Partitioning on Dynamic Behavior for Parallel Discrete Event Simulation

Ketan Bahulkar, Jingjing Wang, Nael Abu-Ghazaleh and Dmitry Ponomarev
Computer Science Department
State University of New York at Binghamton
{kbahulkar, jwang36, nael, dima}@cs.binghamton.edu

***Abstract*—Partitioning plays an important role in PDES performance due to the high communication cost in parallel platforms and the fine-granularity of most simulation models. Traditionally, models are partitioned by deriving the static communication graph of objects and applying graph partitioning to reduce the mincut while load balancing the number of objects. However, many, if not all, models exhibit great diversity in their dynamic behavior: objects communicate with each other with diverse frequencies that are commonly power-law distributed. Similar diversity exists in the activity of objects and the processing requirements of events. In this paper, we argue that partitioning based on static graphs ignores these effects, leading to poor partitioning. We explore how partitioning based on dynamic information should be approached and explore policies that focus on communication cost, load balancing and both. We show that on multicore clusters, dynamic partitioning achieves up to 4x better performance than static partitioning. On the AMD magnycours, where the communication latency is low, dynamic partitioning results in a 2x performance improvement over static partitioning for some of our models. Our future work considers how to derive the dynamic weights (in this study, we do that through profiling), and how to balance the importance of communication and computation in a way that is informed by the underlying architecture.**

***Keywords*-PDES; partitioning; multi-core; many-core;**

I. Introduction

Parallel Discrete Event Simulation (PDES) is a fine grained application with dynamic dependencies: its performance and scalability is highly impacted by the cost of communication between the processors of the host parallel machine. Communication is used to transfer event messages to objects on other processing elements, as well as to implement synchronization algorithms (e.g., GVT for optimistic simulation, or lookahead messages for conservative simulation). The high message sending and receiving overhead slows down event processing. Moreover, the high latency causes events to be late, causing rollbacks (in optimistic simulation) or slow simulation progress.

Partitioning is one of the primary approaches to reducing the impact of communication: by mapping often communicating objects to the same processor or to nearby processors, communication frequency and distance is reduced. Another goal of partitioning is to maintain a balanced workload across the different simulation processes. For partitioning purposes, the simulation model is represented as a graph, where every vertex is a simulation object and every edge represents the fact that the two objects communicate during the simulation. A graph partitioning tool is then used to partition the graph to minimize the cut size (to reduce communication) while maintaining balanced partition sizes (to maintain load balancing) [1], [2].

With few exceptions, partitioning research has focused on graph based partitioning where all objects and edges are considered identical. The advantage to this approach, which we call *static partitioning*, is it requires only information about the static simulation topology. In practice, this topology often does not reflect the dynamic behavior of the simulation model. In particular, some edges in the communication graph may be significantly more important than others because the connected objects communicate frequently, or because the events are more critical (little look-ahead available between the event generation and its consumption); minimizing the impact of remote communication requires taking this information into account. Moreover, some simulation objects require more computational resources, such as processing and memory, than others either because they become active more frequently or because processing their events requires more computation; effective load balancing must take this information into account. Finally, at a higher semantic level, it is likely that dependency patterns play a role in determining effective partitioning.

We argue that taking the dynamic model behavior into account is critical to effective partitioning of simulation models. We motivate the need for incorporating this information by providing the evidence from real models that both edges and objects have substantially different behavior. Thus, given this information, partitioning can much better localize the most important dependencies, and load balance in a way that takes into account the behavior of objects, rather than just their counts.

The primary goal of this paper is to make the case for dynamic partitioning by quantifying the size of the opportunity for different types of models and with respect to different architectures. We develop a configurable synthetic benchmark that allows control of the distribution of communication and object activity. This configurable benchmark allows the abstraction of the dynamic behavior of real models. We observe and extract the dynamic properties of the model through a profiling run. The information is then represented in the form of a weighted graph. We develop

a number of strategies for partitioning to show the relative impact of communication and load balancing.

We study the partitioning strategies on two multi-core architectures: a dual quad-core Intel Xeon cluster, and a 48-core AMD Magny-cours system. The two platforms differ significantly in a number of ways, including the relative cost of communication to computation and the behavior of the memory subsystem. We discover that dynamic behavior based partitioning can outperform static partitioning by a factor of 4x on the cluster, and up to 2x on the Magny-cours system. The best performance is achieved by the partitioning strategy that emphasizes both communication cost and load balancing; this strategy consistently leads to effective partitions for different model types and simulation platforms.

