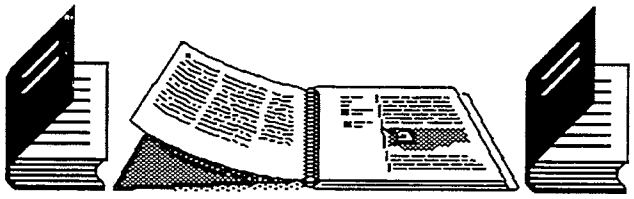


Parallel Computing with Generalized Cellular Automata

By: W. A. Maniatty

In
Parallel and Distributed
Computing Practices

Published by
Nova Science Publishers Inc.
6080 Jericho Turnpike - Suite 207
Commack, NY 11725
Tel: 516-499-3103
Fax: 516-499-3146
Email: Novasci1@aol.com

Parallel and Distributed Computing Practices
Volume 1, Number 1 pp. 31-50
ISSN 1097-2803

PARALLEL COMPUTING WITH GENERALIZED CELLULAR AUTOMATA

W. A. MANIATTY* , B. K. SZYMANSKI*, AND T. CARACO †

Abstract. *Cellular automata* (CA) are fundamental computational models of spatial phenomena, in which space is represented by a discrete lattice of *cells*. Each cell concurrently interacts with its neighborhood which, in traditional CA, is limited to the cell's nearest neighbors. In this paper we discuss *generalized cellular automata* (GCA), an important but unexplored class of CA, in which the cell's interaction domain extends beyond the nearest neighbors. The computational power necessary to run large scale CA (and GCA) models has only recently been available thanks to parallel processing. This paper focuses on implementation and performance of GCA in biological modeling. In particular, we present results of simulating the spread of epidemics and the creation of spatial infection patterns that are important for disease control.

The simulation system is implemented on three different platforms: the MasPar MP-1 SIMD computer, the IBM SP-2 MIMD machine and a network of workstations (NOW) that consists of Sun SPARCstation 5 and UltraSPARC 2's connected via Ethernet. The system presented in this paper has been specialized for simulating a four species spatially explicit model, however, the implementation may be readily modified to represent other models. Simulation results are presented for simple epidemics and vector-borne diseases spread by parasites.

Key words. parallel algorithms, parallel simulation, cellular automata, parallel computing, epidemics, ecology

AMS subject classifications. 68Q22, 68U05, 92D25, 92D30

1. Introduction. Many large-scale natural phenomena arise from concurrent, spatially localized interactions of small entities. Examples are fluid dynamics [46], forest fires [14], population dynamics and spread of epidemics [44]. *Cellular automata* (CA) model such phenomena by discretizing space into a lattice of *cells* [9]. Each cell is in one of a finite set of discrete states and interacts with its neighborhood, referred to as the cell's *stencil*. In CA models each cell updates its value as a function of its current state and its stencil. Despite their simplicity, CA display surprisingly complex behavior. Wolfram [47, 48] classifies the qualitative convergence of two-dimensional CA into the following taxonomies: (1) evolving to a homogeneous state, (2) evolving to a simple separated locally periodic structure, (3) yielding chaotic and aperiodic patterns and (4) yielding complex patterns of localized structures. However, very little is known about predicting CA behavior without running them.

Percolation models [39] are closely related to CA. They embed a graph structure onto a problem space, and model a phenomena as a fluid flowing along the edges from *wet* source vertices to *dry* destination vertices. This means that each cell computes its impact on its neighbors (for regular lattices this forms a dual of CA) [1].

Traditionally, CA, as well as percolation models equate stencils with the nearest neighbors only, although, sporadically more diverse models were sometimes considered[2]. However, many phenomena, especially those studied in biology, ecology and sociology, have longer range interactions. Often interactions between organisms in nature and between humans in society are of such character and the size of a stencil is an important parameter of such interactions. CA models for such phenomena require stencils that extend beyond the cell's nearest neighbors and which can be varied from simulation to simulation. We refer to CA with extended and variable stencils as *generalized cellular automata* (GCA).

This paper focuses on parallel computing environments for GCA used to model biological systems. We concentrate on epidemiological models and compare and contrast several implementations of the simulation system on various parallel architectures. One of the most important modeling

*Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180-3590, ({maniattb,szymansk}cs.rpi.edu).

†Department of Biological Sciences, SUNY Albany, Albany, NY 12222.

[1] One of the anonymous reviewers noted that also lattice gas automata (LGA) and lattice Boltzmann models (LBM) have similar features to the GCA models presented in this paper. For a description of LGA and LBM see for example [12, 41].

[2] For example, Ermentrout and Edelstein-Keshet [19] survey several CA models of biological processes, including an epidemiological model where each cell, (i, j) in a $N \times N$ lattice interacts with the nearest neighbors to the *complementary cell* at location $(N - i, N - j)$

paper we consider GCA for which all stencils are equal in size and in their placement around their generating cells. The stencil size is denoted by δ. The biological results of our research indicate that the stencil size fundamentally influences the qualitative and quantitative behavior of CA and the environment that it models [11, 18].

