PERFORMANCE MODELING FOR ENTITY-LEVEL SIMULATIONS

Alan Su* Fran Berman†* Henri Casanova†*

* Computer Science and Engineering Department,
University of California, San Diego.
alsu@cs.ucsd.edu

† San Diego Supercomputer Center,
University of California, San Diego.
{berman,casanova}@sdsc.edu

Abstract

Advances across many fields of study are driving changes in the basic nature of scientific computing applications. Scientists have recognized a growing need to study phenomena by explicitly modeling interactions among individual entities, rather than by simply modeling approximate collective behavior. This entity-level *approach has emerged as a promising new direction in a number of scientific fields.*

One of the challenges inhibiting the entity-level approach is the substantial resource requirements it entails. Unfortunately, such applications exhibit characteristics and behaviors which render traditional parallel computing techniques ineffective. Well-defined methodologies for achieving scalable performance on distributed computing platforms are needed. As an important first step, we present an abstract application model for entity-level applications, and we instantiate it for a case-study immunology application. Our experiments confirm that this model tracks application performance trends sufficiently well to study scheduling issues pertaining to entity-level applications. We identify a scalability problem inherent to the entity-level approach and use our model to quantify the potential performance improvements that remapping *strategies may yield.*

1 Introduction

Computer simulation offers many benefits to scientists studying the phenomena which define their fields. One approach involves identifying global parameters and properties of the studied system and defining mathematical models to describe how properties are inter-related. We term this a *system-level model*. System-level approaches have been widely applied to scientific problems in many fields. However, as researchers learn more about the fundamental phenomena that define their field, they often develop new theories and technologies which exceed the capabilities of system-level approaches. For example, scientists in several fields are beginning to recognize the importance of studying the mechanisms by which behaviors of individual entities aggregate to form the large-scale emergent phenomena they observe in real environments [8, 11, 12, 19]. We term this an *entity-level model*. Such models can provide greater insight into the studied phenomena, achieve improved accuracy, and make the scientific modeling task more intuitive.

Efforts to design and implement instruments to observe, analyze, and study entity-level properties are becoming increasingly prevalent [2, 9, 12]. In conjunction with these advances in theory and technologies, entity-based algorithms and applications that require massive parallel computing platforms are being developed. Although the behavior of a single entity may be much simpler than the aggregate system-level model, the behavior of each entity in an entity-level model must be tracked individually. Thus, the computational resource requirements of entity-level applications increase as the size of the simulated populations increase. As scientists seek to study larger entity populations, the resource demands of these scientific computing applications are growing dramatically. Unfortunately, the structure of entity-level applications renders traditional parallel computing techniques ineffective or unusable. Traditional approaches fail to effectively cope with the non-deterministic behavior and non-uniform work distribution exhibited by entity-level approaches.

In this paper, we describe a model for the general study

of entity-level applications. As a case-study, we demonstrate and evaluate our model for IMMSIM, an entity-level application in the field of immunology. Our experimental results show that our model tracks application performance trends appropriately. This model is an important first step towards developing effective task mapping strategies for entity-level applications. In this context, we observe that non-deterministic properties of entity-level applications may cause performance to markedly degrade over time. This degradation gives rise to a potential benefit from remapping – modifying the application task mappings as the application is executing. Using simulation based on the entity-level model, we quantify the potential benefit from remapping for our case-study application.

2 Modeling Entity-Level Simulations

2.1 Abstract Model

The basic structure of an entity-level simulation is a parallel iterative application. In each iteration, computation and communication tasks are determined based on the states and locations of entities in the simulation. For this work, we target tightly-coupled parallel architectures (e.g. clusters, MPPs).

In this context, we begin studying entity-level applications by constructing an application model. Our primary goal in constructing this model is to build a tool to analytically study the impact of task mapping heuristics on application performance. To meet this goal, the model is specified to satisfy two critical properties. First, the primary function of an application model is to serve as a **valid** predictor of application performance. Secondly, a model must be **widely applicable** to be effective: it must be both sufficiently general and expressive to describe entity-level applications from many domains.

