

# 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) <sup>1</sup>.

Traditionally, CA, as well as percolation models equate stencils with the nearest neighbors only, although, sporadically more diverse models were sometimes considered<sup>2</sup>. 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.

<sup>1</sup>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].

<sup>2</sup>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  $(N - i, N - j)$

results is the simulation trajectory. We describe the algorithms needed for measuring the simulation trajectory and show their affinity to some classical image processing algorithms. We describe also visualization and interactive simulation tools which we developed to assist users in running and analyzing the simulation.

Ecologists have long recognized that the spatial context of ecological interactions is fundamentally important to understanding population dynamics, community stability, and biodiversity [23, 25]. To simulate the spatial and temporal dynamics of a multi-species ecosystem, the habitat is mapped onto a GCA with each cell's state indicating the presence or absence of species involved in a simulation [44]. A probabilistic model can describe a cell's state transitions as functions of the states of cells within its stencil. The size of the stencil and the location of the affected cell within it are based on the characteristics of the species and the habitat. The implemented model allows diverse analyses [11, 31, 32]. We can examine both direct transmission and the ecologically significant vector-borne disease. The software is parameterized by the modeled species' properties for ease of experimentation. It has been used to investigate the relatively unexplored issue of how systematically varying the stencil size and shape impacts outcomes of epidemics [18] and competition [11]. The simulation's results include the number of cells in each state at the end of simulation, the distribution of the species in the habitat, as well as size, frequency and fractal dimensions of species' spatial clusters.

The model presented in [44] is highly data parallel and its implementation requires regular communication patterns, hence it is suitable for many architectures. This model has been implemented on a MasPar MP-1 Single Instruction Many Data (SIMD) machine, an IBM SP2 distributed memory Multiple Instruction Multiple Data (MIMD) computer, and on a network of workstations (NOW). The PEs on the MasPar MP-1 are arranged in a two-dimensional, eight-nearest neighbor, toroidally-wrapped mesh, connected to a master processor, the DPU. The SP2's processors are interconnected through a multistage switch. The NOW consists of Sun SPARCstation 5 and UltraSPARC 2 workstations linked with  $\frac{10Mb}{sec}$  Ethernet using TCP/IP.

There has been a substantial amount of work in the field of parallel CA, with work done in both generalized tools and tools dedicated to particular applications. Lubachevsky [28] developed an asynchronous CA on a SIMD mesh (MasPar MP-1) suited for a subclass of CA applications (such as some Ising models) where the scheduling of state changes does not depend on the stencil. Margolus developed CAM-8 [34], specialized parallel hardware to support CA applications. Di Grigorio et al. implemented CAMEL [16], which executes models expressed in the CARPET programming language on a mesh of transputers. Our simulation tool implements Szymanski and Caraco's four species GCA [44] for ecological and epidemiological modeling on multiple general purpose parallel architectures. It should be noted that the Szymanski-Caraco model, like many CA models, has state changes that are dependent on stencil content, rendering Lubachevsky's approach unsuitable for this application.

The model used in simulation is discussed in § 2. The implementation of the model, including a brief description of algorithms used, the implementations' performance, and visualization tools are discussed in § 3. The ecological results of some simulations are reviewed in § 4. Finally, conclusions and future directions for this research are given in § 5.

**2. GCA and Its Application.** Consider a  $d$ -dimensional space occupied by discrete entities. We model finite regions of this space on a discrete lattice,  $\Omega$ , of dimension  $d$  with  $N$  cells (discrete points). At any time  $t$ , each point (cell) is a state drawn from a finite (and usually small) set  $S$ . In  $\Omega$  the distance between cells is measured via a distance metric,  $L_k$ , where  $k$  specifies the norm used. The presented applications assume that the lattice is rectangular. Let  $m_i$  denote the length of the  $i$ th dimension (using norm  $L_k$ ). In our application [32, 44], the space is two-dimensional ( $d = 2$ ) and the distance is measured by infinity norm, i.e.  $k = \infty$ . In this case, the modeled space can be denoted  $\Omega = \prod_{n=1}^d \{j | 1 \leq j \leq m_n\}$ . Each cell,  $x \in \Omega$ , is then uniquely identified by its integer coordinates:  $x = [c_1, \dots, c_d]$  and has a state  $s(x, t_i) \in S$  at time  $t_i \in T$ , where  $T$  is a finite time series  $T = \{t_1, \dots, t_{stop}\}$ . The *configuration* at time  $t$  of  $\Omega$  is the global state  $C_t = \bigcup_{x \in \Omega} s(x, t)$ . Each cell  $x \in \Omega$  has a stencil surrounding it, denoted  $\delta_x \subset \Omega - \{x\}$ , of cells that can influence state transitions at  $x$ . In GCA,  $\delta_x$  may include elements which are not the nearest neighbors of  $x$ . In this

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  $\delta$ . 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  $(\forall s_0, s_1 \dots s_\delta \in S : \sum_{s_{new} \in S} p(s_0, \dots, s_\delta, s_{new}) = 1)$ .

A GCA, then, can be formally defined as the quadruple  $(\Omega, S, \delta, p)$ .

**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.

state	host	parasite	pathogen
1	absent	absent	absent
2	present	absent	absent
3	present	present	absent
4	present	absent	present
5	present	present	present

Species	Set of states
Host	{2, 3, 4, 5}
Parasite	{3, 5}
Pathogen	{4, 5}

(a) State to Species Mapping

(b) Species to State Mapping

FIG. 2.1. State numbering conventions

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$j = 1$	$\bar{\rho}^{\sigma(\{2,3,4,5\})}$	$\bar{p}(1, 1)$	0	0	0
$j = 2$	$\mu_2$	$\bar{\mu}_2(\bar{\alpha}^{\sigma(\{3,5\})})$	$\bar{p}(2, 2)$	0	0
$j = 3$	$\mu_3$	$\bar{\mu}_3\mu_d$	$(\bar{\mu}_3)(\bar{\mu}_d)(\bar{\beta}^{\sigma(\{5\})})$	0	0
$j = 4$	$\mu_4$	0	$\bar{p}(4, 4)$	$\bar{\mu}_4(\bar{\alpha}^{\sigma(\{3,5\})})$	0
$j = 5$	$\mu_5$	0	0	$\bar{\mu}_5\mu_d$	$\bar{p}(5, 4)$

TABLE 2.1

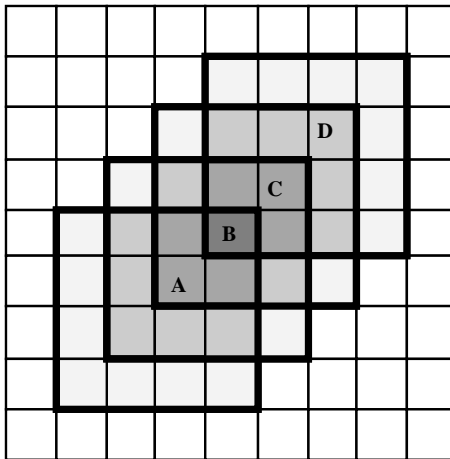
State transition probabilities,  $p(j, k)$ , used in epidemics modeling; for brevity,  $\bar{v}$ , for  $v \in \{p, \mu, \alpha\}$  denotes complementary value  $1 - v$ .