As time advances from t_i to t_{i+1}, the state of each cell x is selected according to the state transition probability function. We assume that this function is independent of the simulation time, so it could be defined as $p : S \times S^{\delta} \times S \rightarrow [0, 1]$, such that $(\forall s_0, s_1 \ldots s_{\delta} \in S : \sum_{s_{new} \in S} p(s_0, \ldots, s_{\delta}, s_{new}) = 1)$. A GCA, then, can be formally defined as the quadruple (Ω, S, δ, p).

2.1. A Four Species Model. Most epidemic models assume that a pathogen is transmitted when an infective host contacts a susceptible host. Recently, however, greater attention has been directed to vector-borne diseases where individuals of a parasitic species sequentially exploit many host individuals, and can transport the pathogen from an infected to an uninfected host. The direct effect of the parasite on the host population is sometimes small, but the indirect effect (via the pathogen) is often severe.

In the most general form the implemented model can simulate a four-species ecosystem composed of two competing hosts, a parasite feeding on both hosts, and a pathogen that employs the parasite as a vector between host individuals [44]. We assume that a vector-borne pathogen can infect individuals of two host species. The host populations compete for space and once an individual of one host species occupies a cell, its mortality (freeing the cell) is independent of other hosts.

A single parasitic species can occupy a cell only if an individual of either host population already occurs at that cell; the parasite's range cannot exceed its host's range. If the parasite exploits both a host individual that is infected with the pathogen and a nearby uninfected host, the parasite may carry the pathogen from the former to the latter host individual. We allow the parasite to transmit the pathogen both within and between host species. The parasite generally will reduce a host individual's survival and reproduction to a lesser extent than will the pathogen. The parasite may prefer to exploit one of the host species. Hence, the model addresses the question of how behavioral selectivity influences the dynamics of an epidemic and so governs a spatial pattern in an interactive biotic community. Assuming that there is one host species means that each cell can be in one of five possible states, $S = \{1, 2, 3, 4, 5\}$, coding of which is shown in Figure 2.1(a).

2.2. Simulating Epidemics with the Four Species Model. A variety of epidemics can be expressed with the model presented in § 2, including simple epidemics discussed in § 2.2.1, vector-borne epidemics presented in § 2.2.2 and general epidemics described in § 2.2.3.

2.2.1. Modeling Simple Epidemics with the Four Species Model. The classic simple epidemic [4] assumes that the individual is either susceptible or infective. Since the simple epidemic excludes recovery, pathogen loss is eliminated. By replacing parasitism of adjacent hosts by contact between individuals, the model described above can provide a stochastic, spatially explicit representation of the simple epidemic [3]. Parameters governing the gain and loss of the parasite allow us to generalize the notion of "contact" between host individuals. The host grid might represent plants, animals with fixed territories, stable home ranges [37], or households in human populations. For this application, the state space is $S = \{1, 2, 3\}$ and its coding is a subset of the coding shown in Figure 2.1(a). The relative frequency of state 1 (empty cells) remains constant in this application according to the simple epidemic assumption that the host population is constant. A site empty at time $t = 0$ remains so and a cell with a host never becomes empty. The only possible transition in this simple model is from state 2 to state 3.

2.2.2. Modeling Vector-Borne Epidemics with the Four Species Model. The model described in § 2 was also a basis for simulation of a vector-borne disease in which the spread of the pathogen from one individual to another involves a *vector*. A host must be exposed to a parasite residing on a "nearby" infected host to acquire the pathogen. Following the assumptions of the simple epidemic, there is neither host nor pathogen mortality. Hence, the initial density and spatial arrangement of hosts are likely to govern the rate of epidemic progress.

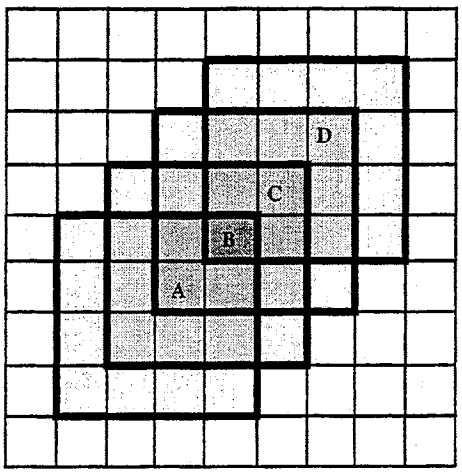


FIG. 3.1. *Overlapping stencils for cells A, B, C, D*

tations are described in § 3.6. The programming language's impact on portability and performance tuning of the MIMD implementation is discussed in § 3.7. Performance results of the various simulation engines are compared and contrasted in § 3.8. Data visualization and interactive simulation tools which were developed to facilitate understanding of the simulation results, are discussed in § 3.9.

3.1. Simultaneous Reduction. Each cell's state transition probability can be expressed as a function of its current state, and the states of cells in the corresponding stencil, as in § 2. Let n_1 and n_2 stand for the lengths of the sides of the stencil, p_1 and p_2 be the offsets (measured from the lower left corner of the stencil) of the affected cell, and $s_{i,j}$ be the current state of the cell (i,j). Then, the state transition probability of cell (m,q) to a new state s' is defined as:

$$\text{Prob}[s_{m,q} \to s'] = \sum_{i=m-p_1}^{m+n_1-p_1} \sum_{j=q-p_2}^{q+n_2-p_2} f(s_{m,q}, s_{i,j}) \tag{3.1}$$

This operation is similar to what the image processing community refers to as a *convolution.* Image filtering transforms a discrete input *image* (represented as a two dimensional array of pixels) into an output image by applying a convolution operation at each pixel. This operation often computes a weighted average of the neighborhood about each input pixel [22, 24].

