Computational Social Science and High Performance Computing: A Case Study of a Simple Model at Large Scales

John T. Murphy

Argonne National Laboratory, Decision and Information Sciences Division,
9700 South Cass Avenue, Bldg. 221, Argonne, Illinois, USA, jtmurphy@anl.gov

Abstract. Advances in High-Performance Computing (HPC) offer Computational Social Science the opportunity to create simulations at scales vastly larger than in previous times. Here an example is presented showing solutions to practical problems of modeling at these scales. The exercise explores a novel variation on a well-known model (called the 'Triangles' simulation), implemented in the Repast HPC toolkit and run at scales up to billions of agents. The results show that model dynamics change when proportions of different kinds of agents are varied, and this is consistent at very large scales. Practical issues of performance, network analysis, file output and data visualization are illustrated with this worked example.

Keywords. Agent-Based Modeling, High-Performance Computing, Social Science Simulation, Visualization, Scalability, Triangles Simulation

1 Introduction

That agent-based models will become larger is a certainty. High-performance computing systems are making it possible to perform simulations at scales unimaginable only a few years ago. To social scientists, the possibilities offered by agent-based simulation are moving from simulations on the scale of small social groups toward simulations that can address the interactions of millions, or even billions, of individuals.

At the outset of this exploration we must admit that we are uncertain what differences this increased scale will reveal. Some simulations may reveal new nuances of complexity at larger scales; others will simply be 'more of the same.' The task undertaken in this article is an experiment in 'what if?': what if a well-known and very simple simulation is expanded to the much larger scale possible on a HPC platform?

This is presented as an exercise. The major contribution of the article is as an explication of a method used to create and work with an agent-based model run on a high-performance computing (HPC) platform. Most HPC platforms today are collections of independent processors running in parallel, each operating to carry out a portion of the larger computational problem being addressed; for the remainder of this essay I will assume that HPC implies high parallelism. Agent-based models on such systems must overcome both practical and theoretical challenges. The theoretical challenges arise primarily from the way that agent actions depend on one another; this can be specific to a given simulation, and may vary from simulations in which agents act mainly independently to those in which agent actions are highly interdependent. In the latter case, parallel approaches become extremely difficult to implement.

Practical challenges include the software and hardware needed to implement the parallelized actions of the agents. Additionally, they also include a number of other issues that are related to the broader modeling effort, and that will have to be addressed if agent-based modeling is to be brought successfully onto HPC platforms. In general these practical challenges are currently met, in contexts where the agent-based models are at typical scales, by software that is insufficient when the scale of the models and the data they produce becomes much larger.

2 The Model

2.1 The 'Triangles' Model

The simulation used for this study is based on the 'Triangles' simulation [1], which is sometimes used pedagogically and even performed as an exercise by human participants. There are several variations of the simulation, but in each one participants are asked to select two other players (henceforth 'A' and 'B'), and then to move about the room using those players as points of reference according to a rule. The different variants include different movement rules; of these, the two that are of concern here are:

1) Position yourself between A and B
2) Position yourself such that B is between you and A

These two rules lead to very different macro-level behavior. In the first, individuals converge toward the center of the room and the agents begin to collide, while in the second movement is centrifugal.

The additional element inserted in this study is to ask what would happen in a population that consisted of a mixture of individuals playing these two roles. A number of conditions seemed intuitively possible, including, for example, a system in oscillation around some average radius or a relatively static distribution around a given radius but with a 'hole' in the center of the system.

This arrangement leads to a categorization around two 'basic' types: agents following rule #1 are "Blockers," while agents following rule #2 are "Hiders." Additionally, an agent could be assigned one of eight 'extended' types, based on that agent's basic type and the basic types of the 'A' and 'B' agents that it had been assigned. The behavior of a 'Hider' might be different depending on whether it was 'hiding' from two Blockers, two Hiders, or one of each in either the A or B role.