**2.2.3. Modeling General Epidemics with the Four Species Model.** For more general epidemic modeling, it is sometimes necessary to have hosts experience mortality and reproduction. A host's mortality is assumed to be independent of its neighborhood, and have the single step probability of  $\mu_i, i \in \{2, 3, 4, 5\}$  which is influenced by the presence or absence of infestation and infection (see Figure 2.1).  $\rho$  is the probability (per time interval) that an open cell (in state 1) receives an offspring from an occupied cell within the open cell's stencil. We assume that propagation is independent of the presence or absence of the parasite<sup>3</sup>. Occupied cells reproduce independently of each other.

The environment is represented by a CA with  $N$  cells, with a transition probability function,  $p$ . For brevity, we denote by  $p(j, k)$  the transition probability from state  $j$  to state  $k$  which is a function of the states of the cells in the corresponding stencil. Let  $\sigma_x(U)$ , where  $U \subset S$ , represent the number of cells in  $\delta_x$  that are in one of the states in  $U$ . Hosts in state  $j$  die with probability  $\mu_j$ . Mortality of a parasite represents recovery from infestation. We assume that it takes place with probability  $\mu_d$  so the average life time of the parasite is  $1/\mu_d$ . We also assume that a parasite has the probability  $\alpha$  of infesting the neighboring hosts.  $\beta$  denotes the probability of the pathogen infecting the host infested by a parasite carrying the pathogen. The state transition probabilities,  $p(j, k)$ , are given in Table 2.1. Note that death probabilities are stencil independent, while reproduction, infection and infestation probabilities are dependent on the states of stencil's cells.

**3. The Model's Design and Implementation.** Performance and utility were our primary design goals. An efficient simulation engine is critical for the system's performance. § 3.1 briefly introduces simultaneous reduction, which is used for computing state transition probabilities efficiently. The simulation trajectory measures, needed for interpreting the simulation's results, motivated the design of interesting parallel algorithms. *Fractal dimension*, which is a measure of spatial complexity, is discussed in § 3.2. The parallel computation of wave front propagation velocity is described in § 3.3. Placement of data and operations executed on a distributed-memory machine can significantly impact the total computation time. Data distribution of the MasPar MP-1 implementation is described in § 3.4. Performance tuning of the MasPar MP-1 implementation is discussed in § 3.5. Data distribution issues for coarse grained distributed memory MIMD (IBM SP2 and NOW) implemen-

<sup>3</sup>The general model presented in [44] does consider effects on infestation and infection on reproduction.

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:

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

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:

$$r = \text{reduce}(A, 1, N, \oplus) = \bigoplus_{i=1}^N A[i] = A[1] \oplus A[2] \oplus A[3] \oplus \dots \oplus A[N] \quad (3.2)$$

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  $p$ ,  $1 \leq p \leq s$  is denoted  $\text{SPR}(A, p, w, \oplus) = \text{spr}_1, \text{spr}_2, \dots, \text{spr}_N$  where  $\forall i, 1 \leq i \leq N$  and it produces a vector of result, each of which is defined as:

$$\begin{aligned} \text{spr}_i &= \text{reduce}(A, \max(1, i-p), \min(N, i+s-p), \oplus) \\ &= \bigoplus_{j=\max(1, i-p)}^{\min(N, i+s-p)} A[j] \end{aligned} \quad (3.3)$$

Parallel computation of  $SPR(A, p, s, \oplus)$  requires concurrently performing many reduction operations on overlapping array sections to obtain each of the  $spr_i$  in (3.3), which is not supported by traditional parallel prefix algorithms for arbitrary  $p$  and  $s$  combinations. The challenge is to maximize the reuse of partial results of many reductions executed simultaneously.

Functions using stencils to define their inputs are ubiquitous in scientific computations, motivating extensive work on their optimization. Many approaches exploit hardware dependent instructions to effectively compute functions over stencils consisting of nearest neighbors. Bromley et al. described an experimental tool for generating functions of stencils called a *convolution compiler*<sup>4</sup> which generates data parallel codes at compile time for the TMC CM-2 using register allocation, loop unrolling optimizations and minimized data motion (both within and between nodes). Brickner et al. [6, 7] extended that work to develop production level stencil compilers based these techniques for the TMC CM-2 and CM-5. Brickner [6, 7] presents examples using nearest neighbor stencils, however the methods described could be extended to compute functions of larger stencils. The functions that could be compiled by these methods are more general than a simultaneous reduction because the function's use of a particular input value may depend on its relative position within the stencil (i.e. neighboring stencils may use the same value in different ways). As a result, the intermediate values generated during the function evaluation are also position dependent, thereby limiting the efficiency of the stencil computation in a way that is not inherent in simultaneous parallel reduction. It should be noted that many stencil functions used in practice are either simultaneous reductions or can be decomposed into a series of simultaneous reductions yielding more efficient computation.

Bordawekar et al. [5] describe compiler techniques for optimizing arbitrary stencil based functions in distributed computations where an individual processor's problem size could exceed the available memory (so that swapping to secondary storage is required). In [43], we presented algorithms for the simultaneous reduction problem that exploit the associativity and commutativity of binary operators used in reduction to yield optimal solutions for different architectures. As an example, for a linear array of processors, the following *prefix-suffix* algorithm can be used.

For this algorithm, the entire range  $N$  of array  $A$  is divided into  $N/n$  disjoint sections, each of size  $n$ . The algorithm consists of two stages:

1. executing the parallel prefix algorithm on disjoint sections of the input array  $A$  to obtain the result for each disjoint section in one location, and partial results for overlapping sections in the others, and
2. applying operator  $\oplus$  to partial results obtained in the previous step.