We designed our initial model to meet these requirements by analyzing applications described in the literature of various scientific domains. The model essentially describes a collection of entities which exist within some application space. Entities may (i) move within the application space, (ii) incur some computational cost, and (iii) require data be transferred to/from other entities. The movement, computation, and communication each entity performs may depend on the entity's internal state, the state of surrounding entities, or factors present in the environment. Each entity also has a processor assignment that represents the computational node at which the entity's data are stored. In our model, computational cycles are spent at nodes that host entities, and bandwidth is consumed on network links connecting nodes that host communicating entities. Finally, entities' internal states are updated at each iteration until the application terminates. The details of the model are fully described in [22].

2.2 Case Study: Immunology

2.2.1 The IMMSIM Framework

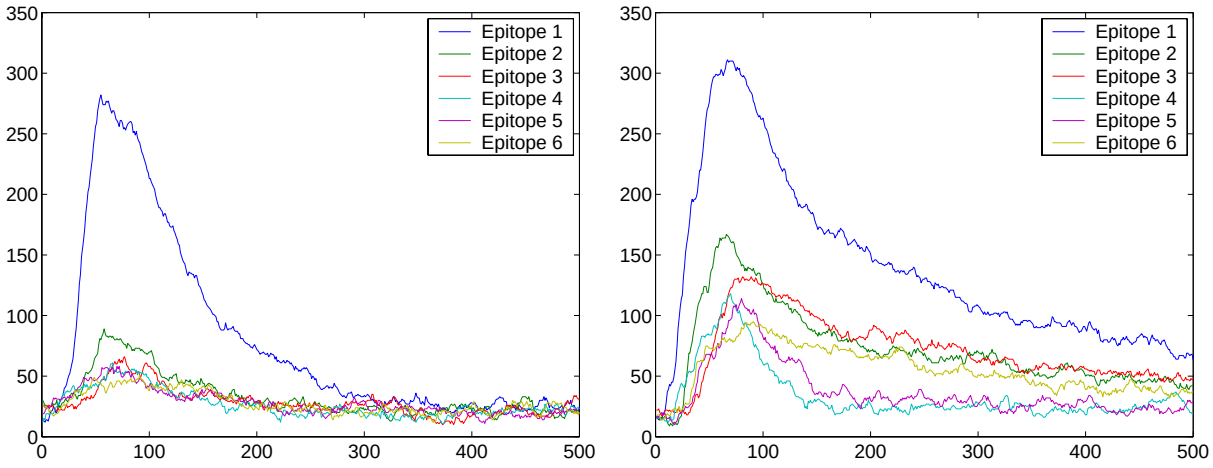
Traditional immunological models seek to describe the gross behavior of the human immune system by characterizing it as a set of interdependent equations governing various system-level properties. For example, an immunologist could express the birth- and death-rates of different types of immune system cells as differential equations. Such system-level models are widely used in various scientific domains other than immunology, and efficient algorithms for using distributed computing environments to attack such problems have been identified [15]. As scientific simulations grow larger and more detailed, effective parallel and distributed computing techniques are critically important.

As immunologists continue to discover new factors which contribute to phenomena they study, they are finding that defining system-level models is becoming increasingly challenging. In an effort to address this difficulty, the IMMSIM simulation model of the human immune response [16, 17] has been proposed. In contrast to system-level models, IMMSIM (**IMM**unology **SIM**ulator) describes the immune system at the entity level: the system dynamics are expressed through behavior algorithms for individual cells and molecules. The system is then described as a collection of independent entities, each associated with appropriate behaviors. Efforts to gain a greater understanding about the behaviors of these fundamental entities have made great strides recently. For example, instruments that allow fine-grained details of live immunological specimens to be observed have recently been developed [23]. Increased attention is being given to cellular- and molecular-level analysis as important tools for the future of immunological research.

2.2.2 Entity-Level Implementation of IMMSIM

Castiglione et. al. [5] have realized a parallel IMMSIM implementation, which structured as a modified cellular automata model: the simulated space is divided into discrete "sites" which contain the immune cells and molecules. Entities move about the space in discrete increments (between site locations), and entity interactions occur only between cells and molecules occupying the same site. Application performance was shown to be scalable for a significant range of parallel computing platforms.

