Parallel Simulation of Lyme Disease Spread Amongst Mice

Anthony Kim, Sara Khedr, Nicholas Fay, and Anders Maraviglia

***Abstract*— Lyme disease is estimated to have 300,000 cases per year in the United States alone [1]. Simulations which model the spread of Lyme disease are critical tools for researchers and policy makers attempting to combat it. To better understand Lyme disease within an ecosystem, this paper proposes a model to simulate its spread through mouse and tick interactions. This proposed model simulates these interactions during a fixed period of 180 days, which covers the spring and summer months. This system models the problem by using a parallel, fixed time-step simulation paradigm in order to represent the spread of Lyme disease as a function of infection rates among mice and ticks over a bounded region. Additionally this model allows for a high degree of customization with the use of global mouse and tick parameters. Lastly, a strong scaling analysis of the system is conducted on one of IBM's Blue Gene/Q supercomputers to show that it is a massively parallel system.**

I. BIOLOGICAL OVERVIEW

Lyme disease is a bacterial infectious disease that can be transmitted to humans from animals. The disease is transmitted to humans through an infected tick bite. Normally, ticks feed on animals and these animal hosts help spread the disease. If an animal becomes infected by a tick, it can then spread Lyme disease to any other tick which feeds on it. The proposed model simulates the time in the life cycle where both ticks and mice are most active (i.e. late spring to summer); during this period, mice are searching for nesting sites and ticks (both larvae and nymph stages) are questing for mice. This simulation model takes advantage of the combination of these two factors.

The life cycle of the tick lasts around two years. During its life cycle, there are four stages: egg, larva, nymph, and adult. Ticks need to have a blood meal at each of the later 3 stages to survive [2]. Larvae and nymphs usually feed on mice, while an adult usually feeds on deer. Ticks that are unable to find a host after a certain period die off. After feeding, a tick ”drops off” its current host and gets ready to quest for its next host as it moves on to its next life stage [2]. Ticks are born uninfected, but a tick can get infected while feeding from a host that is infected. Once infected, a tick can transmit the disease for the rest of its life. A host animal that the infected tick feeds on can get infected. Therefore, Lyme disease is spread through the ticks traveling hosts (mice and deer).

II. RELATED WORKS

The structure of the model is inspired by a study of mice and tick bites, by Deelman et al [3]. Both this model and Deelman's run for 180 days with a simulation space initialized as a two dimensional lattice wrapping in each dimension. The events that occur during each day of the

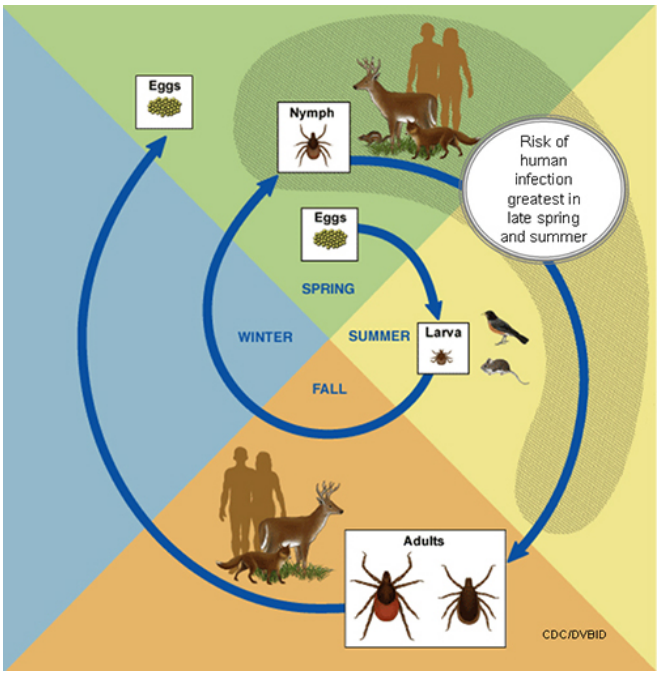

Fig. 1. The life cycle of a tick [2]

simulation (which include mouse movement and tick bites) are incorporated into an individual-based model by treating each distinct mouse as an individual. The assumption that a mouse is bitten by ticks when entering a nest is an element from Deelman's paper [3] that is applied in this model. The Deelman study proposes a Discrete Event Simulation model; however this study proposes a parallel time step model that utilizes both MPI ranks and pthreads. Deelman et al assumes that using a time-step simulation model unnecessarily checks empty areas for activity [3]. This simulation does not make that assumption, instead keeping track of the location of each mouse, and only accessing data from nests where an event has occurred.

