Parallel Discrete Event Simulation of Lyme Disease

Ewa Deelman[†], Thomas Caraco[‡] and Boleslaw K. Szymanski[†]

[†]Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180

{deelmane, szymansk}@cs.rpi.edu

[‡]Department of Biological Sciences, State University of New York at Albany, NY 12222

Abstract

Our research concerns the dynamic processes underlying the rapid increase in the geographic distribution of Lyme disease, currently the most frequently reported vector-borne disease of humans in the United States [10, 1]. More specifically, we ask how spatially localized ecological interactions drive the Lyme disease epidemic at extended spatial and temporal scales. We have developed a parallel discrete event simulation system in C++ for the IBM SP2. The simulation model discussed here models the mouse-tick interaction, an essential element of the epidemic's ecology. The main entities of the simulation are ticks in various stages of development (larval, nymphal, and adult) and mice. We track the behavior of mice and the spread of disease over the course of 180 days (late spring, summer, and early fall). Our goal is to understand patterns in the Lyme disease epidemic at the regional scale through studying the spread of the pathogen across a single white-footed mouse deme.

1 Lyme Disease

Lyme disease is a zoonosis. Humans acquire a pathogen, the spirochete *Borrelia burgdorferi*, that normally infects small mammals and an insect vector [12, 8, 11, 5]. The blood-feeding vector is the deer tick *Ixodes scapularis*. Immature ticks (larvae and nymphs) usually feed on the white-footed mouse *Peromyscus leucopus*. However, immature ticks also may bite a variety of mammals and birds. Humans inadvertently bitten by an infectious nymph may subsequently develop Lyme disease [15]. Adult ticks are less generalized; they feed on white-tailed deer, *Odocoileus virginianus* [16, 11].

The Lyme disease phenomenon is driven, at its ecological basis, by the cycle of infection passing from tick to mouse to the next generation of ticks. Larval *I. scapularis* hatch during summer. The larvae that obtain a blood meal from a mouse then overwinter as inactive nymphs. The following spring these nymphs quest for a second blood meal. Those nymphs successfully attacking a mouse advance to the adult stage. The adults soon attack deer, where they feed again and also mate. Gravid females drop off the deer they have parasitized and lay their eggs, completing the two-year life cycle [1, 5]. The "inverted" seasonal abundance of the immature-tick stages maintains the spirochete. Nymphs infected transmit the pathogen to susceptible mice in the spring. When summer arrives, newly hatched larvae feed on the same mice, acquire the spirochete, and so complete the cycle of infection.

The Lyme disease epidemic is ordinarily depicted at the regional geographic (i.e., spatial) scale and among-year temporal scale [15]. Local spatial scale implies an area occupied by a single deme of the white-footed mouse. In our model individual mice shift their home range within the local area [17]. As they disperse, mice can experience heterogeneity in numbers of infectious ticks. Furthermore, dispersing mice may carry attached ticks between sites within the local area. Either the mice or the ticks parasitizing the mice may carry the pathogen. Our simulations extend over the part of the year during which the cycle of infection occurs, the 180 days elapsing between the appearance of questing nymphs and the completion of feeding by the next generation's larvae.

Our computational model is termed "individual-based" [7]. Our mouse population is an ensemble of different individuals. We track each individual according to its age, its spatial location on a rectangular lattice, its load of parasitic ticks, and its infection status (susceptible or infected). At each lattice node we count susceptible and infected ticks in each of the three developmental stages. The individual-based model treats population descriptions, such as frequency of infection among mice and the spread of infected ticks across the lattice, as consequences of events occurring at the level of individual mice [9]. A single simulation includes mortality and dispersal, but not reproduction. However, the output of one simulation can be mapped through natality functions to produce the initial condition for the next simulation.

2 Individual-Based Modeling

Ecological modeling often involves simulating differential equations which describe various species' rates of population growth. Increasing system complexity, however, requires more complex equations; the resulting model may be hard to understand or analyze. Individual-based modeling may be an attractive alternative. One can establish relatively simple rules for events that affect individual organisms. Simulation of the individual-based model allows different individuals to experience different events; system behavior summarizes the ensemble of individual demographic histories. Differential equations represent "homogeneous" behavior of a mass of individuals and require a significant number of individuals at each grid point; therefore, each grid point represents a relatively large space which is treated homogeneously. Since individual-based models proceed from simpler assumptions and avoid the demographic averaging of differential equations, they offer important advantages for theoretical ecology.