For all simulation runs, N agents are initially placed randomly within a a square bounded at corners (-5000, -5000) and (5000, 5000). Agent location is a c++ double value, and agents can move without restriction (even beyond the initial boundaries). Each agent is assigned a type and selects two other agents at random from the entire population. At each time step, each agent assess the locations of its two target agents and calculates a target destination from these. The target destination will lie on a line that runs through the other two agents. If the agent is a 'Blocker', the target destination will be the closest point on this line that lies between the other two agents; if the agent is a 'Hider', it will be the closest point on this line that lies on the side of agent B that is opposite agent A. With this target in mind, all agents then move simultaneously, but with movement limited to no more than 10 units toward the target location.

2.2 The Repast HPC Toolkit

The simulation was implemented using the Repast High-Performance Computing (Repast HPC) toolkit, under development at Argonne National Laboratory [2]. The toolkit brings a subset of the functionality found in the Repast Simphony toolkit [3] to an HPC environment. Basic functions in the Repast HPC kit that echo some from Repast Simphony include managing the schedule of agent actions, allowing model parameterization via property files, and providing useful data collection structures. The functions more specific to the HPC environment deal with the theoretical challenge of sharing agent information across multiple processes. Repast HPC deals with this through a two-stage process. In the first, agents are 'requested' from other processes; the process that manages the agent will answer a request by providing the other process with a copy of the original agent. In the second, these copies are updated with the borrowed agents' current information. The effect is that an agent on a given process can derive information from and react to the current state of agents on any other process. While in the simulation presented here the 'request' for other agents is explicitly scheduled, Repast HPC also provides a way for spatially located and moving agents to interact with agents near themselves in simulation space but across a process boundary; when this functionality is used both the request and synchronization are handled automatically.

During testing, a bottleneck was discovered that limited the ability to use a purely native Repast HPC implementation. This bottleneck applied to only to the initial 'agent request' procedure and not to the exchange of agent information during the simulation run, and was a product of an aspect of this simulation that was somewhat unusual[1]; nevertheless, it limited the ability to use the initial code, called TrianglesDemo01. Instead, a second version, TrianglesDemo02, was created in which the code managing the exchange of agent information was written using low-level C/C++ and directly invoking code from the Message Passing Interface (MPI) library that provides the cross-process communication in the parallel HPC environment. Other functions, including the scheduling of agent actions, were still performed by the

[1] Specifically, the issue was the need for each process with N agents to have up to 2N non-local agents; in most simulations the proportion of non-local agents is likely to be much lower.

Repast HPC library. This second implementation both permitted larger runs and improved performance (see Section 4, below), but only at the cost of many of the advantages of a true ABM toolkit. In the context of this study, the most salient example of this cost is the inability in the TrianglesDemo02 code to perform aspects of the network analysis that are achieved easily in the TrianglesDemo01 code (see section 3.1, below). Additionally, in this version of the simulation agents are represented only by spatial coordinates and do not interact with or affect other agents except through the A/B dependencies; the possibility of making agents 'solid', so that they have a size and can obstruct other agents' movement or perception, would be more easily facilitated in Repast HPC but added only with great difficulty to TrianglesDemo02.

2.3 Hardware

Although the code could be executed in nearly any parallel environment, the model presented here was created for and executed on the Blue Gene/P supercomputer (BG/P) at Argonne National Laboratory, maintained by the Argonne Leadership Computing Facility (ALCF). All simulation runs described here were done using 'vn' mode, which divides 2GB RAM per compute node among 4 core processors, giving a 500 MB RAM per process limit. For the full specifications of the BG/P, see [4]).

3 Practical Isssues

Three practical issues will be described here. The first is the analysis of the network that results from the association among the agents and their A & B targets. Although many other tools for network analysis exist, in this study an anlysis was built into the simulation code, to deal with networks of sizes beyond those that currently available software can accommodate.

The two other practical issues are file output and visualization. In an inductive study such as this one, visualization is a key tool for gaining insight into the dynamics of the simulation. However, care must be taken to select the appropriate data to be collected as well as to employ a format amenable to visualization, and thus file output and visualization must be considered together.

3.1 Network Analyses

What is meant by 'network analysis' here is far short of what might be done; social network analysis and graph theory offer much more than is examined here (see [5], for example). But in the event that an agent's behavior in the simulation is to be examined more closely, some aspects of the agents' network might be relevant.