The IMMSIM framework enables scientists to describe immunological systems by modeling individual cells and then replicating them throughout the system. Such an



(a) Original cellular automata code

(b) Modified code

Figure 1. Simulated T-cell counts using cellular automata and entity-level versions of the IMMSIM simulator

entity-level approach is more intuitive than traditional modeling techniques. The site-based modified cellular automata implementation is scalable, but substantially limited in the range of entity behaviors that can be expressed. To study entity-level properties without these constraints, we modified the IMMSIM application to allow entities to move freely about the simulated space and interact across site boundaries. We also implemented and studied modified codes based on both the PVM (**P**arallel **V**irtual **M**achine) [10] and MPI (**M**essage **P**assing **I**nterface) [21] inter-processor communication libraries. We found that the results obtained using the PVM and MPI codes were indistinguishable. Our modifications result in two significant changes in the application. First, cell and molecule movement through the simulated space now occurs in a continuous fashion, allowing for more realistic movement patterns. Second, interactions between cells and molecules are no longer constrained by arbitrary site boundaries. Rather, interactions are determined solely based on entity proximity, enabling more expressive and general interaction patterns.

Our implementation effectively preserves the fundamental scientific properties of cells and molecules described in the original IMMSIM model, but relaxes the site-based constraints on movement and interactions. In experiments comparing our modified code to the cellular automata code, we made three significant findings.

1. For the same values of application parameters, both applications yielded substantially similar system-level results. Figure 1 presents a comparison of sample runs using the two versions of the code. These graphs depict T-cell counts, a commonly measured property in immunological simulations. In both cases, the simulation involves an initial injection of antigens into the system, causing a spike in T-cell presence, followed by a gradual return to lower levels. We also determined our entity-level code produces results which represent scientifically valid scenarios after consulting with immunologists at the Scripps Research Institute. They conducted experiments infecting mice with lymphocytic choriomeningitis virus (LCMV). Figure 2 presents T-cell counts in infected tissue measured in these experiments [14]. Although the data differ in scale and granularity, the entity-level simulation captures the fundamental character of the immune response.

2. Using the cellular automata model, entity-level observations are somewhat limited. In contrast, using our modified code, entities move through the space continuously, achieving much more realistic entity behavior. As mentioned in Section 2.2.1, instrumentation technology in the field of immunology is advancing rapidly, driving the need for realistic simulations at the cellular level.

3. Our continuous-space implementation performs much worse than the unmodified version. The parallel application scheduler in the original code is able to capitalize on properties of site-limited cell and molecule interactions. For example, in a set of 500-iteration runs

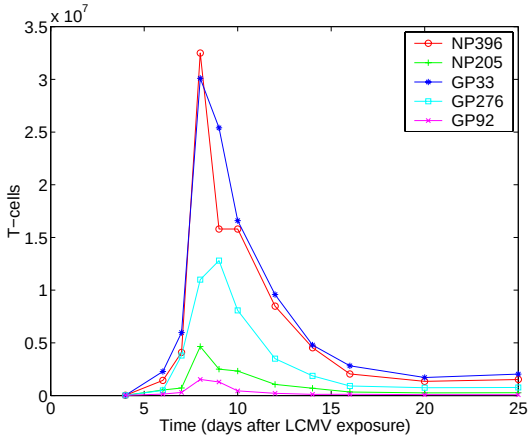


Figure 2. Sizes of T-cell populations observed by immunology group at the Scripps Research Institute

with cell counts ranging from 5,000 to 20,000 cells and platforms ranging in size from 2 to 16 nodes, the execution time of the site-based implementation was between 25 and 75 times faster. We plan to use the model developed in this work to better understand and address these performance challenges.

The entity-level approach is a promising paradigm for applications in immunology, as well as a number of other scientific fields. However, unlike more traditional designs, there are no techniques which have been shown to enable high-performance entity-level applications on distributed computing platforms. Strategies to negotiate the performance challenges inherent to the entity-level approach are needed to make these applications feasible.

2.3 Model Validation

Although entity-level scientific computing is becoming increasingly important in many domains, it is evident that achieving good application performance will be challenging. As a first step towards gaining a better understanding of the issues, we describe the instantiation of the model introduced in Section 2.1 for the modified IMMSIM application described in Section 2.2. We also show that the performance projected by the model correlates with actual application performance.

2.3.1 Model Parameters

The instantiation of our abstract model for the IMMSIM application has the following properties:

1. **Entity Classes**: We define five classes of entities, corresponding to the five major immune system cell types found in the application: B-cells, CD4 "helper" T-cells, CD8 "killer" T-cells, antigen-presenting cells, and epithelial cells. Their role in the IMMSIM framework is described in [17].

2. **Entity Computation Models**: In each loop iteration of the IMMSIM application, entities perform computation operations according to their state at that time step.