Our simulations advance time in units of days. Therefore, each event's occurrence is assigned to a specific day. This choice of time unit introduces some reasonable constraints on simulation structure. For example, a dispersing individual never occupies two different locations during a single day; the location changes on the dispersal date and cannot change again that same day. The simulation system can, however, accommodate a finer temporal granularity if the need arises.

The space where mice and ticks reside has a toroidal shape (wrapped in both directions). This eliminates edge effects. The mice are treated as individuals. Ticks, because of their density (as many as 700 larvae/400m^2), cannot be treated as individuals. This means that every node of the spatial lattice will contain a "*Tick Blob*", which consists of several categories of ticks: questing larvae, questing nymphs, non-questing nymphs (nymphs that have transformed from larvae and are in an inactive state), and adult ticks (adults that have transformed from nymphs and are in an inactive state). We subdivide each tick category according to the presence/absence of the spirochete infection. We assume no transovarial infection; all larvae hatch uninfected. Larvae may acquire the spirochete during their first meal.

The mice can disperse in the environment. They can be bitten by ticks, infected with the spirochete, or infect the feeding ticks with the parasite. The mice can also die, either as a result of failing to find an open site (lattice node) during dispersal, or due to some other causes.

3 Discrete Event Simulation

The general components of the simulation system are simulation objects (in the case of Lyme disease these would be mice), events (birth, move, death, getting bitten by ticks, or ticks dropping off an animal), and a simulation engine, consisting of state variables, an event queue, and a clock [2]. Events are kept in a priority queue, where the event with the lower scheduled time has higher priority. The simulation is started by introducing some initial events, which are inserted into the event queue. The simulation progresses as events are first removed from the event queue and then processed. The simulation clock is advanced to the time of the event's occurrence. When an event is processed, the system changes from one state to another.

Individual-based modeling does not require discrete event simulation; discrete time simulation is more common. A discrete time simulation inspects each location in the environment at each time tick, and must often perform random draws for highly improbable events. If the number of locations is large and the time tick is small, discrete time simulation may be relatively slow. In comparison, discrete event simulation does not inspect empty locations (consider the advantage in a large environment with a sparse population). Furthermore, discrete event simulation need not continually re-evaluate locations where transitions are rare. Consequently, discrete event simulation should prove faster for models of the type we analyze.

A population model often envisions a large number of individuals inhabiting a large area. The size of the problem might make sequential simulation too slow. Simulation speed is limited by processor speed as well as the cache and virtual memory performance. The memory needed by a simulation grows with problem size, so the limited memory and cache size on a single processor may severely degrade simulation performance for large problems. In such cases, an attractive alternative is to use a parallel or a distributed machine. Such a machine possesses the sum of memory and cache on all processors to fulfill the memory requirements, and the combined processing power to speed up the simulation. Our preliminary results were produced using an IBM SP2.

The general concept of Parallel Discrete Event Simulation (PDES) is similar to the sequential version; however, each processor has at least one simulation engine (termed a Logical Process (*LP*)), each of which has its own clock, event queue, and state variables. *LP*s communicate via messages. Consider the example of a space defined by two sub-lattices of sites l_1 and l_2, populated by mice, and two processors assigned to them. The process *LP*$_1$ simulates the behavior of mice in l_1 and *LP*$_2$ simulates the behavior of mice in l_2. When a mouse m_1 moves from l_1 to l_2 at time t_1, *LP*$_1$ sends a message to *LP*$_2$ containing the event of arrival of a mouse m_1 at time t_1. The event is received and eventually processed by *LP*$_2$.

The problem that arises is that each *LP* advances at its own pace. Some LPs may be early in the simulation, some farther along. For example, in the considered pair of processors, *LP*$_2$ could be at logical time t_2 when it receives a message from *LP*$_1$ with a timestamp (event's scheduled time) $t_1 < t_2$. Such a situation is called a *causality error,* and it can invalidate the simulation if the event happening at t_1 can affect the event at time t_2.

There are two main approaches dealing with causality errors[4]: avoiding them, or undoing the potentially incorrect computation, and restarting the simulation. The first approach, known as conservative, was developed by Chandy and Misra [3].