Two aspects of the network that seemed particularly likely to be important drove the analyses here. One was the existence of cycles: the method by which the network is created guarantees that the network will include at least one cycle, a 'loop' such the movements of an agent in the loop depends on agents whose movements ultimately depend on the original agent's. While there will always be at least one of these, it is also possible that there will be multiple, independent loops; the network is not guaranteed to be fully connected. If this were the case, the effect would be that it would be possible to have one set of agents being simulated simultaneously with, but never affecting nor being affected by, another set of agents; these would essentially be independent simulations. If ordered behavior was found to emerge within these groups, then independent subsets of the simulation could diverge from each other in distinctive ways.

Second, while there will definitely be such loops, a particular agent need not be positioned within any of them: some agents will be at the borders of the network, with no other agents depending on them at all. These agents are information sinks; one reason to identify these 'trailers' is that they may have interesting behavior, but another is that they may be removed completely from the simulation without affecting the agents who are participating in the cycles and mutually affecting each other.

Network analysis presents special challenges when the structure of the network (the association of one agent with another) is distributed across multiple processes. To overcome these challenges, a pattern of processing and exchange of information across processes was designed; the specifics are beyond the scope of this brief article, but one point is salient. The algorithm was implemented as a two-step process. During the first step, information flows 'down' the network, from agents to their A and B agents; this is the reverse of the flow of information during the simulation, and required specialized communication code to facilitate the coordination necessary among the processors. During the second step, information flows 'up' the network; this matches the natural flow of information through the system, and, despite the fact that the information passed was of a complex structure, was therefore easily achieved using Repast HPC's native agent synchronization functions.

The end results of these analyses are that agents are identified in several ways. First, they are marked as either 'trailers' or not; trailers additionally receive a number specifying the 'distance' they are from the nearest cycle from which they receive information. Second, distinct subsets of agents are revealed: any cycle into which no outside information flows is marked with a unique ID. These IDs, as well as the indications of trailer status, become attributes of the agent that are stored with the rest of the simulation data and made available for visualization.

A final, special analysis was also implemented: the ability to select one agent and mark the agents from which that agent draws information. The target agent's immediate A and B nodes are marked with 1, their respective A and B nodes are marked with 2, etc. This was intended to help understand the behavior of a specific agent identified via the visual analysis of some set of simulation output. For examples, see Figures 1 and 2.

3.2 File Output

Large-scale simulations can generate very large data sets. In common cases, most of the simulation’s state would be allowed to be lost, and only summary data, such as the number of agents meeting certain criteria, would be collected. For this study, the challenge of outputting what amounted to the entire state of the model- strictly, the invariant attributes of all of the agents and their X,Y locations at specified intervals- was undertaken as an exercise and to allow for thorough data visualization.

An initial attempt was made to output the data as a garden-variety comma-separated-values (CSV) file; when the simulation was run in parallel, this would mean either outputting each process's data to a separate file (which would then need to be merged together later) or having each process write to the same file in turn. The latter approach was found to slow the simulation performance down to unreasonable levels.

A specialized toolkit for high-performance computing was adopted: the HDF5 library [6]. HDF5 is a file format that allows for compact storage of data in any desired hierarchical structure. It is accompanied by a library (with versions in C and FORTRAN) that allows data to be written and read from files in this format, and this library includes a version that permits parallelization. In the parallel version, each process maps a section of its memory from which to write, and also maps a section of the file to which to write; the destination sections are patterned, and individual components of this pattern are allocated among the running processes so that the data, when written, fit together as if written from one large block of contiguous memory. In this case this was further aided by the underlying hardware of the BG/P, which uses a parallel virtual filesystem that allows multiple processes to write simultaneously.

While very effective, the HDF5 output does carry a processing cost. A benchmark run (see below, Section 4) for 512 processors and 16,384 agents per process took 3 seconds for 100 time steps without reporting, and 86 seconds when HDF5 reporting occurred every time step. Further, when the total number of timesteps was increased to 500, the run took 1991 seconds, or ~400 seconds per 100 time steps.

