Distributed Individual-Based Simulation

Jiming Liu, Michael B. Dillencourt, Lubomir F. Bic, Daniel Gillen, and Arthur D. Lander

University of California
Irvine, CA 92697
bic@ics.uci.edu
http://www.ics.uci.edu/~bic

Abstract. Individual-based simulations are an important class of applications where a complex system is modeled as a collection of autonomous entities, each having its own identify and behavior in the underlying simulated space. The main drawback of such simulations is that they are extremely compute-intensive. We consider the class of individual-based simulations where the simulated entities interact with one another indirectly through the underlying simulated space, significant performance improvement is attainable through parallelism on a network of machines. We present a data distribution and an approach to reduce the communication overhead, which leads to significant performance improvements while preserving the accuracy of the simulation.

Keywords: individual-based simulation, parallel and distributed computing, performance.

1 Introduction

Individual-based simulations, also known as entity or agent based models, are applications used to simulate the behaviors of a collection of entities, each having its own identity and autonomous behavior, and interacting with one another within a two- or three-dimensional virtual space. At each time step, an entity decides its behavior by interacting with its nearby environment and/or other entities. Typical examples of such applications are interactive battle simulations [1], particle-level simulations in physics [2], traffic modeling [3], and various individual-based simulation models in biology or ecology [4, 5, 6, 7]. Advanced graphics and animation applications, especially those involving large numbers of individuals, such as the animation of a flock of birds, have also taken advantage of spatially-oriented individual-based modeling [8].

There are two types of interaction in individual-based simulation: (1) between entities, and (2) between entities and the environment. Different applications require different types of interaction. For example, n-body problems or the simulation of schooling/flocking behaviors of various animals requires only entity-to-entity interaction because the surrounding space is empty or homogenous. Ecological simulations, on the other hand typically require both types of interaction. For example a species of fish interacts with the environment (e.g. by consuming renewable resources) but also with one another. A third type of simulation requires only entity-to-environment interaction. Many particle-level simulations fall into this category because the size of any given particle (e.g. a molecule) is so small relative to the space that collisions can largely be ignored. For example, simulating the propagation of particles (e.g. chemical agents) through living tissue falls into this category. Note however, that while there is no direct entity-to-entity communication, entities still influence each other by modifying the environment, i.e., they communicate with one other indirectly. In particular, multiple entities may compete for a given resource but only one can obtain it. Such simulations where entities communicate with the environment and thus indirectly with one other are the subject of this paper. Specifically, we address the problem of performance: How to distribute and parallelize an individual-based simulation to take advantage of the resources of a computer network or cluster. The major issues that arise in implementing such a parallel computing system include: (1) dividing the problem into small portions, (2) providing a distributed computing environment to support the implementation, and (3) ensuring that the simulation is correct, in the sense that it provides results consistent with what sequential implementation produces.

2 The Particle Diffusion Model

The target application of this research is particle diffusion in a biological intercellular space, based on the molecules diffusion theory of random walks in biology [9]. The biological system consists of cells arranged in a 2-dimensional space. New particles (molecules) enter from one side and, using Brownian motion, propagate through the space between the individual cells. Each cell is equipped with receptors, capable of capturing particles that come close. A captured particle may be released or it may be absorbed after some time. The receptors also move along the cell walls thus aiding the transport of the particles though the space.

Simulated Space. The simulation model represents the cells as equal-sized rectangles arranged in space as shown in Figure 1(a). The left boundary is a closed boundary; particles attempting to go back past the left boundary are bounced back to the simulated space. The right boundary is open; particles that walk across the right boundary disappear from the simulated space. The top and bottom boundaries are connected. Thus, topologically, the space is a cylinder, closed on the left and open on the right.

The Cells. Each cell (Figure 1(b)) is a square with a size of $10\mu m$ by $10\mu m$ and the distance between cells is one tenth of the cell size, i.e., $1\mu m$. The wall of each cell is divided into a number of segments (generally 20).