The algorithm uses two auxiliary arrays, a prefix array,  $P[1..N]$ , and a suffix array,  $S[1..N]$ , defined as follows: for the  $k$ -th disjoint section of array  $A$  extending from  $kn - n + 1$  to  $kn$ , the arrays  $P, S$  are:

$$(\forall kn - n < i < kn) \quad P[i] = \bigoplus_{j=kn-n+1}^i A[j] \quad \text{and} \quad S[i] = \bigoplus_{j=i}^{kn} A[j] \quad (3.4)$$

Once the arrays  $P$  and  $S$  are known, the solution can be evaluated at the left end<sup>5</sup> of the section as follows:

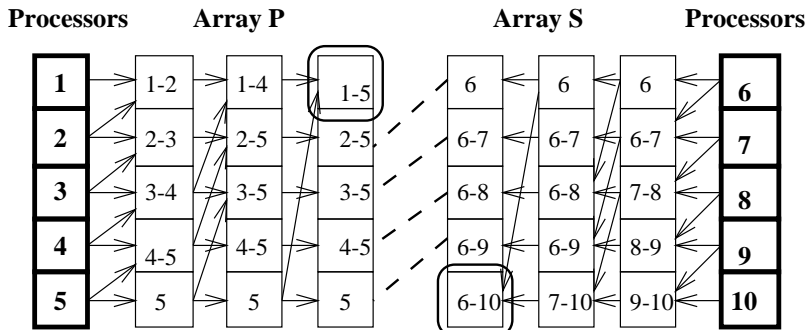
$$res[i] = \begin{cases} S[i] & \text{if } i = kn - n + 1, \\ S[i] \oplus P[i + n - 1] & \text{otherwise.} \end{cases}$$

The above computation is inexpensive and requires a single communication *shift* for the distance  $n - 1$  and a single application of the operator  $\oplus$ . The cost of evaluating arrays  $P$  and  $S$  dominates the complexity of this algorithm.

A SIMD mesh architecture (such as the MasPar MP-1) is sensitive to the stencil's size,  $\delta = n_x n_y$ , where  $n_x$  is the stencil's horizontal length, and  $n_y$  is its vertical length. The optimal SIMD mesh algorithm uses  $(n_x - 1) + (n_y - 1)$  communication distance, with no more than  $2(\log n_x + \log n_y)$  communication latencies, memory and operations cost.

<sup>4</sup>The convolution compiler team at TMC won the Gordon Bell Award in 1990.

<sup>5</sup>Right-end placement of the result can be achieved by switching  $P$  and  $S$  in their roles, i.e., by using the expression  $P[i] \oplus S[i - n + 1]$ . Hence, the placement of the result at the given offset can always be achieved by a single shift for the distance of at most  $n/2$ .

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  $\Omega$ ) 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 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  $\Omega$  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  $i$ th 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  $\bar{x} = \sum_{i=1}^n x_i/n$  and  $\bar{y} = \sum_{i=1}^n y_i/n$ . Fractal dimension,  $\phi$ , can be expressed as:

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

On a SIMD mesh machine, such as MasPar MP-1, computing the image component labels was done in  $O(\sqrt{N})$  time using Cypher's algorithm [15]. Then, for each cell a record is created containing the image component, the processor id, and a flag indicating whether the cell is on the perimeter or not. Sorting such records in ascending order using the image component label as the primary key and the processor id as the secondary key on a SIMD mesh takes  $O(\sqrt{N})$  messages and operations. As a result, all the members of  $c_i$  are placed in contiguous locations in a one dimensional array, so both  $a_i$  and  $perimeter_i$  can be computed by a segmented scan operator using  $O(\sqrt{N})$  messages. The fractal dimension is obtained using reduction operators with a complexity of  $O(\sqrt{N})$  in number of operations and messages which is also the overall complexity of the entire computation (see [31] for more details).

Implementation on a coarse grained MIMD architecture is nontrivial. We are adopting Bader and JáJá's image component labeling algorithm [2] and are extending it to compute the area and perimeter of each image component.

**3.3. Discrete Voronoi Diagram and Velocity Computation .** The well known Voronoi diagram problem was first posed almost ninety years ago by G. Voronoi [45], and after decades of study is still an open area of research. Consider the  $d$ -dimensioned space  $\mathcal{S}$  of elements (typically called points) using the  $L_k$  distance metric and let  $\mathcal{P}$  be a distinct finite nonempty subset of  $\mathcal{S}$ . The *Voronoi diagram* of  $\mathcal{S}$  is a partitioning on  $\mathcal{S}$  induced by  $\mathcal{P}$ . For points  $a, b \in \mathcal{S}$  the distance between  $a, b$  is denoted  $\|a - b\|$ . Each point  $p \in \mathcal{P}$  is assigned its *Voronoi partition*,  $VP(p)$  which contains all points of  $\mathcal{S}$  for which  $p$  is the nearest point in  $\mathcal{P}$ . Let  $d_{\mathcal{P}}(q)$  denote the distance from  $q \in \mathcal{S}$  to the nearest point in  $\mathcal{P}$ , i.e.,  $d_{\mathcal{P}}(q) = \min_{p \in \mathcal{P}} \|p - q\|$ . Given a point,  $p, p \in \mathcal{P}$ , the Voronoi partition about  $p$  is:

$$VP(p) = \{q \in \mathcal{S} \mid \|p - q\| = d_{\mathcal{P}}(q)\} \quad (3.5)$$

When  $S$  is a discrete rectangular space with  $N$  cells, it corresponds to the lattice  $\Omega$  described in § 2, with  $\mathcal{P}$  being a set of cells in the lattice. Adamatzky [1] defines the *discrete Voronoi diagram* (DVD) as a Voronoi diagram in discrete space and presents solutions for the  $L_1$  and  $L_\infty$  norms on fine grained SIMD mesh architectures using a modified breadth first search (BFS). Maniatty and Szymanski present improvements to Adamatzky's algorithms in [33]. The optimal sequential solution for computing DVDs for the  $L_1$  and  $L_\infty$  metrics uses a BFS forest with roots at  $\mathcal{P}$  and runs in  $O(N)$  time.

Informally, *wave front propagation velocity* (called velocity for short) is a measure of the rate at which a phenomena emanates out from a set of initial sources. Use of velocity measurements in population biology began with Fisher's [20] model for the spatial advance of an advantageous gene, and with Skellam's [40] study of the ecological spread of invading species. Recent biological applications include measuring the velocity of epidemics (e.g. rabies in foxes by Murray, et al. [37]), and developing analytical solutions for equations defining such velocities in population dynamics [36]). This fundamental spatio-temporal measurement might have image processing applications, particularly for measuring propagation in digitized time series images.

Let  $U \subset S$  be a subset of states. Informally, states in  $U$  are meant to identify cells that are a part of pattern of interest (such as the pattern of species presence in epidemiology). Let  $\omega_U(t) \subseteq \Omega$  denote the set of cells (discrete points) such that at time  $t$ ,  $\omega_U(t) = \{x \mid s_{x,t} \in U\}$ . Let  $v_U(t)$  denote the velocity of the phenomena indicated by  $U$  at time  $t$ . Informally, velocity measures the maximum observed displacement divided by the time. More formally, velocity is defined as:

$$v_U(t) = \begin{cases} \max_{x \in \omega_U(t)} \left( \frac{d_{\omega_U(0)}(x)}{t} \right) & \text{if } \omega_U(t) \neq \emptyset \wedge t > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (3.6)$$

Velocity can be computed using the following algorithm: (i) at  $t = 0$ , compute the DVD induced by  $\omega_U(0)$ , labeling each cell with its nearest neighbor in  $U$ , (ii) at time  $t$ , for each cell  $x \in \omega_U(t)$  compute  $d_{\omega_U(0)}(x)$ , and (iii) compute  $\max_{x \in \omega_U(t)} d_{\omega_U(0)}(x)$ .

