JOURNAL OF COMPUTATIONAL PHYSICS 84, 76–89 (1989)

A VLSI Architecture for Percolation Simulation*

PETER D. HORTENSIUS,† HOWARD C. CARD, AND ROBERT D. MCLEOD

Department of Electrical Engineering, University of Manitoba, Winnipeg, Manitoba, Canada R3T 2N2

Received January 5, 1988; revised October 11, 1988

The basis of an efficient VLSI architecture for parallel computations involved in the simulation of the percolation model is presented. This architecture provides a spatially distributed set of pseudorandom numbers, which are required in the local non-deterministic decisions at the various sites in the lattice, using pseudorandom number generators based upon cellular automata. It is shown that the time-intensive task of sampling the percolation configurations is expedited by the inherent parallelism of this approach. Furthermore, the architecture can also be used to group occupied sites into clusters in parallel and report pertinent information to a host computer. In this sense it acts as a hardware expert, or percolation coprocessor, on the computer system bus. This architecture can provide computational speedup of many orders of magnitude over conventional simulation using a serial computer. Measurements from a prototype constructed using a custom VLSI chip implementation indicate that a hypothetical 1000×1000 square lattice could be completely updated in 50 ns. The validity of this approach is verified by computer simulation of the behaviour of the architecture which derived the correct critical exponents for the percolation model.

I. INTRODUCTION

In this work we will use concepts developed in [1, 2] which are based on the recent discovery that effectively random behaviour may be derived from elementary or primitive 1-dimensional (1D) cellular automata arrays even though the local logical rules are deterministic [3, 4].

A novel architecture for the simulation of the percolation model will be described. Here we will make extensive use of computer simulation since the objective of this correspondence is to show that correct results are obtained from a computer architecture which will speed up such simulations. Subsequently, results will be shown which are derived from the proposed architecture. Many of the 2-dimensional problems which will be discussed here have exact analytic solutions [5] but are nevertheless employed since they are the easiest to understand and to make quantitative comparisons with. One should also note that many percolation

* This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Province of Manitoba's Strategic Research Grant Program, and the Canadian Microelectronics Corporation.

† Current address: IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

0021-9991/89 $3.00

problems, when considered in a higher dimensionality than two can only be analysed via computer simulation. Here we will examine three of the six common critical exponents for the percolation model using solution techniques taken from Sur [6] and Kirkpatrick [7]. The calculated critical exponents are

(i) The rate of growth of the largest cluster as a function of site probability, derived from the scaling relation [8]

$$S(L) \approx L^{1/\alpha}, \tag{1}$$

where $S(L)$ is the largest cluster in a lattice of size L.

(ii) The lattice size dependency of the probability that percolation has occurred, derived from the scaling relation [9]

$$\frac{dP'(p, L)}{dp} = L^{1/\nu}. \tag{2}$$

where $P'(p, L)$ is the probability that percolation has occurred in a lattice of size L.

(iii) The percentage of sites in the largest cluster as a function of p, derived from the relation [10]

$$R(p, L) \sim L^{-\beta/\nu} X_1\left(L^{1/\nu} \frac{p - p_c}{p_c}\right). \tag{3}$$

where

$$R(p, L) = \frac{\text{number of sites in largest cluster}}{\text{number of sites in the sample}} \tag{4}$$

and X_1 is an appropriate scaling function of $L^{1/\nu}(p - p_c)/p_c$.

Other common quantities in the percolation model for which critical exponents are often calculated are site correlation or spanning length (the maximum separation of two sites in a cluster), pair connectedness (the probability that two sites separated by a given distance are members of the same cluster), and the conductivity (the conductance across a corresponding random resistor network). Here we have not considered these other quantities, but the three quantities which we do study are representative of the calculations which must be carried out in order to study the percolation model. It is not expected that any uncalculated critical exponent will deviate further from its known or expected value than those critical exponents which are calculated in this work.