The study by Rao et al [4] remarks on Deelman et al's [3] usage of spatially explicit models to simplify and facilitate development, analysis, and optimization. A spatially explicit model requires the context of space in a conceptual framework. As with the Deelman et al study, the model utilizes a two dimensional lattice where each cell represents a mouse nest. Rao et al also remarks upon Deelman et al's usage of spatially explicit models to subdivide the relevant domain into smaller, non-overlapping regions to increase efficiency [4]. Rao et al also proposes subdividing the space dynamically using Dynamic Component Substitution (DCS).

Rao et al shows that DCS decreases the time required for simulations by 7% to 28% [4]. This model utilizes the strategy of subdividing a spatially explicit model.

III. SIMULATION MODEL

The simulation space models a grid of mice nesting sites, each of size 400 m^2. Ticks are present in these nesting sites, yet are immobile. Within the simulation, each mouse moves in a certain direction to find an empty nesting site. If a nesting site is empty, the mouse will make the nest its home. If the nesting site contains other mice, the mouse will find food in the nest for that day; however each nest contains only enough food for a certain number of mice per day. If mice are unable to locate food as they travel, they will die of hunger.

As a mouse passes through a nesting site, any ticks present have a chance to latch onto the mouse and feed. For a set number of days, the ticks will feed on that mouse as it travels before falling off at the mouse's new location, where it will progress to the next stage of its lifecycle. If it feeds as a larva it will drop off as a nymph, and if it feeds as a nymph it will drop off as an adult. In a nest, there may exist ticks at different stages of their life cycle: larva (which are born uninfected), uninfected nymphs, infected nymphs, uninfected adults, and infected adults. The type of tick that feeds on the mouse is chosen by the probability distribution of ticks within the nesting site. When a mouse is bitten by ticks, it will carry a predetermined number of those ticks. Mice will carry ten larvae and five nymphs, since only nymphs and larvae may feed on mice.

At the beginning of the simulation a grid of nesting sites is initialized, wrapping in both the vertical and horizontal directions. At the start of the simulation, nymphs are placed in every fourth row of the grid within a set column-band size; the amount of nymphs placed in each nest can be as many as 1200, where these initial nymphs represent those that have survived over the winter months. To model the spread of the disease, infected nymphs are placed in the center of the universe in a cross-like structure within the column band. The simulation begins by populating every other nest with a set number of mice, where all but one will begin moving in the search of an empty nesting site. On the 90th day of the simulation, larvae are placed in the nests where nymphs were originally initialized, representing tick eggs hatching at the start of the summer.

Many of the parameters within the model can be configured, including:

- The grid size
- Initial number of mice and ticks per nests
- Number of mice each nest can feed per day
- Number of days a mice can travel without finding food

IV. IMPLEMENTATION

In order to best simulate how mice spread Lyme disease throughout an ecosystem, the system represents both mice and the ecosystem (henceforth referred to as the 'universe') as entities upon which ticks impose themselves. Therefore in the implementation both mice and nests (400 m^2 sections) within the universe are represented as structures which contain values representing the quantities of ticks on each.

The universe is modeled as a two dimensional array of nest structures, where this array is evenly divided into sections based on the number of MPI ranks being used, and each MPI rank is assigned one of these sections to perform the simulation upon. Each MPI rank is responsible for all computations of the nests and mice that fall within its boundaries. Every nest in all MPI ranks may contain a linked list of mice structures that represent the number of mice present in that nest at that point in time. These nest structures also store the number of each type of tick currently present, as a statistic instead of individual structures per tick.

Mice are modeled as structures which store an assigned direction in which to move, the statistical measures of any ticks it may be carrying at some point in time, and whether or not it carries Lyme disease. A mouse may contract Lyme disease if it passes through a cell containing infected ticks, and is bit by some of the infected larva or nymphs, but not adults (which do not bite mice in the real world). Once a mouse is infected with Lyme disease, it will always be a carrier and will infect any subsequent larva or nymphs that bite it.