The remainder of the paper is organized as follows. Section II motivates the need for partitioning that is sensitive to the dynamic model behavior by showing that such behavior exists in real models. We overview partitioning based on model behavior and introduce the partitioning strategies in Section III. In Section IV, we design a configurable synthetic benchmark that enables simulation behavior and structure to be configured independently to reflect the dynamic properties of observed real world models. Section V describes our experimental environment. Section VI then exercises our benchmark and partitioning approaches on two multi-core platforms to provide an estimate of the possible range of benefits. Section VII presents some related work. Finally we present our concluding remarks and future work in Section VIII.

II. Motivation: Dynamic Behavior and Static Structure

Real world simulation models exhibit dynamic activity patterns which static partitioning approaches are unable to exploit. Many phenomena exhibit skew in their behavior often in a way that does not correlate with their structure. In this section, we present two representative examples to demonstrate that such activity patterns exist.

A. Protein-Protein Interaction Networks

Systems biology is the study of the functional biological systems observed through the use of both wet lab and dry lab experiments. However, due to cost, most experiments are observed in labs first and then simulated to acquire further results [3]. One particular area in systems biology that exhibits interesting communication patterns is protein-protein interaction networks. In these networks, the degree of connectivity between proteins follows a power law distribution [4]. Another interesting aspect of protein interaction networks is how likely two connected proteins will interact. An example of the distribution of protein-protein interaction likelihood can be found in table I. In this table an interaction score of 2 means that two proteins are twice as likely to interact than arbitrary routine pairs [5]. Clearly, a large skew in interaction can be observed; this impacts both the communication between these objects, as well as the amount of processing they do.

B. P2P Networks

We also analyzed a P2P networking simulation benchmark and observed similar trends. In particular, Table II shows percentages of communicating object pairs and percentage of total communication instances when we only consider the object pairs that communicate at least the number of times defined by the communication frequency threshold parameter that is shown in the first column of this table. The results present the average communication frequencies (second row), as well as the communication frequencies observed at various other thresholds (including 10 standard deviations away in the last row). Even at this high threshold value, a significant percentage of all communication events is encountered (16%), while the number of distinct object pairs that contribute to it is very small (less than one tenth of one percent of all communicating object pairs).

The conclusion is that the communication patterns exhibit high skew, and the communication graph has a small number of edges with very high weights (activities) and a much larger number of edges with smaller weights. Clearly, the information about the structure alone is not sufficient to capture these dynamics.

As the two examples above demonstrate, skewed, and even power-law distributed behavior, is quite common in real models. Quite often, this behavior does not match the structure (or is even hidden by it). As a third example, the Internet topology is known to display power-law connectivity (structure) [6]. Commonly held understanding [7], translated into widely used simulation models [8], assumed that the core of the network is where the highest degree nodes existed. However, it was later shown that the edge routers have the highest degrees (to connect end customers) while core routers had small degrees due to the difficulty of scaling the number of interfaces on high speed routers [9]. Thus, structure would not identify core routers as the most active, potentially partitioning neighboring core routers apart, or failing to account for their disproportionate activity when load balancing.

Traditionally, PDES partitioning has been focused on static partitioning which can exploit the structural properties of the models. However, as we have shown, the activity patterns can differ significantly from the underlying static connectivity. Static partitioning can be ineffective or even harmful for such models. This is the motivation behind the work in this paper.

Table I
PROTEIN-PROTEIN INTERACTION

Interaction Score Cut off	Number of Protein-Protein Interactions Above Cut Off Score
0.25	79441
1	37606
2.5	25598
25	5394
250	1232
2500	498

* Source:http://www.compbio.dundee.ac.uk/www-pips/dbStats.jsp

Table II
P2P NETWORK SIMULATION

Communication Frequency Threshold	Number of Communicating Object Pairs	Percentage of Communicating Object Pairs	Number of Communication Instances	Percentage of Total Comm. Instances
1	37020	100	645436	100
17 (Avg)	6396	17	586740	90
131 $(Avg + stddev)$	1260	3.4	358556	55
245 $(Avg + stddev * 2)$	116	0.3	169224	26
1157 $(Avg + stddev * 10)$	26	0.07	100672	16

III. DYNAMIC PARTITIONING BASED ON MODEL BEHAVIOR

We have motivated the need to incorporate dynamic model behavior into partitioning decisions. In this section, we consider the problem of how to implement such a partitioning scheme.