II. Percolation Architecture

The computational work in any percolation simulation on a typical serial computer consists of the actual generation of the percolation lattice with site probability

p and the calculation of the appropriate quantity of interest. The architecture which we consider here will be oriented towards the simple 2-dimensional square lattice. Other lattice types and dimensions can be simulated using the same architectural technique with an approrpriately modified interconnection scheme. To simulate percolation on a lattice we must generate an independent pseudorandom number for each site, compare it with a given probability p, and occupy the site accordingly. This operation is repeated over the entire $N = L^2$ sites of the square lattice. Therefore, we require at least $O(N)$ time to generate a single copy of the lattice using a serial computer. For the three critical exponents above we must calculate the probability of percolation and the size of the largest cluster at any probability p. To calculate whether the lattice has a percolating cluster and the size of the largest cluster requires no more than $O(N^2)$ time using the Hoshen and Kopelman cluster labelling algorithm [11].

An obvious question to ask is which aspects of the percolation model simulation can be accomplished in parallel. While some portions of the algorithm which can be parallelised are fairly obvious[1] the best method to implement such a parallel computer is not. We note that the larger the lattice which can be simulated the greater the interest in the simulation. The largest simulation which has presently been carried out used a $160{,}000 \times 160{,}000$ square lattice [12].

The remaining problem is the calculation of the critical exponents. It is possible to build a special computer which can both simulate the system and calculate the critical exponents for the percolation model. However, this is not necessarily the most expedient solution. The disadvantages of such an approach stem from the fact that the actual calculation of the critical exponents requires data memory and floating point calculations. However, it is well understood by anyone who has attempted to simulate the percolation model that very little time is actually spent calculating the desired exponents. Most of the computer time is used in generating new lattice configurations and grouping the occupied sites into clusters. Furthermore, operations using the clusters are generally very rapid given that site clustering has already occurred. Therefore, little is to be gained by building a computer dedicated solely to the calculation of critical exponents. Much can, however, be gained by building a device which can generate new lattices and form clusters quickly. This device would act as a special purpose *coprecessor* to a general purpose host computer and because of the nature of its specialised task could be made to operate very efficiently. Therefore, we will consider an implementation where a host computer will determine the actual critical exponents and do operations on clusters generated by a special purpose *percolation coprocessor.*

In order to efficiently execute the percolation simulation the architecture of Fig. 1 is proposed. Each processor consists simply of a pseudorandom number generator (PRNG), comparator, and storage element, or site latch. The site probability, p, is made available to each comparator over a global bus and the pseudorandom number from the PRNG is compared to it. Finally, the site latch is turned high

[1] Occupying sites based on the site probability.

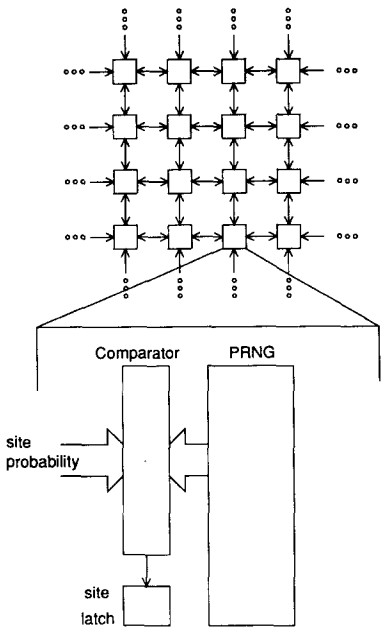

FIG. 1. Basic percolation simulation architecture.

($>p$) or low ($\leqslant p$) accordingly. Each site in the lattice is assigned a unique processor. Therefore, after each clock cycle we have defined a new percolation lattice, as compared to at least $O(N)$ time for a serial updating technique. In addition, the time for a single clock cycle is quite small ($\leqslant 50$ns). Each simulation step on a typical serial computer with a software PRNG is comparatively large ($\geqslant 5\mu$s) for a single site. In addition of course, to update the entire lattice the serial method must be applied N times; the present approach only once. The overall speed improvement is approximately $100N$. It should be noted that the size of the lattice which can be simulated is restricted by the number of processing sites available which is in turn dependent in an inverse way on the size of each processor.

The heart of any nondeterministic computation is the pseudorandom number generator. For simulations such as those discussed here it is known that a *non-random* PRNG will result in erroneous simulation results. Thus, it is critical that the PRNG used be of high quality. However, at the same time we are restricted to using only a small area for the PRNG since we are targeting a VLSI implementation. It has been shown that conventional algorithmic PRNG techniques such as the multiplicative congruential and additive feedback generators are not suitable for an architecture such as that described here because of silicon area and processing time considerations [1]. In addition, typical hardware techniques such as those based on shift registers are also unsuitable due to poor quality randomness or excessive area and time. In [1, 2] a hardware PRNG based upon cellular automata

(CA) is developed and is shown to possess more desirable implementation properties (small area, local communication) and/or randomness than conventional PRNG techniques. However, the architecture is not PRNG dependent and another PRNG technique may be substituted if desired.

