SAND2008-1306C

Parallel Particle Data Model

Zachary J. Heath, Max S. Shneider, Jaideep Ray, Benjamin A. Allan,
Keith B. Vanderveen, Michael E. Goldsby and Robert C. Armstrong
{*zheath, msshnei, jairay, baallan, kbvande, megolds, rob*}*@sandia.gov*
Sandia National Laboratories
7011 East Avenue, MS 9915, Livermore CA, 94550-0969

Abstract

We present the design and performance of a parallel entity simulation framework called the Parallel Particle Data Model (PPDM). Based loosely on a Particle-In-Cell algorithm, the PPDM orchestrates and supports Discrete Event and Agent Based Simulations on a parallel high-performance platform. The PPDM is targeted at social simulation applications and is designed to be portable to a variety of high-performance platforms. On cluster platforms the PPDM performs very well for Discrete Event calculations and moderately well for Agent Based simulations. We hope that this work will form the cornerstone of a reusable toolkit for modeling and simulation.

1 Introduction

Enterprise modeling, social simulation, system-of-systems war gaming, and related techniques simulate complex environments and systems (hereafter referred to as Human Dynamical Systems (HDS)) and compute observables which manifest themselves primarily as emergent phenomena. To increase fidelity of these HDS systems, increasingly large computations are needed and are accessible only through high-performance computing (HPC). Computer-based simulation of HDS has long been used within the military for war games, to carry out training, assess new technologies, and evaluate tactics. Recently, the Department of Homeland Security (DHS) and other federal agencies charged with disaster preparedness and response have embraced simulation for modeling a variety of complex phenomena including response to attacks by Weapons of Mass Destruction (WMD) and natural disasters. Sandia National Laboratories has developed a number of simulation applications to support different programs examining WMD countermeasures and concepts of operation, defense applications, terrorist networks, and economic consequences from natural disasters or terrorism affecting critical infrastructures. These simulation applications differ in several respects, yet they share an underlying commonality: all concern themselves with modeling complex "systems of systems" where the system components are discrete (such as people, autonomous land vehicles, or companies) and interact in highly complex ways with other system components. Historically, modeling of HDS has not taken advantage of parallel processing hardware or techniques. There are at least three reasons for this:

1. HDS generally exhibits poor data locality and is thus less amenable to parallelization.
2. The phenomena modeled were poorly understood, requiring many assumptions to be made, so that making additional assumptions to speed up execution time was deemed acceptable. This obviated the need for parallel processing to speed up execution times.
3. The history of simulation of HDS has favored rapid development, use of widely available computing hardware and software, and incorporation of sophisticated graphical user interfaces (GUIs) over speed of execution. This emphasis on rapid, slick-looking deliverables portable to a wide variety of platforms led to the use of high-level prototyping and scripting languages such as Java, Python, and Tcl over C and C++, because most programmers are much more productive in these languages. Most programming for high performance parallel processor systems has taken place

in languages like C, C++, and (historically) FORTRAN because they give the programmer fine-grained control over allocation of memory and other processor resources.

Increasingly, however, HDS must reproduce significant phenomena/effects at credible fidelity while being fast enough to enable human-in-the-loop and timely batch-oriented analysis. Recently developed requirements for the next generation of social simulation applications have shown that the current capability, based on conventional computing architectures, falls short in both respects, requiring extension to HPC platforms using parallel processing. We have developed components and tools for HDS simulation methods in a parallel HPC setting. In particular, we are addressing the challenges outlined above by creating a portable, scalable, and general engine for particle simulations as a facility for multi-use in agent-based simulations.

2 Related Work

Many simulations have been created to study complex Human Dynamical Systems. Two excellent examples from Virginia Tech's Network Dynamics and Simulation Science Laboratory include TRANSIMS [6] which uses urban travel data to model transportation networks and EpiSims [7] which uses that same travel data to model disease epidemics. Another example of work in this arena is Generative Social Science, which uses simulations to help discover the underlying dynamics of complex social systems [11].