The general idea behind the conservative approach is that *LP*s do not process any new events until there is a certainty that no event with a lower timestamp can arrive. The messages sent between *LP*s are assumed to be sent in chronological order. The receiving *LP* has preallocated buffers for each *LP* that can send it a message, and it reads off messages from each buffer in a FIFO manner. In order for the *LP* to process a message from the event queue, it has to check all incoming buffers. If all buffers contain messages with timestamps larger than the timestamp of the event in the event queue, then it is *safe* to process that event. Conservative simulations derive their good performance from *lookahead*, the ability to predict when an event will occur. In the case of Lyme disease simulation, lookahead results from the fact that mice move only one lattice node at a time. If a mouse is four lattice nodes distant from the processor boundary it is safe to move the mouse if neighboring processors are behind no more than three days. If, however, the mouse is on the processor boundary, the neighboring processors must be practically synchronized. This lookahead is further shortened when deer are present because they move faster than mice. This tight synchronization at the processor boundaries required by the conservative methods prompted us to investigate the optimistic approach in the parallel simulation.

The second protocol, known as optimistic, deals with causality errors differently. Each *LP* processes events according to their timestamp, and when an event e_s with a timestamp (t_s) smaller than the local clock arrives (that event is also known as a *straggler*), the computation is rolled back to time t_s. The computation is then restarted, and the event e_s is processed. Time Warp [6] is the best known optimistic protocol. There are several aspects to the rollback. The first one is the need to restore the system state to that just before the time of the *straggler*. Another aspect is the need to restore the queue of incoming messages. This entails saving all the states of the computation leading up to the causality error, as well as all the messages that have been received. Finally, the messages that have been sent since the time of the *straggler* have to be cancelled, since they might become invalid through the restarted computation. The messages cancelling the original messages are known as *antimessages*. When an *LP* receives an *antimessage*, it must roll back the computation to the time before the corresponding original message was processed. The original message then has to be cancelled.

To see the rollback mechanism in action consider the following causality error example (see fig.1). The mouse m_1 moves to location y at time $t_1 = 100$ (assume that the mouse already had four unsuccessful attempts to find a free space and that after five unsuccessful attempts it must die). The mouse survives dispersal and occupies location y. The logical process (*LP*$_1$) that processed the move event is now at time $t_1 = 100$. On *LP*$_2$ there is a mouse m_2 at location z. *LP*$_2$ processes a move event from location z to location y for mouse m_2 at time $t_2 = 80$. Since *LP*$_2$ is not responsible for location y, it sends a message to *LP*$_1$ containing the move event of m_2. *LP*$_1$ receives the message, and since the timestamp of the message is lower than the logical time of *LP*$_1$, the message is a *straggler*. The state of *LP*$_1$ has to be restored to that at the time $t_2 = 80$. The simulation on *LP*$_1$ is then restarted, the first event will be the move of mouse m_2 to location y. Next the move of mouse m_1 to location y is processed, and under our assumptions (maximum of five steps for dispersal), the mouse m_1 dies and is removed from the simulation. The situation after the rollback is shown in fig.2.

Figure 1: Optimistic simulation before rollback.

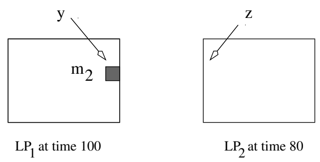


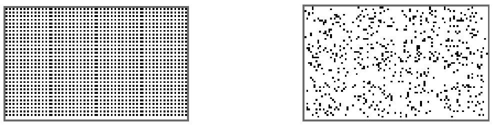
Figure 2: Optimistic simulation after rollback.

The disadvantage of the optimistic method is obviously the amount of memory it consumes by keeping track of all computed states and messages received and sent. In order to minimize the amount of information which needs to be saved, the global virtual time (*gvt*) is calculated. The *gvt* is the minimum virtual (simulation) time of all the *LP*s and the timestamps of all messages present in the system. By definition, the computation cannot be rolled back beyond the *gvt*, since there are no events with a timestamp smaller than the *gvt* in the system. The *gvt* calculation is an important part of reducing the simulation's memory usage, because any states and messages that happened before the *gvt* will never be referred to and can be therefore removed from memory. The process of memory reclaiming is known as *fossil collection*.