A cellular automaton evolves in discrete steps with the next value of one site determined by its previous value and that of a set of sites called the neighbour sites. The extent of the neighbourhood can vary depending, among other factors, upon the dimensionality of the cellular automaton under consideration. In a simple 1-dimensional cellular automaton, the next value at a site depends only on its present value and the values of the left and right neighbours. The cellular automaton may posses null boundary conditions (i.e., the first and last sites consider their missing neighbour site to always have a zero value) or be cyclically connected (i.e., one considers the cellular automaton to form a ring thereby making the first and last sites neighbours). Here only binary 1-dimensional cellular automata with two neighbour sites (left and right) will be considered, but it is possible to use any desired modulus, dimension, or neighbour set. For binary cellular automata of this type each site must determine its next value on the basis of the eight possible present values of itself, and the left and right neighbours (i.e., 000, 001, 010, etc.). The truth table for the next state values corresponding to each possible input form a binary number the decimal equivalent of which is referred to as the rule number under the classification scheme of Wolfram [13].

While the description of 1-dimensional cellular automata is very simple, the different CA rules are capable of a very wide range of global behaviour. Wolfram has characterised four basic classes of behaviour in 1-dimensional cellular automata [14]. Class 1 automata evolve to homogeneous final global states, class 2 to periodic structures, class 3 exhibit chaotic behaviour, and class 4 yield complicated localised and propagating structures. For pseudorandom number generation it has been shown that several class 3 CA rules produce high quality pseudorandom number sequences [1, 2, 4]. In the present paper we employ the CA rule based PRNG discussed in [1, 2]. It should be noted that, as with all PRNGs, some correlation in the number sequence is present [15]. But it is unclear whether the particular correlations present in CA-based PRNGs will affect the results of a percolation simulation. Again, we emphasize the PRNG technique independence of the architecture.

The regularity of processing sites in such a percolation processor makes it an

FIG. 2. CMOS layout of 16 bit percolation site processor.

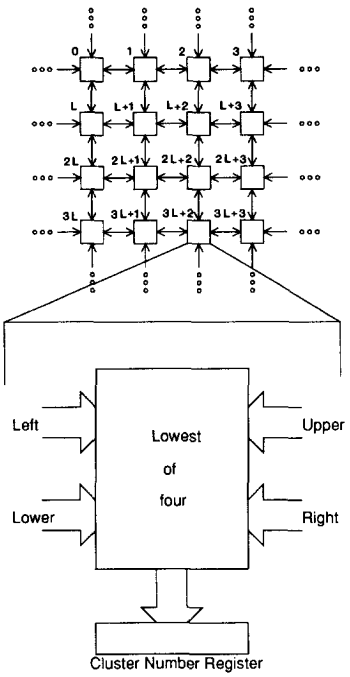

FIG. 3. Architecture to group occupied sites into clusters.

ideal candidate for VLSI implementation. The size of such a processor is directly dependent on register size. However, we can make estimates based on a fixed register size and scale up or down as appropriate for different register sizes. A 16-bit site processor layout is pictured in Fig. 2. The size of this processor using a 3μm single metal CMOS technology is 0.838 mm^2. Therefore it is possible to have 25 such processors simulating a 5×5 lattice on a single 4.8×4.8 mm die.[2] It is straightforward to implement the site processors in such a way as to be able to combine chips to form larger lattices. However, it is not realistic to consider employing a unique processor for each site in the lattice for lattices of arbitrary size. Therefore, we will restrict ourselves to lattices of $L \leqslant 1000$ for which we assume there is a unique processor for each site in the lattice. We will return to the problem of lattices larger than 1000×1000.