The Receptors. Receptors reside on cell walls. Each receptor has two states: free or occupied. When a receptor captures a particle its state changes from free

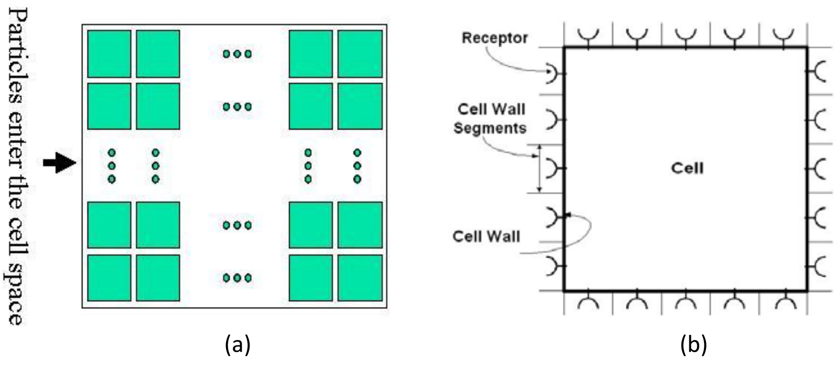


Fig. 1. (a) The simulated space; (b) One cell

to occupied. When the stuck particle is released or degraded (absorbed), the receptor changes its state back to free. Receptors, both free and occupied, move between neighboring cell segments at a predefined rate at every simulation time step. Thus the number of receptors in a given segment varies over time. This movement balances the number of free and occupied receptors on the cell segments.

The Particles. Particles are identified in the system by their position and state. The position of a particle is defined by its coordinates, x and y, in the simulated space. A particle does not own a territory. Therefore the density of particles in the simulated space has no limit. The total number of particles entering into the system (as well as many other spatial or behavioral characteristics) can be varied to simulate different cases, according to biological assumptions.

New particles enter into the simulated space periodically during the course of simulation. A particle starts as a free particle and can travel freely through the alleys (intercellular spaces) or be captured by a free receptor on a cell segment.

In the basic model (simulating the Brownian motion), each free particle moves during every iteration for a fixed distance (the micro step, or $10^{-9}m$) in one of four compass directions chosen randomly with equal probability. If it comes within a certain distance of a free receptor, it is captured. The main problem with this model is that it is too slow. For example, to simulate one biological minute requires on the order of 10^8 iterations. An improved model has been presented in [10]. This moves particles in macro steps using a Gaussian distribution and it determines the events of capture, release, and degradation probabilistically. This model improves performance by several orders of magnitude. The following sections describe how performance can be improved further by distributing the data and parallelizing the code.

3 Basic Distributed Implementation

The simulated space is partitioned into horizontal strips (using an Eulerian decomposition) and each strip is assigned to a different node. This partitioning preserves the locality of particles, which minimizes communication. It also preserves

```
1. while(current time < simulation time)  {
2.    Receive messages from left and right neighbor
3.    Update own simulation space
4.    Move Receptors for all cells
5.    Process Stuck Particles
6.    Process all Free Particles
7.    Move all emigrating particles into left and right messages
8.    Send messages to left and right neighbor
9.    Increment current time
10. }
```

Fig. 2. Basic simulation cycle

load balance, because there are large numbers of particles, all of which have the same behavior, and thus every node manages the same number of particles on average. Each node executes the same code shown in Figure 2, which implements a basic time-driven simulation. At the beginning of each iteration of the simulation run, each node waits for a message from its left and right neighbors (line 2). These messages contain particles that have migrated into the space managed by the current node from its neighbors. The node places them into its simulation space (line 3). Next the node processes all receptors by moving them probabilistically along the cell walls according to the simulation parameters (line 4). Some of the stuck particles are degraded and disappear from the system while other are released as free particles (line 5). Next all free particles are processed by moving them to their respective new positions and determining if they become stuck (line 6). All particles whose new position is outside of the nodes assigned space are extracted (line 7) and sent to the corresponding neighboring nodes (line 8). Finally, the simulation time is incremented to the next step (line 9).

4 Reducing Communication Overhead

Our goal is to distribute and parallelize the simulation model to speed up its execution. The problem with the straight-forward implementation outlined in the previous section is its excessive communication overhead: Nodes must exchange information at every iteration. Because of this overhead, this implementation is actually slower than the sequential implementation. The obvious solution is to exchange information less frequently. We define an epoch as a time interval (number of iterations) between two consecutive data exchanges. This optimistic version of the simulation reduces communication overhead but introduces two new problems:

1. A particle can move across to a neighbor node during any iteration within an epoch but because nodes do not exchange information at every iteration, the original node must continue tracking the particle even though it is no longer in its assigned region. That problem can be solved by introducing shadow cells.