A. Overview and Scope

There are two primary challenges that must be addressed.

1) *Extracting dynamic model behavior.* Obtaining dynamic activity information is not straightforward; it requires either profiling the model, static analysis of the model, or hints from the model developers.
2) *Partitioning based on dynamic information.* Once the behavior information is obtained, the second step is to exploit it in partitioning algorithms. Our approach annotates both the edges and the objects of the connectivity graph with weights derived from the dynamic information. The weighted graph is then partitioned. We discuss the approach in more detail in the remainder of this section.

In this paper, our focus is on the second problem: once the dynamic model information is available, how do we exploit it to produce better partitioning. We obtain the dynamic model information through profiling. The reason for focusing only on the partitioning problem is that it allows us to evaluate the size of the available opportunity. Once we establish that dynamic model behavior based partitioning can yield superior performance, our future work will address the first problem to enable practical exploitation of behavior information in partitioning.

B. Partitioning Approach

We profile the models to obtain the dynamic communication pattern between the objects, as well as the activity pattern of the objects themselves. This communication information is used to derive weights for the edges in the static connectivity graph. Similarly, the object activity is used to derive weights for the vertices in the same graph. We can now apply partitioning on the weighted graph; by minimizing the weighted mincut, the partitioning tool minimizes the dynamic cutsize (the number of remote messages, rather than remote edges). Similarly, by load balancing the object weights, the run-time is load balanced across PEs (rather than the number of objects across PEs).

To evaluate the importance of dynamic partitioning, and the relative importance of object weights to edge weights, we investigate the following six partitioning strategies:

1) **Random:** Random partitioning does not take into consideration any connectivity or activity information. It places an equal number of arbitrary objects on each processor regardless of their relationships to each other. Random strategy represents a baseline of no partitioning algorithms applied to determine object placement.
2) **Static:** Static scheme partitions a static connectivity graph, using static information for both edges and objects. All edges and objects are treated equally regardless of the their varying importance. Static partitioning represents the baseline of existing partitioning approaches.
3) **Object-Only:** Object-only partition strives to balance total weight of all the processors. It does not, however, consider inter-object relationships. Thus, it considers

dynamic object weights, but ignores connectivity information. This strategy is expected to perform well when the impact of balanced object workload is much more important than the inter-object communication.

4) **Activity:** This strategy takes into account the dynamic edge activity information, but only the static object weights (e.g., all objects have the same weight). The weight of the edge indicates the importance of the relationship between two objects, for example, as a function of how often they communicate or the criticality of these events (lookahead available between generation and execution time of the event). In models dominated by communication costs, this strategy performs well.
5) **Object-Activity:** This strategy takes into account both the dynamic object and edge weights. It is expected to provide the best performance across different simulation models as it minimizes communication as well as balances the workloads.

C. hMetis Partitioning Tool

We use the hMetis partitioning package [10] for graph partitioning. hMetis is a state of the art partitioning tool implementing both bi-partitioning and k-way partitioning for general weighted graphs. hMetis works by first creating a mincut partition, and then attempts to exchange objects to satisfy the load balance requirement. As a result, it is not necessary to normalize the edge and object weights to each other since they are not jointly optimized. The load imbalance constraint is specified by providing an imbalance factor to inform the partitioning tool of how much imbalance can be tolerated.

All of the partitioning strategies other than Random and Object-Only use hMetis. For Object-only, we use a simple bin packing heuristic where the next largest weight object is assigned to the processor with the least total weight.