For each cell at every time step of the simulation, the right hand side of equation (3.1) is evaluated. This computation takes a significant portion of the total execution time of the simulation. Equation (3.1) is an example of a reduction computed simultaneously over many overlapping contiguous sections of an array, as shown in Figure 3.1.

Reduction applies associative binary operators across a list of elements [13, 27]. Let A be an N element array, i.e. $A[1..N]$, and $\oplus$ be an associative binary operator. The reduction of A produces a single result defined as follows:

$$r = reduce(A, 1, N, \oplus) = \oplus_{i=1}^{N} A[i] = A[1] \oplus A[2] \oplus A[3] \oplus \ldots \oplus A[N] \tag{3.2}$$

Traditional parallel algorithms for solving reduction focused on reducing either the entire array or disjoint sections of the array using variations of the well known parallel prefix algorithm [13, 27].

Simultaneous reduction applies reduction operators to uniformly sized overlapping contiguous array subsections called *stencils.* The *simultaneous reduction* of $\oplus$ on array $A[1..N]$, with stencil size, s, and result position $p, 1 \le p \le s$ is denoted $SPR(A, p, w, \oplus) = spr_1, spr_2, \ldots, spr_N$ where $\forall i 1 \le i \le N$ and it produces a vector of result, each of which is defined as:

$$\begin{aligned} spr_i &= reduce(A, \max(1, i-p), \min(N, i+s-p), \oplus) \\ &= \oplus_{j=\max(1,i-p)}^{\min(N,i+s-p)} A[j] \end{aligned} \tag{3.3}$$

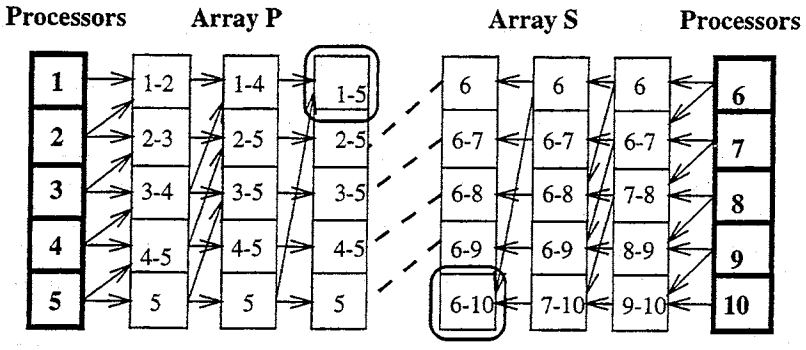


FIG. 3.2. *Prefix-Suffix Algorithm for* $n = 5$.

The optimal coarse grained distributed memory MIMD algorithm requires block partitioning as described in § 3.6 and applies a sequential algorithm locally to each block. Assuming that the stencil is smaller than the block size, there are $O(1)$ communication latencies and the communication volume per each processor is $O(\delta\sqrt{\frac{N}{P}})$, where N is the problem size (number of cells in Ω) and P is the number of processors. The operation and memory cost is $O(\frac{N}{P})$. The efficiency of the algorithms was measured on the MasPar MP-1 and reported in [29], and (indirectly as part of the run time) on both the IBM SP2 as reported in § 3.8.2 and a network of workstations (see § 3.8.3).

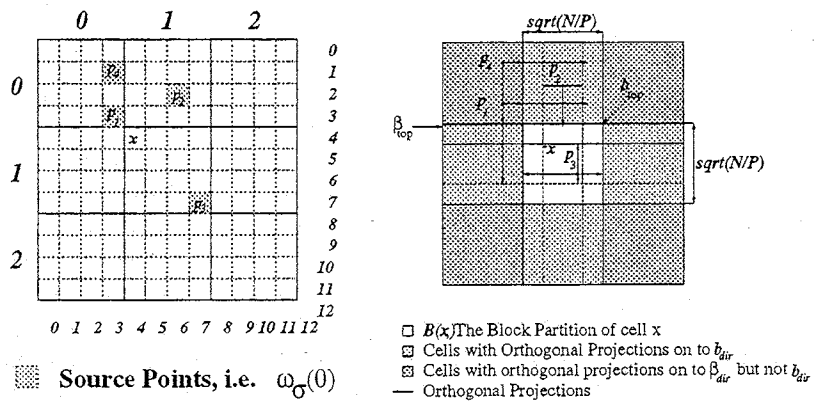
3.2. Fractal Dimension Computation. The complexity of the spatial patterns arising in the habitat may influence both community dynamics and its control, such as disease control which difficulty increases with the growth of spatial complexity. *Fractal dimension* measures spatial complexity because larger fractal dimension correspond to a more complex spatial patterns. Two-dimensional habitats have fractal dimensions in the interval $[1, 2]$, where 1 corresponds to a line and 2 corresponds to a plane.