3.3 Visualization

While the actual maximum number of agents that would be simulated was unknown, it was assumed that it would be very large; hence a data visualization package that could support extremely large data sets was needed. The package selected was ParaView, which is built on top of the Visual ToolKit (VTK) [7]. ParaView has been used for data sets involving tens of millions of elements, and can be run in parallel, allowing a cluster of graphics processors to be used in tandem. The ALCF has such a cluster, and by using it real-time interaction with extremely large data sets can be achieved.

Having said this, however, it is not straightforward to import data into ParaView from an agent-based model. ParaView has no native format for the basic data in this simulation: the 2-Dimensional X,Y coordinates of the agents through time. The solution was the creation of an XDMF file [8] that provided ParaView with the structure of the data and translated it into visualizable elements. Unfortunately, an XDMF file appropriate to a given data set must be customized according to the properties of that data set- the number of agents, the number of time steps, and the different agent attributes recorded, inter alia- and thus a program (this one in Java) had to be created to generate the XDMF file required to view any given set of simulation data in ParaView. Eventually a more streamlined process may be achievable.

A final concern was how to represent the data. Three approaches were selected (each requiring a different XDMF file, to transform a given data set into the objects to be projected by ParaView). In the first, the data are presented as 'nodes' in 3-D space; the x and y coordinates are as they are in the simulation, and time steps are represented by the z coordinate. In the second, what were nodes in the preceding view become endpoints for line segments, showing, essentially, the trajectory of the agent. In the third, a 2-d representation is used, with the third dimension being animated using ParaView's time series data animation function. See Figures 1 - 5 for examples.

4 Performance and Scaling

Standard measures of the performance of parallel computation include tests of scalability. One kind of scaling is the improved performance gained by taking a problem of given size and dividing it across processes; a second is the ability to maintain a consistent performance when the number of processes is increased but the problem is increased commensurately [9].

Table 1 presents the results of performance benchmark tests for the Triangles simulation. The tests were performed by recording the time taken to complete 100 time steps; the benchmarks do not include initialization or data collection. Only one run was performed for each test, because each test run is effectively an average for the 100 time steps (but the values, in seconds, are given for the total test). In every test half of the agents were Hiders and the other half Blockers. For these benchmarks, the version of the simulation used was the second, more purely MPI version (TrianglesDemo02). It should be noted that performance tests were also done on TrianglesDemo01, and that the performance of TrianglesDemo02 is faster due to the direct MPI implementation.

The table records the number of processors in columns, and the number of agents *per process* in rows. Note that along each axis the values increase in powers of two. The table can be read in two ways. Scanning across a row gives an measure of the second form of scaling. Each column represents the doubling of the number of agents; the time required increases, but at a rate far less than multiples of two[2].If the values are read diagonally upward to the right, the other form of scaling is demonstrated. Each cell represents the same number of agents as the other cells along this diagonal. It can be seen that the simulation represented at the bottom left (2,097,152 agents x 16 processes = ~33 million) is the same number of agents as that of the top right (1024 agents x 32,768 processes), but the performance is dramatically faster.

[2] Also of interest is the transition that occurs between 1024 and 2048 processes, or 256 and 512 compute nodes, when time *decreases* noticeably. The reason for this is the architecture of the Blue Gene/P. For jobs using at least 512 nodes an additional dimension in the communication network path is available, transforming the 'mesh' used on smaller jobs into a full 'torus' grid, and improving performance accordingly by providing an additional, often shorter, path via which to exchange information from process to process.

Table 1. Performance from start to finish of 100 time steps, by number of processes (columns) and number of agents/process (rows), showing time, in seconds.