2. Because a node does not find out about incoming particles until the end of each epoch when information is exchanged, it is operating with out-of-date information, which can cause particles to become stuck when they should not, or vice versa. This must be resolved using an explicit conflict resolution scheme at every epoch.

4.1 Shadow Cells

We create shadow cells to extend the local node boundary in order to allow the node to track particles that have emigrated from its assigned space. A shadow cell is simply a copy of the neighbor cell taken at the beginning of the epoch. Consider for example a rectangular space of 5x5 cells. Assume the each row is mapped on a different node. That is, node 1 holds the cells 11, 12, 13, 14, 15; node 2 holds the cells 21,22, 23, 24, 25; node 3 holds the cells 31, 32, 33, 34, 35; and so on. Then node 2, in addition to its own cells, will also hold copies of the cells 11, ..., 15 (of node 1) and 31, ..., 35 (of node 3) as shadow cells. Note that nodes 1 and 5 are also neighbors because the space is an open cylinder. Thus node 1 will hold copies of the cells 51, ..., 55 (of node 5) and 21, ..., 25 (of node 2). Shadow cells are updated at every epoch when nodes exchange information with one another.

4.2 Conflict Scenarios

The shadow cells allow nodes to continue tracking particles that have left their assigned space. The problem is that the receiving nodes will not know about the new incoming particles until the end of the epoch. As a result, they may take actions that would be different from those taken if the new particles were visible immediately. Such conflict situations must be recognized and corrected at the end of every epoch. There are two different scenarios under which a conflict occurs.

Scenario 1: A free particle is captured when it should not Figure 3 shows a situation where, due to out-of-date information, a particle is captured by a receptor that is no longer free. The left half of the diagram shows the sequential execution. At T2, a free receptor captures a particle (solid circle). At T3, another particle (hollow circle) approaches the same receptor but continues as free.

The right half of the diagram shows a distributed execution of the same situation. Node 1 controls the top row of the original space; node 2 controls the bottom row. The corresponding shadow cells are labeled with the letter S. At time T0, the beginning of the epoch, the system is synchronized by exchanging data between neighbor nodes. Thus the shadow cells are exact copies of their real counterparts. At time T1, one particle (hollow circle) walks across the space boundary of Node 2. It enters the shadow region of Node 1 but that node does not see it in its real space. At time T2, Node 1 captures the solid particle but this is not visible to Node 2. Consequently, at time T3, Node 2 captures the

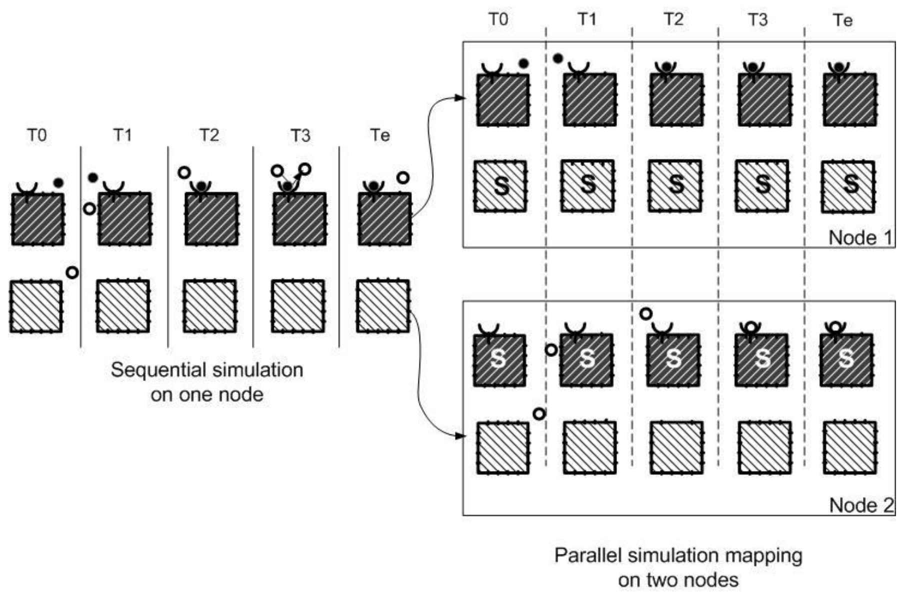

T0
T1
T2
T3
Te
Sequential simulation on one node
S
Node 1
Node 2
Parallel simulation mapping on two nodes

Fig. 3. Sequential and parallel execution of scenario 1

hollow particle. The resulting state at the end of the epoch, Te, is inconsistent with the sequential execution because both particles have been captured by the same receptor. The conflict must explicitly be resolved.