IV. Benchmark and Experimental Methodology

PDES performance evaluations often use synthetic benchmarks, such as PHOLD [11], [12], because of the lack of large scale portable models to enable comparison across simulators and infrastructures. In PHOLD, the PEs are allocated an equal number of objects, and each object is initialized with the same number of events (on average). During simulation, each object randomly picks a target and sends an event to that target. Upon receipt, the target picks another object and sends an event. The total event population is preserved at all times. PHOLD is simple, and is generally effective for testing system performance in a controlled way. Extensions of PHOLD [13] to control remote communication percentage, as well as more general synthetic models [14] have been proposed. However, the behavior remains different from dynamic models and it is difficult to use them to evaluate model-related algorithms and techniques. Other environments [15] appear promising in that they take in the model topology and activity, but are specific to a simulation environment and were not available to us.

We developed a configurable synthetic benchmark that enables composition of abstract models with various structure and dynamic properties to represent real world models. Importantly the model allows independent specification of topology and dynamic behavior. We describe the model in the remainder of this section.

A. Topology

Topology defines the static structure of the simulation: which objects communicate to what other objects. Without loss of generality, we implemented two classes of topology: (1) *Hierarchical* models create a tree hierarchy of object groups. Objects have dense connectivity to nearby objects in the tree, there are sparse connections between remote objects. Structurally, this model is similar to systems such as road transport networks where there is dense network of roads inside a city while on a second inter-city level they are connected with a sparse highway network. The model is controllable in the number of levels in the tree, and the connectivity intensity at each level; and (2) *Uniform* model, on the other hand, does not exhibit structural variations in connectivity. It's unclear whether such models exist in practice, but we wanted to be able to evaluate partitioning performance when the structure carries no clustering information (which may help or harm partitioning depending on the dynamic behavior of the model). The model is configurable in the intensity of connectivity.

B. Dynamic Behavior

To represent various communication patterns, we use a Pareto distribution to control activity. The Pareto distribution allows us to controllably vary the activity pattern, creating skew that is independent of the underlying topology. In addition, the Pareto distribution is also optionally applied to object computation requirements to simulate different workload for each object. The distribution can be configured so that at one end nearly uniform distribution is obtained, while at the other, a heavy tailed distribution is obtained where some edges (or objects) have much higher importance than others. The Pareto distribution is described as follows.

$$Frequency = \left[\frac{-(UH^{\alpha} - UL^{\alpha} - H^{\alpha})}{(H^{\alpha}L^{\alpha})}\right]^{-1/\alpha}$$

where, U is uniformly distributed on (0, 1) , H is the upper bound while L is lower bound.

Figure 1 shows the frequency distributions at 6 values of parameter α. The communication pattern becomes gradually skewed with the increasing value of α. These communication levels however indicate the frequency of messages at each level in the topological model.

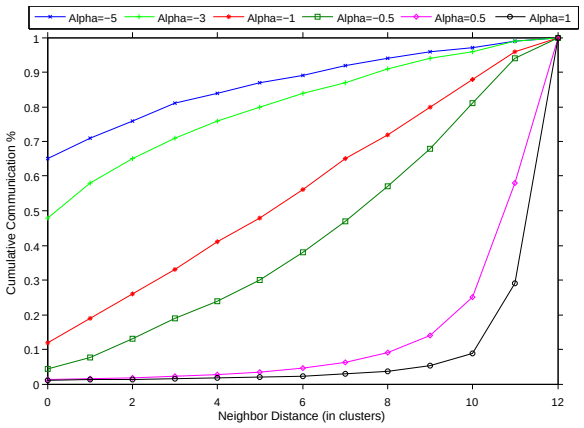


Figure 1. Communication Frequency Distributions

C. Benchmark Example

To ground the description above, and to provide a description of the simulation model used in the experimental section, we describe how we instantiate a hierarchical model example. A primary parameter in this model is number of *Levels* (L) in the hierarchy. At the leaf level, we start with a predefined number of objects x. Every subsequent level consists of two of the clusters below it. Therefore, the total number of objects in this model is x^L. A third parameter controls the number of neighbors each object has at each level. During initialization the object randomly chooses that many neighbors from the cluster at corresponding level and guarantees that the communication will be forwarded to one of these neighbors. As the candidate group size increases at upper levels but the number of edges remains the same, the density of edges is higher at lower level groups while it gradually reduces at higher levels. This creates a hierarchical static connectivity structure.

On top of this topology, we use the Pareto distribution with the specified α parameter to control the frequency of communication at each level. The steeper the curve, the more intensely we skew the communication frequency. For example, if we use α=-5 then most of the communication will be forwarded to the neighbors in nearest cluster. On the other hand, if we use α=1, most of the communication will be forwarded to the neighbors in the farther clusters.