4 Computational Model

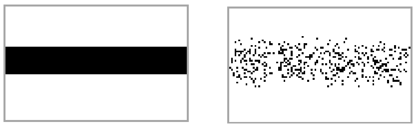
We have decided to model Lyme disease using the optimistic protocol. The main reason, besides the small lookahead, is that the ecological simulation has a lot of inherent parallelism, which can be exploited in an optimistic way. The basic components of the simulation are the space inhabited by individuals and the individuals themselves. The space is divided equally among the processors, and, for prototyping purposes, we have used strip decomposition in the dominant (larger) direction, i.e., this direction is equally subdivided between processors. For four processors, if the space is 400x50, each processor will receive a 100x50 strip. There is one *LP* per processor, and each *LP* (which we call a SpaceManager) is responsible for the part of the space on that processor and all the objects assigned to that space. The parallelism is derived from the fact that several independent events can occur at the same time. For example, events associated with mice getting infected by ticks or mice dying are independent; therefore, they can be processed in parallel.

Since we deal with large areas and large numbers of individuals, it is prohibitively expensive to save all the states needed for rollback; therefore, we have turned to an



(a) day 1 (b) day 180

Figure 3: Even distribution.



(a) day 1 (b) day 180

Figure 4: Band distribution.

approach similar to that used in SPEEDES [13], which uses incremental state saving. In our system, when an event is processed, it is placed at the head of the *processed event list*. When a rollback occurs, the events are removed from the head of the list and are "undone". The events are removed from the list until an event with a smaller timestamp than the timestamp of the event that caused the rollback is found at the head of the list. The method of course assumes that an event can be undone given the information contained in the original event. This is the case in our system. There is another structure necessary to support incremental state saving, namely the *ghost list*, which contains objects that have been deleted from the simulation or objects that have been sent to another processor due to a *Move Event*.

Our *gvt* calculation might take more than one round of synchronization messages in order to flush out all messages that are in transit, so, in order to make the computation efficient, it is important to overlap computation and communication [14]. In our system, when an *LP* initiates a *gvt* calculation or receives a *gvt* calculation message, it enters a *quiet* mode. In that state the *LP* receives *antimessages* and *event messages* and processes events. The *LP* might also roll back, and, as a result, send out *antimessages*. It does not, however, send any *positive* messages resulting from processing events. The *LP* puts these messages on the *withheld messages* list. When the *gvt* calculation is completed, and the new *gvt* has been broadcast by the processor who initiated the *gvt* calculation, all *LP*s release the messages that were withheld.

5 Simulation

Mouse movements are modeled as follows: the mouse enters the dispersal state drawn from a geometric distribution function with expectation $1/\theta_d$. If the mouse is to disperse, the animal randomly selects one of eight directions for dispersal. It moves in that direction with a waiting time derived from the exponential distribution with expectation $1/\theta_m$. If the first site the animal encounters is open (no mouse), the mouse stops and survives dispersal with certainty. If the first site is occupied by another mouse, we update the number of steps (n_s) the mouse has taken since dispersal. The mouse then dies with probability $(n_s/5)^2$. If it survives, it attempts to disperse to the next site in the same direction. After five unsuccessful steps ($n_s = 5$) the mouse dies with certainty.

Fig.(3) shows mice movements in an environment where the mice are evenly distributed, with free space available between the nodes occupied by the mice. We have placed more mice at the boundaries of processors to show where the strip-wise space decomposition occurs (the darker vertical lines in part a of the figure). The simulation starts with 1,560 mice and ends after 180 days with 52.4% of mice dead due to natural causes, and 0.32% dead due to lack of space. Fig.(4) shows the mice movements in an environment where each of the nodes in a "populated region" is occupied. On the first day there are 1,500 mice in the simulation, and by the last day 22.2% of the mice die due of lack of space, and another 41.7% die by other causes.

5.1 Mice Simulation Events

Several events can affect a mouse. There are several object functions that create events, which are in turn inserted into the event queue. The following object functions are related solely to the mouse. *Disperse* creates a new *Disperse Event* and enqueues it. The event time is derived from an exponential distribution with a mean of 20 days. *Start_moving* puts the mouse in a dispersal state (sets dispersal flag), resets the direction of move, and calls *move*. *Move* picks a new direction (if it is not defined yet), figures out when to move (dictated by an exponential distribution), creates a corresponding *Move Event*, and enqueues it. *Next_move* "flips a coin". If the number is within the death probability, the function makes a new *Kill Event* (instantaneous) and enqueues it; if, on the other hand, the mouse survives, *move* is called.