It is possible to use the proposed percolation architecture solely to dramatically increase the speed of updating the lattice. However, if we could calculate the size of the clusters and other relevant properties we could accelerate the simulation even

[2] This technology (3 μm CMOS) available to us at present is not the state of the art. Implementation using more advanced technology would dramatically increase the number of site processors per chip. For example, on a triple metal 1μm technology using a 10×10 mm die one could easily place over 1000 site processors.

more dramatically. It is possible to quickly group the occupied sites of the lattice into clusters if we superimpose the multiprocessor architecture of Fig. 3 onto the architecture of Fig. 1. Here we assign each processor a unique cluster number corresponding to its location in the lattice. For example, in Fig. 3 we have assigned processors in the first row to have values 0 to $L-1$, the second row processors are assigned numbers L to $2L-1$, and so on. This *percolation computer* operates as follows. First we utilise the underlying architecture of Fig. 1 to decide which sites are occupied. Occupied sites take their assigned cluster value while unoccupied sites take on the value of ∞, or some other appropriately large number. We then proceed to synchronously update all sites according to the following algorithm. If a site is occupied, the next cluster value is selected as the lowest of its four neighbouring cluster values (remember we are presently considering only square 2-dimensional lattices) and itself. For example, in Fig. 4a we see an 8×8 lattice with $p = 0.5313$, Fig. 4b shows the same lattice initialised using the above cluster numbering assignment, and in Fig. 4c we see the cluster numbers one synchronous update later. The synchronous updating procedure continues to take place until all sites belonging to the same cluster have had their cluster numbers merged together. The worst case time for this procedure would be $L(L-1)/2$. The final cluster numbering configuration for Fig. 4a is shown in Fig. 4d.

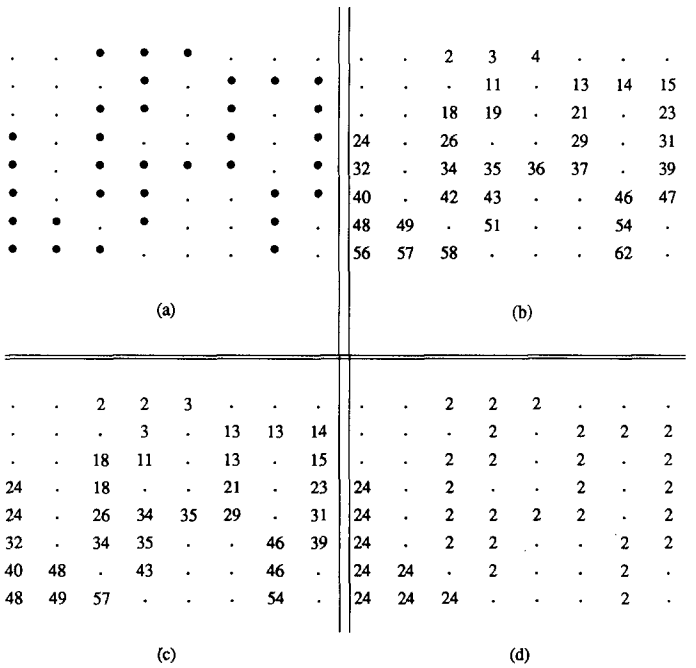

FIG. 4. Operation of percolation computer on an 8×8 square lattice: (a) 8×8 square lattice with $p = 0.5313$; (b) initialised lattice with cluster number assignment; (c) cluster numbers after one update; (d) final cluster numbering assignment.

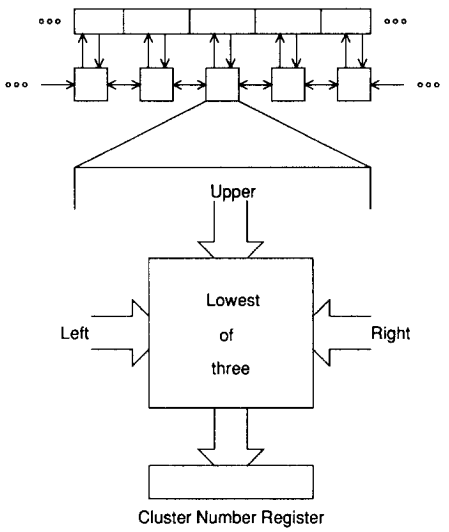

FIG. 5. Architecture to group sites into clusters for row at a time lattice generation.

Determining whether or not the largest cluster is infinite is quite easy if we realise that a spanning cluster must be present both at the top and bottom of the lattice. Therefore, if any sites on the bottom of the lattice have a final cluster number less than L then we have a spanning cluster. Therefore, for a 1000×1000 lattice we can group the occupied sites into clusters and determine if a spanning cluster exists in at most 500,000 update steps (Fig. 5). A small prototype has been constructed using a 3μm single metal CMOS technology and measurements of the circuit have shown the operating speed to be at least 20 MHz. In order to simulate large lattices it is necessary to construct the percolation coprocessor using many chips (Fig. 6).