V. Experimental Setup

In this section, we describe our experimental environment and methodology. Several PDES kernels have been developed over the years, both in academia [16], [17], [18], [19] and industry [20]. For our studies, we use Rensselaer's Optimistic Simulation System (ROSS) simulator developed at RPI [19]. ROSS is a state-of-the-art simulation engine with support for conservative and optimistic time warp simulations. However, little support is provided for partitioning or dynamic load balancing. Default strategies to distribute

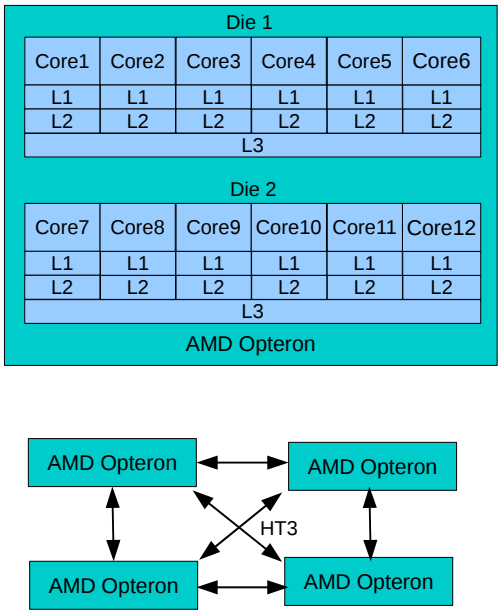


Figure 2. AMD Magny-cours (Total 48-cores)

objects in round robin fashion one object at a time, or a block of objects at a time are provided.

We have implemented support in ROSS to collect dynamic behavior information and to implement partitioning. Once we collect the dynamic behavior information, we use it to generate the weighted model graph. We use hMetis to partition the graph, and use its output to generate an object mapping file. We implemented a manual object placement scheme that reads the object mapping data and places objects on the corresponding processors.

We evaluate partitioning using two modern platforms. The first platform is a cluster of dual quad-core Intel Xeons. The second platform is an AMD Magny-cours machine, with four AMD opteron 12-core chips connected with a hyper-transport connection (Figure 2). The Magny-cours cores are tightly integrated providing low communication latency between the cores, but has Non-Uniform Memory Access (NUMA) latencies. Thus, the two environments have a different balance between the cost of communication and computation, and significantly different memory hierarchies, which affects load balancing decisions.

VI. Performance Evaluation

In this section we present a number of experiments to evaluate the impact of dynamic partitioning on simulation performance.

A. Impact of Topology

In the first experiment, we show the performance of dynamic behavior based partitioning for different activity patterns overlayed on top of the two different static topologies. Figure 3 shows the execution times with Static and

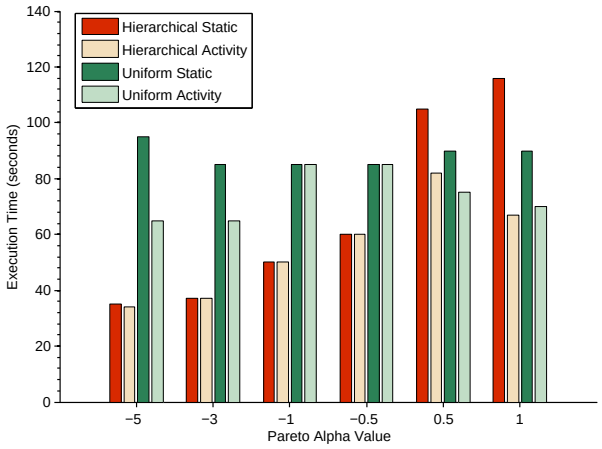


Figure 3. Hierarchical Vs. Uniform (Cluster 8 nodes)

Activity partitioning strategies for *Hierarchical* and *Uniform* models with all six frequency distributions (Refer to Figure 1). The results show that, in the case of *Hierarchical* model, for the first four *Alpha* values static connectivity based partition performs equally well as the activity partitioning. In these cases, activity patterns match the underlying structure making Static partitioning effective. However, for last two *Alpha* values activity differs significantly from the underlying connectivity. This information is captured and utilized by Activity partition and results in 20 to 40% benefit over Static partitioning.