We adopted Sugihara's technique for computing fractal dimension of images composed of a lattice of *pixels* (which assume a value of either 0- set off, or 1 set on) in [26, 42]. Computing the fractal dimension involves creating an image from the model's configuration C_t and then processing it.

In the lattice of the model presented in § 2, cells have values assigned from a set of states S usually of higher cardinality than 2. However, we can distinguish a subset $U \subset S$ of local states that are of interest to the user (e.g., testing for the the presence of a pathogen implies $U = \{4, 5\}$). Fractal dimension computation requires measuring clustering of cells in a state which belongs to U. We will call such cells set on, and the cells which have a state in $S - U$ will be called set off. Hence, a pair (C_t, U) defines an image. The set of all cells set on at time t will be denoted $SetOn(C_t, U) = \{x | s_{x,t} \in U\}$. Two cells in $SetOn(C_t, U)$ are *connected* if they are adjacent, or if they can reach each other by a path through Ω such that all cells on the path are in $SetOn(C_t, U)$. The *image component* (also called a *cluster*) containing $x \in SetOn(C_t, U)$ is the set of cells connected to x in $SetOn(C_t, U)$. The *image component label* of cell $x \in SetOn(C_t, U)$ uniquely identifies which image component x is participating in. The well known *image component labeling problem* [27] consists of computing the image component label for each cell in $SetOn(C_t, U)$. A cell in $SetOn(C_t, U)$ is said to be on the *perimeter* if it is adjacent to some cell not in $SetOn(C_t, U)$. The set of all perimeter cells will be denoted $OnPerimeter(C_t, U)$.

Fractal dimension is the regression of the logarithm of image component perimeter on the logarithm of image component area. Suppose that (C_t, U) has n image components, numbered $1 \leq i \leq n$. Let c_i denote set of cells in the ith image component, the *area* of c_i be $a_i = |c_i|$ and perimeter of c_i be $perimeter_i = |c_i \cap OnPerimeter(C_t, U)|$. Let $x_i = \ln a_i$ and $y_i = \ln perimeter_i$, with means $\overline{x} = \sum_{i=1}^{n} x_i/n$ and $\overline{y} = \sum_{i=1}^{n} y_i/n$. Fractal dimension, ϕ, can be expressed as:

$$\phi = 2\frac{\sum_{i=1}^{n}(x_i y_i) - n\overline{xy}}{\sum_{i=1}^{n} x_i^2 - n\overline{x}^2}.$$



(a) $D_U(x) = \{p_1\}$ (b) Orthogonal Projections

FIG. 3.3. *A coarse grained DVD example using block partitioning with* $\omega_U(0) = \{p_1, p_2, p_3, p_4\}$ *and* p_1 *closest to* x*, as seen in (a). Note that* p_1 *does not orthogonally project onto* b_{dir}*, hence,* β_{dir} *is needed, as per (b).*

For this section we will concentrate on L_∞ solutions in two dimensional lattices, these techniques can be used for the L_1 norm and can be extended to higher dimension. For a SIMD Mesh architecture (i.e. MasPar MP-1), we adopted a method similar to Adamatzky's DVD algorithm [1, 33], which uses synchronous BFS to compute $d_{\omega_U(0)}(x)$. This algorithm's complexity is $O(\sqrt{N})$ in number of operations and parallel messages, as well as in communication distance. Computing $v_U(t)$ given $d_{\omega_U(0)}$ requires sending messages $O(\sqrt{N})$ distance and involves $O(\log N)$ operations and parallel message passing operations (which is optimal for this architecture).

The same three-step algorithm can be used for block partitioned lattices on coarse grained MIMD architectures (e.g. IBM SP2 and NOW), however, step (i) (solving for $d_{\omega_U(0)}(x)$) via direct application of the synchronized BFS DVD algorithms described in [1, 33]) requires $O(\sqrt{N})$ iterations, each performing $O(N/P)$ operations on P processors, therefore, it is not work optimal. Below, we sketch a work optimal MIMD algorithm which can be used for either the L_1 and L_∞ distance metrics. In the interest of brevity, only the L_∞ case is discussed below.

The algorithm first computes the DVD induced by $\omega_U(0)$. Consider a cell $x \notin \omega_U(0)$ resident on processor P_x. Let $y \in \omega_U(0)$ be the nearest neighbor to x, and let y reside on processor P_y. Then either (i) $P_x = P_y$ and a sequential BFS application on P_x will correctly compute $d_{\omega_U(0)}(x)$ or (ii) $P_x \neq P_y$ and computing $d_{\omega_U(0)}(x)$ requires remote information from P_y. Let $B(x)$ be the block partition to which x belongs (i.e. the set of cells resident on processor P_x) and $D_U(x)$ denote the set of cells in $\omega_U(0)$ closest to x, i.e. $(\forall y \in D_U(x) :\| x-y \| = d_{\omega_U(0)}(x))$. DVD (and velocity) computation requires handling the case that $D_U(x) - B(x) \neq \emptyset$ (see Figure 3.3). Let $\hat{D}_U(x)$ be the set of cells closest to a partition boundary, $b_{dir}(x)$, where $dir \in \{left, right, top, bottom\}$ (all boundaries can be treated similarly due to symmetry). It can be shown (using the triangle inequality) that $\hat{D}_U(x) \supseteq D_U(x) - B(x)$, and since the number of cells on the boundary is limited, $|\hat{D}_U(x)| \in O(\sqrt{N/P})$. The algorithm computes $\hat{D}_U(x)$ as follows (see Figure 3.3):