The full course of the simulation can be broken down into initialization and 180 iterations (where each iteration represents a day in simulation time). Initialization within an MPI rank is done in serial by that rank and sets every other nest to contain a set amount of mice, and every fourth node (within the middle of the universe) to contain a set amount of nymphs. If the node falls within the center of the universe, half of the nymphs will be set to be infected with Lyme disease. Computation of the iterations within an MPI rank are assigned to however many pthreads per MPI rank that the system has been assigned to use. At every iteration, all nests within the rank that contain mice are divided among the pthreads and the mice present within each of the nests are subjected to interact with ticks. Iteration over all mice within a nest is done in serial in order to properly compute the percent chance a mouse may be bitten, which is based on the number of ticks still present in that nest after other bites occur. If there are more mice present than is sustainable by the nest, those extra mice are marked as having to move to another nest.

A nest may feed more mice than it can sustainably house, giving those mice which it can feed an extra day that they can travel (done by simply not decrementing the number of days it can still travel before starving to death). All of the mice present within the rank are then iterated over in parallel, where each pthread takes a fraction of the mice (based on the number of pthreads) and if marked as needing to do so, computes where it needs to move. Mice that need to move outside the boundaries of the universe section of its current MPI rank are placed in a linked list of other mice that also need to do so. After all mice have been iterated over by the pthreads, one pthread per rank takes the lists of mice that must move to another MPI rank and sends that list over. For

every mouse in the list, that neighbor MPI rank assigns them to the nests they must be in, and distributes the responsibility for their computation evenly across all pthreads of said rank. By not going over all nests in an MPI rank each iteration, computational resources are focused only on mice and nests where events actually occur.

Communicating the mice that must travel between ranks is done by taking each mice that need to travel from one rank to another and transforming their representation into an offsetted array of integer values.

A. Optimizations

A significant increase in performance was observed after changing how pthreads iterated over mice and nests. Initially, a thread-safe linked list was used as a queue in order to allow each thread to pop a mouse or nest off for computation without fear of multiple access errors. However, due to the liberal use of mutexes within these linked lists and the high cost associated with using them, performance suffered dramatically enough that universe sizes larger than 512 by 512 nodes could not be simulated with the resources available.

In order to combat mutex inefficiency, a system of assigning lists of mice for each pthread to process individually was implemented and mutexes on linked lists were removed. This alleviated the threat of multiple threads attempting to modify the same mouse or nest by assigning each of them to the care of exactly one thread at a time. Subsequent performance increases unlocked the ability for the system to process much greater universe sizes, and were then used in the configurations of the experiments conducted, the results of which can be found in the following section.

V. EXPERIMENTS, RESULTS, AND ANALYSIS

A. Strong Scaling

The simulation model was tested on AMOS, RPI's Blue Gene/Q supercomputer using several parallel configurations. Parameters of the simulation such as the number of mice to initially spawn at each nest and the size of the universe, were held constant through all tests. When initializing the simulation, 5 mice were assigned to every other nest and 1000 ticks to every fourth nest in the middle half of the universe, essentially creating a 'band' of ticks running horizontally through the universe. A nest was set to be able to temporarily feed 5 mice per day but only allow one permanent resident, and a mouse could travel without feeding for 3 days. The four tests outlined in Table 1 were all run using 1, 2, 4, 8, 16, and 32 compute nodes, which taken together encompass a complete strong scaling experiment. In Figures 2, 3, 4, and 5, the x-axis represents the total number of parallel computations possible in a given configuration. This value is the number of compute nodes times ranks per node times threads per rank. Within Figures 2, 3, 4, and 5, 'Total Ranks' represents the number of nodes times ranks per node.