Scenario 2: A free particle is not captured when it should Figure 4 shows a situation where, due to out-of-date information, a node fails to capture a particle by a free receptor. The left half of the diagram shows the sequential execution. At T1, a previously captured particle (solid circle) is degraded and the receptor becomes free. As a result, it captures a new particle (hollow circle) that approaches it at T2.

In the distributed implementation, Node 2 does not learn about the degradation of the solid particle; the receptor appears as occupied in the shadow cell for the duration of the epoch. Consequently, Node 2 fails to capture the hollow particle approaching the receptor at T3. The resulting situation at the end of the epoch, Te, is thus inconsistent with the sequential execution.

5 Conflict Resolution

5.1 Solution to Scenario 1

The conflict resolution scheme is to take snapshots of every event of particle capture that occurs during the epoch; these remain tentative until the end of the epoch, when the snapshots are exchanged and each capture either confirmed or rejected as part of the conflict resolution.

The record of every capture contains information about the particles location prior to capture, including the cell segment and the number of free receptors of the capturing segment, and other information necessary to replay the event at the time of conflict resolution. The record also differentiates between two different types of capture:

Own: The particle was captured within the cell belonging to the node
Shadow: The particle was captured within a shadow cell

At the end of every epoch, all records of tentatively captured particles are exchanged among neighboring nodes. A conflict occurs whenever one node records a capture of type own while its neighbor records a capture of type shadow in the neighbors shadow copy. This indicates that both nodes performed a capture of some particle in the same cell but, because their decisions were based on out-of-date information, one of the captures may not be valid.

Whenever a conflict is detected, each node reprocesses all tentatively stuck particles based on the now accurate information contained in the exchanged records. The goal of the resolution is to find the tentatively stuck particles that should not become stuck, and then release them back to remain as free particles.

To illustrate the above conflict resolution process, consider again the scenario of Figure 3. Node 1 records a capture at time T2. This capture is of type own because the particle resides in the cell assigned to Node 1. Node 2 records a capture at time T3; this is of type shadow because the particle resides in the shadow cell corresponding to the actual cell of Node 1. The two records are exchanged by the two nodes at the end of the epoch and, because they are of different types, conflict resolution is carried out by both nodes:

1. At time T2, the first record is processed. The information in the tag matches the actual local information, i.e., there is a free receptor available in that cell segment at this time. Consequently, the tentatively stuck particle is confirmed as stuck and the free receptor becomes occupied.
2. At time T3 the second record is processed. At this time, there is no free receptor remaining in the cell segment (as a result of the previous step) and hence the second tentatively stuck particle cannot become stuck. It is released as a free particle and its new location at the end of the epoch is calculated.

The result of the conflict resolution then matches that of the sequential simulation shown in Figure 5, where only the first particle has been captured.

5.2 Solution to Scenario 2

Scenario 2 is more difficult because it is caused by the absence of a capture event. Obviously, there is no record of such a non-occurring event (it is not possible to record all events that have not occurred.) Hence there is no information for the conflict resolution to use. Our solution to this problem is to overstate the number of free receptors in the shadow cells at the beginning of the epoch. Overstating this number by one allows one additional particle to be captured that should

