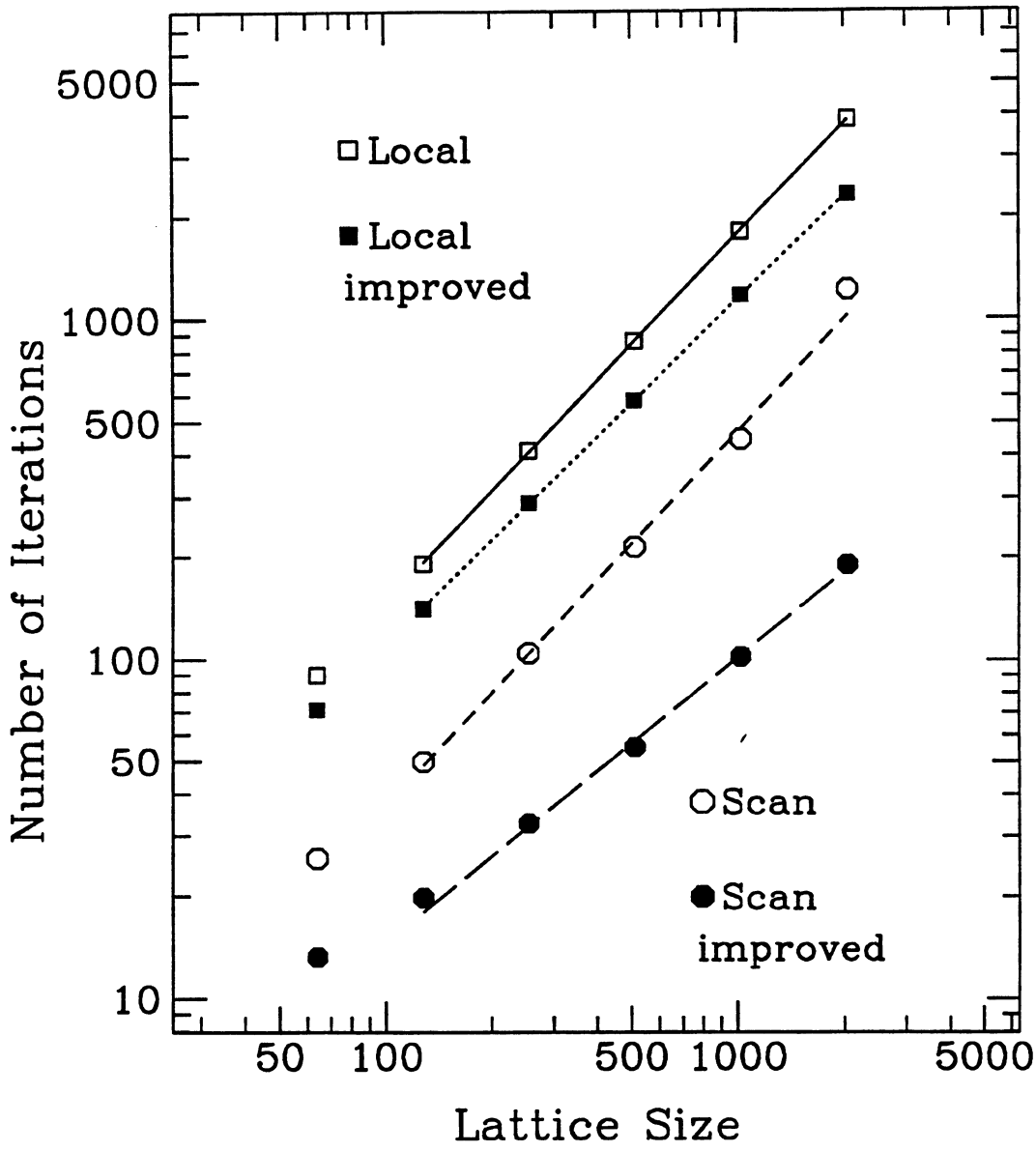

5000
1000
500
100
50
10
Number of Iterations
Local
Local improved
Scan
Scan improved
50
100
500
1000
5000
Lattice Size