A trend can be observed in Figure 2 that as more compute nodes were used, execution time decreases, which intuitively follows since using more compute nodes also increases

Ranks/Nodes	Threads/Rank
64	1
32	2
16	4
8	8

TABLE I
STRONG SCALING TEST CONFIGURATIONS

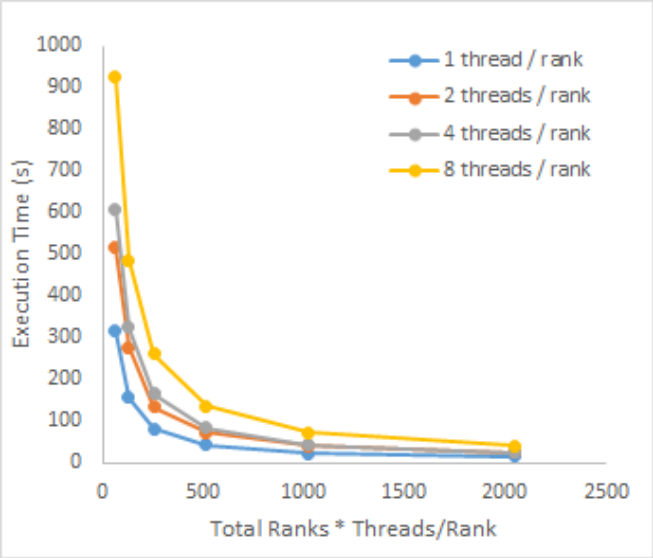

Fig. 2. This figure shows the execution time depending on the number of MPI ranks x number of Compute Nodes x number of Threads/rank

the overall number of MPI ranks being used, making the computational load per rank proportionally less. This trend is further characterized by that decrease in execution time also being nonlinear for all of the plotted configurations. Such a trend is to be expected in a strong scaling experiment, and proves the system to be massively parallel. However, it is incorrect to assume the superiority of the one pthread per MPI rank configuration above all others based only on runtime.

Fully evaluating parallel efficiency from Figure 2 must take into account the effect of the necessary use of highly inefficient mutexes and barriers within an MPI ranks parallel computation of iterations. In order to properly illustrate this, consider the following example: when the test using one compute node using 64 MPI ranks with one pthread per rank is run, that one pthread is responsible for 65536 nests and their respective mice. Now consider in the 32 MPI ranks and 2 pthreads per rank configuration, where again one compute node is used, each pthread is still responsible for about 65536 nests, but within the rank the addition of the overhead of barriers and mutexes that keep each iteration of a day running properly negate any initial efficiency benefits of parallelizing rank computation.

Speedup of the system can be observed in Table 2, which was calculated by dividing the execution time for 64 MPI ranks on 1 compute node (315.45 seconds) by all other execution times, making the values present in the table equate

Ranks/Nodes	1 Threads/Rank	2 Threads/Rank
64	1	0.6117981
128	2.018824879	1.1401072
256	3.900392906	2.34872523
512	7.444012383	4.40073935
1024	13.54402176	7.56533296
2048	20.55811903	12.6023224

TABLE II
SPEEDUP RESULTS

Ranks/Nodes	4 Threads/Rank	8 Threads/Rank
64	0.51772124	0.34058473
128	0.97365513	0.64828182
256	1.8957991	1.21208491
512	3.71344985	2.3195814
1024	7.25611771	4.32108268
2048	12.355076	7.9612889

TABLE III
SPEEDUP RESULTS

to the speedup of each configuration for a range of parallel computations. This then revealed how maximum speedup was 20.56, which occurred when using the 64 ranks per node with 1 thread per rank configuration, applied to run using 32 compute nodes. Having 1 pthread per rank configuration experiences the greatest speedup is not surprising, considering how (as discussed previously) 1 pthread per rank does not suffer the slowdown of mutexes and locks within an MPI rank inherent in the other configurations.