As the modeling of HDS has grown in popularity, a number of reusable Agent Based Modeling (ABM) toolkits have emerged to simplify the work of the developer. Repast [10] is arguably the most popular framework in the ABM community and provides developers with libraries for agent creation, event scheduling, and data charting and visualization. Mason [16] provides similar functionality but focuses more on light-weight models meant to be run many times for Monte Carlo-type studies. Neither framework lends itself for use developing models that run across a cluster of computers, thus limiting the size and complexity of such models. Other popular ABM toolkits include AnyLogic [24] and Ascape [18].

Most ABM toolkits employ Discrete Event Simulation (DES) at their core. Instead of advancing time in regular steps, DES assigns an exact time to each event [20]. DES can achieve performance gains when events are irregularly spaced in time or when different parts of the simulated system operate at different time scales. Recent research has shown DES to bring greater accuracy and numerical stability to some numerical problems, including some particle-in-cell (PIC) simulations [15]. The idea of parallel DES first appeared in the late 1970's [17], and parallel implementations have been plentiful since the early 1990's. Parallel DES has scored its greatest successes in military war gaming, the simulation of transportation systems, digital logical circuits and telecommunications systems [13]. Practical DES simulations have achieved parallelism of up to 1500 nodes [12]. Parallel DES engines have been shown to successfully provide time management for agent based simulations such as in the SASSY system [14].

A more traditional PIC-based, time stepped model has also been used for simulating HDS such as the evacuation of large crowds [22]. In this work, the Social Forces algorithm was implemented such that the simulated geometry and populations were divided up amongst a cluster of computers. A similar implementation was done in [23] but using the Play Station 3 as the parallel processing environment.

3 Architecture

In our work modeling various HDS, we identified a great deal of functional commonality in our simulation codes. We also observed that future simulations in the problem domain would be growing in population size and complexity. This led us to create the Parallel Particle Data Model (PPDM), a Java-based, reusable programming toolkit that supports HDS simulations and can scale to a large number of processors.

The HDS simulated by Sandia are generally composed of large populations of agents (people, vehicles, etc.) spread across a geography of segmented locations such as cities, census tracts, or even households and businesses. Each location contains a set of agents that are able to interact with each other as well as with agents in neighboring locations. The agents can move between locations as time evolves. Various modules of the simulation need to update or gather information from the agent populations in different ways based on the phenomenon they model. Examples include infection of people due to a biological hazard plume or detection that a car should trigger a radiation sensor alarm.

We designed the PPDM to accommodate this method of interacting with the simulated populations while operating in a distributed environment. The PPDM uses a particle-in-cell (PIC) data structure which divides the simulated geography into cells that contain populations of agents. Our PPDM then distributes these cells across a number of processors. Agents

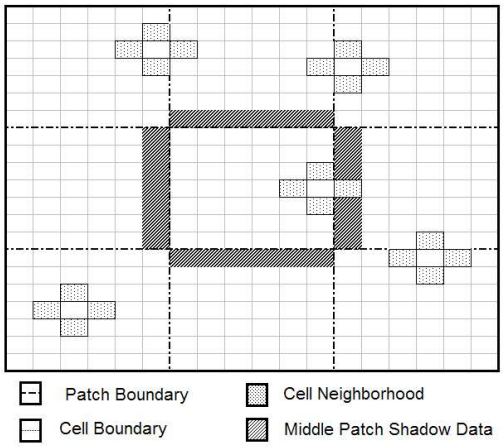

Figure 1. Domain Model

directly affect other agents residing in the same or neighboring cells. Non-neighboring particles can interact through special mechanisms.

3.1 Domain Geometry

To make use of the PPDM, the user must first define the geometry of the problem domain. Currently, we have implemented two types of geometry. A user can specify either an n-d lattice or a graph of cell locations. Along with the geometrical structure, the user must define the number of functional patches into which they would like the geometry broken. A patch represents a grouping of cell locations that are guaranteed to be collocated on the same processor. Each processor is allocated a number of patches to process. Modelers write their algorithms and agent update code to work on a patch-by-patch basis which enables the PPDM to distribute the user's program across many processors.

