On the Feasibility of Running Entity-Level Simulations on Grid Platforms

Alan Su[*] Fran Berman[†‡] Henri Casanova[†‡]

[*] Laboratoire de l'Informatique du Parallélisme
École Normale Supérieure de Lyon, Lyon, France
alan.su@ens-lyon.fr

[†] San Diego Supercomputer Center [‡] Dept. of Computer Science and Engineering
University of California, San Diego, U.S.A.
{berman,casanova}@sdsc.edu

Abstract

Scientists have long relied on abstract models to study phenomena that are too complex for direct observation and experimentation. As new scientific modeling methodologies emerge, new computing technologies must be developed. In this paper, we focus on entity-level modeling, *a modeling approach that is gaining prevalence in many scientific fields. Although the principles of entity-level modeling are straightforward, entity-level simulations require a large amount of compute resource and grid platforms can meet such resource needs. Unfortunately, efficient large-scale distributed entity-level simulations have proven elusive, typically due to non-determinism that renders classical distributed application deployment strategies ineffective. In this work, we propose a method for dynamically remapping application tasks to cope with this inherent non-determinism. We evaluate the efficacy of this method in a simulated grid computing environment and discuss the feasibility of executing entity-level applications on grids.*

1 Introduction

High-performance computing technology long played an critical role in scientific research as scientists often turn to *modeling* as a means to indirectly describe and study otherwise intractable problems. One popular approach is *system-level modeling*, which relies on abstraction: abstraction: significant aggregate properties of the system under study are identified, and the fundamental behavior of the system are manifested as mathematical relationships between these properties. For example, system-level vehicular traffic models characterize flows of traffic as fluid flows, using properties such as density (i.e., how tightly vehicles are packed) and velocity (i.e., average speed of traffic) and linking them via partial differential equations (PDEs) [8]; such models have proven highly valuable in many fields (e.g., [8,19,31]).

However, a number of emerging scientific models are increasingly focused on the importance of understanding emergent behaviors – the mechanisms by which individual actors in a system interact to give rise to observed system-level behavior. Consequently, efforts to explicitly describe complex systems using relatively simple descriptions of entities which comprise the systems being studied are growing progressively more prevalent (e.g., [1,3,5,15,27]). We term this approach *entity-level modeling*. For instance, an entity-level vehicular traffic model could consider cars as individuals with possibly complex and diverse behaviors that may be vehicle- and driver-specific. Such a modeling approach has been cited in a number of fields in which system-level models are undesirable or insufficient (e.g., [7, 9, 12]).

Entity-level simulations typically have much higher computational requirements than their system-level counterparts. This is because each individual entity is modeled separately, and thus requires memory (e.g., to store a potentially large data structure) and computation (e.g., to treat large numbers of entities running potentially complex algorithms). Therefore, running on grid platforms that aggregate large amounts of resources is promising. Unfortunately a critical factor that limits the utility of entity-level modeling is the fact that the issues regarding computer-based entity-level simulations are not well-understood. The main difficulties are model structure irregularity due to the heterogeneity of entities in the model, and the fact that entity behaviors can be arbitrary, non-deterministic, and complex. The heterogeneous and non-uniform structure of grids themselves exacerbate this complexity.

In this paper, continuing our study of entity-level applications in distributed environments [25, 26], we focus on the problem of application non-determinism on grid platforms: the behavior of an entity-level application (and correspondingly, its computational resource requirements) typically changes during run-time. Based on our prior parallel task mapping work, we propose a *task remapping strategy*. We use an entity-level model from ecology to study the efficacy of this strategy in a realistic context, and we show that the approach yields measurable improvements in simulated application performance on grid platforms.