3. **Entity Communication Models**: Inter-cellular mechanisms are simulated in the interaction phase of the IMMSIM code. In our model, we project the amount of data each entity sends and receives in this phase.

Our primary modeling technique is based on detailed analyses of the actual application code. We observe that the computation and communication kernels used in IMMSIM operate on individual instances of cells. Although this structure is not surprising, it impacts the process of application modeling substantially. In particular, the model behavior cannot be evaluated based solely on static code analysis. Since each segment of code only governs the behavior of a single entity (instead of an aggregate system-level feature), much more information is needed to determine its impact on application performance. For example, factors such as the number, density, and locations of specific entities may affect the performance of entity-level computation and communication kernels. Consequently, we incorporate entity-level factors into our model in order to achieve more accurate performance projections.

Furthermore, as observed in Section 1, entity-level applications often exhibit non-deterministic behavior. The same code may perform differently depending on the specific entities involved in the behavior the kernel represents. Incorporating such detailed information into the entity-level application model would render the model prohibitively complex. Consequently, we seek to effectively capture the behavior of each kernel by using a *probabilistic* representation of its expected performance. This is done by analyzing the initial parameters of the IMMSIM application to estimate the relative frequency of particular events. The cost of each possibility is weighted by the probability of that outcome, resulting in a composite probabilistic model of the expected application behavior.

2.3.2 Validation Methodology and Results

The utility of our model for studying entity-level applications depends on its ability to predict application performance. Essentially, we seek to answer the following question: *does the model accurately track the application performance when properly instantiated for the current state of the application entities*? To answer this question, we instrument our code to evaluate the model accuracy at each

iteration of the application. The performance evaluation methodology is based on the framework used in [22]: we instantiate entities based on the active cells in the application and build a simulator to project the application performance. Our simulator employs the Simgrid [3, 20] simulation toolkit, which provides a rich set of facilities for evaluating parallel mapping and scheduling heuristics in distributed computing environments. To determine the accuracy of the model, we measure the time needed for the application iteration to execute and compare this time with the iteration time projected by the model.

We ran experiments for a number of sets of typical application parameters. In particular, we focused on the parameters which define the initial concentrations of each type of immune cell in the simulation. At the start of each iteration, we recalculate the projected computation and communication times for the given state of the application. We have also instrumented the application code to record the actual time spent performing computation and communication operations as the application proceeds.

Our experiments compared the actual execution times to the model-predicted execution times for computational platforms ranging in size from 2 to 16 processors. The pool of computational resources was composed of AMD Athlon and Intel Pentium III processors with processor speeds from 500 MHz to 800 MHz. Nodes were connected with non-dedicated, but generally quiescent 100 Mbit/s ethernet switches. The accuracy of the model did not vary significantly as the platform size changed. Figure 3 presents data for two runs that utilized an eight-processor platform and are representative of validation data we obtained. These plots compare the model-projected computation and communication times and the corresponding actual times, measured by our instrumented code. The runs depicted differ only in initial cell counts. The first run was instantiated with approximately $10,000$ initial cells (Figure 3(a)), and the second with $15,000$ cells (Figure 3(b)). For both cases, we compare the communication and computation times projected by our model with the actual times observed over the 500-iteration run.

In the initial phases of the application the immune cell populations increase dramatically in response to the antigens injected at the first time step. This growth in cell populations accounts for the dramatic increase in communication costs; as there are more cells in the system, more interactions take place between cells hosted on different nodes. Both the model predictions and the actual application communication times reflect this phenomenon. Although the average model prediction error per iteration was 29.5% for the $10,000$-cell run, and 26.0% for the $15,000$-cell run, our model tracks application performance trends appropriately. It is our experience that application models which accurately track performance trends are sufficient for developing and evaluating scheduling strategies.