B. Communication Overhead vs. Computation Time

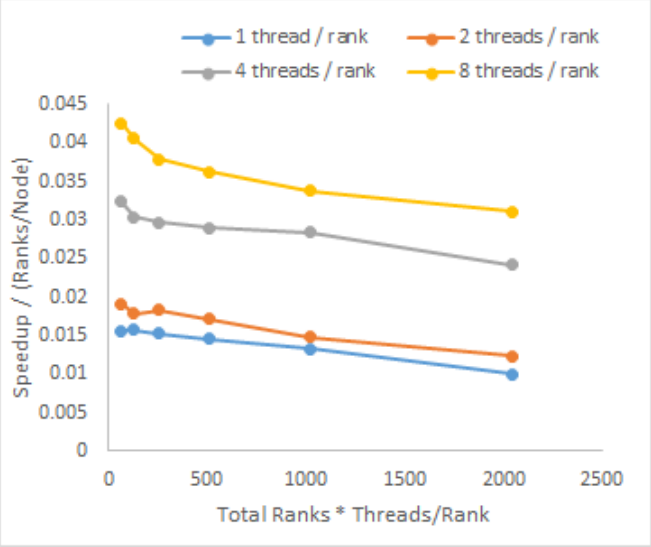

Fig. 3. This plot shows the parallel efficiency (speedup / MPI ranks). These data points were obtained by dividing by each column in the Speedup Table (Table 2) by the number of MPI ranks per node. This means the 1 thread per rank values were divided by 64, 2 threads per rank values were divided by 32, 4 threads per rank values were divided by 16, and 8 thread per rank were divided by 8.

The conclusions obtained from Table 2 should not be taken as definitive proof of the superiority of the single thread per rank configuration over all others, as having the best speedup does not necessarily mean that configuration also experiences the greatest parallel efficiency. Indeed, concluding which configuration experiences this should instead be determined by the analysis of Figure 3, which plots parallel efficiency for each configuration as a function of the number of parallel computational tasks. For all such values the 8 pthread per MPI rank configuration is consistently the greatest. This claim is verified when observing how the line representing said configuration falls above all the others, which indicates that its parallel efficiency is the greatest. Further verification lies in Figure 2, in which the greatest change in execution time as the number of parallel computations increases is seen by none other than the 8 pthread per rank configuration, proving that it exhibits the best parallel efficiency.

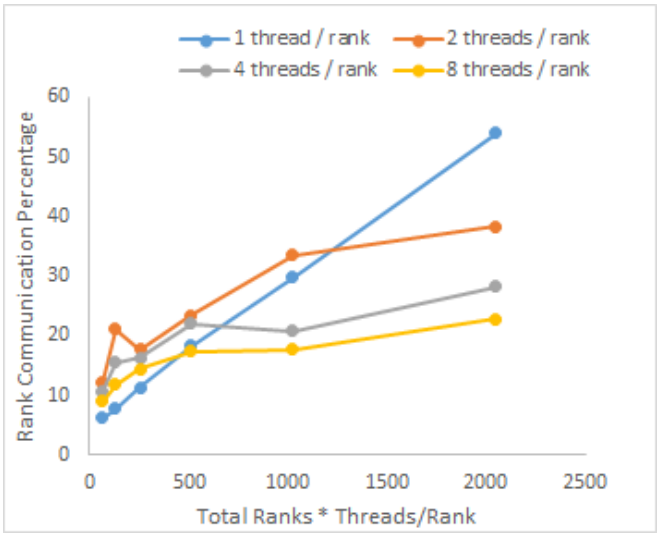

Fig. 4. Percentage of time spent on inter-rank communication

The disadvantage of the single pthread per MPI rank configuration lies in the percentage of time spent on communication between ranks as the system scales. Since rank universe sizes are smallest at this configuration, this leads to a proportional increase in the number of communications that need to happen between ranks. No matter the size of each ranks universe, the amount of data being transmitted between any two of them is always constant, as the boundary between ranks will always see the same level of flow of mice across it. Constant boundary size is inherent when partitioning the universe in only one dimension, and since mouse direction is randomized, on average there will always be the same number of mice that wish to move from the last row of one rank to the first row of the nest, and vice versa. Therefore, as rank universe size decreases, the number of mice traveling across their boundaries and necessitating communication increases proportionally, creating a directly inverse and linear relationship. This then means that more time is spent communicating between ranks for the single thread per MPI rank configuration relative to the others, and is verifiable by observing the linear trend of said configuration in Figure 4. Is is important to also note that the

configurations with more than one pthread per rank spend more of their time communicating when there are fewer parallel computational processes, due once again to mutex inefficiency. However, Figure 4 demonstrates that percent time spent on communication for those configurations is not linearly proportional to parallel computational processes, instead surpassing the 1 thread per rank configurations at very high levels of MPI ranks. This happens when the time lost due to mutexes for each of the three other configurations becomes less than the time the single thread per rank case spends on communicating between so many more ranks.