2 Background

2.1 Entity-level Application Template

The entity-levelmodeling idea has been proposed in many scientific fields (e.g., [1,3,5,6,15]). Examples of these efforts range from creating large-scale traffic systems from models of individual vehicles, to studying the biological immune response from the perspective of interactions between immune cells and foreign bodies (e.g. viruses and bacteria). From a computer science perspective, entity-level models logically correspond to iterative applications in which entities are represented by parallel tasks, each of which exhibits computational and communication requirements based on the fundamental properties of the entity itself. Based on a careful examination of entity-level models from numerous application domains, we believe that it is possible to design strategies that are generally applicable across application domains. Guided by these commonalities, we have defined an *entity-level application template*, which serves to specify the model-specific properties that impact application performance (see [24] for all details). These common properties are the basis for the entity-level template:

- **State information**: Most entity-level models we reviewed involve entities that encapsulate state data that differentiate individuals in the system. Collectively, these data represent the state of the system.
- **Entity interactions**: Large-scale emergent behaviors result from the combination of many small-scale interactions; entities in our template exhibit interaction models that effect appropriate modifications to entities' states. In the context of simulation, these mechanisms generally correspond to inter-entity (and potentially inter-processor) communication operations.
- **Internal behavior**: Internal behaviors represent the mechanisms by which entities respond to environmental factors. Effectively, these behaviors specify the state-change operations that each entity should undergo as the simulation progresses from iteration to iteration. Internal behaviors would primarily correlate to computational load incurred by the entity task.
- **External behavior**: Environmental properties of the system iften affect entity behavior. External behaviors of entities are essentially algorithms that govern the environmental effects on entities and the effects of entities on the environment. These functions would normally result in a mixed workload of entity task computation and inter-task communication.

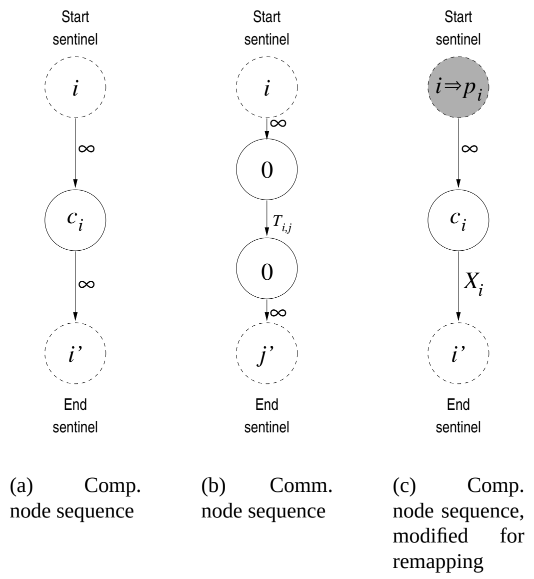


Figure 1. DAG node sequences corresponding to computational operations in entity-level models

2.2 DAG Representation of Entity-level Models

We have previously studied the viability of employing well-known DAG (Directed Acyclic Graph) scheduling heuristics to find effective mappings of entity-level application tasks to computational resources. We thus expressed instances of entity-level models as DAGs [26]; the basic components of the DAG representation for instances of entity-level models are:

- *unique root and end nodes*: common ancestor and descendant, respectively, for all DAG nodes;
- *sentinel nodes*, of which there are two for each entity in the entity-level model, and from which entity task mappings are constructed by assigning each entity to the processor corresponding to the one chosen for its sentinel node by the DAG scheduling heuristic;

- *infinite-weight edges*, a specialized edge that is used to link DAG nodes that must be co-scheduled on the same processor in order to ensure that the DAG scheduling algorithm generates a feasible entity task mapping; and
- *computation and communication node sequences*, which are DAG subgraphs (depicted in Figures 1(a) and 1(b), respectively) that correspond to each entity computation and inter-entity communication operations warranted by the current state of the model.

Our DAG construction proceeds as follows:

1. The distinguished root and end nodes are instantiated.
2. Sentinel nodes are created for each entity in the system. *Start sentinels* are connected by zero-weight edges from the root node; *End sentinels* are connected by zero-weight edges to the end node.
3. Computational node strands (with node weight c_i representing the complexity of entity computation) are created for each entity i and are linked to that entity's start and end sentinel nodes by infinite-weight edges.
4. Communication node strands (with edge weight $T_{i,j}$ corresponding to the amount of data to be transfered) are created for each pair of entities (i, j). These strands are linked to the sender's start sentinel and receiver's end sentinel by infinite-weight edges.