Figure 2

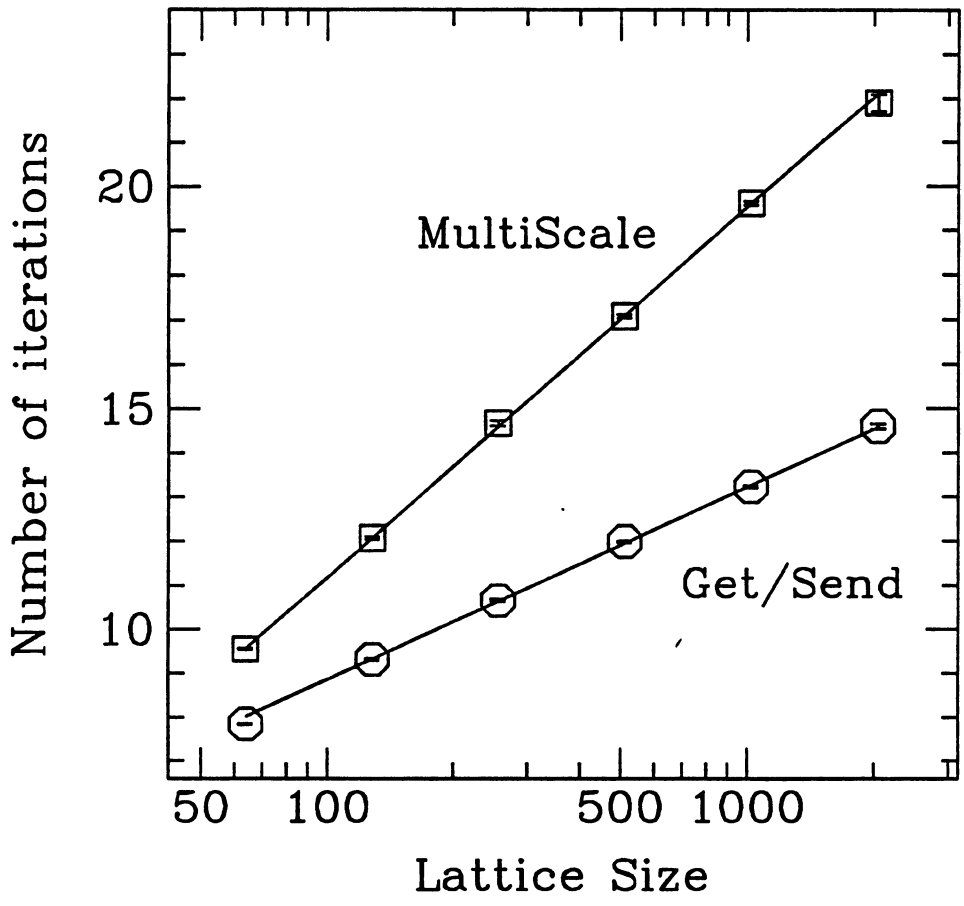

Number of iterations
20
15
10
MultiScale
Get/Send
50
100
500
1000
Lattice Size

Figure 3

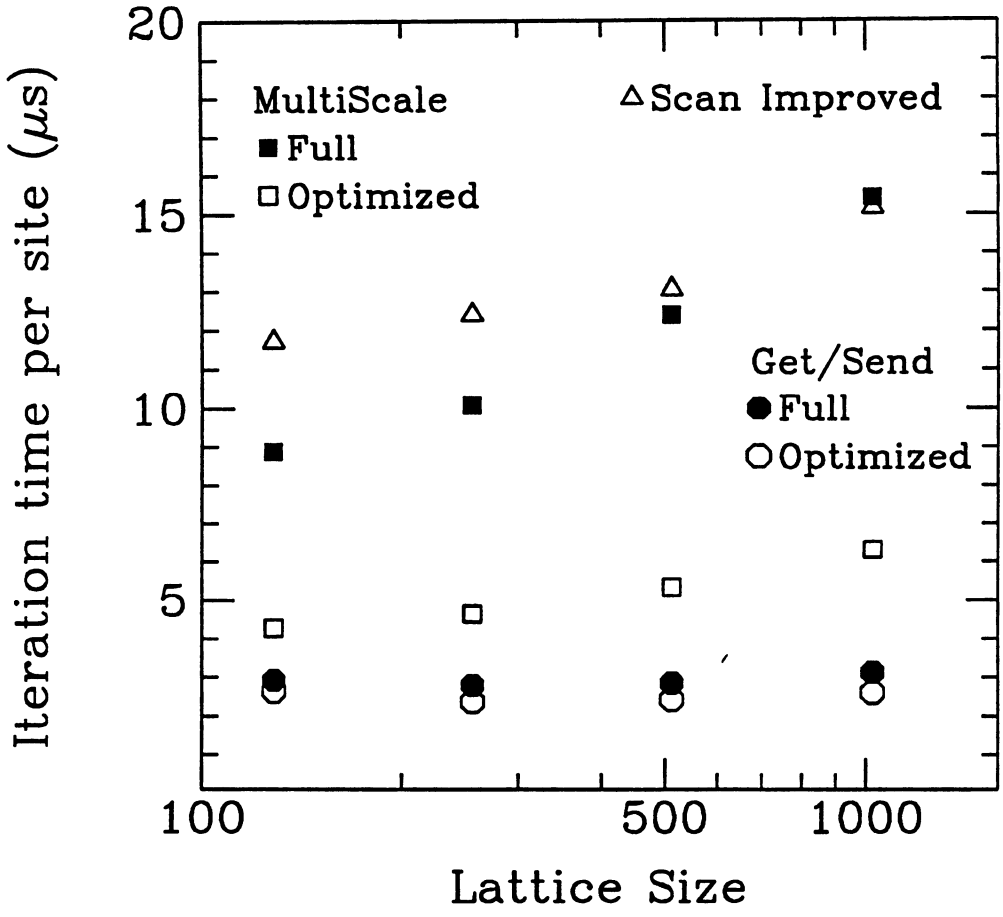

20
15
10
5
100
500
1000
Lattice Size
Iteration time per site (μs)
MultiScale
Full
Optimized
Scan Improved
Get/Send
Full
Optimized