The *SpaceManager* dequeues the events from the event queue and processes them according to the following rules defined for each event type. *Move Event*: Check if the object being moved belongs to the processor executing the *SpaceManager*. If so, remove the object from the location where the object resides. If the object is going out of bounds, and if the *LP* is processing in *quiet* mode, put the message on the *withheld* list. If the object is going out of bounds and the *LP* is not in a quiet mode, send the event message to the processor to which the object is moving. If the object is not going out of bounds, put the object at the new location, check if that location is already occupied. If it is, increase the number of steps that object has taken and call *next_move*, described above. If the site is empty, reset the number of steps that the object has taken to zero, change the object's dispersal status to non-dispersing, and call *disperse*, mentioned above. *Disperse Event*: call *start_moving*. *Kill Event*: find the space where the object is located, and remove the object from that location. Now all other events that have been scheduled in the future for that object are impossible and are removed from the event list and put along with the object on the *ghost list*. When an event is processed, it is inserted into the *processed event* list.

5.2 Ticks

Although ticks are not modeled as individuals, their densities are updated at the time mouse events occur. We do not count all individual tick bites, since studies show that there can be as many as five individual larval bites per day. Accordingly, we decided to combine multiple bites into one: 10 larval bites or five nymphal bites at one time. At the beginning of the simulation we only have nymphs that have over-wintered. They are then questing nymphs. At about the 90^{th} day eggs hatch, larval ticks enter the simulation, and the *Tick Blobs* at each spatial node are updated. Right now, we are using 90 days as a constant, but in the final version of the simulation larvae will hatch within some interval around the 90^{th} day. When a mouse is bitten by a *Tick Blob*, the number of ticks at the lattice node where the mouse is located is decreased by the number that bit the mouse. When the *Tick Blob* drops off the animal, tick densities at the lattice node are increased. We also make the assumption that when the mouse dies, the *Tick Blob* (if any) present on the mouse dies. We assume that mice are bitten by a *Tick Blob* as long as there are enough questing ticks on the node of the lattice. This assumption implies that there is a threshold for both larval and nymphal ticks beyond which we do not "notice" any new bites. We have two types of bites: a nymphal bite and a larval bite. They are treated independently, because they are temporally independent for most of the simulation. The events involving ticks are *Tick Bite* and *Tick Drop*. These involve decreasing/increasing the number of ticks at the space object. In case of a bite a *Tick Blob* is created and placed on the mouse, in case of a drop, the *Tick Blob* is removed form the mouse. Undoing the events entails restoring the *Tick blob* at the node and restoring the infection status of the mouse. It can be seen that the tick densities at lattice nodes will be updated only when a mouse is present in the area, but ticks die even in an area where no mice are present. The solution we use is to update the tick densities at empty (mice-free) locations during the *fossil collection* stage (right after the new *gvt* calculation).

6 Preliminary Results

We wanted to see if our simulations show that the spread of Lyme disease is directly related to the dispersal of mice. The simulation was run on an IBM SP2, a distributed memory MIMD machine. Each of the four processors we used had about 400 mice and a 100×60 lattice. We assumed each lattice node to be 400m^2. Initially all mice are uninfected by the spirochete. We have used the distributions of mice depicted in fig.3 and fig.4. The following figures represent the presence of the disease in a given location; if there is at least one infected tick in the *Tick Blob* at a given location, the figure will show a data point. The infected ticks can be either questing ticks or ticks that have had their blood meal. Mice are not explicitly depicted in the figures. The figures depict simulations at different points in time.

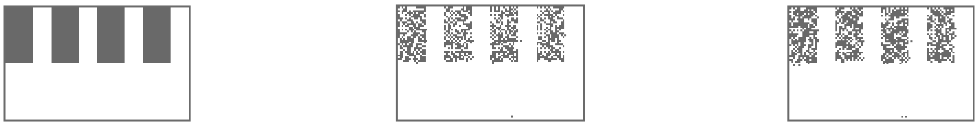
Fig.5(a) shows the initial configuration of infected nymphs. The uninfected nymphs are placed similarly and the larvae will be added to the simulation on the 90^{th} day in the same pattern. Notice that the larvae will not be infected, since we assume no transovarial transmission of disease. The mice are distributed evenly as in fig.3. In fig.5(b) we see that the disease is dying out due to the nymphal mortality. It is sustained in places where the nymphs have fed on mice and dropped off as adult ticks. Fig.5(c) shows the presence of the disease at the end of the 180^{th} day. We can see that initially uninfected larvae fed on infected mice, received the pathogen, and dropped off as non-questing nymphs.