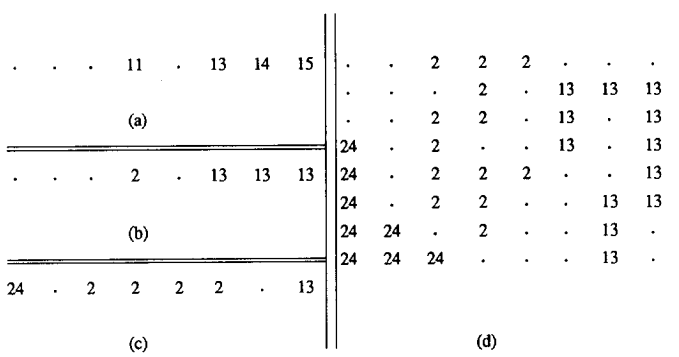

FIG. 6. Operation of row at a time percolation computer on lattice of Fig. 4; (a) initialised second row; (b) second row after cluster-numbering completed; (c) fifth row after cluster-numbering completed; (d) fully cluster-numbered lattice.

Therefore, the simulation speed may be limited by the rate at which interchip communication can be accomplished. It is not unreasonable to consider a board level switching rate of 50 ns, so for the purposes of this work we will consider this to be the maximum clock speed of the current architecture even though the advanced VLSI technologies which will be required for a final version of this architecture will allow the internal clock rate on the chip to be much greater. Thus, in about 25 ms we can generate a 1000×1000 lattice, group the occupied sites into clusters, and determine if a spanning cluster is present.

To determine the size of the largest cluster is a much more difficult problem. However, it is possible for the host computer to offload the final cluster numbers from the percolation computer and count the number of sites in each cluster. This remains a significant enhancement over serial computer simulation techniques since much of the time is spent grouping occupied sites into clusters. Other calculations for quantities such as pair connectedness and site correlation are also significantly faster since the clusters have already been formed. Finally, we note that is also possible to place processing elements which can perform cluster sizing calculations into the architecture. However, such processing elements are considerably more complex, especially since they require data memory, so they are not considered in this work.

III. Simulation Results for the Percolation Computer

Simulations of the percolation computer with cellular automata-based PRNGs were carried out and yielded the following results for the exponents defined

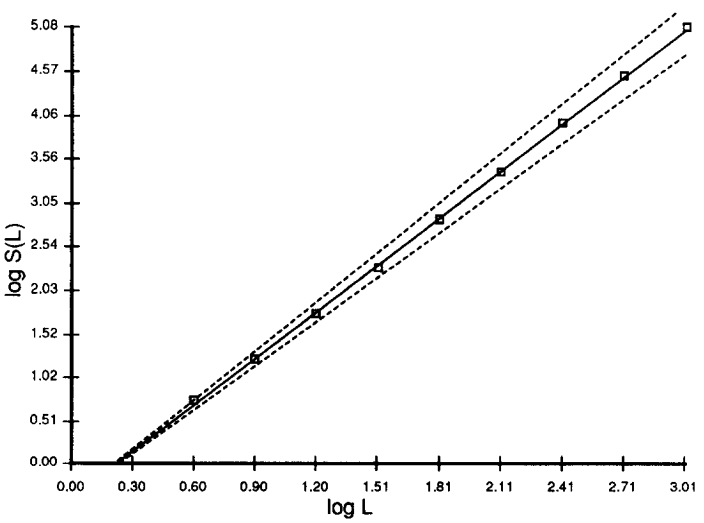

FIG. 7. Size of $S(L)$ at p_c versus L using the percolation computer.