A DAG thus constructed encapsulates all relevant computation and communication costs dependencies exhibited by an entity-level simulation. In [26], we found that by creating such DAGs based on the initial states of entity-level models, invoking DAG scheduling algorithms on these graphs, and creating entity-level task mappings based on the results of the DAG scheduling algorithm, we were able to deliver substantially improved performance to entity-level simulations (e.g., over traditional space-partitioning techniques).

2.3 DAG Scheduling Heuristics

Using the DAG transformation methodology introduced in Section 2.2 effectively generates DAG-structured representations of entity-level simulation workloads. This enables the use of numerous well-known DAG-based parallel application scheduling heuristics for entity-level applications. In this paper, we focus specifically on two heuristics that we have previously shown to be efficacious in this context [24]: the "earliest task first" (ETF [14]) and the "dynamic level scheduling" (DLS [23]) strategies.

2.4 Related Work

The problem of deploying entity-level applications in grid environments is closely related to the problems of dynamic load balancing [4, 29, 30] and irregular workload scheduling [2, 18]. Compared to the dynamic load balancing literature, our work considers an application model that exhibits a qualitatively greater degree of dynamism in two aspects. First, the costs of rebalancing workload (i.e., migrating an entity's data and computation) may vary significantly depending on runtime factors. Second, in the process of modeling emergent behaviors, the performance of entity-level applications may fluctuate significantly more than application models in much of the load balancing literature. Of particular relevance from the irregular workload scheduling literature are graph scheduling problems, which include the DAG scheduling heuristics, as mentioned in Section 2.3; and irregular mesh applications, which resemble entity-level models insofar as the workload is distributed non-uniformly over the problem space.

Several efforts have examined the problem of entity-level simulations; an example of such an effort is the Swarm project [20]. Although Swarm is essentially based on the entity-level model, efforts to realize parallel implementation have not generally been scalable. Our work seeks to provide practical parallel task management strategies that could make a Swarm implementation targeted to grid platforms feasible. We also seek to benefit from work being done in the areas of molecular dynamics [21, 22] and particle-in-cell [11] methods. These application models, for which effective parallel implementations have been realized, are based on the same basic premise of emergent behaviors. These parallel applications rely on specific properties exhibited by the fundamental entities to facilitate efficient and scalable methods for managing large-scale parallel simulations.

3 Entity-level Task Remapping

3.1 DAG Representation for Remapping

To preserve application performance in the face of nondeterminism, we seek to adapt the DAG-based entity task mapping approach discussed in Section 2.2 to perform *application task remapping* – dynamic reassignment of entity tasks according to changes in the entity-level model state. The problem of remapping primarily differs from the mapping problem in that, in a remapping scenario, entity tasks are already deployed on platform resources. This factor has a critical impact on application performance when remapping: a new mapping that is similar to the current mapping will require far less time to implement than one that is dissimilar. Moreover, entity tasks may need to be remapped several times over the course of a single application execution, making efficient remapping schemes even more critical. To enable remapping that accounts for the existing task mapping, we propose two modifications to the DAG construction: (i) We let the start sentinel nodes of the DAG construction represent the *existing* entity task mapping. Thus, when the remapping scheme is invoked, we construct the DAG and pre-assign the start sentinels nodes according to the current entity task mapping. (ii) We assign a weight of X_i, based on the size of the entity state data, to the edge from the node in the entity's computational node sequence to its end sentinel. This weight is considered by the DAG scheduling routine to account for the costs of remapping an entity. The resulting computational DAG node sequence is shown in Figure 1(c).

To effect a remapping using such DAGs, we simply invoke a DAG scheduling algorithm to obtain a mapping of DAG nodes to platform resources. The new entity task mapping is constructed by assigning each entity to the processor on which its end sentinel DAG node is scheduled.

3.2 DAG Unrolling

Multi-iteration task remappings can be achieved via *DAG unrolling* – extending the DAG to encapsulate multiple application iterations – in a straightfoward fashion:

1. Start with the basic DAG, constructed as in in [26]. We term this a *one-iteration DAG*.
2. To construct an n-iteration DAG, we create $n-1$ replicas of all DAG node sequences corresponding to computation and communication operations required in the current entity-level application state. We also create $n-1$ additional sets of *internal sentinel nodes*.
3. Edges (with weights for computational strands equal to the remapping cost, as depicted in Figure 1(c)) connect each set of replicated DAG node sequences to a set of internal sentinel nodes, as detailed in [26].
4. The $n-1$ new blocks of DAG nodes are inserted into the DAG by breaking the links between the existing end sentinels and the terminal DAG node, and successively inserting edges from those nodes to the DAG node sequences of the newly created DAG subgraph blocks. Finally, the edges from the sentinel nodes in the last block are connected to the end DAG node.