The one thread per MPI rank configuration also displays a linear decrease of real computational work being done as the number of MPI ranks increases, which is observable in Figure 5, inversely proportional to the trends seen in Figure 4. As discussed, the percent of time spent communicating also increases linearly over this domain, which explains where the rest of the time is going. The other three configurations also exhibit the inverse of the Figure 4 trends, including the points at which they each surpass the one pthread per MPI rank configuration.

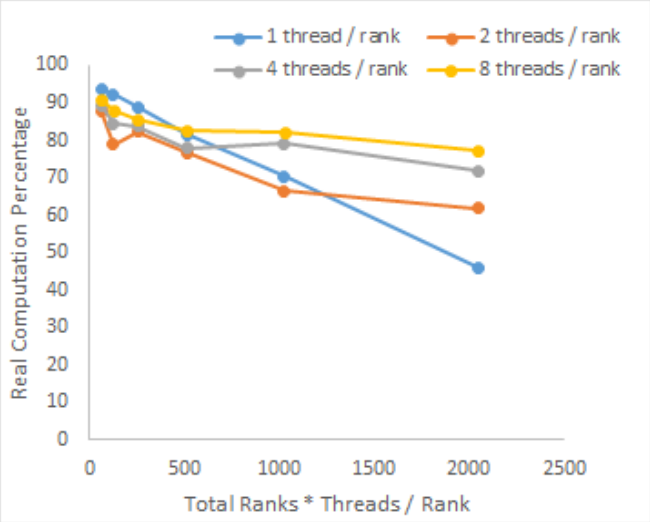

Fig. 5. Percentage of time spent on real computational work

C. Dispersion Visualization

In order to verify the correctness of the dispersion of lyme disease across the universe, heat maps displaying the positions of infected mice over the duration of the simulation were created. In order to model a domain in which carriers of Lyme disease could more thoroughly proliferate, the size of the universe was reduced to 1024 by 1024 nests. In larger universe simulations, Lyme disease could only spread as far as a mouse could travel in 180 days, which means Lyme disease could exist at most 180 nests from any initially infected one. In Figures 6 through 8, infected mice present in a nest at some point in simulation time are represented as black pixels, where nests with no infected mice show up as being white.

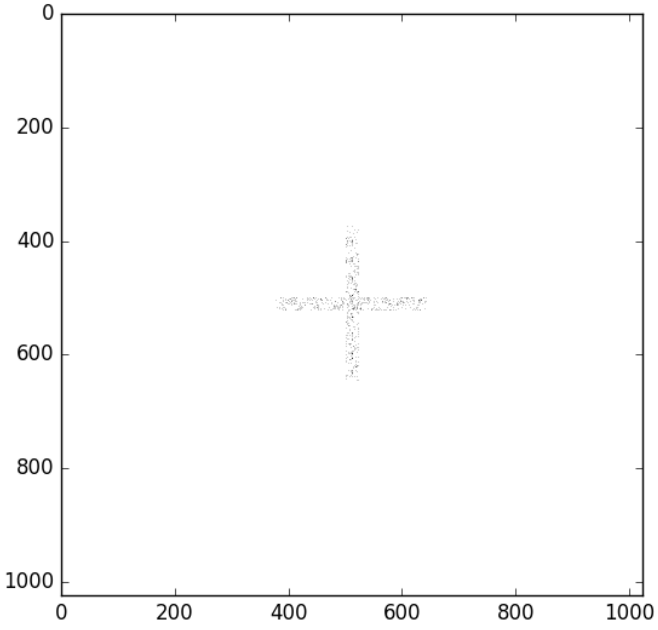

Fig. 6. Day 10 - Axis represents position in space

Figure 6 shows what the universe looked like at day 10 in the simulation, where a black cross can be seen in the middle of the universe. This represents the initial wave of mice that were infected by ticks that were initialized in nests with Lyme disease, which were assigned in such a way as to form a cross. Thus a cross appearing so early in the simulation is exactly what should happen, as no infected mouse has yet had a chance to journey farther afield, and the cross is of sufficient density as to indicate that mice are indeed traveling. Figure 6 therefore proves that infected ticks were initialized correctly, mice travel as intended, and mice are indeed contracting Lyme disease.