	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1024	<1	<1	<1	<1	1	<1	1	1	2	7	13	18
2048	<1	1	1	<1	1	1	1	1	2	8	14	24
4096	<1	<1	<1	1	1	2	2	2	3	9	16	24
8192	1	1	1	1	2	3	3	2	6	11	18	29
16384	2	2	3	2	3	4	5	3	7	23	24	39
32767	4	5	5	5	5	7	8	7	11	37	48	46
65536	9	9	10	10	11	14	16	13	21	40	52	91
131072	19	19	21	21	22	28	30	24	38	72	105	100
262144	37	40	45	48	49	61	64	51	73	127	149	163
524288	74	81	91	98	104	125	131	105	147	239	261	241
1048576	150	162	182	196	210	256	265	215	287	430	482	506
2097152	299	324	363	391	418	510	530	432	568	839	891	941

Each agent-based model will scale differently, and the performance of this one may eventually prove to be atypical. Nevertheless, the implementation of an agent-based model with ~68 billion agents (at the maximum represented in the table, 2,097,152 agents x 32,768 processes) suggests that very large scale models are now within reach.

5 Observed dynamics

The observed dynamics of the simulation show an interesting effect of the mixture of the two agent types, and a (disappointing) lack of effect due to the expansion of scale.

Generally speaking, the simulation begins with all agents evenly filling out a square space, and moves to a condition in which the agents are distributed rather evenly in expanding circles from some apparent center point; typical start and end distributions for a simulation are shown in Figures 3 and 4. If the simulation is allowed to run long enough, some agents arrive at a point virtually atop their target agents, and can track them indefinitely; others stop moving entirely, and some runs have been observed that approach a stable state, though the exact circumstances that allow this are not fully understood.

Figure 5 shows the results of the test runs of the simulation, using three graphs for three values representing the proportion of 'Hiders' in the total population. Ten runs for each parameter set were run; each run consisted of 2000 time steps, with data collected every 20 time steps. The results presented show the average of the last 10 data collection events, or 200 time steps. The data series in the graphs represent different agent types; basic types of 'Hider' and 'Blocker' are represented, as are extended types in a notation that indicates Basic Type: A's Basic Type, B's Basic

Type. The values on the y-axis are the distance from the points that marks the average location of all agents.

As expected, varying the proportions of Hiders and Blockers noticeably changes the character of the simulation. When the simulation is run with 90% Blockers, agents of all types are closer to the center; when run with 90% Hiders, agents generally move further from the center. Moreover, the subtypes show the effects of this most dramatically: Hiders whose A and B agents are also Hiders tend to scatter the furthest, while Blockers whose A and B agents are both Blockers tend to be found closest to the center. Note also that Hiders' positions tend to be determined by their B agents' types: if their B is also a Hider, they scatter very far, regardless of their A agent's type. Conversely, Blockers cluster into two groups depending on whether their A and B agents are the same: Blockers with A and B agents that are both Hiders scatter, while Blockers with A and B agents that are both also Blockers cluster toward the center; Blockers whose A and B agents are mixed arrive at the same distance, regardless of which agent is which kind. This is in keeping with the implementation: Blockers deal with their A and B agents symmetrically, moving to get between them regardless of which one is which type, whereas Hiders distinguish between the B agents behind which they conceal themselves and the A agents from which they are hiding. The averages for the basic types appear to shift as well, and even to change the ordering of the series, but this may be primarily due to the fact that their subtypes occur in different proportions. Additional work might be done to determine if the shifts seen here are smooth or inflected, but this is not undertaken here.

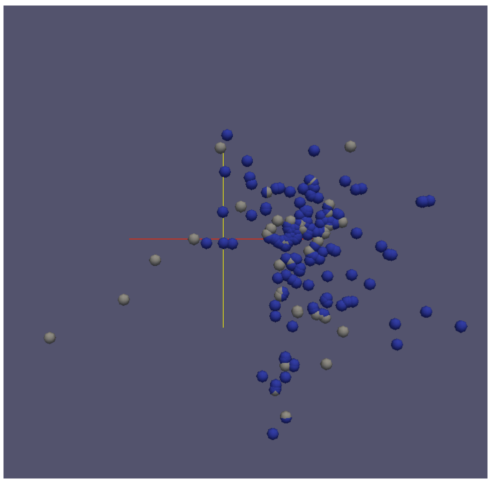
Figure 6 shows a simulation in Paraview for a typical run. Only four kinds of agents are shown (Blocker: B-B, Blocker: B-H; Hider: H-H; Hider: H-B). Blue of both shades are the Hiders; yellow represents Blockers. In this run, with a high fraction of Hiders, Blockers can be seen converging from their initial state (bottom) toward the center of the simulation, then as time moves forward (up along the central axis), remaining in close proximity to the center. Hiders, conversely, spread outward.