3.2 Data and Particle Fields

After specifying the domain geometry, modelers define multiple data and particle fields across this geometry. A data field holds a single unit of data (such as an int or float) for each cell in the domain, while a particle field allocates to each cell in the domain a list-like data structure called a particle set, within which the modeler can store and organize agents. In creating a data or particle field, the modeler specifies a locality stencil for that field. This defines each cell's local data neighborhood within an overall n-dimensional lattice geometry. In the case of 2-d lattices, a modeler will commonly specify a 4 point stencil in which a cell's neighbors include adjacent cells to the north, east, south, and west. In a graph geometry, on the other hand, the graph structure determines cell neighborhoods directly. The local neighborhood defines a region of shadow data that each patch must make available to the cells it maintains. After doing a shadow update for a particle field, each patch will have the data for the cells it contains as well as a read-only copy of data from its neighboring cells. This enables the simulation to update agents' state using data from their own cell as well as neighboring cells. This method is similar to that used in a parallel implementation of the social forces model [22]. While processing the agents, modelers also have the ability to schedule the movement of agents from cell to cell. This cell movement can span patches and processors. The PPDM takes care of the actual movement of the agents and transfer of shadow data. If the modeler guarantees that particle movement only occurs between neighboring cells, the PPDM can make optimizations that reduce the amount of communications needed between processors, thereby allowing greater speedup. The PPDM's capabilities hide a lot of parallel processing complexity from the modeler.

3.3 Data and Agent Access

To better enable parallel processing, the PPDM limits the modeler's interaction to the data and particle fields by allowing only cell-by-cell or patch-by-patch access to the data. A modeler's algorithms have full access to all agents owned by the

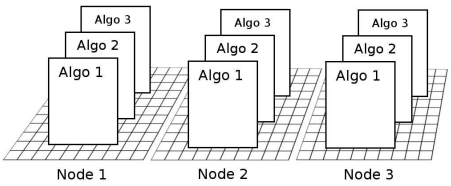

Figure 2. Visitor Algorithms

patch they process, as well as read access to agents of neighboring cells of that patch. The PPDM's feature set encourages modelers to build their algorithms with locality in mind. This allows the algorithms and data to be more easily and efficiently distributed across multiple processors. From our experience in prior simulations, this restriction is acceptable. We also offer various convenient yet less efficient mechanisms, described later in the paper, to handle global communication between processors.

We provide modelers several patterns for interacting with the particle populations. The most common method of particle access is to submit particle algorithms to the PPDM. Particle algorithms are data structures that implement the visitor pattern. When particle algorithms are submitted to the PPDM, the PPDM duplicates them across all processors and schedules them to be executed at regular intervals. The algorithms visit each cell or patch owned by the processors on which they are located and perform their needed operations on the contained agents. The particle algorithms can optionally coordinate with their copies on other processors and send back aggregated data to their parent modules or other subscribing particle algorithms.

In our simulations, various modules work together in a decoupled manner by submitting particle algorithms that enforce their desired behavior or area of concern on the agents. This deviates from standard agent-based techniques, which initialize the agents with a given behavior and state and allow the agents to evolve their state. This difference aries because the PPDM grew out of our experience with prior simulations composed of separate modules that ran on their own Java virtual machine. Each of these modules needed to interact with the population. Rather than moving the population data to each of the modules, we allowed the modules to move their code to the population data. This is similar to how distributed systems interact with a common database through sql or stored procedure calls. One can duplicate the more traditional agent update behavior in the PPDM through the use of a simple particle algorithm that calls an update method on all or a selected number of the agents at scheduled intervals. The agents themselves would contain all of the logic needed to interact with neighboring data and agents and perform all necessary updates and movement. Another alternative would be to have the various modules inject differing behavior strategy objects into the agents upon initialization which will eventually be called during updates. Dynamic languages like Ruby would provide an ideal environment for this technique. We are also looking into incorporating patch-based event queues into the PPDM to allow for algorithms or agents themselves to make use of the DES programming model. To date, our simulation needs have not warranted the added synchronization and bookkeeping issues that this will require.

3.4 Communication implementation

The PPDM uses the mpi model for parallel data communication. Communication occurs between copies of particle algorithms running on differing patches, when agents move between cells owned by different processors, and to update shadow data located on neighboring processors.

When necessary, the particle algorithms communicate and coordinate with each other across patches using standard parallel processing messaging such as the mpi broadcast and reduction methods. We have provided modelers with simplified interfaces to these functions. We have also implemented a messaging service to allow agents to communicate with other agents that are not located on their patch. The PPDM organizes and sends messages using a regularly occurring batch update process. The messaging service is optional and when in use it requires additional overhead because it must track agent locations to enable routing.