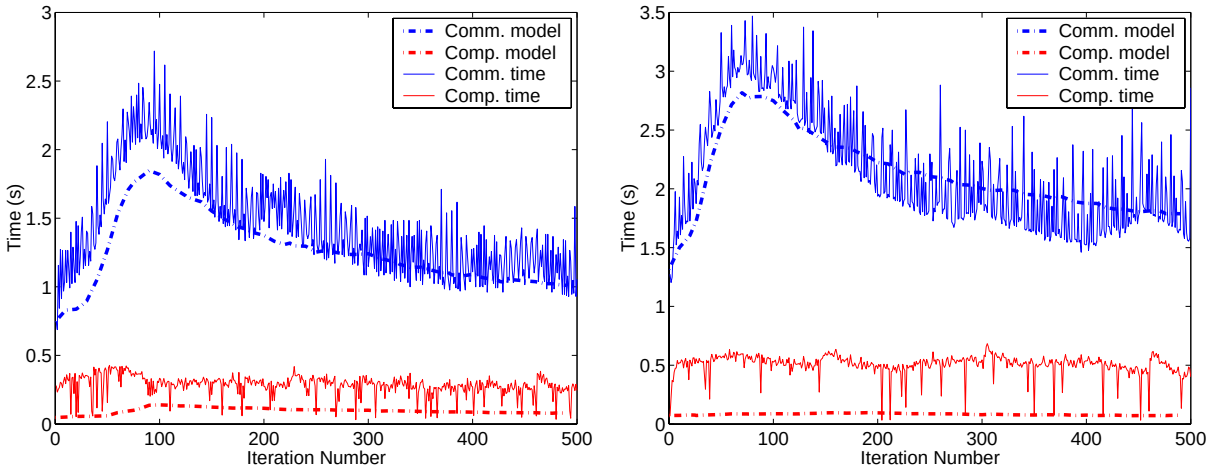
Several factors contribute to the discrepancy. We constructed our model based only on code found in the "interaction" phase of the application's main event loop, neglecting inter-processor communication operations that do not correspond with entity behavior (e.g. "bookkeeping" operations to track global cell locations). Secondly, as stated in Section 2.3.1, we employ a probabilistic approach to capture the non-deterministic aspects of entity behavior. However, the probabilities on which this approach is based may vary during the application run, leading to model inaccuracy. For example, the T-cell population may become specialized to fight a particular infection. Finally, our IMMSIM application uses standard parallel computing libraries – PVM and MPI – to perform communication operations. These facilities incur overhead costs that may vary based on the communication patterns of the application.

Examining the computation projections and costs, we find a substantial error. In a similar manner to the communication model, we built our computation model based solely on the "behavior" phase of the application's main event loop. Unfortunately, this does not provide an accurate predictor of computation costs. The code on which we based this part of the model consists of operations needed to update internal state variables for each cell at every time step. However, we have found that much of the computation time is spent in code which is not related to updating cell states. Consequently, the computation component of the entity-level model significantly mis-predicts the computation time. Note that these phenomena are specific to the IMMSIM application. The intent of our model is to study entity-level applications in general. Application-specific tuning of the model must be used carefully, as such modifications decrease the generality of the model (an important requirement as described in Section 2.1).

Our findings further suggest that the entity-level approach tends to incur a significant performance penalty. Our research seeks to identify the performance issues entity-level models introduce and to design strategies to cope with these problems.

3 Potential Benefit of Remapping

The benefits of entity-level models are balanced by the challenges in achieving good application performance. In order for entity-level applications to be viable for scientific computing, effective parallel execution techniques must be designed and studied. One critical impediment to application performance is the *non-determinism* exhibited by these models. Application non-determinism potentially impacts performance in two important ways. First, efficient work distribution is non-trivial. The computational requirements of the application depend on the positions and states of en-



(a) 10,000-cell run (MPI)

(b) 15,000-cell run (MPI)

Figure 3. Comparison of model-projected execution time and observed application behavior

tities within the model. Second, application state changes over time, resulting in time-varying application resource requirements.

Our initial work addressed the issue of irregular work distribution [22]. We showed that heuristics from the area of DAG scheduling [18] can be used to generate effective task mappings for entity-level applications. In this paper, we turn our attention to the problem of changing resource requirements resulting from application non-determinism. In particular, we consider remapping, a technique analogous to dynamic load balancing efforts, describe in the parallel scheduling literature [1, 4, 6, 7, 13]. Remapping has the potential to cope with the non-deterministic character of entity-level applications. To investigate this potential, we seek (i) to identify the causes of performance degradation as entity-level applications progress, (ii) to quantify the potential performance improvement a dynamic **remapping strategy** may achieve, and (iii) to show that the abstract model presented in Section 2.1 is an effective tool for studying entity-level applications generally.