It is clear, however, that as scale increases, the dynamics remain largely the same: the x axis within each graph represents the total number of agents in the simulation, with values at 2^18, 2^21, 2^24, and 2^27. The lines are essentially flat: scale has only small effects, and these were at the lower ends of the range explored (these were not explored more fully).

The key to understanding this may be network analysis: the idea of independent cycles is a reasonable one, but the actual network generation algorithm used (that is, random assignment of A and B across the entire population) tends to lead the simulation to be a single cycle of mutually interacting agents plus their trailers. Future inspection of this model might involve alternative methods for generating more constrained and structured networks.

6 Concluding Remarks

Although this study did not show any interesting shift in simulation dynamics as scale was increased, other simulations with different structures and dynamics can, and many will be found that will. A key point for future work is to be able to identify which kinds of simulation will vary with increased scale; network analysis will play a key role in this, both practically and theoretically, and the other practical issues discussed here will also be pushed ahead as they serve their expanded purposes. As a hint to the direction that HPC social science can go, consider only this: the maximum simulation size shown in this demonstration is not the maximum possible on the BG/P: only 20% of the Blue Gene/P's capability was used. Next year Argonne plans to install another machine, the Blue Gene/Q, that will exceed the Blue Gene/P's capacity by a large factor in terms of both number of processors, total memory and speed.



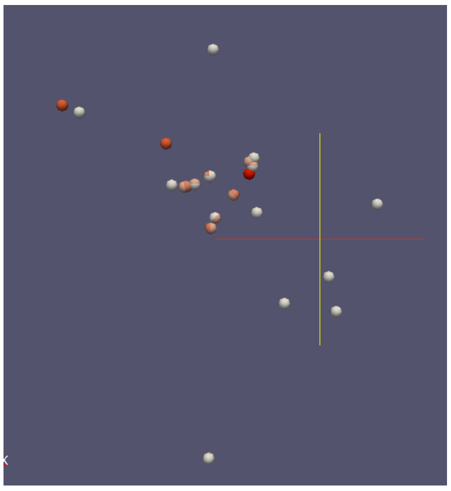
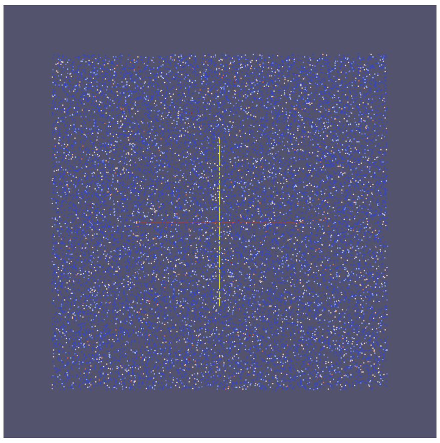


Figure 1. (L) Small simulation showing network analysis; 'trailer' agents are shown in gray.

Figure 2. (R) Filtered view with distances calculated upstream from the red agent near the center of the picture. This agent is following the two bright orange agents on the left. These agents, in turn, are following the other agents shown, with network distance from the original agent indicated by diminishing color, fading to white. Maximum network distance shown is 4.



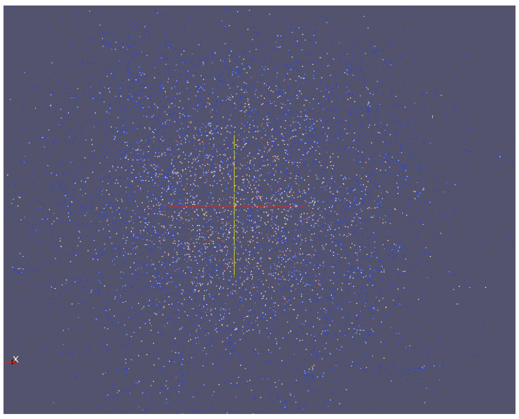


Figure 3 (L). Top-down view in Paraview of simulation in initial state. A high fraction of Hiders exist, and are depicted in blues.