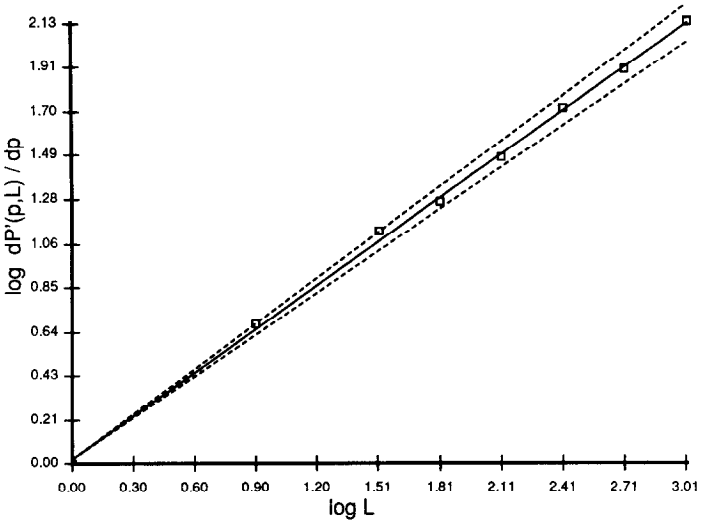

FIG. 8. Log–log plot of $dP'(p, L)/dp$ vs L using the percolation computer.

previously in Eqs. (1) to (4). The percolation threshold was found to be 0.5915 ± 0.0023. The rate of increase of the largest cluster $S(L)$ vs lattice size is shown in Fig. 7 from which we calculate $1/\alpha$ to be 1.800 ± 0.096. A plot of $dP'(p, L)/dp$, where $P(p, L)$ is the probability the percolation has occured on a lattice of size L is shown in Fig. 8, yielding the value of the critical exponent ν to be 1.434 ± 0.030. Likewise for $R(p, L)$, the percentage of occupied sites in the largest cluster, we observe the behavior of Fig. 9 from which we derive a value of

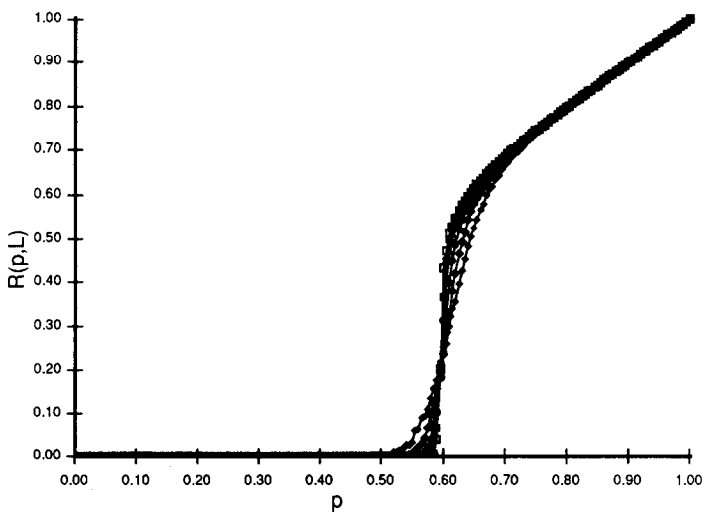

FIG. 9. $R(p, L)$ versus p for various lattice sizes using percolation computer.

TABLE I

Percolation Critical Exponents

Exponent	Standard	Present work	Others	Reference
p_c	0.5916 ± 0.0022	0.5915 ± 0.0023	0.5928	[8]
$1/\alpha$	1.798 ± 0.093	1.800 ± 0.096	1.89	[8]
ν	1.439 ± 0.015	1.434 ± 0.030	1.35	[16]
β	0.144	0.145	0.14	[16]

Note. Standard refers to standard serial computer percolation simulations done for this work; Present work refers to simulation results for the present percolation computer; Others refers to representative results which have been reported elsewhere.

β of 0.145. These results are summarized in Table I. There is a small discrepancy between the results which have been calculated here and those which have been published elsewhere [8, 16]. However, there is close agreement between the results for the percolation computer and a standard percolation simulation based on algorithmic PRNG using a serial computer. Therefore, we can conclude that the proposed parallel percolation computer yields similar critical exponents to normal serial percolation simulation. The discrepancies between the critical exponents calculated here and those published elsewhere may be due to several factors such as smaller register size (here 16-bit was used) and a smaller number of samples. However, it is encouraging that percolation simulations using the proposed percolation computer and a standard serial computer method yielded the same results.

It would be much more expensive in terms of both area and time to use any PRNG other than the CA-based schemes employed in this approach. In addition, one can see that, because of the vast number of PRNGs required, the topological regularity of the CA approach provides a very clear advantage. Finally we note that it has been found that standard LFSR-based and some multiplicative congruential PRNGs are inadequate for Monte Carlo simulations [17] since they do not produce correct critical exponents. We observe that the critical exponents calculated using the CA-based percolation computer provided approximately correct values.