3.1 Experimental Plan

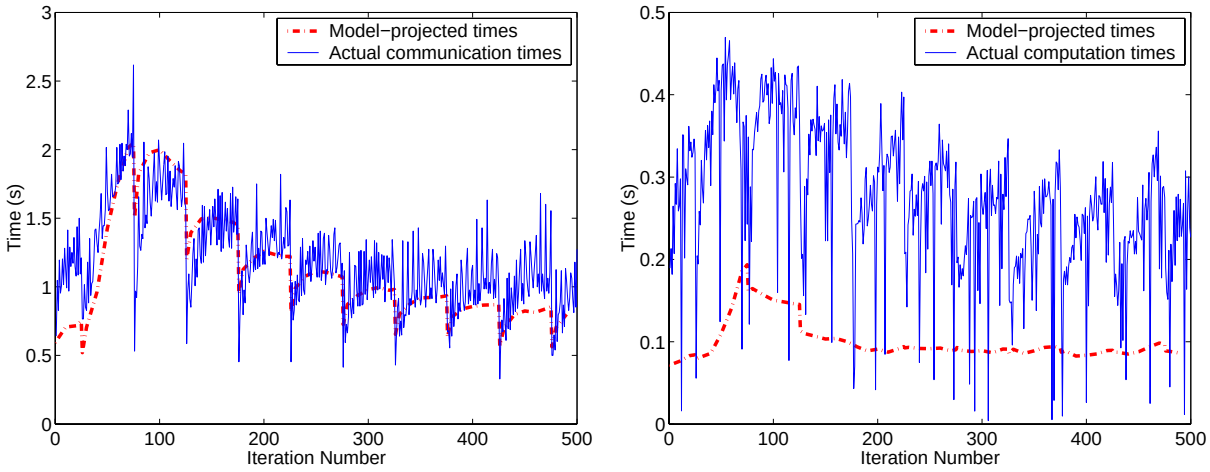
We extended our modified entity-level code to implement a *remapping event*. At the onset of a remapping event, progression of the simulation is temporarily halted. The assignment of cells to processors is recomputed based on the original IMMSIM work partitioning strategy, and processors trade entity data based on the new assignment. After cells have been exchanged, the simulation is resumed.

The goal of the remapping event is to adjust the application workload distribution to *balance computational load* across all processors in the system and to *minimize interprocessor communication* by maximizing the co-location of communicating tasks. Remapping events are scheduled to determine the potential benefit of remapping at various points during the application execution. We also evaluate our model before and after each event to determine if the model is able to project the impact of remapping. In the next section, we investigate the effects of injecting remapping events at various points in the execution.

3.2 Results and Discussion

We use the testbed presented in Section 2.3.2, and our methodology remains the same: at each iteration, we measure the amount of time spent in communication operations and the total iteration time. The total time spent by the processor in computation phases is then derived. To evaluate the potential over a range of application states, we inject remapping events into the application execution at regular intervals. The changes observed in communication and computation times (from the iteration before the remapping event to the iteration immediately after) characterize the potential of remapping. We again find that the accuracy of the model did not vary significantly for different platform and problem sizes. Figure 4 presents the results for our $10,000$-cell run on an 8 processor platform with remapping events injected. This experimental configuration is identical to the one used to obtain the data shown in Figure 3(a).

Figure 4(a) shows the communication times. The remapping scheme – one remapping event every 50 iterations,



(a) Communication times with remapping

(b) Computation times with remapping

Figure 4. Comparison of IMMSIM and entity-level model with periodic remapping events

starting at iteration 25 – is evident from the periodic changes in the communication times. The benefit from remapping is significant. Over the ten remapping events performed in this run, we compared the communication times for the iterations immediately before and after the event. The average decrease in communication times due to remapping was 61.1%. We also see that the application model predicts decreased communication times as a result of remapping.

The computation model, shown as the dashed line in Figure 4(b), consistently underpredicts the computational time required for a given iteration. Moreover, the model does not reflect the periodic remapping-related phenomena seen in the data for actual computation times. However, our suspicion that the communication costs would be the dominant factor in determining application performance proved true. Indeed, the combined application model (shown in Figure 5 and discussed below) is reasonably accurate. As we consider applications that exhibit more balanced communication-to-computation ratios, the computational models may need refinement.