The n-iteration remapping event is realized by (i) initially assigning start sentinels to processors according to pre-remapping entity assignments, (ii) executing the DAG scheduling heuristic on the unrolled DAG, and (iii) generating successive task remappings from each set of end sentinel nodes in the DAG. The task mapping is obtained using a process analogous to the one described in Section 3.1: for the first iteration after the remapping event is triggered, entity tasks are assigned to processors based on the first set of internal sentinel nodes. Each successive set of sentinel nodes denotes the task mapping for the subsequent iteration, yielding an n total task mappings.

3.3 Triggering Remapping Events

To determine *when* to perform a remapping event, we need a metric to measure the extent to which current application performance is degraded due to non-determinism in the entity-level model. First, we define

$$NormIterTime = \frac{IterationTime}{WorkloadMetric}$$

to represent application performance for the current iteration normalized for the amount of "work" in the system; for the application model used in this paper, we use the aggregate amount of data to be transfered in a given iteration as $WorkloadMetric$. During the simulated execution of the application, we maintain a running average of $NormIterTime$ values for all iterations immediately following a remapping event. We then define the current performance degradation factor as

$$CurDeg = \frac{(CurNorm - AvgNorm)}{AvgNorm}$$

where $CurNorm$ is the current value of $NormIterTime$, and $AvgNorm$ is the running average of $NormIterTime$ values. The $CurDeg$ metric serves as an indicator of the cumulative effects of entity non-determinism (e.g., movement, state changes, etc.) on execution performance. For each run, we also define a performance degradation threshold; when $CurDeg$ exceeds this threshold, a remapping event is triggered. In this paper, we consider two threshold values: 10% and 25%.

4 Experimental Results

To test the efficacy of our remapping strategy, we use a popular entity-level model from the field of ecology. Ecologists have long utilized modeling techniques to abstractly study the interactions between animals, plants, and their environments. Traditionally, these models (e.g., the Lotka-Volterra model [19, 28]) have been based on the system-level approach mentioned in Section 1. However, recent work has recognized the value of the entity-level approach in ecological modeling [12].

4.1 Entity-level Application Case Study: ATLSS

We base our studies on an instance of our application template corresponding to the ATLSS [13] entity-level ecological model of an ecosystem in the Florida Everglades. The principal features of this model are

- *predator entities* – representing the Florida panther, the focus of the ATLSS model – each of which is described by its movement behavior (the maximum dis-

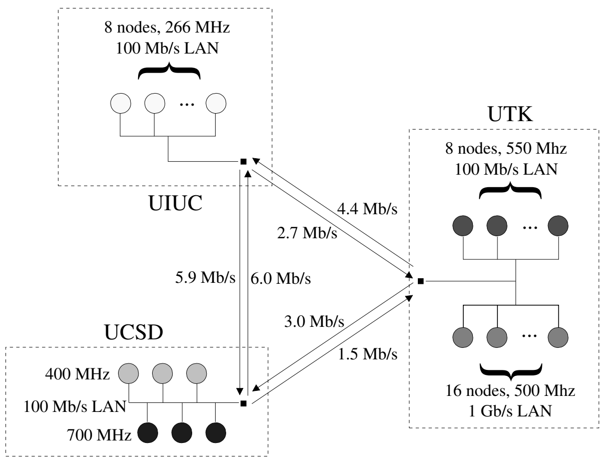


Figure 2. Grid platform: GrADS testbed

tance a predator entity moves in a single model iteration) and its predation behavior (the distance within which a predator entity hunts for prey);

- *prey entities* – representing the white-tailed deer, a common prey for the Florida panther – each of which is described primarily by its movement behavior; and
- an *environment*, defined by the physical extents of the model and characterized by *food sites*, which are effectively randomly located points that are designed to create realistic foraging behavior in the prey entities by introducing bias into their movement patterns.