1. For each partition boundary, b_{dir} create a $d-1$ lattice subspace $\beta_{dir} \supseteq b_{dir}$, which projects the endpoints of b_{dir} to the partition boundaries.
2. Compute the set of cells generating the minimal orthogonal projections on each β_{dir} via cumulative min or max reductions of the lattice coordinates of the cell. This is necessary because the nearest cell in $\omega_U(0)$ to a cell in b_{dir} might not orthogonally project onto b_{dir}. This requires $O(N/P + \sqrt{N})$ operations (where P is the number of processors) and $O(\log P)$ data communication steps per partition boundary and message size of $O(\sqrt{N})$.

was ported to both the IBM SP2 and a NOW.

The IBM 9076 SP2 is a MIMD parallel supercomputer. Each processing element in the SP2 is an IBM RS/6000 with 64 kilobytes of data cache, 128 megabytes of local memory, and an interface into the high-speed inter-processor switch. The high-speed switch topology is a bidirectional multistage interconnection network. The IBM AIX Parallel Environment on the SP2 provides a single-program-multiple-data (SPMD) programming environment. Several versions were written; preliminary ones were written in the C programming language, whereas the later versions use C++ and utilize the MPI [35] message passing library for communication. When executed, parallel programs are automatically distributed and loaded onto the requested number of processors.

The network of workstations in the Computer Science Department of Rensselaer is quite heterogeneous. We chose a (relatively) homogeneous subset of machines, using seven SPARCstation 5s and two UltraSPARC 2s, preferring the (somewhat slower) SPARCstation 5's for smaller problem sizes in an attempt to keep timings uniform. The nodes all ran Sun's Solaris 2.5 and were networked via a Ethernet using TCP/IP. The MPI chameleon/p4 version was used for message passing, so the software was quite similar to the IBM SP2 version.

The MIMD implementation of the model allocates many cells per processor, to efficiently utilize the few powerful processors available. We use a *a block* decomposition, in which a problem size of $N = n_1 \times n_2$ is contiguously partitioned over P processors with each partition containing $\frac{n_1}{P} \times \frac{n_2}{P}$ elements. Typical problem sizes have $n_1 = n_2 = \sqrt{N}$. The implementation is based on a SPMD approach in which each processor asynchronously executes the same program on different data.

Efficient MIMD programs exploit asynchrony in the underlying problem, since synchronization, especially barrier synchronization, has a large performance penalty. On the surface it would seem that time step driven simulations would have strong synchronization constraints, however the locality of interaction permits us to exploit a certain amount of asynchrony. The MIMD implementation has no barrier synchronization calls. At every time step a processor must wait for all of the inputs from its neighbors, before advancing to the next time step. Thus if one processor were fast, it could get a time step ahead of its neighbor, and then it would have to block on receipt of boundary data. The maximum difference (measured in the time step) between any two processors during the simulation is equal to the diameter of the mesh. Whenever the global simulation trajectory is measured, global synchrony is enforced to support the reduction of the data local to each processor across all processors, thereby eliminating the need for explicit barrier synchronization.

Further speedups were obtained by interleaving communication and computation by:

- using nonblocking MPI message passing routines which support noncontiguous message buffers, thereby reducing the number of copy operations [35],
- interleaving communication and computation by precomputing the partial results of stencil reductions and simulation trajectory measurements that do not require data from neighboring processors during the communications phase (see Figure 3.4).

3.7. MIMD Implementation's Language Issues (IBM SP2 and NOW). We faced several language tool difficulties during the development cycle. Initially C++ and MPI were unavailable on the SP1 which was then our initial target, and we resorted to C and used the MPC message passing protocol. The contiguous buffer required to hold the simulation state was allocated at compile time (for efficient multi dimensional array indexing), its dimensions were computed using preprocessor directives. As a result, when the local problem size or stencil dimensions changed, a recompilation was required. This version was directly ported to MPI. The next phase of development replaced the 2-d arrays storing the simulation state and the 3-d arrays used in stencil reduction with dynamic arrays in C++. Diagnosing errors and performance issues under C++ was challenging, since many development tools did not work well in presence of automated internal naming conventions employed by many C++ compilers (so-called *name space mangling* effect). Early experiences with profilers on the SP2 under MPI were not positive because of name space mangling. In particular, the ability of the SUN collector tool's profiler to decode mangled names was critical in diagnosing a programming error which caused a slow down by a factor of 150 of the C++ version (an inadvertent use of call by value instead of call by reference due to a missing "&" character induced frequent implicit copy constructors for a large array).