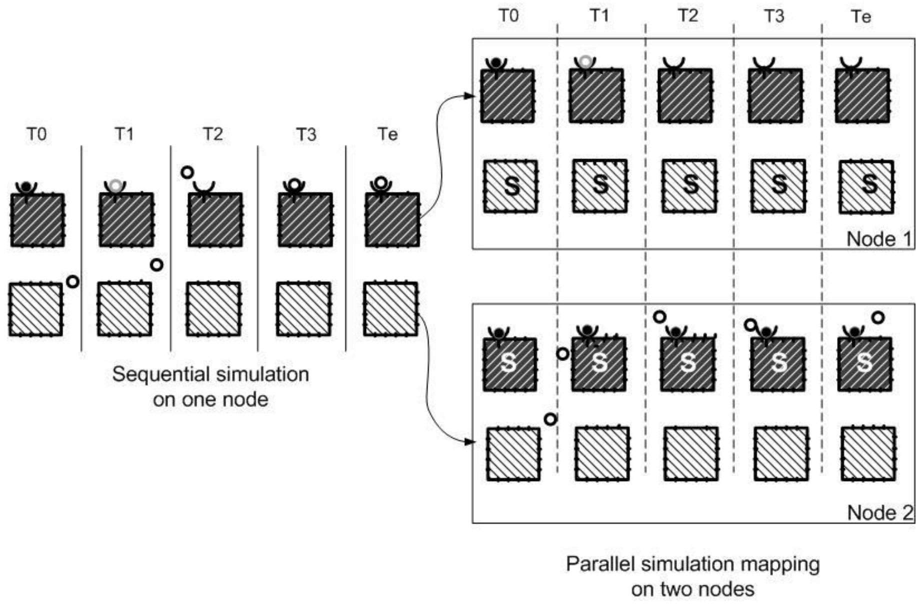

T0
T1
T2
T3
Te
Sequential simulation on one node
S
Node 1
Node 2
Parallel simulation mapping on two nodes

Fig. 4. Sequential and parallel execution of scenario 2

not have been captured. If this occurs, however, the conflict resolution scheme described previously will detect and correct this error.

To illustrate this conflict resolution process, consider again the scenario of Figure 4. The number of free receptors in the shadow cell maintained by Node 2 is now artificially increased to 2 instead of 1. At time T1, Node 1 records the degradation of the captured particle; this event is of type own. At time T3, Node 2 records a capture of type shadow. This is now possible because there are 2 receptors, one of which is still free.

The two records are exchanged by the two nodes at the end of the epoch and, because they are of different types, conflict resolution is carried out by both nodes. The reprocessing now uses the actual (not overstated) numbers of free receptors: At time T1, the particle is degraded and the receptor becomes free. At time T3, the second capture is also correctly confirmed because of the availability of the receptor released during the previous step. The result of the conflict resolution then matches that of the sequential simulation shown in Figure 4, where both particles have been sequentially captured.

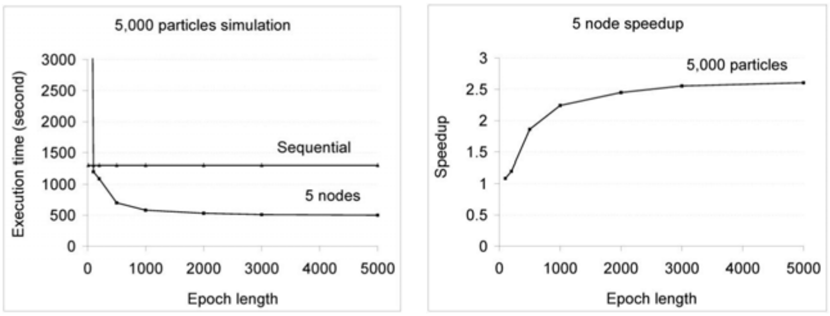
6 Performance Evaluation

The main objective of the distributed implementation is to increase performance and the most important measure is the speedup achieved over the corresponding sequential implementation. This is achieved by increasing the epoch length, which reduces communication overhead. However, with increased epoch length the probability of conflict increases. Furthermore, some of the conflicts become unrecoverable. Specifically, a node cannot track a particle beyond its shadow cells; when a particle walks too far during the epoch, it becomes lost. Similarly, overstating the number of receptors in the shadow cells by 1 (earlier scenario 2) can handle only a single conflict. Overstating by more than one would handle multiple conflicts but would also result in additional recovery overhead caused by the phantom capture events caused by the additional (non-existent) receptors. For all these reasons, the epoch should be kept as short as possible. Hence the ultimate goal is find the ideal epoch length where the combined effects of communication and conflict resolution are minimized.