The combined application and model performance data, depicted in Figure 5, were obtained using the same experimental configuration used to generate Figure 3(a), making it possible to compare the total execution times for these two runs. The application completed the 500-iteration run in 856 seconds without remapping events and 736 seconds with remapping events. No effort was made to enforce identical conditions on the computing environment, but in both cases there was basically no contention for either the computational resources or the interconnection network.

It should be noted that the results in this section basically characterize the *potential* benefit of remapping strategies. Two observations illustrate the efforts needed to realize that potential. First, the algorithm for reassigning cells at each remapping event is unremarkable: the initial cell allocation algorithm is essentially re-run. In our future efforts, we may consider factors like remapping event frequency and entity behavior predictions in making remapping decisions. Secondly, it should also be noted that the costs of performing remapping have not been considered in these results. A viable remapping strategy will need to weigh the potential benefits of remapping against the costs it incurs.

4 Scalability Study

In this section, we discuss the scalability of our entity-level implementation of IMMSIM, i.e. the performance achieved when using different platform sizes. Table 1 shows the speed-up achieved for two problem sizes: *small* ($10,000$ entities) and *large* ($40,000$ entities). The experiments were conducted with 2, 4, 8, and 16 processors on the testbed described in Section 2.3.2. With no remapping events, the parallelization is ineffective. In fact, the enormous amount of inter-processor communication causes the performance to degrade markedly as the number of processors increases. By contrast, we see that the use of remapping events leads to observable performance improvements. We observe speed-up of 1.47 for 8 processors for the small problem size, and 1.49 for 16 processors for the large problem size.

Although our results demonstrate parallel speed-up for

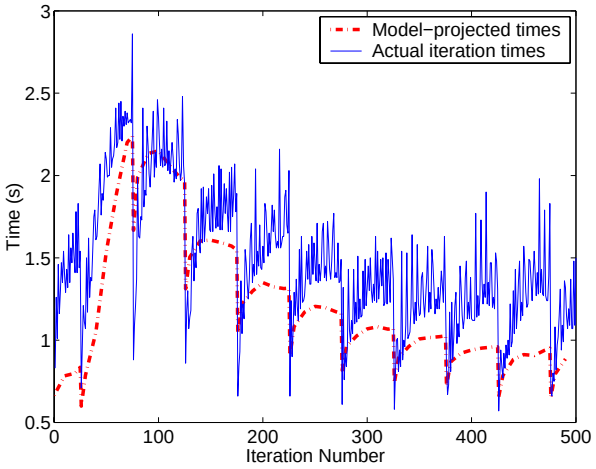


Figure 5. Comparison of combined application model and actual application performance

Number of processors:		2	4	8	16
Without remapping	small	0.70	0.69	0.36	0.16
	large	0.51	0.27	0.13	0.06
With remapping	small	1.09	1.34	1.47	1.45
	large	1.05	1.41	1.47	1.49

Table 1. Scalability results: observed speed-up without and with remapping events.

the IMMSIM application, the performance improvements are modest. Moreover, the results indicate that scalability will be challenging to achieve. The use of a commodity network may be one limiting factor. We suspect that advances in interconnection network technology will enable more tightly-coupled computing platforms and substantially alleviate this bottleneck. Also note that our remapping events are currently scheduled arbitrarily. Further gain might be obtained by more careful timing of these events. Finally, we use a naïve adaptation of IMMSIM code that was not designed for an entity-level implementation. In particular, redesigning the communication kernels to be sensitive to the costs of inter-processor communication will likely improve performance significantly.

5 Related Work

The limitations of system-level modeling are becoming evident in a number of scientific fields. For example, scientists studying vehicular traffic patterns have relied on simulation techniques based on a mixture of system-level models and queuing theory. Studies have shown that this methodology often fails to detect fine-grained phenomena and is only able to accurately model idealized systems [9, 19]. Observations noting the inadequacy of system-level models also appear in other fields, including social science [11] and ecological population modeling [12]. In particular, it has been noted that the mathematical characterization of the aggregate behavior of a population of biological entities is extremely challenging. In many fields, the success of system-level methodologies is based on relatively simple behaviors exhibited by the individual components of the population being modeled (e.g. molecules in a fluid). In contrast, the complexity inherent in a person or an animal casts doubt on the accuracy of any system-level model based on such entities. These trends are the driving force behind numerous efforts to design and implement entity-level models.