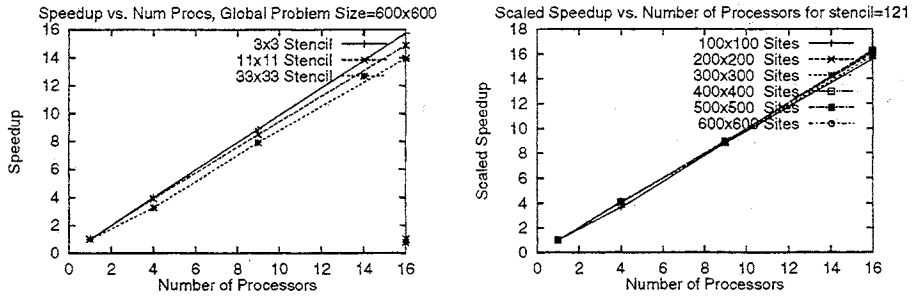
processor. It can be determined as follows:

$$T_{MP-1} = = \frac{100\ timesteps \times 2048 \frac{transitions}{timestep}}{2048\ processors \times 2.25\ sec.} \approx 44 \frac{transitions}{sec} \tag{3.7}$$

This value is useful as a standard of comparison of raw speed between different implementations of the simulations.

Utilization of MasPar processors is obtained from MasPar's advertised speed and speed measured at simulation runs. It is plotted as a function of the number of cells and as a function of the stencil size in Figure 3.5. For the largest stencil and the maximal number of cells, simulation sustained a high utilization, for SIMD architecture[6] of 32.6%. Utilization in SIMD architectures is limited by the way conditional flow of control is executed. Processors not taking a branch have to wait for branch completion, when they are reactivated. There is a nine-way branch in computing the state transition probability inherent in the nine-state ecological model. Increased efficiency of computation over larger stencils and environments is the result of sublinear (logarithmic or square root) complexity of several data sampling algorithms in terms of stencil size.

3.8.2. Performance of the IBM SP2 Implementation. Performance measurements were done for various numbers of processors, problem sizes, and stencil sizes. Stencil sizes were selected so that their areas would vary by (roughly) an order of magnitude. All of the performance graphs presented use a block decomposition. We begin with the traditional parallel speedup measure, where the speedup is defined as a ratio of a sequential implementation's run time to a parallel implementation's run time. It is plotted in Figure 3.6. The effect of scaling the workload over



(a) Raw Speedup Vs. P (b) Scaled Speedup Vs. P

FIG. 3.6. *Speedup on IBM SP2 vs. P = number of processors*

processors (i.e. the problem size is proportional to the number of processors) on run time is shown in Figure 3.7. Scaled workload performance measure [21] scales the problem size (and therefore workload) along with the number of processors. It is shown in Figure 3.6 for the fixed size stencils of $\delta = 121 = 11 \times 11$. The workload increases linearly with the number of processors. The observed speedup is roughly linear with respect to the number of processors, and multi-processor performance improves with the problem size per processor. This improvement results from an increase of the computation to communication ratio caused by faster growth of the computation on each processor than communication volume with the growth of the problem size.

Comparing throughput of architectures can give us insights into their relative performance. Let T_{SP2} be simulation throughput, measured as the number of state transition computations per second per processor, then:

$$T_{SP2} = \frac{100\ timesteps \times 5.76 \times 10^6 \frac{transitions}{timestep}}{16\ processors \times 300\ sec.} = 1.2 \times 10^5 \frac{transitions}{sec.} \tag{3.8}$$

[6]Parallel computation on SIMD TMC machine with utilization of 10% won Gordon Bell Prize [17] and with 30% utilization on MasPar the SuParCup93 Award [30].

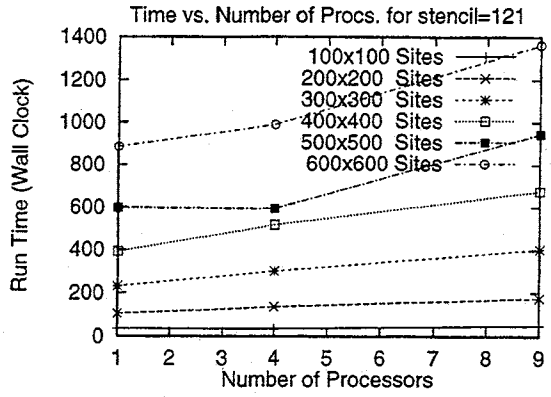


FIG. 3.9. *Run time on NOW for a scaled workload vs. number of processors*

$O(\delta\sqrt{\frac{N}{P}})$ communication volume and $O(1)$ messages on the IBM SP2. As previously, the throughput of the NOW is measured in state transitions per second per processor:

$$T_{NOW} = \frac{100\ timesteps \times 3.24 \times 10^6 \frac{transitions}{timestep}}{9\ processors \times 1355\ sec.} = 2.7 \times 10^4 \frac{transitions}{sec.} \tag{3.9}$$

Comparison of relative throughput of a single processor in the NOW and a single MasPar MP-1 PE yields $\frac{T_{NOW}}{T_{MP-1}} \approx 600$. Similar comparison of an IBM SP2 processor to a processor in the NOW yields: $\frac{T_{SP2}}{T_{NOW}} \approx 4.5$.