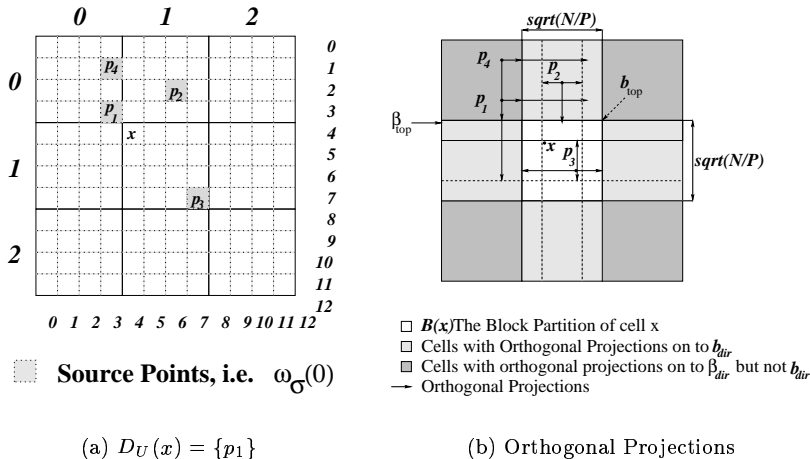


FIG. 3.3. A coarse grained DVD example using block partitioning with  $\omega_U(0) = \{p_1, p_2, p_3, p_4\}$  and  $p_1$  closest to  $x$ , as seen in (a). Note that  $p_1$  does not orthogonally project onto  $b_{dir}$ , hence,  $\beta_{dir}$  is needed, as per (b).

For this section we will concentrate on  $L_\infty$  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  $L_\infty$  distance metrics. In the interest of brevity, only the  $L_\infty$  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  $\beta_{dir}$  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  $O(\sqrt{N})$ .
3. Compute the set of cells nearest to each point on  $b_{dir}$  boundary using a modified BFS, introducing cells at the "right" time, i.e. according to their distance from  $\beta_{dir}$ . This in turn can be done by sorting the set of cells inducing minimal orthogonal projections on  $\beta_{dir}$  in

$O(\sqrt{N})$  operations via radix sort since, for  $L_1$  and  $L_\infty$  norms, the distance from the mesh boundary is  $O(\sqrt{N})$  and the distances are integral.

An application of the process used in Step 3 at higher dimension permits correct local computation of the DVD given the boundary information obtained. The modified BFS is done in  $O(N/P)$  steps, however the boundary points need to be sorted by distance to their nearest point in the distinguished set to ensure correct processing (again using radix sort in  $O(\sqrt{N/P})$  operations), giving a complexity of  $O(N/P + \sqrt{N/P})$  operations per processor. The total asymptotic complexity of computing the DVD is therefore  $O(N/P + \sqrt{N})$  operations, with  $O(\log P)$  message passing operations with message size  $O(\sqrt{N})$ . Computing the local  $d_{\omega_U(0)}(x)$  requires  $O(N/P)$  operations and each processor can store those values in  $O(N/P)$  storage. The reduction step across all the cells takes  $O(N/P + \log P)$  operations, with  $O(\log P)$  message passing steps with message size  $O(1)$ . The asymptotic complexity of computing  $v_U(t)$  is therefore  $O(N/P + \sqrt{N/P})$  in number of operations,  $O(\log P)$  in number of messages, each message of  $O(\sqrt{N})$  size. This means that the algorithm is work optimal when  $P \ll \sqrt{N}$ , an inequality that often holds for coarse grained MIMD systems. To the authors' knowledge the solution presented here is the only work optimal algorithm published to date.

**3.4. SIMD (MasPar MP-1) Implementation's Data Distribution.** The model was implemented on a MasPar MP-1 SIMD parallel computer, using MasPar's parallelized dialect of C, MPL. The array of processing elements (PEs) of the MasPar is organized in a two-dimensional grid with a master processor, the *data processing unit* (DPU). The DPU is connected to a DECstation 5000 workstation, which is called the *front end*. Each processing element has eight nearest neighbors. Each processing element can communicate with its neighbors directly and in parallel with others using *xnet* operations. There is also a global router which allows communication between any two arbitrary PEs at a much higher communication cost than *xnet* operations.

The architecture matches the grid-partitioned environment being simulated. Ecological cells map naturally onto the PEs. Local state changes at each cell are programmed as operations on local variables, whereas the global simulation trajectory measures are stored in the front-end processor. Careful algorithm design minimized the amount of data communicated and the number of communication steps. Furthermore, each communication step was designed to have consistent direction along the *xnet* and minimal distance in our algorithms [31, 43], eliminating contention for communication links and promoting efficient use of the MasPar's interconnection network. The tight synchronization of SIMD architecture execution facilitates the update of the global state at each time step but requires careful design of algorithms to maintain high utilization of PEs.

**3.5. Performance Tuning the MasPar MP-1 Implementation.** Performance tuning on the MasPar MP-1 was done using the MPPE programming tool to get profiling measurements. The simultaneous reduction operator [43] discussed in § 3.1 can be optimized if the stencil is known at compile time. We designed software to generate unrolled loops and sequences of *xnet* communication operations during simultaneous parallel reduction. The efficiency is highly dependent on the stencil area (since the number of operations required is nonlinear with regard to stencil area), giving a speedup of about 1.5 for many stencil sizes over run time iterative algorithms for stencil reduction. Initial implementations utilized the DPU to perform all sequential operations, which greatly reduced simulation throughput (especially for input and output operations). By shifting input, output, and sequential tasks to the much faster DECstation front end we were able to get a speed up by a factor of roughly 2. Additional speedups were obtained by code tuning of the parallel sections of the code, in particular by utilizing MasPar MP-1's much faster additions for 8 bit and 16 bit in place of standard 32 bit integers. By selecting the smallest size of integers suitable for counting the number of cells in each state (which has an upper bound of  $\delta$ ), we were able to obtain another speedup of a factor of 1.5 to 2, giving us a total speedup of a factor of 5.

**3.6. MIMD Implementation's Data Distribution.** This section describes data distribution issues in a coarse grained distributed memory MIMD implementation. This implementation 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 inter-

face 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.