We wondered if the spread of the disease is correlated with the mouse dispersal rate; therefore we increased the dispersal rate by a factor of four. The results are depicted in fig.6. We see that the disease spreads faster among both the questing nymphs (fig.6(a)) and the questing larvae (fig.6(b)). In comparing the final configurations (fig.5(c) and fig.6(c)), we notice that the density of the disease-carrying ticks is much higher when mice are dispersing faster. This is because the faster the mice disperse, the more area they can cover. The points present at the bottom of the figures represent the infected mice that carried ticks from the area at the top of the figures and dropped them off at the bottom (the lattice, as previously mentioned, wraps around).

We also wanted to see how the distribution of mice affects the spread of the disease. The mice were distributed band-wise as depicted in fig.4, and the initial configuration of nymphal and larval ticks remains the same (fig.7(a)). We can see that the disease dies off in the area where no mice are present (fig.7(b)). This is of course because the questing nymphs fail to find a blood meal and therefore die. The final configuration (fig.7(c)) shows that the spread of the disease is increased by the presence of non-infected larvae.

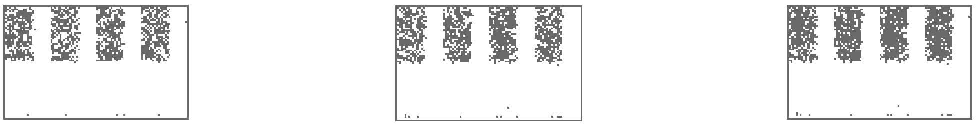
The next set of figures shows the same configuration for ticks and mice, but the mice are dispersing four times faster. We notice that the disease spreads farther and that the spatial density is higher for faster dispersing mice. The results, although preliminary, correspond to our understanding of the spread of the disease. In fig.10. we present a graph of the spread of the disease through the ticks; the decomposition is band-wise and mice disperse fast. The first two bars represent the total number of questing nymphs and the number of infected questing nymphs at the beginning of the simulation. The next two bars depict these nymphs after they took a blood meal and molted into non-questing adults. The fifth and sixth bars represent the larvae on day 90, and the last two bars represent these larvae after they have transformed into non-questing nymphs. The infection ratio might seem high, but the parameters that were chosen for the model are the most favorable for the spread of the disease.

Although our system is only in its early stages, we were able to obtain significant speedups as depicted in fig.9. The decline in performance with two processors reflects the combined cost of the optimistic approach and the communication overhead. With the use of four processors, we can overcome this overhead.



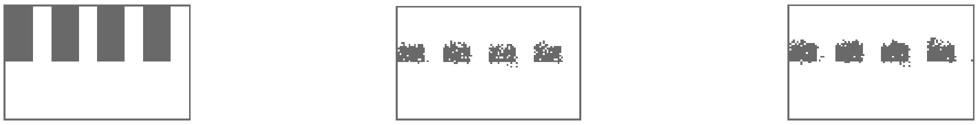
(a) day 1 (b) before larvae (c) day 180

Figure 5: Distribution of infected ticks. The mice are distributed evenly and disperse slowly.



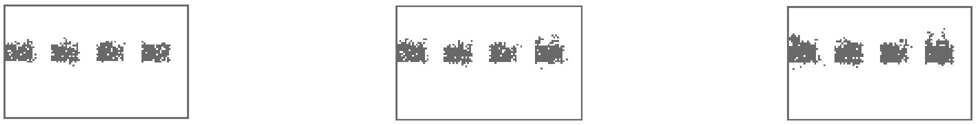
(a) before larvae (b) after larvae (c) day 180

Figure 6: Distribution of infected ticks. The mice are distributed evenly and disperse fast.



(a) day 1 (b) before larvae (c) day 180

Figure 7: Distribution of infected ticks. The mice are distributed band-wise and disperse slowly.



(a) before larvae (b) after larvae (c) day 180

Figure 8: Distribution of infected ticks. The mice are distributed band-wise and disperse fast.