The PPDM handles particle movement during a regularly schedule batch update where all processors work together to redistribute agents to the correct locations. We handle copying of shadow cell data for neighboring cells in a similar manner. This requires the various processors to remain synchronized in time.

Modelers can configure the PPDM to make use of mpiJava [4] or MPJ Express [5] for the mpi services. Both of these are object-oriented implementations of the mpi protocol using the Java programming language. They differ in that mpiJava

provides a JNI wrapping over a native mpi environment (such as MPICH of LAM), while MPJ Express is a complete implementation of the MPI protocol using only Java. In Java, serialization of object data to byte data causes a major cpu cost. We hope that custom serialization code will reduce some of this cost and are looking into the serialization and communication libraries from the Ibis project [8].

4 Example Simulations

In this section we demonstrate the use of PPDM in two different examples and conclude with an illustration of its parallel capabilities. The problems have been formulated in an agent-based fashion, but carry a PIC flavor in the sense that agents/particles are collated into "bags"/cell with in-cell (in-bag/near-field) behavior modeled at a far greater fidelity than the far-field (out-of-bag) behavior.

4.1 Spread of disease in a 2-species population

This example approximates the spread of a disease in two separate species, humans and (non-human) primates, with primates acting as a reservoir for the pathogen. The pathogen, modeled loosely on Ebola, is communicable within each of the species and is also capable of primate-to-human jumps. This example models both humans and primates as individual agents. A number of primates and humans are collocated into "bags"/cells, representing settlements where humans may come in contact with primates and contract the disease. Both humans and primates may move between cells, though primates' movements are significantly more circumscribed than humans. The epidemic model is best understood in terms of (1) the disease progression in the human and primate agents, and (2) the geographical movement of the agents.

4.1.1 Disease model

Ebola is a Filovirus, a virus family which has not been well studied. It causes a hemorrhagic fever in humans and primates, and its case-fatality rate varies between 70%-100%. A review of Ebola and its subtypes can be found in [21].

Ebola can spread in a human population via contaminated bodily fluids. Common ingress points are eyes and minor cuts and bruises. Contaminated needles (common in backward regions with less than satisfactory medical infrastructure) have been implicated in almost all large Ebola outbreaks. Contaminated bodily fluids, from cadavers, are another source of infection, one that can be significantly reduced if they are disposed of expeditiously. Given these mechanisms for the spread of the disease, studies have attempted to fit an SEIR model [3] to it [9]; the R_0 is 1.83 and the average infectious/contagious period is 5.6 days

In this work we use a compartmental disease propagation model. Both the humans and the primates go through five compartments/states of health - Susceptible(S), Exposed (E), Mildly Symptomatic (MS), Severely Symptomatic (SS), and Recovered/Dead (R/D). The residence time in each of these states of health is a random variable, which follows a distribution (usually log-normal); the parameters of the distributions are not very well known. The process of infection of humans is governed by their contact with other humans, cadavers and primates, i.e. it depends on their closeness of contact and the probability of meeting an infected individual, primate or cadaver. Transmission within the primate population is entirely by contact. We will assume that both the Mildly Symptomatic and Severely Symptomatic stages are contagious, for both humans and primates.

4.1.2 Movement model

We assume that there exist M human settlements of size $N_i, i = 1 \ldots M$. Human settlement i also contains n_i primates. Interactions between human and primates occur between those present in the same settlement. The human and primate populations in S_i are drawn from the log-normal distribution $\mathcal{L}(N_{median}, S)$, where N_{median} is the median and S the standard deviation.

$$\begin{aligned} N_i &\sim \mathcal{L}(N_{median}, S), N_{median} = 10^3, S = 500 \\ n_i &\sim \mathcal{L}(n_{median}, s), n_{median} = 50, s = 12.5 \end{aligned} \tag{1}$$

For each settlement we define a mobility factor Mh and Mm for the human and monkey populations. This mobility factor determines the percentage of each population that will relocate on a given movement step. The indexing of settlements reflect

geographical proximity. Primates travel to other human settlements primarily because of proximity. Once the connection routes are determined, a settlement's mobile population is distributed across its connecting settlements using the above probabilities as a weighting factor.