Partitioning for the *Uniform* model reveals more about the connectivity-activity interplay. Since the *Uniform* model does not have a partitionable static connectivity pattern, the activity patterns influence the model behavior even in case of smaller *Alpha* values; the emergent dynamic pattern can be identified and used to partition the model. Thus, the effectiveness of partitioning is a function of the dynamic behavior of the model, regardless of the static relationships.

In the next experiment, we study the impact of dynamic partitioning on a *Hierarchical* fine-granularity, communication intensive application. Figure 4(a) shows the execution times for three partitioning strategies: Random, Static and Activity. For this scenario, activity-based partitioning substantially outperforms static partitioning by a factor of 2x on 4 nodes and 4x on 32 nodes. To explain this performance, Figures 4(b) and 4(c) show the static and dynamic mincut achieved by the partitions. The mincut represents the cut size once the graph is partitioned. The static cutsize considers all edges of equal weight (1). In contrast, the Active cutsize takes into account the frequency of communication on each edge. While static partitioning achieves a better static mincut, Activity partitioning achieves substantially better dynamic mincut size. In effect, the number of communicated messages reflected by the Active mincut, is greatly reduced, resulting in the better performance achieved in comparison to static partitioning.

In Figure 5(a) we repeat the experiment for a *Hierarchical*, computation intensive model where event execution requires substantial time (simulated using a delay loop). For high granularity models, Activity partitioning does not clearly outperform Static partitioning. Since the simulation time is dominated by event processing, the primary consideration for effective processing is load balancing. Figures 5(b) and 5(c) show that the Static and Active Mincuts of Activity and Static partitions closely follow each other.

In Table III, we show the dynamic load balancing achieved by the activity based partitioning for the fine-granularity model. The activity based partitioning only considers the dynamic communication information, but only load balances in terms of the object counts. Since object behavior also varies significantly, we can see in the table that it results in large deviations in load balance in terms of total object weights. We need to take into account dynamic resource usage of the objects.

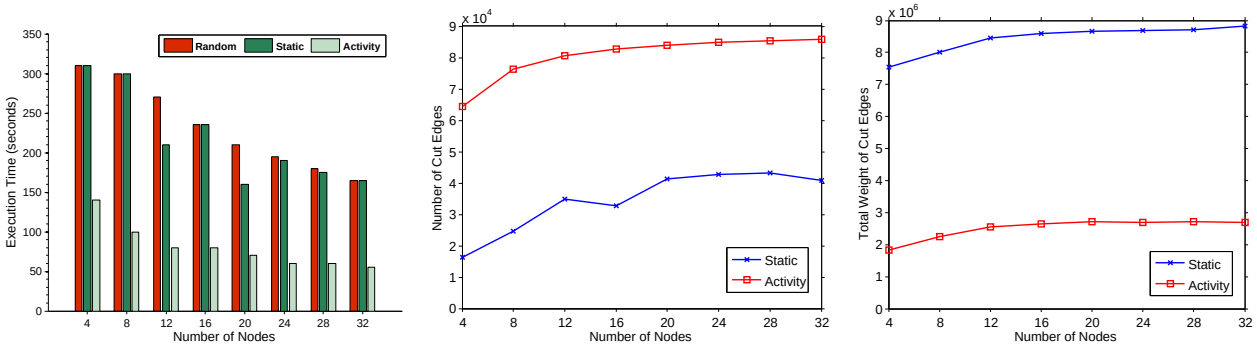
B. Combined Communication and Load Balancing

Table III indicated that there is further opportunity in case of Activity partition as it does not consider load balancing. Object-Activity partitioning strategy is useful where load balancing is important but we also need to consider communication. Figure 6(a) shows around 25 to 30% benefit of Object-Activity partition over basic Activity partition in a communication intensive model. Figure 6(b) shows the reduction in object deviation in the case of Object-Activity partition. Object-Only partition which solely focuses on load balancing does not perform well in this case as it fails to consider the communication.

Figure 6(c) shows the impact of Object-Only and Object-Activity partitions in the case of a computation intensive model. In this model, even though communication patterns exists, computation plays a more important role in determining overall performance. This is reflected in up to 3x benefit of Object-Only partition over Activity and Object-Activity partition. Object-Activity partition is primarily an Activity partition with a load balancing component. However in this case, where the communication is not significant, the Object-Only partition which purely focuses on load balancing performs better.