6 Conclusion and Future Work

The state of the art in scientific computing is rapidly evolving. As advances are made in scientific tools and instruments, researchers are developing new methodologies that challenge the current practices in parallel and distributed computing. New models are pressing the field of computer science to develop techniques to enable the next generation of scientific computing applications.

Entity-level applications comprise one class of applications that is emerging as an important computing paradigm in a number of fields. In this paper, we showed that we can effectively model the performance of entity-level applications. Additionally, we have shown that the entity-level approach entails a substantial cost in terms of application performance. Our model is able to characterize and predict performance penalties which result from the entity-level approach. We have validated our model by presenting a case-study of an entity-level application, the IMMSIM immunological simulation model. Finally, we have analytically and experimentally illustrated the substantial potential benefit of remapping strategies for entity-level applications.

We plan to continue this research by using our model to evaluate remapping heuristics for a greater variety of entity-level applications. We also plan to study aspects of the remapping strategy in greater detail. One particular area of interest is the scheduling of remapping events. In essence, we would like to answer the question: *when is the application likely to benefit from remapping?* The answer to this question involves many issues, including (i) the nature of the entities in the application, (ii) the cost of generating and implementing new task mappings, and (iii) the impact of future application behavior on the efficacy of remapping.

References

[1] R. Biswas, S. K. Das, D. J. Harvey, and L. Oliker. Parallel Dynamic Load Balancing Strategies for Adaptive Irregular Applications. *Applied Mathematical Modeling*, 25:109–122, 2000.

[2] J. N. Blattman, R. Antia, D. J. Sourdive, X. Wang, S. M. Kaech, K. Murali-Krishna, J. D. Altman, and R. Ahmed. Estimating the Precursor Frequency of Naive Antigen-specific CD8 T Cells. *Journal of Experimental Medicine*, 195(5), March 2002.

[3] H. Casanova. Simgrid: A Toolkit for the Simulation of Application Scheduling. In *Proceedings of the IEEE/ACM International Symposium on Cluster Computing and the Grid*, May 2001.

[4] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In *Proceedings of Supercomputing 2000*, November 2000.

[5] F. Castiglione, M. Bernaschi, and S. Succi. Simulating the Immune Response on a Distributed Parallel Computer. *International Journal of Modern Physics*, 8:527–545, 1997.

[6] G. Cybenko. Load balancing for distributed memory processors. *Journal of Parallel and Distributed Computing*, 2(7):279–301, 1989.

[7] R. Elsässer, B. Monien, and R. Preis. Diffusion Schemes for Load Balancing on Heterogeneous Networks. *Theory of Computing Systems*, 35:305–320, 2002.

[8] J. Epstein and R. Axtell. Artificial Societies and Generative Social Science. In *Proceedings of the First International Symposium on Artificial Life and Robotics*, Feb 1996.

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

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine*. MIT Press, 1994.

[11] N. Gilbert. Simulation: an Emergent Perspective, `http://www.soc.surrey.ac.uk/research/simsoc/tutorial.html`, 1996.

[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] T. Hagerup. Allocating Independent Tasks to Parallel Processors: An Experimental Study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[14] D. Homann, L. Teyton, and M. B. A. Oldstone. Differential regulation of antiviral T-cell immunity results in stable $CD8^+$ but declining $CD4^+$ T-cell memory. *Nature Medicine*, 7(8):913–919, August 2001.

[15] E. N. Houstis, J. R. Rice, S. Weerawarana, A. C. Catlin, P. N. Papachiou, K.-Y. Wang, and M. Gaitatzes. PELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms. *Transactions on Mathematical Software*, 24(1):30–73, 1998.

[16] IMMSIM webpage at `http://www.cs.princeton.edu/immsim/`, 1999.

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

[18] 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.

[19] J.-S. Oh, C. Cortets, R. Jayakrishnan, and D.-H. Lee. Microscopic Simulation with Large-network Path Dynamics for Advance Traffic Management and Information Systems. In *Proceedings of the 6th ASCE International Conference on Applications of Advanced Technologies in Transportation Engineering*, Jun 2000.

[20] Simgrid webpage at `http://grail.sdsc.edu/projects/simgrid/`, 2002.

[21] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The Complete Reference*. MIT Press, 1998.

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

[23] S. Valitutti and A. Lanzavecchia. Serial Triggering of TCRs: A Basis for the Sensitivity and Specificity of Antigen Recognition. *Immunology Today*, 18(6):299–304, 1997.