To instantiate the model for the purposes of simulation, an ecologist would typically specify the size of the environment, the locations food sites, and the number and initial locations of predator and prey entities in the system. Based on our experience with this model we determined that the two main factors affecting application performance are the number of predators and the degree of physical mobility [24], which we study in this paper.

4.2 Different Platform Configurations

While in [24] we focused on cluster environments, here we target grid platforms. We simulate a grid testbed used in the Grid Application Development Software (GrADS) project [10] composed of clustered computing resources and network infrastructure from three participants in the project: the University of California, San Diego (UCSD); the University of Tennessee, Knoxville (UTK); and the University of Illinois, Urbana-Champaign (UIUC), as depicted in Figure 2. In this context, we are particularly interested in the disparity between cluster-level interconnection networks (LANs with capacities of 100 Mb/s and 1 Gb/s in the example given) and the inter-cluster networks (commodity

Remapping Heuristic	UTK only	UTK and UIUC	UTK, UIUC, and UCSD
Spatial	1.00	0.89	0.48
ETF	1.43	1.62	1.59
ETF, 2-iter unroll	–	–	1.84
DLS, 2-iter unroll	–	–	1.90

Table 1. Application speedup

Internet links). With such a wide range of processor and network capacities, we suspect that complex task remapping events may potentially overtax low-capacity links, limiting its effectiveness. Moreover, note that the capacity of the network between the UCSD and UTK sites is lower than the other two wide-area connections; in this case, this indicates that the throughput of data transfers may actually be higher if data are routed through the UIUC network.

Based on the above platform configuration, we designed an application performance simulator using the SIMGRID grid simulation toolkit [17]. Using this tool, we first analyze the application performance using a spatial remapping strategy using only the UTK resources (i.e. 24 relatively homogeneous nodes with LAN interconnection networks) in the GrADS testbed depicted in Figure 2. We run a small suite of application executions using representative configurations, take the average execution time for these runs, and define this as the reference value for the speedup analysis. We then run the same experiments, using the same platform and application configurations, for the basic ETF remapping heuristic without unrolling. The application iteration times are averaged and an effective "speedup" is computed. The leftmost column of Table 1 indicate that on a tightly coupled cluster, the using the DAG-based task remapping approach yields a measurable benefit over the naïve spatial remapping, due to the fact that the former is able to account for inter-entity communication costs.

We then add the UIUC computational resources into the platform, and re-ran the same two suites of experiments; the resulting speedup values are given in the second column of Table 1. We immediately observe the effects of introducing lower-performance wide-area links into the scenario on the efficacy of spatial remappings: because it does not account for the potentially significant variation of network performance between arbitrary nodes in the platform, the overall application execution time actually increases. On the contrary, the ETF DAG remapping heuristic accounts for the disparity in network link performance *and* the communication operations incurred by the application. Combining these, the DAG approach is able to effectively utilize the additional resources to improve execution time.

Finally, we add the UCSD resources to the modeled computational platform and again perform the same experiments. We expect and observe a sharp decline in performance using the spatial remapping scheme. However, we also note that the execution time using the DAG-based approach is essentially unchanged in most cases, and even degraded in others. On average, the entire suite of experiments using the ETF remapping heuristic ran slightly slower after adding the third site. By unrolling the DAGs before invoking the DAG-based remapping routines, remapping events that make more effective use of the additional resources are generated, and overall application execution times improve.

4.3 Efficacy of DAG Unrolling

We speculate that DAG unrolling would limit performance degradation on grids. Using our simulation framework we consider the DLS and ETF algorithms, deriving an initial entity task mapping using the methodology described in [26]; for each heuristic, we consider runs using remapping DAGs corresponding to one-, two-, and three-iteration unrolled application instances. We compare these to the “naïve” spatial approach, which neglects communication costs when assigning entity tasks to processors.