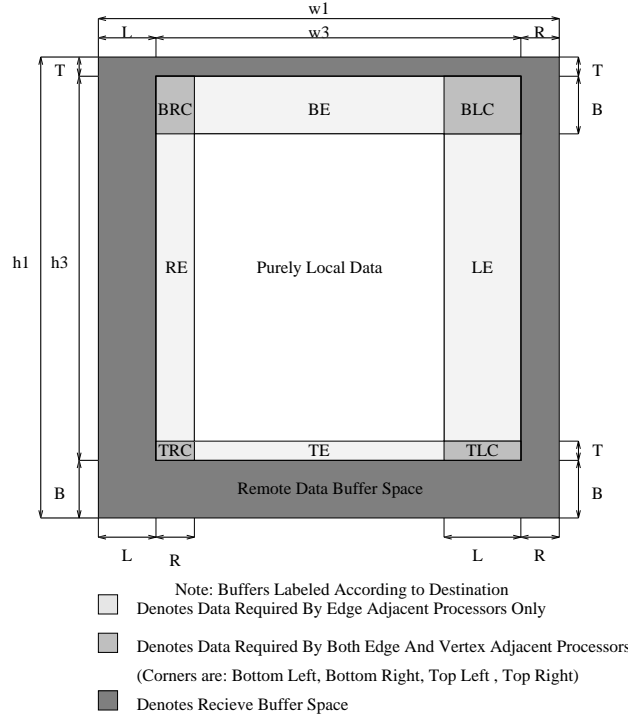
Efficient MIMD programs exploit asynchrony in the underlying problem, since synchronization, especially barrier synchronization, has a large performance penalty. On the surface it would seem that time step driven simulations would have strong synchronization constraints, however the locality of interaction 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).

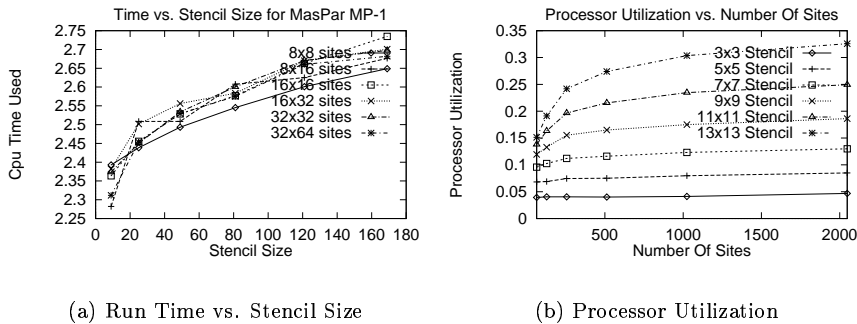
**3.7. MIMD Implementation's Language Issues (IBM SP2 and NOW).** We faced several language tool difficulties during the development cycle. Initially C++ and MPI were unavailable on the SP1 which was then our initial target, and we resorted to C and used the MPC message passing protocol. The contiguous 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).

**3.8. Performance Results of Parallel Simulation.** We compare and contrast the computational performance of the SIMD and MIMD implementations. Trajectory measurements were disabled during the performance runs to measure computational performance without impact of I/O

FIG. 3.4. *Stencil size's relation to a partition's buffer allocation*

subsystem. A summary of the MasPar MP-1 performance is presented in § 3.8.1 and the MIMD IBM SP2 performance is analyzed in 3.8.2.

**3.8.1. Performance of the MasPar MP-1 Implementation.** Thirty six executions of the model were timed using 32 bit integer and 64 bit floating point arithmetic on the MasPar MP-1 (with 2048 PEs). The simulations differ in both the number of cells in the environment (one PE is allocated to a cell on the MasPar) and the area of influence of each cell (i.e., the size of the stencil). Raw timings of the MasPar MP-1 implementation as a function of both stencil size and number of cells are presented in Figure 3.5. The timings indicate that the implementation scales

FIG. 3.5. *Performance of the MasPar MP-1 implementation*

well with respect to the number of processors, but exhibits sensitivity to the stencil size. Let  $T_{MP-1}$  be simulation throughput, measured in number of state transition computations per second per processor. It can be determined as follows:

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

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<sup>6</sup> 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

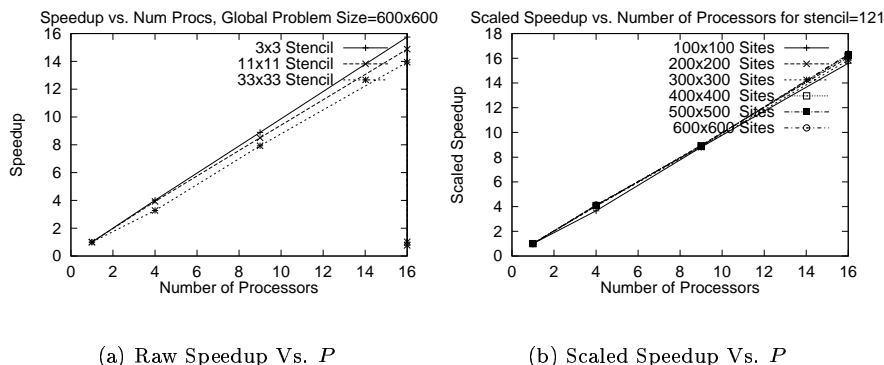


FIG. 3.6. *Speedup on IBM SP2 vs.  $P = \text{number of processors}$*

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:

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

The SP2 speed relative to the MasPar MP-1 is  $\frac{T_{SP2}}{T_{MP-1}} = 2700$ .

<sup>6</sup>Parallel computation on SIMD TMC machine with utilization of 10% won Gordon Bell Prize [17] and with 30% utilization on MasPar the SuParCap93 Award [30].

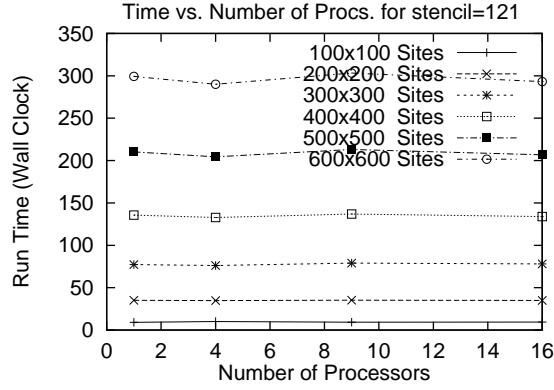
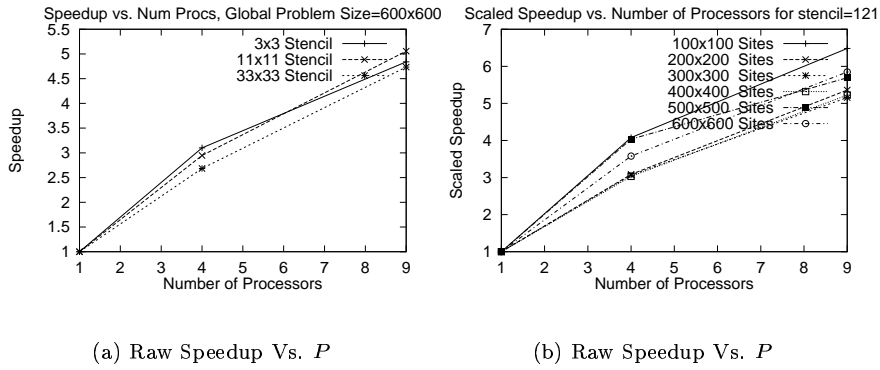


FIG. 3.7. Run time on IBM SP2 for a scaled workload vs. number of processors

(a) Raw Speedup Vs.  $P$ (b) Raw Speedup Vs.  $P$ FIG. 3.8. Speedup on NOW vs.  $P =$  number of processors

**3.8.3. Performance on a Network of Workstations.** We employed a benchmarking structure similar to the one used on the IBM SP2, to facilitate comparisons of results. Runs were done for the identically scaled workload sizes as on the IBM SP2 (see § 3.8.2) and for number of workstations  $P \in \{1, 4, 9\}$ . Getting accurate timings on NOW was nontrivial since the machines are in public use, and even if the pool of processors was currently unused the network was subject to load variations during runs. Therefore, if a major load fluctuation was detected during a benchmarking run, it was discarded and restarted.