Experiment 1. The simulated space in this experiment consists of 50 cells organized in 5 rows with 10 cells in each row. New particles enter into the system from the left boundary of the simulated space at a rate of 0.001 particles per iteration per cell row. The experiment runs for 1,000,000 iterations thus simulating a total of 5000 particles. The parallel implementation uses 5 nodes with each cell row assigned to a different node. We run the experiment with the epoch lengths of 10, 100, 200, 500, 1000, 2000, 3000 and 5000.

Figure 5(a) shows the execution times for the sequential and parallel simulations. Figure 5(b) shows the corresponding speedups. The execution time for the sequential simulation is 1303 seconds. The break-even point where the sequential and the parallel implementation need the same time is with an epoch length of less than 10. There is a steep improvement in performance when the epoch length is increased from 10 toward 1000. Past that point, the improvement is only marginal because both communication and conflict resolution overheads are very small. The speedup with epoch lengths of 5000 is 2.62 (on 5 nodes).

Experiment 2. In this experiment, we increase the simulated space to 100 cells organized in 10 rows with 10 cells in each row. Each row is mapped on a separate node. New particles enter into the system at a rate of 0.001 per iteration per cell row. There are total of 10,000 particles simulated in this experiment. The epoch lengths are the same as in the previous experiments.



(a) (b)

Fig. 5. Execution times and speedup of experiment 1 on 5 nodes

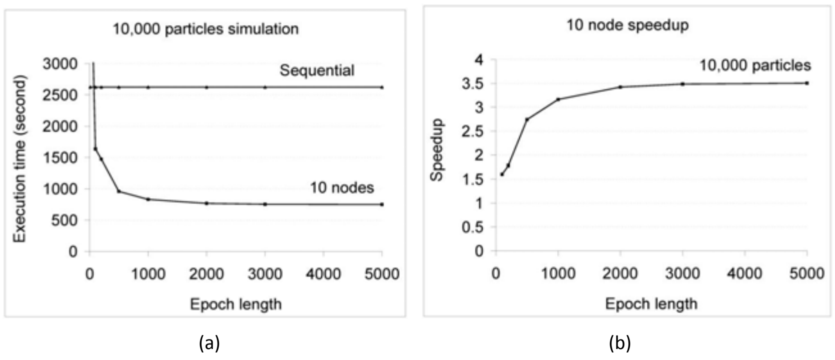


Fig. 6. Execution times and speedup of experiment 2 on 10 nodes

Figure 6 shows the execution times and speedups of this experiment. The execution time of the sequential simulation is 2625 seconds with a break-even point between 10 and 100 iterations per epoch. As before, the speedup improves significantly when epoch length increases toward 1000 but then becomes flat. The maximum speedup attained is 3.5. This is significantly better that the 2.62 speedup with 5 nodes (experiment 1) but 10 nodes were required to achieve this.

Experiment 3. This experiment investigates if a better speedup is obtainable when the problem size (in terms of the number of particles in the simulation) is greater. We keep the same simulated space but increase the number of incoming particles to 0.004 per iteration per row. There are now 40,000 particles simulated in this experiment, 4 times more than it in experiment 2.

Figure 7 shows the execution times and the speedups for this experiment. The execution time of the sequential simulation is 20712 seconds and the break-even point is again between 10 and 100 iterations. For comparison, Figure 7(b) also includes the speedup of experiment 2 (10,000 particles). The maximum attained

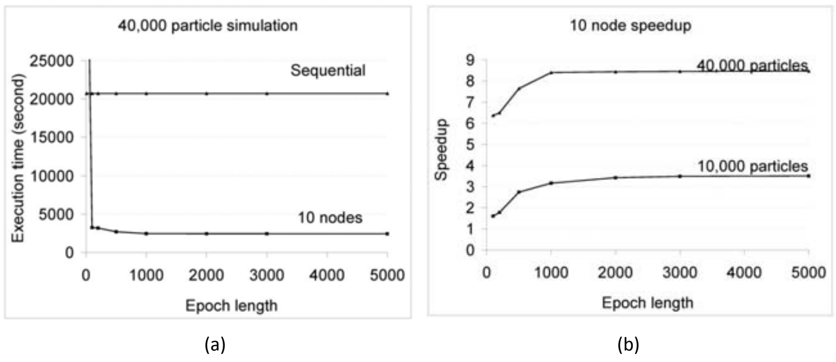


Fig. 7. Execution times and speedup of experiment 3 on 10 nodes

Table 1. Comparison of the speedup and accuracy at different epoch lengths

Iterations	500	1000	2000	3000	5000
Accuracy	100%	100%	**99.7%**	94.8%	92.1%
5 node speedup	4.06	**4.43**	4.43	4.41	4.43
10 node speedup	7.64	**8.40**	8.44	8.45	8.47

speedup of experiment 3 is 8.47 (on 10 nodes), which is significantly better than the speedup of 3.5 (attained on 5 nodes in experiment 2).