The model can be configured with two modes of population movement. In the first, agents are assigned a fixed day-night routine. In the second mode, agents move randomly several times a day to connected settlements using the above probabilities and weights, and at night they return to their home settlement. Mildly symptomatic individuals move, but severely symptomatic ones do not.

4.1.3 Disease Model Scaling

The disease model initializes the PPDM with a 2-d lattice geometry. The model represents each settlement by a cell in the lattice where a cell is given an x and y location to determine its distance from other cells. We initializde two separate particle fields for the human and monkey populations. We submit four configurable particle algorithms to the PPDM to evolve the disease agents. These algorithms create the initial population distributions, initialize the agent movement patterns, perform agent movement at a set rate, and evolve the disease state of the agents. The algorithms are modular and can be replaced with different implementations of varying complexity to study how changes to the population or disease dynamics effect the outcome of the epidemic. For this model, two types of inter processor communication take place. During agent movement some agents will need to relocate to neighboring processors, and after each disease update the master disease algorithm coordinates with its neighbor algorithms to output the aggregated disease statistics for that time.

We performed experiments testing both the strong and weak scalability of this model on the PPDM. In the strong scaling test, we initialized the population with a fixed 2,000 settlements, each containing an average population size of 1,000 for a total population of 2 million agents. We then varied the number of processors allocated to the PPDM and ran the simulation for a simulated 2 months. Figure 3 shows the results of the strong scaling test. The results show near-ideal scaling up to about 8 processors, after which the particular problem setup begins to scale poorly. This poor scaling results because we keep the problem size fixed. As we increase the number of processors, each processor has less to do and communication and synchronizations costs take over.

In the weak scaling test, we initialized the population with a fixed 2,000 settlements. We then scaled the population size up linearly with the number of processors allocated to the PPDM so that each processor would maintain a fixed number of agents. We ran the model for a simulated 2 months. Under the weak scaling case where the problem size grows with the number of allocated processors, we achieve fairly good scaling with the 32 node case within twenty percent of the ideal result. We see this good scaling because the communication load for each processor is balanced by the increasing computational load. These results show that we can scale up our modeled size by increasing the available number of processors.

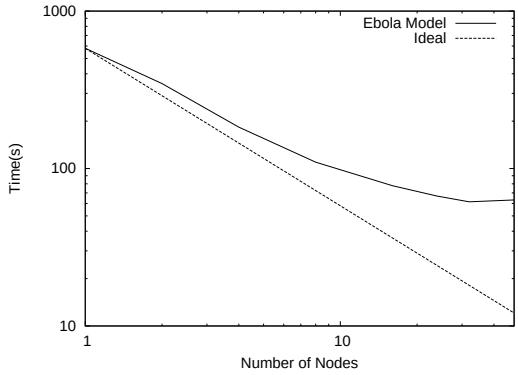

Figure 3. Disease Strong Scaling

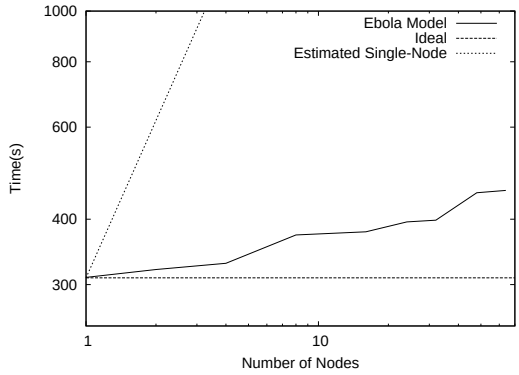

Figure 4. Disease Weak Scaling

4.2 Seldon

Seldon is a software agent-based modeling toolkit that combines technology and concepts from a variety of different research areas, including psychology, social science, and agent-based modeling and simulation. It has been used to study

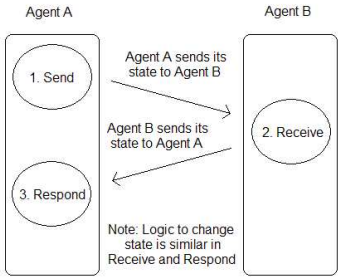

Figure 5. Seldon Interactions

urban gang recruitment and terrorist network recruitment. This second example of the use of PPDM demonstrates the utility of this methodology in analyzing effect of media on populations.