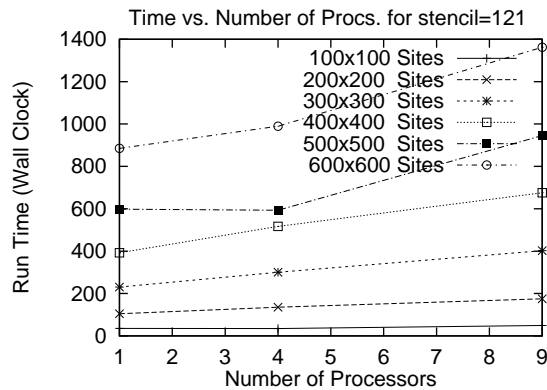


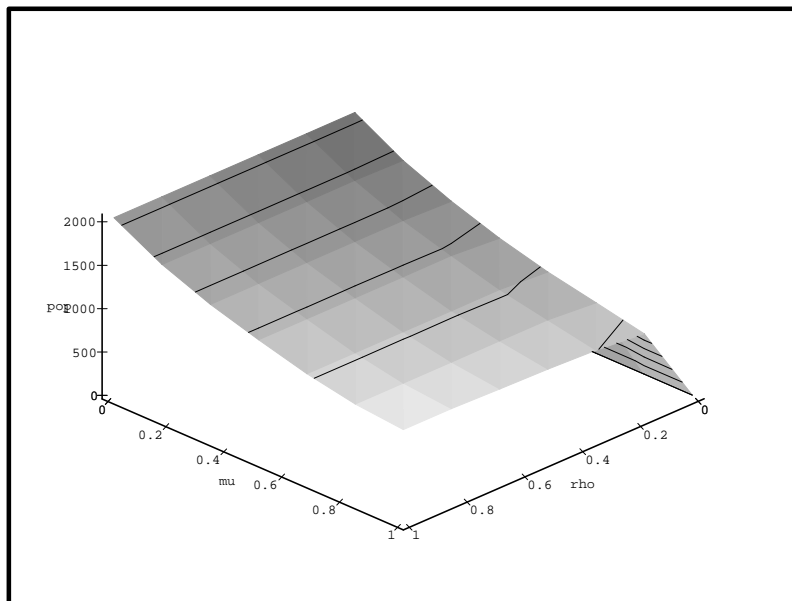
FIG. 3.9. Run time on NOW for a scaled workload vs. number of processors

Traditional speedup vs. the number of processors is shown in Figure 3.8. The scaled speedup (compare § 3.8.2) for the  $11 \times 11$  stencil is shown in Figure 3.8. The run time (wall clock) for scaled work loads as a function of the number of processors for the  $11 \times 11$  stencil (similar to § 3.8.2, Figure 3.7) is shown in Figure 3.9.

The NOW implementation is scalable, but to the lesser degree than the IBM SP2 implementation (compare § 3.8.2). On the IBM SP2 the interconnection network can concurrently route many messages between processors. The amount of communication between a processor and its neighbors in each time step is  $O(\delta\sqrt{\frac{N}{P}})$  in distributed parallelized systems, where  $N$  is the problem size,  $\delta$  is the stencil area and  $P$  is the number of processors. In contrast, the Ethernet connections employed in NOW have smaller bandwidth, larger latency, and they serialize communication. For NOW, serial communication increases as we add processors, so the serialized communication incurs a  $O(\delta\sqrt{NP})$  communication volume transferred in  $O(P)$  messages, compared to the parallelized  $O(\delta\sqrt{\frac{N}{P}})$  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:

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

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:  $\frac{T_{SP2}}{T_{NOW}} \approx 4.5$ .



**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 visualizations of simulation results are possible, including simulation trajectory displays for multi-media presentation of results and comparison of simulation trajectories across various parameter sets. The latter can be used to show a relationship between several parameters or to demonstrate temporal correlation between the runs (e.g. Figure 4.2, see § 4). One such example is shown in Figure 3.9 which displays population densities (along the  $z$  axis) obtained using a post processing pass to format the data for input to a visualization tool. Frames (as in Figure 3.9) are displayed in ascending order of time, and may be animated or concatenated together into solids. The surface plot function of *Maple*, a symbolic math package, was used. We also use the University of

Florida's *SciAn* data visualization tool. The *TEMPEST Viewer*, called *TV* was our initial interactive visualization tool [32]. *TV* was written in C and used the *Simple User Interface Toolkit, SUIT*, from the University of Virginia. This version runs on SUN workstations using SUN OS and Solaris. Recent availability of Java, with its high level of portability, its accessibility via the world wide web and its ease of user interface development made it practical to create a new interactive visualization tool. *GUST*, the *Graphical User interface Simulation Tool* provides an accessible mechanism for interactively exploring simulation trajectories of our model. A sample *GUST* window is shown in Figure 3.10.