IV. Using Renormalisation on the Percolation Computer

Extensions to the percolation architecture could include the use of renormalisation group principles [18] to extrapolate the infinite lattice critical exponents. The basic concepts required to implement renormalisation as applied to percolation are quite straightforward. Essentially we slowly integrate out small scale fluctuations and obtain information on successively larger and larger scales. This is done by replacing a small block of sites on the lattice with one site representing gross, or

average, behaviour. For example, in majority rule renormalisation, a block of 3×3 sites is represented by one occupied site if the majority of sites are occupied and an unoccupied site if the majority of sites are not occupied. This procedure is repeated many times progressively reducing the lattice size by a factor of l for each renormalisation, where l is the size of the block of sites being replaced by a single site. The result is that for $p > 0.5$ the new p_1, representing the density on the renormalised lattice, moves quickly towards a value of 1.0, while for $p < 0.5$ the new p_1 moves towards 0.0. However, for $p = 0.5$ the new p_1 will also equal 0.5. This critical value of $p = 0.5$ derives simply from the majority renormalisation rule and is not associated with the critical percolation value. The critical exponents are extracted from the rate at which the value of p_1 moves towards 1.0 or 0.0. The problem here is that for many lattices simple majority rule renormalisation is not adequate to extrapolate infinite lattice behaviour. In the above example we saw that for $p > 0.5$ the value of p_1 moved quickly towards 1.0. Thus, for $p = p_c = 0.5928$ on the square lattice p_1 will move towards 1.0 and it is not possible to extract critical behaviour since p_1 is not equal to p_c. Therefore, another renormalisation rule is required if we are to study critical behaviour for site percolation on a square lattice using renormalisation techniques. For example, [19] studied renormalisation on a square lattice by replacing a block of sites with an occupied site only if a spanning cluster,

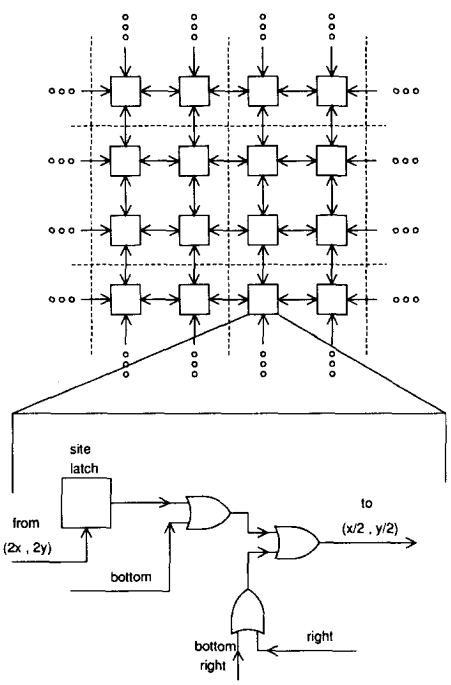

FIG. 10. Renormalisation architecture operating on 2×2 blocks using a renormalisation rule due to [18].

or connecting path, existed in the block. Reynolds [20, 9] utilised a position-space renormalisation procedure whereby a block of 2^d sites was replaced by a single site and d bonds, requiring that the d bonds reflect the connectivity of the block which it is replacing.

In any case we see that construction of hardware to implement any renormalisation procedure, other than the simple majority rule case, requires significant processor resources. An example of a simple renormalisation group architecture is shown in Fig. 10. Here we implement the renormalisation rule of [19]. For simplicity we use a block size of 2×2. To determine whether an infinite cluster exists in a 2×2 block merely requires checking if each row has an occupied site. A renormalised site representing sites (x, y), $(x+1, y)$, $(x+1, y+1)$, and $(x, y+1)$ in the old lattice will be stored in position $(x/2, y/2)$ in the new $L/2 \times L/2$ lattice, necessitating a shift to the left and up by $x/2$ and $y/2$ site processors. This will require additional shifting hardware at each site processor. Finally, we assign cluster numbers to each occupied site in the new lattice and invoke the site clustering process. As larger blocks or more complicated renormalisation rules are considered the associated computing hardware becomes considerably more complex. Thus, a percolation computer implementing renormalisation will not be further considered in this work. However, we note that if a percolation computer is to be constructed which itself calculates the critical exponents, it is probably best to use a renormalisation approach to quickly reduce the amount of cluster data which must be processed and offloaded to the host computer.