Performance versus Accuracy. These experiments show that the simulation is highly scalable. When nodes are added or the number of simulated particles is increased, the simulation attains significant speedup, especially when the epoch length increases past 1000 iterations. Increasing the epoch length past 1000 still gains some performance but lowers the accuracy. That is, some portion of the simulated particles are in different states or positions in the sequential and distributed implementations. This is because the conflict resolution scheme cannot resolve certain situations that occur when the epoch is very long.

Table 1 shows the trade-offs between performance and accuracy. With an epoch length of 1000 or less, accuracy is still at 100%. That is, the number and locations of all particles are identical in the sequential and distributed implementations. Between 1000 and 2000, some unrecoverable discrepancies begin to appear. On the other hand, the speedup increases dramatically when the epoch length is increase from 1 toward 1000 but is only marginal past 1000. This gives the user the option of choosing between 100% accuracy with a slightly lower performance or to aim for maximum speedup with a slight loss of accuracy.

7 Conclusions

Individual-based simulations are an important class of applications that are able to general result unattainable from statistics-based simulation models and hence are used frequently in various science disciplines. The main drawback of such simulations is that they are extremely compute-intensive. This paper considered a class of individual-based simulations where the simulated particles interact with one another indirectly by interacting with the underlying simulated space (biological cells and their receptors, in our case). We have demonstrated that an Eulerian space partitioning allows the simulation to effectively utilize multiple nodes, provided the communication overhead is decreased by the introduction of epochs and the necessary conflict resolution mechanisms to correct any inaccuracy resulting from the delayed information exchange.

The implementation presented in this paper is an optimistic simulation [11] but it differs from other approaches in several important ways. First, our simulation is time-driven rather than discrete-even-driven. All processes proceed at the same pace in virtual time and thus one cannot get ahead of another. Second, the periodic exchange of information is delayed by the epoch length, which means that, by design, all messages are straggler messages and require corrective action. However, a complete rollback is not necessary. Instead, we are able to correct individual conflicts at specific receptors through the mechanisms introduced in this paper. Thus the cost of the optimistic view and the need for recovery (conflict resolution) is by far outweighed by the gains in performance resulting from the dramatically reduced communication overhead.

References

1. DIS Steering Committee, The DIS vision: A map to the future of distributed simulation. Institute for Simulation and Training (1994)
2. Hockney, R.W., Eastwood, J.W.: Computer Simulations using Particles. IOP Publishing Ltd., Bristol (1988)
3. Resnick, M.: Changing the centralized mind. Technology Review, 33–40 (1994)
4. Huston, M., DeAngelis, D., Post, W.: New computer models unify ecological theory. BioScience 38(10), 682–691 (2001)
5. Huth, A., Wissel, C.: The simulation of the movement of fish schools. Journal of Theoretical Biology 156, 365–385 (1992)
6. Villa, F.: New computer architectures as tools for ecological thought. Trends in Ecology and Evolution (TREE) 7(6), 179–183 (1992)
7. Hartvigsen, G., Levin, S.A.: Evolution and spatial structure interact to influence plant-herbivore population and community dynamics. Proc. Roy. Soc. London Ser. B 264, 1677–1685 (1997)
8. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. Computer Graphics 21(4), 25–34 (1987)
9. Berg, H.C.: Random Walks in biology. Princeton University Press, Princeton (1993)
10. Liu, J.: Distributed Individual-Based Simulation, PhD Dissertation, Dept. of Computer Sci, University of California, Irvine (2009)
11. Fujimoto, R.: Parallel and Distributed Simulation Systems. In: Proc. 2001 Winter Simulation Conference, vol. 2 (2001)