As the simulation went on, the positions of infected mice at day 90 were again collected to form Figure 7. This figure is more interesting than the last in that patterns have begun to emerge based on the constraints and behavioral rules of entities within the system. It can be seen that a darker band in the shape of a box has formed in the middle of the universe, indicating the highest concentrations of infected mice lie in this band. Beyond the band, the pattern of an eight pointed star emerges in lighter shades of black, indicating the farthest points any infected mouse could have traveled from the beginning of the simulation.

At the end of the simulation (day 180), a heat map was again made to gauge the full extent of the infections dispersal pattern, seen in Figure 8. The infection again forms the shape of an eight pointed star, proportionally larger than before as mice have had 90 more days to travel. The checkerboard pattern that now appears comes from having set ticks to drop off after traveling more than one day, and potentially indicates how infected ticks that drop off their original host are still capable of biting and infecting subsequent mice they come across.

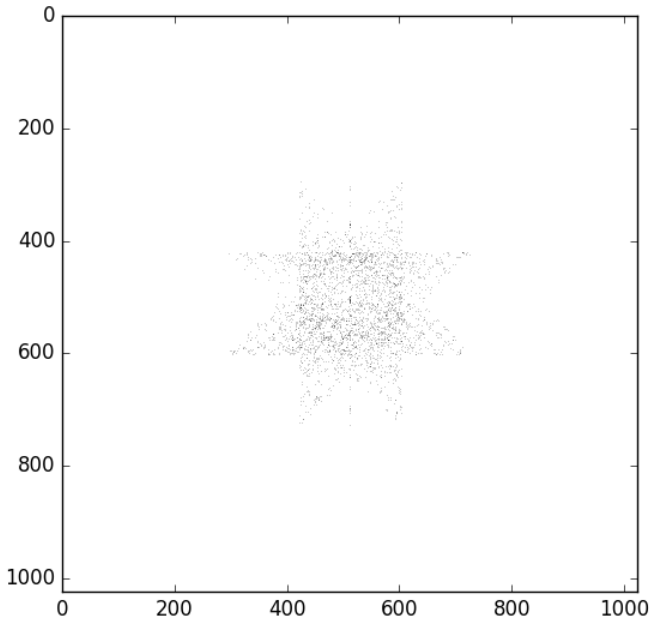

Fig. 7. Day 90 - Axis represents position in space

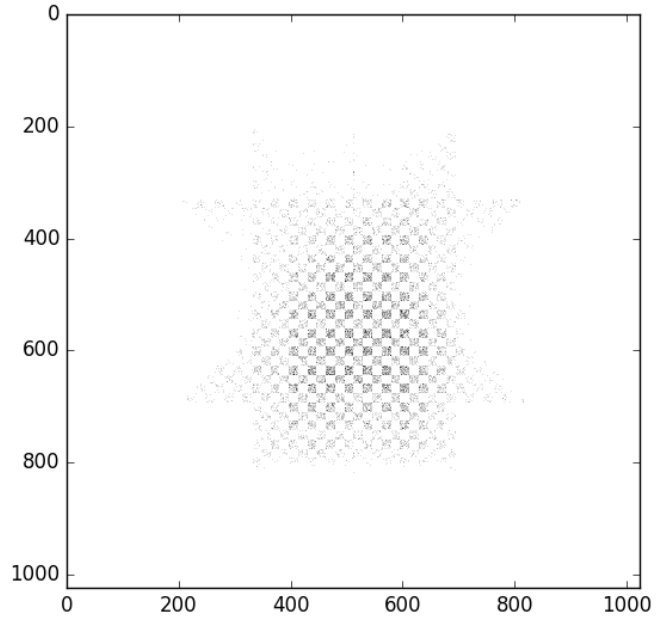

Fig. 8. Day 180 - Axis represents position in space

VI. CONCLUSION AND FUTURE WORKS

The results of the strong scaling experiments on the Lyme disease model show the direct relationship between increasing parallel computational tasks and performance, which prove the model to be massively parallel. Along with this, the dispersion of Lyme disease throughout the universe is verified as being consistent between implementation and model by the heat maps.