Figure 4

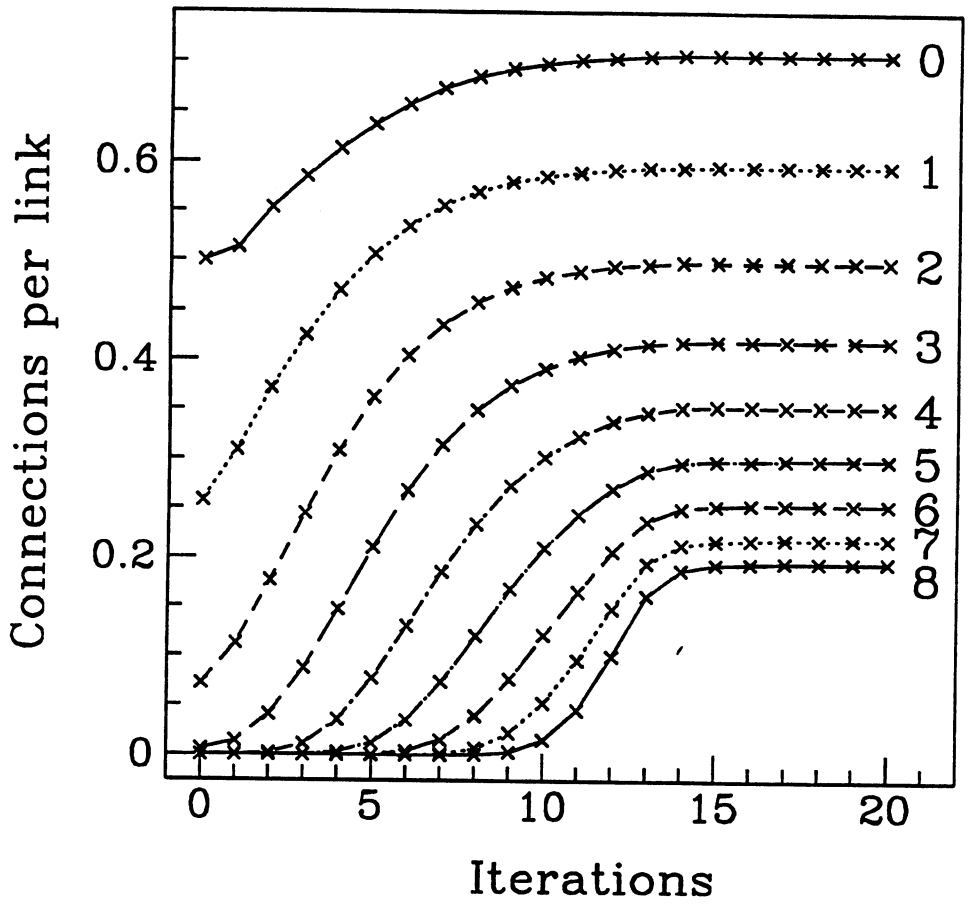

Connections per link
0.6
0.4
0.2
0
0
5
10
15
20
Iterations
0
1
2
3
4
5
6
7
8

Figure 5