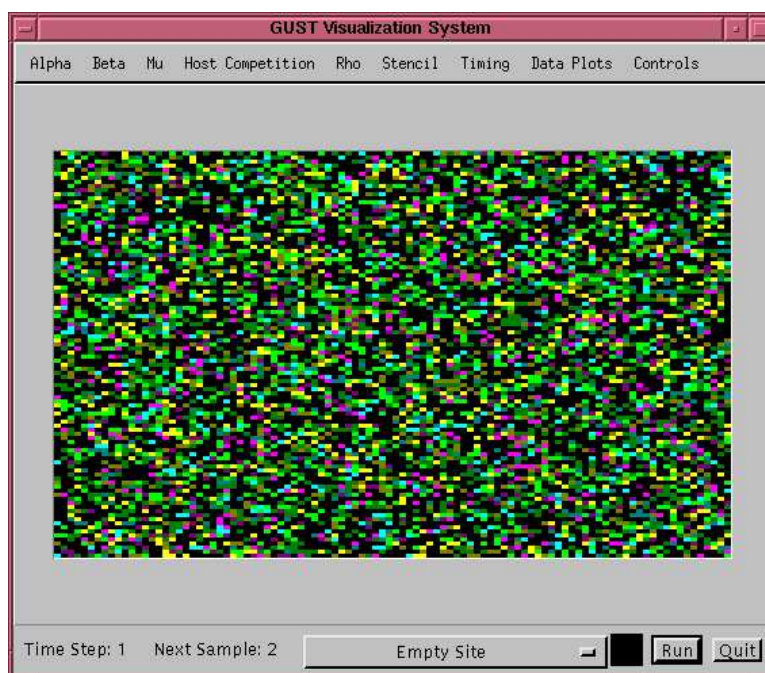


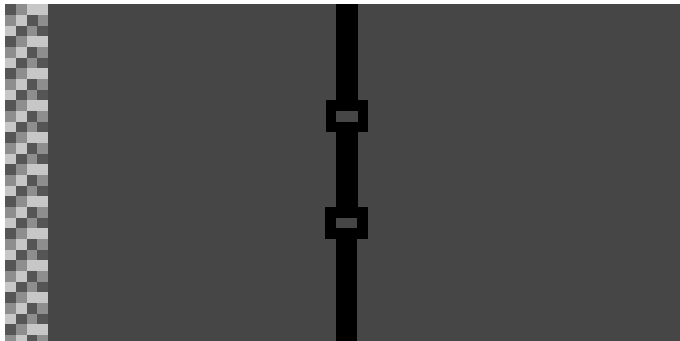
FIG. 3.10. A sample *Gust* screen

We used the Java AWT kit for user interface design. The *Gust* window contains a *MenuBar* object which supports pull down menus to access simulation control parameters and permits the user to view data plots of simulation trajectory measures, using Brookshaw's *Graph2D* library [8]. For world wide web access we used a publicly available applet [10], *AppletButton*, from SUN and "launched" the *Gust* application from the applet. Remote access involves down loading the *Gust*'s byte codes and executing the intermediate representation (byte code) via either an embedded interpreter or a just in time compiler in a browser, making *Gust*'s performance highly browser dependent. *Gust* can be accessed via the URL <http://www.cs.rpi.edu/research/gust/>.

**4. Ecological Results.** In this section we consider two sets of experiments. The experiment described in § 4.1 runs on the MasPar MP-1 and investigates the transient spatial patterns of a vector-borne epidemic. Another experiment, described in § 4.2, runs on the IBM SP2 to model transient and stable state behavior of a *general epidemic*, that is an epidemic where the disease is spread by direct transmission, and hosts experience mortality, birth, and recovery from the pathogen.

**4.1. Simulating the Transient Behavior of a Vector-Borne Epidemic.** Each simulation started with the same ecosystem. Hosts without parasite or pathogen occupy most cells (90.2%). A thin barrier that cannot be occupied splits the environment into left and right halves. The two "islands" are ecological "stepping stones" for the parasite and pathogen<sup>7</sup>, as seen in Figure 4.1. A

<sup>7</sup>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].

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*

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.

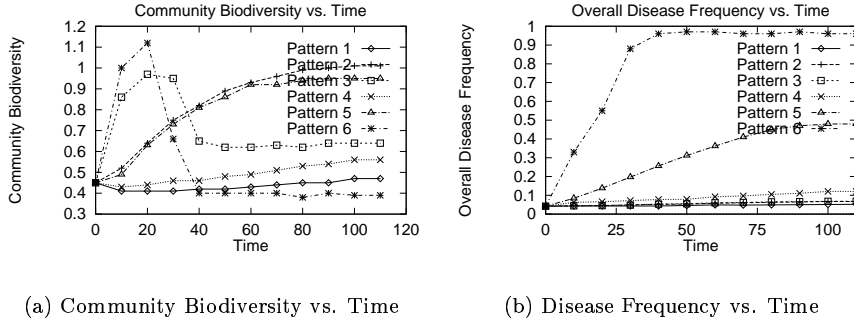
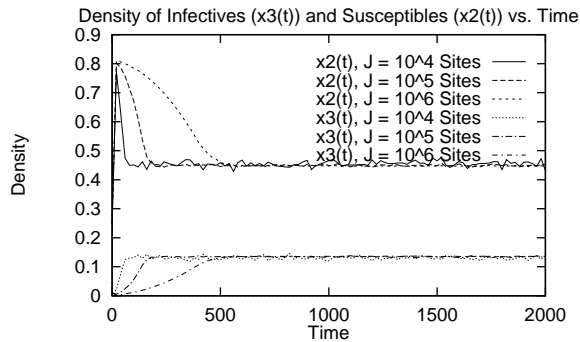
Two simulation results are discussed here:

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,  $\alpha$ , the probability of infection,  $\beta$ , and host recovery from infestation via  $\mu_d$ . 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  $i$ , ( $i = 1, \dots, 5$ ). 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  $\mu_i$ , where  $i = 2, 3$ , denotes mortality of the host in each of the possible states. We assume that  $\mu_3 \geq \mu_2$ , so that  $(\mu_3 - \mu_2)$  is the additional mortality imposed on each diseased individual. All hosts survive or die independently.

We simulated the spatially explicit processes of birth, death and parasite infestation in environments of three sizes ( $N \in \{10^4, 10^5, 10^6\}$  cells). We fixed mortality of uninfested hosts at  $\mu_2 = 0.1$

FIG. 4.2. *Ecological results*FIG. 4.3. *Host densities vs. time for  $\rho = 0.03, \mu_3 = 0.7, \delta = 5 \times 5$  case*

throughout, and fixed the parasite attack probability at  $\alpha = 0.09$  in all simulations. We varied stencil size ( $\delta \in \{25, 2601\}$ ), host-propagation ( $\rho \in \{0.03, 0.09\}$ ), and the mortality probability for infested hosts ( $\mu_3 \in \{0.1, 0.4, 0.7\}$  and  $\mu_2 = 0.1$ ).

To initiate a simulation, susceptible hosts were dispersed across the lattice randomly and independently at a density of 0.25 per cell. Then 1% of the hosts (in the center of the lattice) were designated as infested. Each simulation continued for 2000 iterations. Both species (host and parasite) remained extant in all simulations. Furthermore, densities of both susceptible and infested hosts equilibrated by time 1000 in each simulation. Density variances over the second 1000 iterations all fell in the interval  $(10^{-8}, 10^{-4})$ , so that variation compared to the mean was extremely small.

Figure 4.3 shows a qualitatively typical result; densities of susceptibles and infested hosts are plotted against time for the three different environment sizes,  $N$ . Time required for the densities to reach their long-term averages increases with  $N$ . But the levels at which the densities equilibrate are essentially independent of the size of the environment. The latter pattern was anticipated and held for every combination of  $\delta, \rho, \mu_3$  simulated.