Previous uses of Seldon involved rather few ($O(10^3)$) agents, and serial implementations generally sufficed. However, media affects entire populations or societies, so adequate models required large agent populations which necessitated a recourse to parallel computing. Further, the individual agent models have undergone an enhancement to include a cognitive model (essentially a semantic graph of concept activations and edge weights), allowing a realistic processing of media information. This led to an increase in the computational intensity of individual agents, further spurring the need for parallelization.

Seldon has two types of agents: individuals and abstract. Abstract agents represent social or institutional concepts that can influence an individual in some way (e.g. schools and mosques). Abstract agents have a set of individual agent members, and can thus be highly-connected nodes in the overall structure.

Interaction of two agents involves significant processing. Agents have a customizable set of attributes that can vary according to exchangeable rule sets (i.e. linear attraction, linear reinforcement). During an interaction, concept activation vectors are exchanged and interpreted by the semantic graph (nodes fire), leading to changes in their cognitive states.

The simulation commences with an unconnected set of agents. Homophily is used as the basis of attraction, and as interactions proceed, relationships are formed. Each agent has a maximum amount of relationship energy that it can expend, thus providing a cap. This provides the flexibility to either have a large number of weak relationships or a small number of strong relationships. Agents also have personality factors, which affect their interactions. For instance, an extroverted agent might have more interactions than an introverted agent. As relationships evolve, they form social networks, which range from acquaintances to cliques. These, in turn, drive subsequent interactions, so that an agent will more likely interact with a close friend (i.e. in its clique) than a mere acquaintance.

We now detail the parallelization process. First, we replaced all direct agent references by agent IDs, and created a central mechanism for dereferencing the agents (where required). This also presented an opportunity to reduce the total number of dereferences. Secondly, we modified the interaction method (in fact any method that was directly called by another agent), as illustrated in Fig 5. This allowed agents to interact even if they were located on different processors. The agents first compare their attributes and cognitive information prior to deciding whether to interact. Interactions are transactional, resulting in a modification of the agents emotional states if they proceed. The interaction procedure consists of three parts, which are referred to as send, receive, and respond. Agent A sends its information to Agent B, who then receives the information, compares it to its own, and responds to Agent A. If the agents decide to interact, A and B change their state in the receive and respond methods, respectively.

In order to invoke the receive and respond methods remotely, we defined a new particle set for messages. We replaced each direct call with the creation of a new message particle and its subsequent addition to a queue (which then delivers a large number of messages concurrently). Messages consist of the caller's ID, attributes, top cognitive activations (since cognitive models can contain numerous concepts), and the callee's ID and method name. Each timestep in the algorithm consists of a number of loops. In the current simulation the individual agents first determine their membership with abstract agents in the first loop. Then, in the second loop, abstract agents determine their interactions with individual agents. Finally, in the third loop, the individual agents identify other individual agents with which to interact. Each loop results in the creation of message streams, which the simulation collectively processes at the end of the loop. This involves delivering the message to the callee's processor and then invoking the callee's intended method. Since receive messages usually create their own response messages, message processing continues until the queue empties. This barrier synchronization ensures that each loop finishes completely before the next loop starts.

The third part of the parallelization involved decomposition of the problem across processors in a load-balanced manner. This involved assigning processors to agents in a manner that maximized the likelihood of intraprocessor communication. We used Zoltan [2], a load-balancing library, to invoke graph-partioning algorithms in ParMETIS [1]. Zoltan used the network structure (using relationship strengths as edge weights) to calculate the optimal agent-to-processor mapping. Zoltan repartitions the agents at the end of every timestep. Repartioning, using Zoltan, exploits the current decomposition to reduce data migration. Zoltan also provides a distributed directory capability to track the processor assignments of off-processor agents. The actual data migration (rather than the processor assignments) are performed separately and involve packing and unpacking agents at the source and target processors. This requires synchronization of (each agent's) state data. The actual data migration is performed in a manner analogous to the delivery of messages.

An additional difficulty involved multi-language integration. Zoltan is implemented in C/C++, while Seldon is implemented in Java. We created a JNI wrapper around Zoltan using Swig, which then provided access to Zoltan from Seldon. Further, we had to rationalize the MPI implementation, i.e. link both Zoltan and Seldon against the same MPI library. The need for rationalization required significant time to identify and rectify.