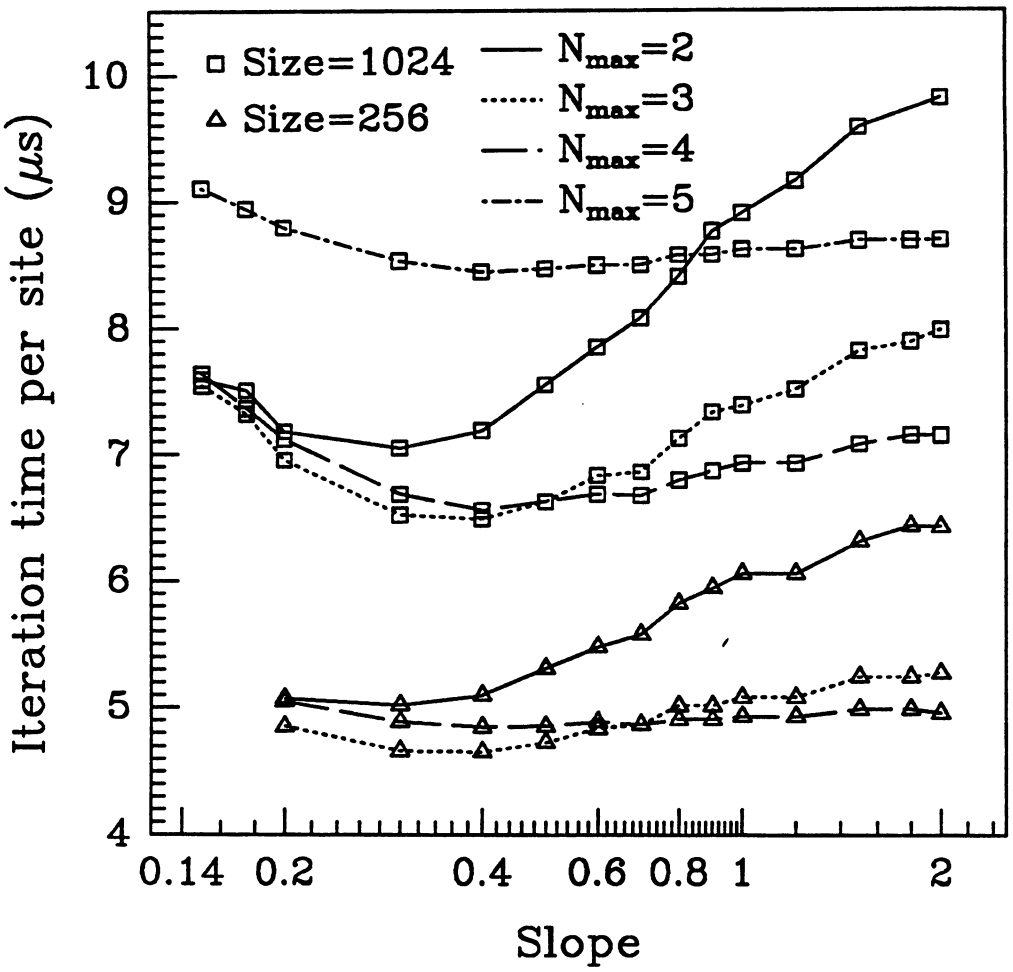

Size=1024
Size=256
N_{max}=2
N_{max}=3
N_{max}=4
N_{max}=5
Iteration time per site (μs)
10
9
8
7
6
5
4
0.14
0.2
0.4
0.6
0.8
1
2
Slope