Another extension to the percolation computer is the inclusion of different lattices and dimensions other than the 2-dimensional square lattice which we considered here. To include other lattice types, for example, the triangular or honeycomb lattices, one need merely increase the connectivity of the site processors to account for the increased number of neighbours. Otherwise the method of operation is precisely the same. Similarly for higher dimensions one need merely increase the neighbour connections at each site processor to account for the increased neighbour set. Admittedly for $d > 2$ longer physical distances between neighbouring processors will occur due to the increased connectivity, but the method of implementation and operation remains the same. For the processors under consideration here the distance between processors is not the dominant factor relating to operating speed. Rather, actual processing time dominates. No simulations were performed on percolation operating on different lattice types or higher dimensions since it is not expected that the validity of the percolation computer will be affected by having more neighbours. Neither do we expect the computer time for simulations on the percolation computer to increase dramatically as the neighbour set increases.

V. Conclusions

In this correspondence we have shown that VLSI is an appropriate medium for specialised hardware with which to simulate the percolation model. Speedup of

several orders of magnitude has been accomplished. Due to an efficient PRNG and clustering algorithm we can exploit high speeds and minimal clock cycles. Having correct critical exponents justifies the cellular automata-based PRNG, since near the phase transition, correlations in the pseudorandom number sequence will adversely affect the results.

REFERENCES

1. P. D. HORTENSIUS, Ph.D. dissertation, University of Manitoba, Winnipeg, Canada, 1987 (unpublished).
2. P. D. HORTENSIUS, H. C. CARD, AND R. D. MCLEOD, *IEEE Trans. Comput.*, in press.
3. S. WOLFRAM, D. A. LIND, M. S. WATERMAN, P. GRASSBERGER, AND S. J. WILLSON, in *Cellular Automata, Proceedings of an Interdisciplinary Workshop* (North-Holland, Amsterdam, los Alamos, NM, 1984).
4. S. WOLFRAM, *Adv. Appl. Math.* 7, 127 (1986).
5. J. W. ESSAM, D. S. GAUNT, AND A. J. GUTTMANN, *J. Phys. A* **11**, 1983 (1978).
6. A. SUR, J. L. LEBOWITZ, J. MARRO, M. H. KALOS, AND S. KIRKPATRICK, *J. Stat. Phys.* **15**, No. 5, 345 (1976).
7. S. KIRKPATRICK, "Models of Disordered Materials," in *Ill Condensed Matter*, edited by R. Balian, R. Maynard, and G. Toulouse (World Scientific, Singapore, 1983), p. 323.
8. D. STAUFFER, *Introduction to Percolation Theory* (Taylor & Francis, Philadelphia, 1985).
9. P. J. REYNOLDS, H. E. STANLEY, AND W. KLEIN, *J. Phys. A* **11**, No. 8, L199 (1978).
10. M. E. FISHER, in *Critical Phenomena, Proceedings of the International School of Physics, "Enrico Fermi Course 51,"* edited by M. S. Green (Academic Press, New York, 1971), p. 73.
11. J. HOSHEN AND R. KOPELMAN, *Phys. Rev.* **14**, 3428 (1976).
12. D. C. RAPAPORT, *J. Phys. A* **18**, L175 (1985).
13. S. WOLFRAM, *Rev. Mod. Phys.* **55**, 601 (1983).
14. S. WOLFRAM, *Physica D* **10**, 1 (1984).
15. A. COMPAGNER AND A. HOOGLAND, *J. Comput. Phys.* **71**, 391 (1987).
16. R. ZALLEN, *The Physics of Amorphous Solids* (Wiley, New York, 1983).
17. G. PARISI AND F. RAPUANO, *Phys. Lett. B* **157**, No. 4, 301 (1985).
18. K. G. WILSON, *Rev. Mod. Phys.* **47**, 765 (1975).
19. S. KIRKPATRICK, *Phys. Rev. B* **15**, 1533 (1977).
20. P. J. REYNOLDS, H. E. STANLEY, AND W. KLEIN, *J. Phys. C* **10**, L167 (1977).