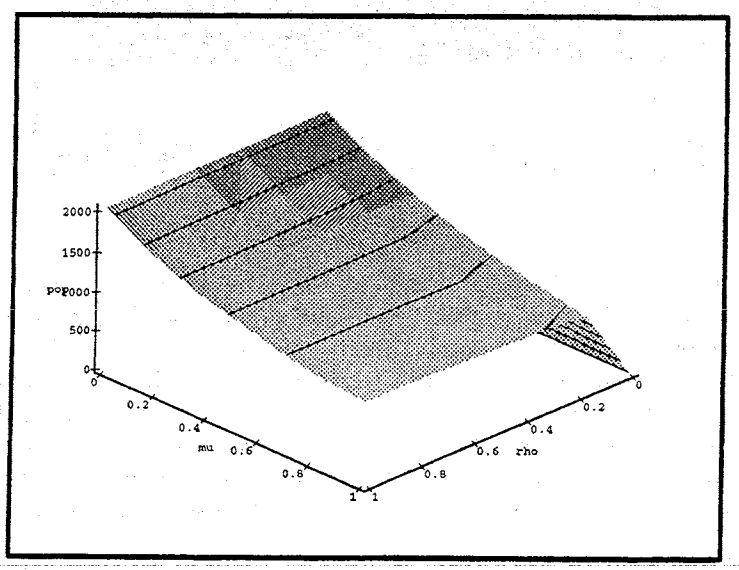


FIG. 3.10. *An animation frame, plotting population (pop) as a function of mortality (mu = μ_2) , and fecundity, (rho = ρ_2)*

3.9. Visualization and Interactive Tools. Large scale simulations require tools to foster understanding of the model's behavior given the large volume of data generated. Model validation and result interpretation benefit from interactive simulation and post processing of the results of several simulation trajectories for comparison of simulations run under different parameters. Many

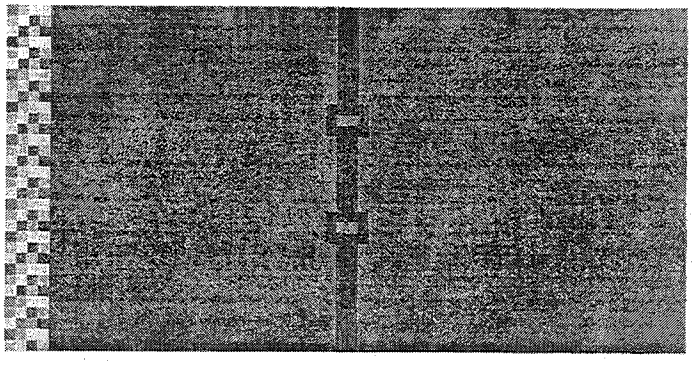


FIG. 4.1. *Tomography of epidemic about to cross the river*

	Parasite's Progress	Pathogen's Progress
1	No Spread	No Spread
2	Stopped at Barrier	No Spread
3	Saturates Environment	No Spread
4	No Spread	Contained by Parasite
5	Stopped at Barrier	Contained by Parasite
6	Saturates Environment	Saturates Environment

TABLE 4.1
Patterns observed in vector borne epidemics

"islands" are ecological "stepping stones" for the parasite and pathogen[7], as seen in Figure 4.1. A few hosts carrying the parasite, pathogen, or both (2% of each state) are at the far left; hence, the epidemic spreads from left to right.

Two simulation results are discussed here:

1. The overall frequency of diseased hosts at each sampling points.
2. The overall biodiversity of the environment at each sampling point.

The extent of infection is an epidemic's most fundamental ecological attribute. The spatial frequency of diseased hosts describes the epidemic's temporal evolution. Community biodiversity increases with the number of distinct ecological (local) states and strongly influences extinction trends. We varied some of the parameters described in § 2.2: the probability of infestation, α, the probability of infection, β, and host recovery from infestation via μ_d. Community biodiversity is used to measure global ecological diversity by quantifying variability by identity [38]. Since the extinction can occur, there are five states in the simulation, numbered according to Figure 1(a). Let f_i be the fraction of cells in state i, $(i = 1, \ldots, 5)$. Let p_i represent the fraction of sites with state i, and S be the number of states satisfying $p_i > 0$.. A community biodiversity index measures the spatial ecological diversity at a given time and since a community biodiversity index should grow as the variance of p_i's decreases [38], we selected the entropy, $H = \sum_{i=1}^{S} f_i \ln f_i$, as its measurement. The simulations were run for 100 time steps, with sampling every 10^{th} step. Patterns emerged in the simulations indicating ecologically significant outcomes [31] as described in Table 4.1. Graphs of average frequency of infected hosts over time and mean community biodiversity over time for the patterns listed in [31] are shown in Figure 4.2.

4.2. Simulating Steady State and Transient Behavior of a Spatial Epidemic. Consider a general epidemic where a pathogenic parasite is communicated via direct contact, and exploits a single host species, in which hosts do not experience recovery, i.e. $\mu_d = 0$. Using the notation presented in § 2.2, the set of possible states for each cell in this system is $\{1, 2, 3\}$ (see Figure 2.1) and μ_i, where $i = 2, 3$, denotes mortality of the host in each of the possible states. We assume that

[7]This is an abstraction of more realistic case of wild fox rabies moving from continental Europe to Great Britain across the English Channel via the tunnel [49].

the NOW is equivalent to about 600 PE's on a MasPar MP-1. Although the tight synchronization is required in our simulation, the cost of communication and synchronizations is proportional to square root of the data allocated to each processor on the IBM SP-2 and to the square root of the product of the number of processors and global problem size on the NOW, whereas the computation is linearly proportional to the problem size. Hence, our simulations are scalable and the larger the problem analyzed, better the performance of the parallel machines. Thanks to this property we were able to simulate efficiently spatially explicit models which revealed phenomena not present in the homogeneous models of smaller computational complexity. The software described in this paper is in public domain and can be accessed via the URL `http://www.cs.rpi.edu/research/tempest/`.