Figures 3(a) and 3(b) depict two typical application execution time traces corresponding to runs utilizing the DLS and ETF heuristics, respectively. In general, we find that for most application instances, a measurable performance improvement in terms of decreased overall application execution time (up to 12%) is achieved using a two-iteration unrolled DAG compared to a basic one-iteration DAG. Compared to their two-iteration counterparts, the overall execution times using one-iteration DAG remappings are 7.0% slower with DLS and 6.8% slower with ETF. Using a three-iteration unrolling yielded considerably lesser benefit; for example, for the DLS and ETF runs depicted in Figure 3, an additional level of DAG unrolling improved execution time by 0.4% and 0.7%, respectively.

4.4 Application Volatility

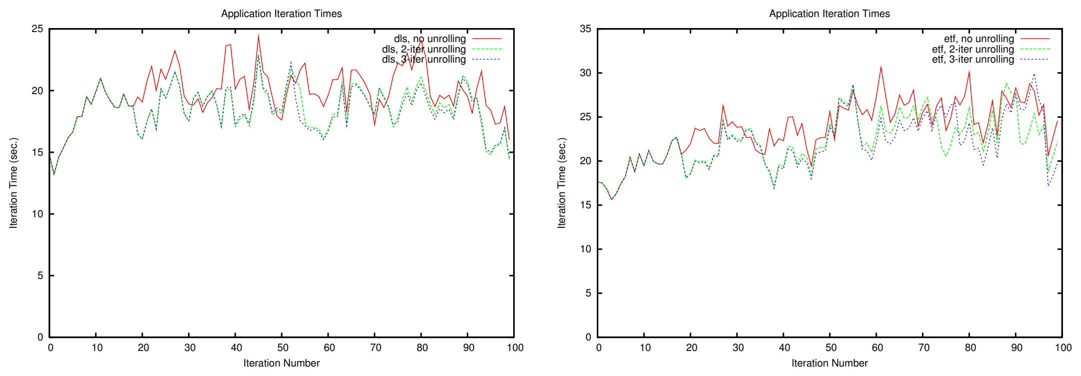
We identified several instances of volatile application configurations in which application performance modulates constantly and dramatically. In these instances, the efficacy of “smart” schedulers is limited, as the task mapping decisions they make are based on snapshots of application state that become inaccurate relatively quickly. In both cases, it is apparent that minimal benefit is derived from using a multi-iteration unrolled DAG to perform the remapping events. In particular, the overall projected execution time of the application using the three-iteration DAG compared to the one-iteration DAG was 0.9% faster for the DLS heuristic and 1.1% faster for the ETF heuristic. Furthermore, we conducted a minimal set of experiments with the same DAG remapping heuristics using four-iteration DAG unrolling. As expected, no significant overall performance benefit was observed. We believe the diminishing benefit of additional DAG unrolling is due to the baseline level of volatility in the entity-level model, and the effects of “shape” of the grid platform on the complexity of remapping events that are likely to be advantageous. By studying these factors, we hope to develop a methodology to automatically determine the appropriate degree of DAG unrolling, given an instance of an application and a targeted grid platform.

4.5 Aggregate Results

In aggregate, we ran three simulation runs for each of 64 applications configuration, totaling 192 runs. In each run, the entity-level model is evaluated for 100 iterations, and seven task mapping/remapping heuristics are compared: spatial decomposition, the ETF and DLS heuristics with one-, two-, and three-iteration DAGs. We evaluate the simulated performance using a metric commonly used to compare scheduling strategies known as *average degradation from best* [16]. For each run, the best scheduling strategy is noted and assigned a degradation value of 0%. All other execution times are compared to this value, and degradation values, corresponding to the differences in execution times, are assigned to each heuristic. Table 2 gives average degradation over the three runs for each configuration, categorized by values of application parameters (see Section 4.1).

Above all, we observe that a task remapping using a spatial decomposition results in application execution times that are substantially higher than those observed with DAG-based heuristics. This performance degradation is explained by the fact that the spatial decomposition focuses on load balancing the computational workload of a parallel application; communication costs incurred by the remapping events and the application itself are not considered. In the ecological entity-level model presented, there are thousands of individual entities that incur extensive interaction operations and that transfer substantial amounts entity-specific data, rending remapping costs non-negligible. Also, spatial decomposition fundamentally relies on the assumption that application workload is distributed fairly uniformly over the application space; in the ecological model, there exist numerous factors that may cause entities to cluster spatially, affecting the validity of this assumption from run to run.