Figure 4. (R) Final state of the same simulation shown in Figure 3.

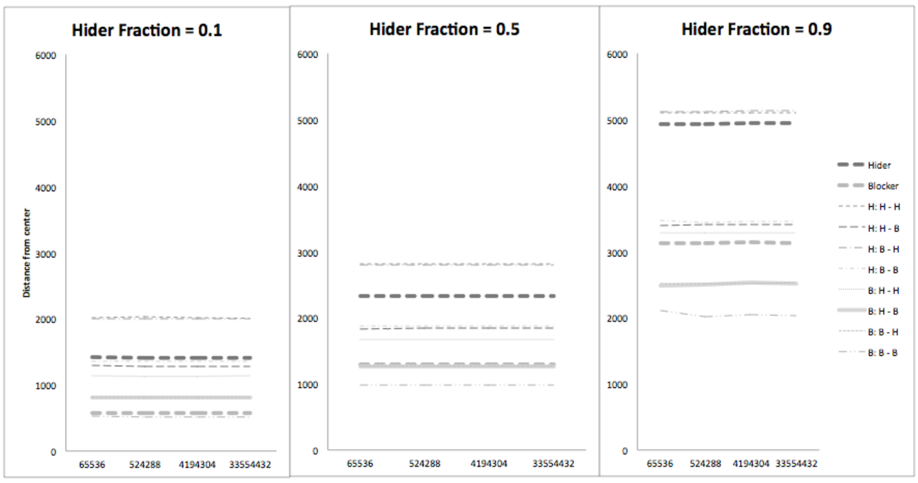


Figure 5. Graphs showing average final distances of agents from center of all agents for Hider Fraction = 0.1, 0.5, and 0.9. Horizontal axes show number of agents.

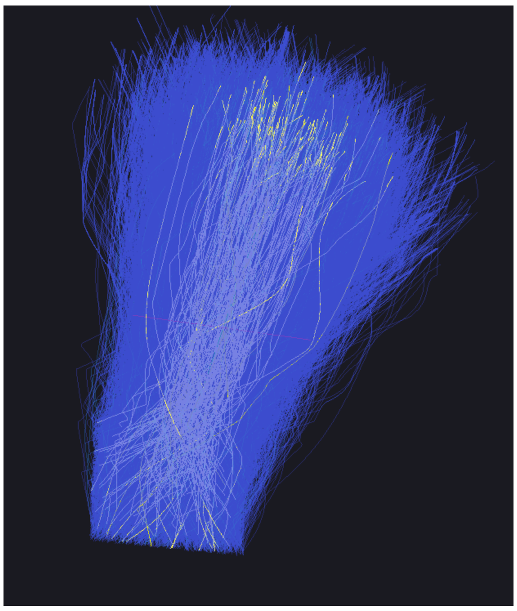


Figure 6. Filtered 3-D view showing the progress of a simulation from start to finish. Only Hiders: H-H, Hiders H-B, Blockers: B-H and Blockers: B-B are shown.

Acknowledgements. This work was supported by American Recovery and Reinvestment Act (ARRA) funding through the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357 and used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. Thanks are given to Michael North, Nick Collier, and Timothy J. Williams. All errors are my own.

7 References

1. Booth Sweeney, L., Meadows, D.: The Systems Thinking Playbook. Chelsea Green Publishing, White River Junction, Vermont (2010)
2. Repast Development Team: http://repast.sourceforge.net/repast_hpc.html
3. Repast Development Team: http://repast.sourceforge.net/index.html
4. Argonne Leadership Computing Facility: https://wiki.alcf.anl.gov/index.php/Quick_Reference_Guide
5. Wasserman, S., Faust K.: Social Network Analysis: Methods and Applications Cambridge University Press, Cambridge (2009)
6. The HDF Group: Hierarchical Data Format Version 5, 2000-2010; see http://www.hdfgroup.org/HDF5/
7. Paraview: http://www.paraview.org/
8. XDMF: http://www.xdmf.org/
9. Foster, I.: Designing and Building Parallel Programs, Addison-Wesley, Reading, Massachussetts (1995)