The distribution and abundance of all species exhibit some degree of spatial variation. Spatial heterogeneity in abiotic factors or biotic processes may govern population dynamics and the resulting characteristics of ecological communities. Despite a long-standing recognition of the importance of spatial variation, analytical and computational models of spatially detailed ecological interactions have only recently become available [23, 25]. Our spatially explicit model addresses the population dynamics of (as many as) four species through simulation of the epidemiological landscape of a carrier-borne disease. Related ecological questions have been approached by modifying the model.

Acknowledgments. The authors thank W. Kaplow and J. Waiveris for their help in developing the previous MIMD implementations of the software and M. Duryea for providing comments on this paper and using the software, the anonymous reviewers and M. Nibhanupudi for comments on the paper. This work was partially supported by NSF grants BIR-9320264 and CCR-9527151, and by fellowship from IBM Corporation. The contents of this entry does not necessarily reflect the position or policy of the U.S. Government—no official endorsement should be inferred or implied.

REFERENCES

[1] A. I. ADAMATZKY, *Voronoi-like partition of lattice in cellular automata*, Mathematical and Computer Modelling, 23 (1996), pp. 51–66.

[2] D. BADER AND J. JÁJÁ, *Parallel algorithms for image histogramming and connected components with an experimental study*, Journal of Parallel and Distributed Computing, 35 (1996), pp. 173–190.

[3] N. T. J. BAILEY, ed., *The Mathematical Theory of Infectious Diseases and Its Applications, 2nd. Edn.*, Charles Griffin, London, 1975.

[4] D. J. BARTHOLOMEW, *Stochastic Models for Social Processes, 3rd Edn.*, Wiley, New York City, NY USA, 1983.

[5] R. BORDAWEKAR, A. CHOUDHARY, AND RAMANUJAM, *Automatic optimization of communication of out-ofcore stencil codes*, in Proceedings of 10th ACM Intl. Conference on Supercomputing, Philadelphia, PA, May 1996, ACM Press.

[6] R. G. BRICKNER, *Designing a stencil compiler for the connection machine model cm-5*, Tech. Rep. LA-UR-94-3152, Los Alamos National Laboratory, 1994.

[7] R. G. BRICKNER, W. GEORGE, S. L. JOHNSSON, AND A. RUTTENBERG, *A stencil compiler for the connection machine models cm-2/200*, in Proceedings for the Fourth International Workshop in Compilers for Parallel Computers, Delft, The Netherlands, December 1993. Additional information available on line at http://www.c3.lanl.gov/~rgb/CM2_Delft.html.

[8] L. BROOKSHAW, *Java 2d graph package*. Additional information available on line at http://www.sci.usq.edu.au/staff/leighb/graph/Top.html, April 1997.

[9] A. W. BURKS, *Theory of Self-Reproducing Automata, John von Neuman*, University of Illinois, 1968.

[10] M. CAMPIONE AND K. WALRATH, *The Java Tutorial*, Addison-Wesley, Reading, MA, 1996. Additional information available on line at http://www.javasoft.com/nav/read/Tutorial/index.html.

[11] T. CARACO, W. MANIATTY, AND B. K. SZYMANSKI, *Spatial effects and competitive coexistence*, in Proceedings of the Workshop on Spatioteomporal Models in Biological and Artificial Systems, F. L. Silva and et al., eds., vol. 37 of Frontiers in Artificial Intelligence and Applications, Sintra, Portugal, November 1997, IOS Press, Amsterdam, pp. 9–16.

[12] S. CHEN, G. D. DOOLEN, AND K. G. EGGERT, *Lattice-Boltzmann fluid dynamics: A versatile tool for multiphase and other complicated flows*, Los Alamos Science, 22 (1994), pp. 100–109. Additional information available on line at http://lib-www.lanl.gov/pubs/la-sci.htm.

[13] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction To Algorithms*, McGraw Hill, 1990.

[14] J. COX AND R. DURRETT, *Limit theorems for the spread of epidemics and forest fires*, Stoch. Proc. and their Applications, 30 (1988), pp. 171–191.

[15] R. CYPHER AND J. L. C. SANZ, *Algorithms for image component labeling on simd mesh-connected computers*, IEEE Transactions on Computers, 39 (1990), pp. 277–281.

[16] S. DI GRIGORIO, R. RONGO, W. SPATARO, G. SPEZZANO, AND D. TALIA, *High performance scientific computing by a parallel cellular environment*, Future Generation Computer Systems Journal, 12 (1997).

[17] J. DONGARRA, A. H. KARP, K. KENNEDY, AND D. KUCK, *1989 Gordon Bell prize report*, IEEE Software, (1990).