**5. Conclusion.** Implementation of the generalized cellular automata model described in § 2 posed challenging problems such as, simultaneous reduction operation, computing the fractal dimension, and wave front propagation velocity, which motivated the development of novel parallel algorithms. The results confirmed the important conclusion, reached by many authors, that both the general purpose MIMD machines and NOW performed very well running data parallel simulations. Notwithstanding the good fit of the model of epidemics with SIMD architecture, a single node of an SP-2 was equivalent to about 2700 PE's on a MasPar MP-1 and a single SPARCstation 5 node of 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 the NSF grant BIR-9320264, and by a grant and 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-of-core 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. JOHNSON, 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](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. MANIATY, AND B. K. SZYMANSKI, *Spatial effects and competitive coexistence*, in Proceedings of the Workshop on Spatiotemporal 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).
- [18] M. DURYEA, J. GARDNER, T. CARACO, B. K. SZYMANSKI, AND W. A. MANIATY, *Host spatial heterogeneity and extinction of an SIS epidemics*, Journal of Theoretical Biology, (1998). to appear.
- [19] G. B. ERMENTROUT AND L. EDELSTEIN-KESHET, *Biologically motivated cellular automata*, Journal of Theoretical Biology, 160 (1993), pp. 97–133.
- [20] R. A. FISHER, *The wave of advance of advantageous genes*, Annals of Eugenics, 7 (1937), pp. 355–369.

- [21] J. L. GUSTAFSON, *Reevaluating Amdahl's law*, CACM, 31 (1988), pp. 532–533.
- [22] R. M. HARALICK AND L. G. SHAPIRO, *Computer and Robot Vision*, vol. I, Addison Wesley, Reading, MA, 1992.
- [23] A. HASTINGS, *Spatial heterogeneity and ecological models*, Ecology, 71 (1990), pp. 426–428.
- [24] R. JAIN, R. KASTURI, AND B. G. SCHUNK, *Machine Vision*, McGraw-Hill, New York, NY, 1995.
- [25] P. KAREIVA, *Population dynamics in spatially complex environments: theory and data*, Philosophical Transactions of the Royal Society, London, B, 30 (1990), pp. 175–190.
- [26] J. R. KRUMMEL, G. SUGIHARA, R. V. O'NEILL, AND P. R. COLEMAN, *Landscape patterns in a disturbed environment*, Oikos, 48 (1987), pp. 321–324.
- [27] F. T. LEIGHTON, *Introduction To Parallel Algorithms and Architectures: Arrays · Trees · Hypercubes*, Morgan Kaufman Publishers, San Mateo, California, 1992.
- [28] B. D. LUBACHEVSKY, *Efficient distributed event-driven simulations of multiple-loop networks*, CACM, 32 (1989), pp. 111–123.
- [29] B. MANIATTY, B. SINHARROY, AND B. SZYMANSKI, *Efficiency of data alignment on MasPar*, SIGPLAN Notices, 28 (1993), pp. 48–51. Online information available at <ftp://ftp.cs.rpi.edu/pub/maniattb/dataalign.ps.Z>.
- [30] B. MANIATTY, B. SZYMANSKI, AND T. CARACO, *Virtual epidemics - ecological modeling on a parallel machine*, Tech. Rep. CS 93-18, Dept. of Computer Science, Rensselaer Polytechnic Institute, 1992. SuParCup '93 Student Prize award.
- [31] W. MANIATTY, B. SZYMANSKI, AND T. CARACO, *Implementation and performance of parallel ecological simulations*, in Proceedings IFIP WG10.3 International Conference on Applications in Parallel and Distributed Computing, Amsterdam, The Netherlands, 1993, Elsevier Science Publishers B.V. Online information available at <ftp://ftp.cs.rpi.edu/pub/maniattb/ifips93.ps.Z>.
- [32] ———, *Tempest: a fast spatially explicit model of ecological dynamics on parallel machines*, in Proceedings of the 1994 International Simulation Conference, San Diego, CA, 1994, The Society For Computer Simulation. Online information available at <ftp://ftp.cs.rpi.edu/pub/maniattb/hpc94.ps.Z>.
- [33] W. A. MANIATTY AND B. K. SZYMANSKI, *Fine-grain discrete Voronoi diagram algorithms in  $l_1$  and  $l_\infty$  norms*, Mathl. Comput. Modelling, 26 (1997), pp. 71–78. Also Technical Report 97-3 Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180.
- [34] N. MARGOLUS, *Pattern Formation and Lattice-Gas Automata*, American Mathematical Society, 1996, ch. CAM-8: a computer architecture based on cellular automata.
- [35] MESSAGE PASSING INTERFACE FORUM, *MPI: A message Passing Interface Standard*, University of Tennessee, Knoxville, Tennessee, 1995. Additional information available on line at <ftp://ftp.mcs.anl.gov/pub/mpi/mpi-1.jun95/mpi-report.ps.Z>.
- [36] D. MOLLISON, *Dependence of epidemic and population velocities on basic parameters*, Mathematical Biosciences, 107 (1991), pp. 255–287.
- [37] J. D. MURRAY AND W. L. STEWARD, *On the spatial spread of rabies among foxes with immunity*, Journal of Theoretical Biology, 156 (1992), pp. 327–348.
- [38] E. C. PIELOU, *Ecological Diversity*, Wiley-Interscience, New York, NY, 1984.
- [39] M. SAHIMI, *Applications of Percolation Theory*, Taylor & Francis, Bristol, PA USA., 1994.
- [40] J. G. SKELLAM, *Random dispersal in theoretical populations*, Biometrika, 38 (1951), pp. 196–218.
- [41] S. SUCCI, G. AMATI, AND R. BENZI, *Challenges in lattice Boltzmann computing*, Journal of Statistical Physics, 81 (1995), pp. 5–16.
- [42] G. SUGIHARA AND R. M. MAY, *Applications of fractals in ecology*, Trends in Ecology and Evolution, 5 (1990), pp. 79–86.
- [43] B. SZYMANSKI, W. MANIATTY, AND B. SINHARROY, *Simultaneous parallel reduction*, Parallel Processing Letters, 5 (1995). RPI Dept. of Computer Science Tech Report CS 92-31.
- [44] B. K. SZYMANSKI AND T. CARACO, *Spatial analysis of vector-borne disease: A four species model*, Evolutionary Ecology, 8 (1994), pp. 299–314. Online information available at <ftp://ftp.cs.rpi.edu/pub/szymansk/eec.ps>.
- [45] G. M. VORONOI, *Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième Mémoire: Recherches sur les paralléloèdres primitifs*, J. Reine Angew. Math., 134 (1908), pp. 198–287.
- [46] S. WOLFRAM, *Cellular automata fluids: Basic theory*, Journal of Statistical Physics, 45 (1986), pp. 471–526. Also in Wolfram's Cellular Automata and Complexity.
- [47] ———, *Cellular Automata and Complexity*, Addison Wesley, Reading, MA, 1994. Online information available at <http://www.wolfram.com/s.wolfram/books/ca-reprint/>.
- [48] S. WOLFRAM AND N. H. PACKARD, *Two-dimensional cellular automata*, Journal of Statistical Physics, 38 (1985), pp. 901–946. Also in Wolfram's Cellular Automata and Complexity.
- [49] S. YACHI, K. KAWASAKI, N. SHIGESADA, AND E. TERAMOTO, *Spatial patterns of propagating waves of fox rabies*, Forma, 4 (1989), pp. 3–12.