In the ecology model, the “volatile” configurations (as mentioned in Section 4.4), were generally those with the smaller number (i.e., 50) of highly-mobile (i.e., predator movement index = 80) predator entities. In these runs, entity interaction patterns change quickly and dramatically as the application progresses. This volatility exacerbates the fact that remapping heuristics using unrolled DAGs effectively assume that entities maintain the same state for the



(a) DLS

(b) ETF

Figure 3. Application performance under a typical application configuration

Remap Threshold	# predators	Pred. mvmt.	Spatial	DLS 1-iter	DLS 2-iter	DLS 3-iter	ETF 1-iter	ETF 2-iter	ETF 3-iter
0.10	50	20	62.52	5.66	2.71	1.61	13.09	10.05	8.77
		80	40.22	1.67	1.66	1.66	5.17	5.16	5.18
	100	20	68.43	3.36	0.69	0.18	15.50	12.57	11.96
		80	35.88	4.08	0.80	0.00	12.86	9.27	8.43
0.25	50	20	57.79	8.69	2.83	1.56	13.36	7.37	5.84
		80	42.84	1.38	0.67	0.26	11.94	11.20	10.70
	100	20	66.44	7.89	0.94	0.40	19.65	12.02	11.35
		80	42.80	8.42	2.61	0.93	14.20	8.07	6.19

Table 2. Aggregate data: average percentage degradation from best

duration of the remapping event. In these cases, we find that the error introduced by this faulty assumption nullifies much of the benefit of using these methods.

In all cases, remapping time is included in the execution time corresponding to the iteration immediately preceding the remapping event. In most cases, we found that the benefit of performing the remapping using a DAG-based heuristic outweighed the cost of migrating entity data between nodes in the platform. Moreover, we observed that the heuristics using multi-iteration unrolled DAGs generally resulted in fewer remapping events over the entire run and in aggregate; although the time spent transferring entity data on a per-remapping basis were marginally greater, the benefit of having fewer remapping events generally meant that a comparable (if not smaller) amount of time was spent remapping when using multi-iteration unrolled DAGs. Additionally, although the remapping costs proved to have a negligible overall effect on iteration time for our ecology application, these factors may significantly affect aggregate application performance in models composed of “large” entities (in terms byte size of each entity’s state).

5 Conclusion and Future Work

As researchers in numerous scientific domains continue to turn towards entity-level models, the lack of appropriate scientific computing technologies to realize large-scale entity-level simulations is increasingly apparent. We believe grid platforms are well-suited to deliver the extraordinary aggregate computational power needed by entity-level applications, but to realize this potential, further study of the challenges faced by those seeking to implement entity-level grid applications is needed. In this paper, we showed that naïve distributed computing strategies are ineffective in realizing high-performance entity-level grid applications. Our simulation results indicate that application- and platform-sensitive task remapping strategies are needed.

We plan to continue experimenting with the approach described in this paper to determine if the application deployment process can be further automated. For example, in this paper, we considered various values for the unrolling level and remapping threshold parameters. Although we identified efficacious values for our experiments, we recognize that these are tailored to the specific entity-level model we consider and the grid platform we chose. We intend to experiment with other entity-level models and consider other grid platforms to determine if the effectiveness of our application task remapping strategies can be improved through algorithmic tuning according to these factors.

References

[1] C. A. Abbott, M. W. Berry, E. J. Comiskey, L. J. Gross, and H.-K. Luh. Parallel Individual-based Modeling of Everglades Deer Ecology. *IEEE Computational Science and Engineering*, 4(4):60–78, 1997.

[2] M. J. Berger and J. Oliger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *Journal of Computational Physics*, 53:484–512, 1984.

[3] S. Brunett, D. Davis, T. Gottschalk, P. Messina, and C. Kesselman. Implementing Distributed Synthetic Forces Simulations in Metacomputing Environments, 1998.

[4] G. Cybenko. Load balancing for distributed memory processors. 2(7):279–301, 1989.

[5] P. de Vries. An Individual Oriented Approach to Modelling Demography, Malaria and Illness. In *Proceedings of the Workshop on Spatial Aspects of Demography*, 2001.