5 Integration

In creating the PPDM we integrated with several existing simulation frameworks on an experimental basis. Our first integration involved connecting a moving population simulation using the PPDM to an existing High Level Architecture (HLA) federation [13]. The population simulation ran in distributed mode across a number of machines in a compute cluster with a population size in the millions, representing a large metropolitan area. The HLA federation ran on hardware outside of the cluster. We separated the simulations this way due to differing platform requirements. To link the two simulations, we created an adaptor federate that participated in the HLA federation outside of the cluster. This adapter federate forwarded relevant object updates and interactions to and from the PPDM on the head node of the cluster using a Java remote method invocation (RMI) package we wrote. The two systems exchanged a small amount of data relative to the size and complexity of the cluster, and this data consisted mainly of the particle algorithms previously mentioned as well as aggregated output statistics from the population. By using this bridged approach, we integrated the use of a compute cluster into an existing HLA simulation.

Our second integration attempt involved using the PPDM in the RePast Agent Based Modeling framework [10]. A similar approach of adding parallelism to RePast can be found here [19]. We made use of the PPDM as the underlying data structure for one of the RePast demo application models. We did this using a Sequential Program Multiple Data (SPMD) approach where we executed the same RePast model concurrently on all nodes of the cluster. Each instance of the model initialized the PPDM code but maintained a different section (several patches) of the overall domain based on its processor id. Each node made its agent initializations and updates accordingly. We had to make some changes and enforce some limitations on RePasts `Schedule` class to maintain synchronization and to allow for agent movement and shadow agent updates between processors. We scheduled the particle movement and update at fixed intervals on all nodes. RePast enjoys much popularity in the ABM community, so a more formal integration of RePast with the PPDM could be useful.

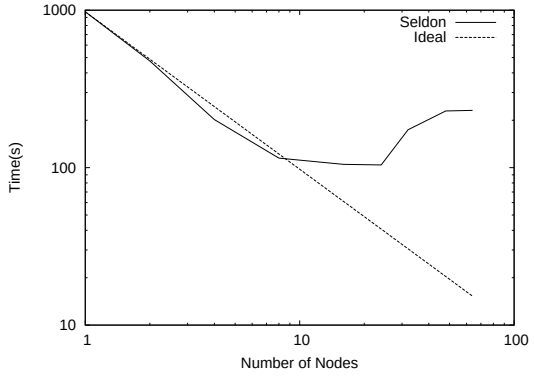

Figure 6. Seldon Strong Scaling

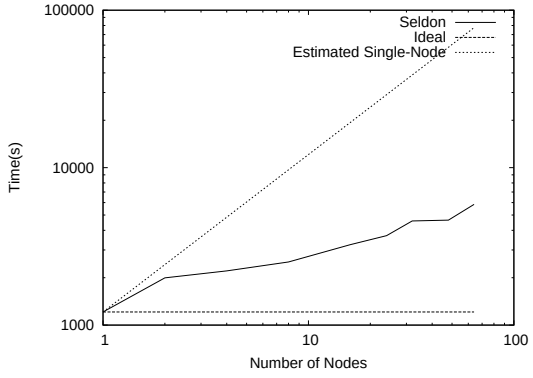

Figure 7. Seldon Weak Scaling

6 Conclusion

DES/AB modeling is arguably the standard tool for understanding and making predictions about complex systems. To some degree both DES and AB modeling make the argument that, unlike statistical methods, simulations must be performed at scale. For such social systems, there is no *a priori* idea of a scaling law or average that would predict their emergent behavior. Because the phenomena simulated are usually large and complex, and because they must be computed at scale, parallel high performance computing is required to enable successful simulations of social systems.

We hope that our work in developing the Parallel Particle Data Model (PPDM) instigates the development of a library of general purpose components for large-scale entity modeling. From a software engineering point of view, separable components for DES/AB modeling will aid repeatability and methodical experimentation. Beyond reusable software to speed software development, the ability to change out entity models and even the framework and schemes for updating without changing the parallelization scheme is an important contribution of the PPDM. Often when comparing two different models that purport to acheive the same result, the phenomenological particulars of the solutions differ in so many ways that comparisons are difficult. Separating out the parallel implementation as we have done in the PPDM, hopefully will make comparison between different models easier in the future.