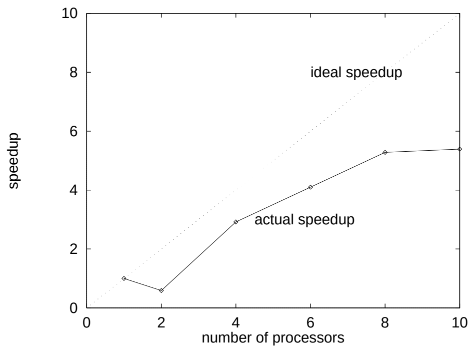


Figure 9: Speedup for 2,4,8 processors.

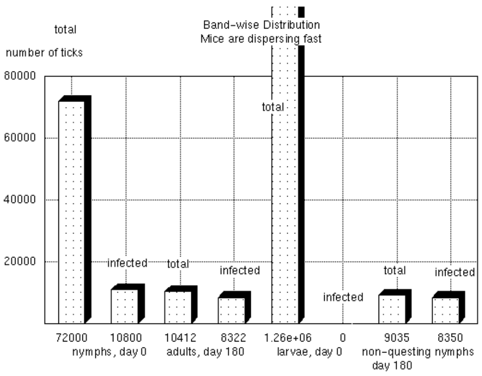


Figure 10: Spread of disease in various tick types.

7 Future Work

Taking the simulation from 180 to 365 days is one of our goals. Parallel Discrete Event Simulation paradigm seems to fit well such an extension because winter months when nothing happens will be "skipped over" and the simulation time will move to the spring thanks to the events at the head of the *event list* having timestamps corresponding to the spring months. The main point is that, unlike the time-step simulation, we are not wasting computation by checking for events every time step or making calculations that do not affect the state of the system.

An important aspect of the spread of Lyme disease is the presence of deer in the environment. Including deers in the simulation will enable us to model the life cycle of adult ticks. The cycle will include taking a blood meal on a deer, dropping off a deer, laying eggs, and will end in death. Our *Tick Blob* will have to be expanded to accommodate the new categories. Finally, we would like to model reproduction for both mice and deer.

Acknowledgments This material is based upon work supported by the National Science Foundation under Grant BIR-9320264 and by the Office of Naval Research under Contract N00014-93-1-0076. The content of this paper does not necessarily reflect the position or policy of the U.S. Government – no official endorsement should be inferred or implied.

References

[1] A. Barbour and D. Fish. *Science*, 260:1610–1616, 1993.

[2] C.G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. 1993.

[3] K.M. Chandy and J. Misra. *IEEE Transactions on Software Engineering*, 5:440–452, 1979.

[4] Richard M. Fujimoto. *Comm. of the ACM*, 33(10):31–53, 1990.

[5] H.S. Ginsberg. *Ecology and Environmental Management of Lyme Disease*, 1993.

[6] D.R. Jefferson. *Trans. Prog. Lang. and Syst.*, 7:404–425, 1985.

[7] O.P. Judson. *Trends in Ecology and Evolution*, 9:9–14, 1994.

[8] R.S. Lane, J. Piesman, and W. Burgdorfer. *Annual Review of Entomology*, 36:587–609, 1991.

[9] E. McCauley, W.G. Wilson, and A.M. deRoos. *American Naturalist*, 142:412–442, 1993.

[10] G.L. Miller, R.B. Craven, R.E. Bailey, and T.F. Tsai. *Laboratory Medicine*, 21:285–289, 1990.

[11] S. Sandberg, T.E. Awerbuch, and A. Spielman. *J. Theor. Biol.*, 157:203–220, 1992.

[12] A. Spielman, M.L. Wilson, J.F. Levine, and J. Piesman. *Annual Rev. Entomology*, 30:439–460, 1985.

[13] J. S. Steinman. *Winter Simulation Conference*, pages 687–696, 1993.

[14] J. S. Steinman, C. A. Lee, L. F. Wilson, and D. M. Nicol. *Workshop on Parallel and Distributed Simulation* pages 139–148. 1995.

[15] D.J. White, H.G. Chang, J.L. Benach, E.M. Bosler, S.C. Meldrum, R.G. Means, J.G. Debbie, G.S. Birkhead, and D.L. Morse. *Journal of the American Medical Association*, 266:1230–1236, 1991.

[16] M.L. Wilson, A.M. Ducey, T.S. Litwin, T.A. Gavin, and A. Spielman. *Medical and Veterinary Entomology*, 4:151–159, 1990.

[17] J.O. Wolff, K.I. Lundy, and R. Baccus. *Animal Behavior*, 36:456–465, 1988.