Figure 6

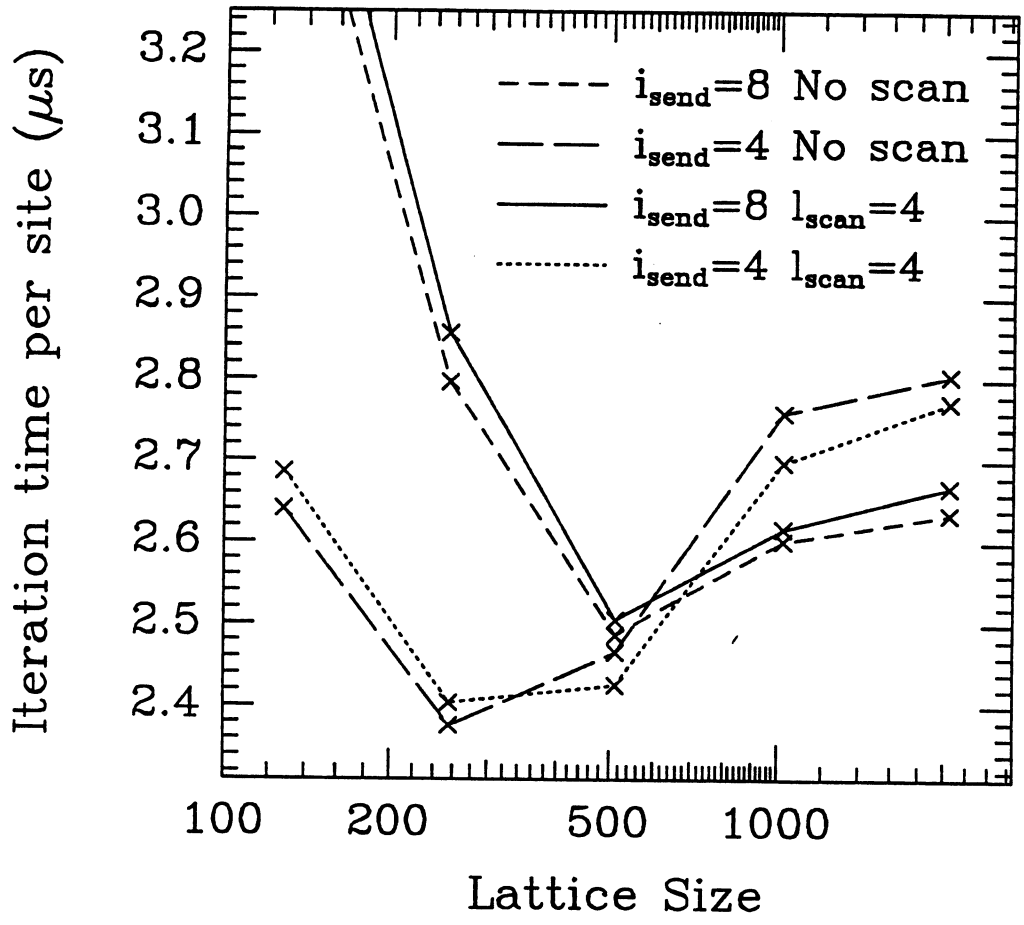

Iteration time per site (μs)
3.2
3.1
3.0
2.9
2.8
2.7
2.6
2.5
2.4
i_{send}=8 No scan
i_{send}=4 No scan
i_{send}=8 l_{scan}=4
i_{send}=4 l_{scan}=4
100
200
500
1000
Lattice Size

Figure 7