7. Acknowledgments

This work has been supported in part by the U. S. Dept. of Energy's Scientific Discovery through Advanced Computing initiative, through the Center for Component Technology for Terascale Simulation Software, of which ORNL and SNL are members.

References

[1] Parmetis homepage. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview.

[2] Zoltan homepage. http://www.cs.sandia.gov/Zoltan/.

[3] R. M. Anderson, R. M. May, and B. Anderson. *Infectious Diseases of Humans : Dynamics and Control*. Oxford University Press, New York, 1992.

[4] Mark Baker and Bryan Carpenter. mpijava. http://www.hpjava.org/mpiJava.html.

[5] Mark Baker and Bryan Carpenter. Mpj express project. http://acet.rdg.ac.uk/projects/mpj.

[6] C. Barrett, K. Bisset, R. Jacob, G. Konjevod, , and M.V. Marathe. Transims: Transportation analysis simulation systems. http://ndssl.vbi.vt.edu/transims.html.

[7] C. Barrett, S. Eubank, V.S. Anil Kumar, and M. Marathe. The epidemiological simulation system (episims). http://ndssl.vbi.vt.edu/episims.html.

[8] M. Bornemann, R. V. van Nieuwpoort, and T. Kielmann. Ibis: Efficient java-based grid computing. http://projects.gforge.cs.vu.nl/ibis/.

[9] G. Chowell, N. W. Hengartner, C. Castillo-Chavez, P. W. Fenimore, and J. M. Hyman. The basic reproductive number of Ebola and the effects of public health measures: The case of Congo and Uganda. Technical Report LA-UR-03-8189, Los Alamos National Laboratory, Los Alamos, NM, 2003.

[10] Nick Collier. Repast agent simulation toolkit. http://repast.sourceforge.net/index.html.

[11] Joshua M. Epstein. *Generative Social Science*. Princeton University Press, New Jersey, 2007.

[12] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. Large-scale network simulation: How big? How fast? In M. Calzarossa and E. Gelenbe, editors, *11th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2003)*, volume 2965 of *Lecture Notes in Computer Science*, pages 116–123. IEEE Computer Society, 2006.

[13] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley and Sons, New York, 2000.

[14] M. Hybinette, E. Kraemer, X. Yinand G. Matthews, and J. Ahmed. Sassy: A design for a scalable agent-based simulation system using a distributed discrete event infrastructure. In B. Lawson, F. Perrone, J. Liu, and F. Wieland, editors, *Proceedings of the 2006 Winter Simulation Conference*, pages 926–933. IEEE Computer Society, 2006.

[15] H. Karimabadi, J. Driscoll, V. A. Omelchenko, and N. Omidi. A new asynchronous methodology for modeling of physical systems: Breaking the curse of the Courant condition. *J. Comp. Phys.*, 205:755–775, 2005.

[16] Sean Luke, Gabriel Catalin Balan, and Liviu Panait. Mason: Multi-agent simulator of neighborhoods. http://cs.gmu.edu/ eclab/projects/mason/.

[17] J. Misra. Distributed discrete-event simulation. *Comp. Surv.*, 18(1):39–65, 1986.

[18] Miles Parker. Ascape. http://ascape.sourceforge.net/.

[19] Hazel R. Parry, Andrew J. Evans, and Alison J. Heppenstall. Millions of agents: Parallel simulations with the repast agent-based toolkit. In *International Symposium on Agent Based Modeling and Simulation*, 2006.

[20] K. Perumalla. *Model execution*. Chapman and Hall/CRC, 2007.

[21] C. J. Peters and J. W. LeDuc. An introduction to Ebola: The virus and the disease. *J. Inf. Dis.*, 179(Suppl 1):ix–xvi, 1999.

[22] M. J. Quinn, R. A. Metoyer, and K. Hunter-Zaworski. Parallel implementation of the social forces model. *Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics*, 2003.

[23] C. Reynolds. Big fast crowds on ps3. In *Sandbox (an ACM Video Games Symposium), Boston, Massachusetts*, 2006.

[24] XJ Technologies. Anylogic multi-method simulation software. http://www.xjtek.com/anylogic/.