In terms of future work, the system could be extended to include the other 180 days of the year. This would include having ticks lay eggs, ticks reaching adulthood, ticks dying by reaching the end of its life cycle, and the impact of seasonal changes. For example, during the winter an animal host may be able to travel a smaller distance per day, which in turn reduces the rate at which Lyme disease would spread. Deer could also be added as hosts for adult ticks to feed on, increasing the rate at which Lyme disease would spread over the universe, since deer can travel much farther than mice in a day. The effect of mouse and deer (if also included) reproduction processes on the dispersion rate of Lyme disease could also be included in the model if the duration of the simulation were increased to a full year. The inclusion of reproduction would provide a constant influx of new hosts, replacing ones that die in the normal course of the simulation, and potentially allowing Lyme disease to more fully spread over larger universe sizes.

Adding nodes which are impassable to both mice and deer create the possibility to model terrain beyond a simple field, and would allow for the simulation of how Lyme disease spreads to more remote areas. In order to do this, smarter algorithms for movement of mice and deer would need to be implemented as well, otherwise the disease could not move into more complex areas. The simplest way to overcome this hurdle is taking a brute force approach, randomizing movement, and flooding the situation with enough entities to force flow to all areas of the universe.

TEAM MEMBERS

Every team member contributed to the success of the overall project. All team members helped run all the experiments conducted.

Anders Maraviglia: Jointly worked with Sara on the initialization of the universe, as well as the implementation of the iterations (including mouse movement and tick bites) and rank communication. Jointly worked with Sara on refactoring the code to optimize parallel performance. Wrote Implementation and Experiments, Results, and Analysis sections.

Sara Khedr: Jointly worked with Anders on the initialization of the universe, as well as the implementation of the iterations (including mouse movement and tick bites) as well as rank communication. Jointly worked with Anders on refactoring the code to optimize parallel performance. Developed performance graphs used in Experiments, Results, and Analysis sections and wrote the Simulation Model section.

Nicholas Fay: Wrote the code for producing heat maps. Worked on formatting the final report and creating tables. Jointly worked with Anthony for linked list implementation for the data structures used.

Anthony Kim: Jointly worked with Nicholas for coding the linked list implementations. Worked on formatting the final report and wrote the Abstract, Related Works, Biological Overview, References, and parts of the Experiments and Results.

CODE LOCATION AND INSTRUCTIONS

The project website, which contains the source code and details on how to run the simulation, is hosted on GitHub

and can be found at: https://sirmarcis.github.io/Massively-Parallel-Lyme-Disease-Simulator/.

The code and scripts to run the simulation can also be found in Anders Maraviglia's home directory on Kratos under the project folder. The full path is PPCmarava2kratos.cs.rpi.edu:/home/parallel/2017/PPCmarava2/final-project. Refer to the included README file for information on how to run code and scripts.

ACKNOWLEDGMENTS

We would like to thank Professor Christopher D. Carothers for teaching us the parallelization tools such as MPI and pthreads that were used in this project. Professor Carothers also helped us come up with our project idea. We would also like to thank Rensselaer Polytechnic Institute for allowing us to use its Advanced Multiprocessing Optimized System (AMOS). All of our experiments were run on this IBM Blue Gene/Q supercomputer.

REFERENCES

[1] ”Lyme Disease: Introduction to Symptoms, Diagnosis and Treatment.” LymeDisease.org. N.p., n.d. Web. 29 Apr. 2017. https://www.lymedisease.org/lyme-basics/lyme-disease/about-lyme/.

[2] ”Transmission.” Centers for Disease Control and Prevention. Centers for Disease Control and Prevention, 04 Mar. 2015. Web. 29 Apr. 2017. https://www.cdc.gov/lyme/transmission/index.html

[3] E. Deelman, T. Caraco and B. K. Szymanski, ”Simulating lyme disease using parallel discrete event simulation,” Proceedings Winter Simulation Conference, 1996, pp. 1191-1198. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=873423

[4] D. M. Rao and P. A. Wilsey, ”Accelerating spatially explicit simulations of spread of Lyme disease,” 38th Annual Simulation Symposium, 2005, pp. 251-258. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1401972&tag=1