[6] K. Erol, R. Levy, and J. Wentworth. Application of Agent Technology to Traffic Simulation, `http://www.tfhrc.gov/advanc/agent.htm`, 1998.

[7] N. H. Gartner, C. J. Messer, and A. Rathi. Revised Monograph on Traffic Flow Theory. Technical report, Oak Ridge National Laboratory. Available at `http://www.tfhrc.gov/its/tft/tft.htm`.

[8] D. L. Gerlough and M. J. Huber. Traffic Flow Theory: A Monograph. Technical Report Special Report 165, Transportation Research Board, 1975.

[9] N. Gilbert and K. G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 1999.

[10] GrADS webpage at `http://www.hipersoft.rice.edu/grads/`, 2004.

[11] Y. N. Grigoryev, V. Vshivkov, and M. P. Fedoruk. *Numerical "Particle-in-cell" Methods: Theory and Applications*. Utrecht, 2002.

[12] V. Grimm. Ten Years of Individual-based Modelling in Ecology: What Have We Learned, and What Could We Learn in the Future? *Ecological Modelling*, 115:129–148, 1999.

[13] L. J. Gross, D. L. DeAngelis, and M. A. Huston. Approaches to Large-scale Ecosystem Modeling Across Multiple Trophic Levels: Some Early Lessons from the South Florida ATLSS Experience. In *Proc. of the Workshop on Aquatic Ecosystem Modeling and Assessment Techniques for Application within the U.S. Army Corps of Engineers*, 1998.

[14] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[15] S. H. Kleinstein and P. E. Seiden. Simulating the Immune System. *Computing in Science and Engineering*, pages 69–77, Jul–Aug 2000.

[16] Y. Kwok and I. Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms . *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.

[17] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the 3rd IEEE Symposium on Cluster Computing and the Grid*, 2003.

[18] K. Li. Analysis of the List Scheduling Algorithm for Precedence Constrained Parallel Tasks. *Journal of Combinatorial Optimization*, 3(1):73–88, 1999.

[19] A. J. Lotka. *Elements of Physical Biology*. Williams and Wilkins, 1926.

[20] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations. Technical Report 96-06-042, The Santa Fe Institute, 1996.

[21] L. Nyland, J. Prins, R. H. Yun, J. Hermans, H.-C. Kum, and L. Wang. Modeling Dynamic Load Balancing in Molecular Dynamics to Achieve Scalable Parallel Execution. In *Proceedings of the International Symposium on Solving Irregularly Structured Problems in Parallel*, 1998.

[22] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1997.

[23] G. C. Sih and E. A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. 4(2):175–187, 1993.

[24] A. Su. *Task Mapping and Remapping Strategies for Parallel Entity-level Simulations*. Dissertation, University of California, San Diego, 9500 Gilman Dr., La Jolla, CA 92093, 2003.

[25] A. Su, F. Berman, and H. Casanova. Performance Modeling for Entity-Level Simulations. In *Proceedings of the Workshop on Parallel and Distributed Scientific and Engineering Computing Applications*, Apr 2003.

[26] A. Su, H. Casanova, and F. Berman. Utilizing DAG Scheduling Algorithms for Entity-Level Simulations. In *Proc. of the Tenth High Performance Computing Symposium*, Apr 2002.

[27] C. Ünsal and J. S. Bay. Spatial Self-Organization in Large Populations of Mobile Robots. In *Proceedings of the IEEE International Symposium on Intelligent Control*, 1994.

[28] V. Volterra. Variazioni e fluttuazioni del numero d'individui in specie animali conveiventi. *Memorie della R. Accademia Nazionale dei Lincei, Ser. VI*, (2):31–113, 1926. (Translation in Chapman, Royal N. Animal Ecology. McGraw-Hill, 1931. pp. 409–448).

[29] C. Walshaw, M. Cross, and M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. 47:102–108, 1997.

[30] R. D. Williams. Performance of Dynamic Load Balancing Algorithms for Unstructured Mesh Calculations. *Concurrency: Practice and Experience*, 3:457–481, 1991.

[31] R. Zvan, K. R. Vetzal, and P. A. Forsyth. PDE methods for pricing barrier options. *Journal of Economic Dynamics and Control*, 24:1563–1590, 2000.