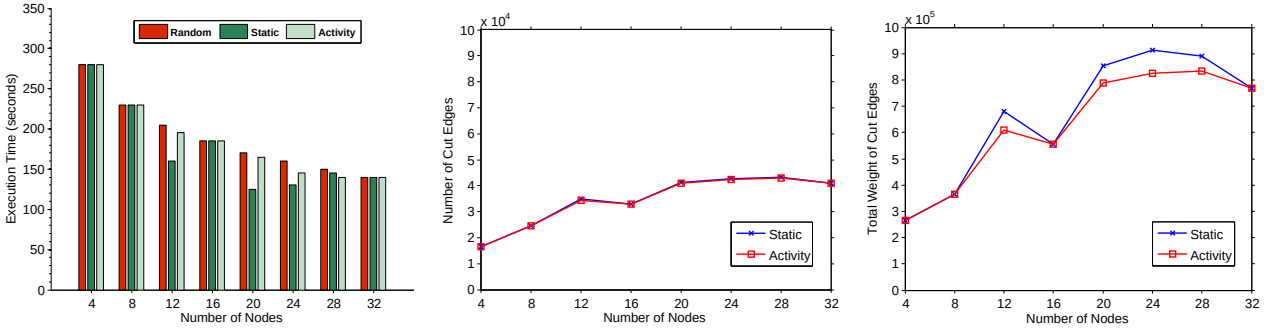
C. Effect of Processing Granularity

Figures 7 and 8 show the comparison of a communication-bound and computation-bound model on a cluster of multi-core machines and an equivalent configuration on the AMD Magny-cours respectively. Figures 7(a) and 8(a) show 4-way to 32-way execution of a communication-bound model (Pareto *Alpha* 1). Despite having slightly slower cores, the AMD machine was able to achieve significantly better performance than the cluster. This advantage can be explained by the low communication costs on a multi-core machine, in



(a) Execution time (b) Static Mincut (c) Active Mincut

Figure 4. Communication Intensive Model



(a) Execution Time (b) Static Mincut (c) Active Mincut

Figure 5. Computation Intensive Model

Table III
COMMUNICATION INTENSIVE MODEL: OBJECT DEVIATION

Nodes	4	8	12	16	20	24	28	32
Static	363	333	23484	169	16499	16034	14907	51
Activity	87708	55720	50401	68579	48606	45604	40209	41287

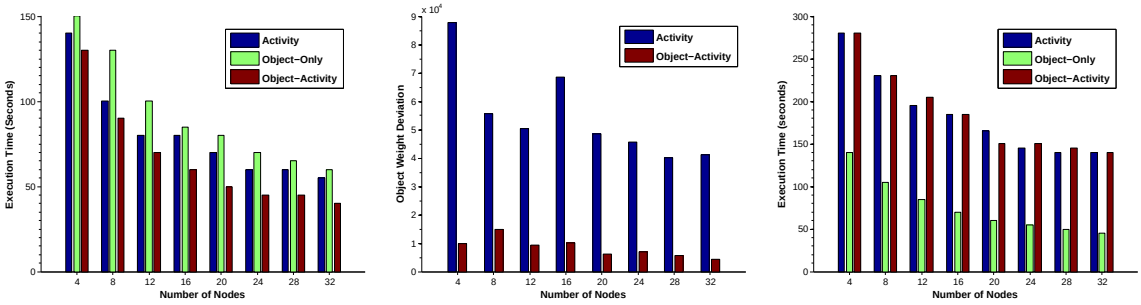
comparison with the high cost of network communication. The second observation is that the Object-Activity partition gives significant benefit on the cluster but not nearly as much on the multi-core machine where the low latency places a premium on load balanced partitions. Figures 7(b) and 8(b) show a computation bound model with a large, 3000 iteration delay loop for each event. As expected, the execution times are much higher. In addition, execution times on AMD Magny-cours are now only 10 to 15% better than those on cluster. In this case, the communication properties of the model were the same as the communication-bound model used here. Therefore this trend indicates that the computation intensive event processing had much higher impact on the Magny-cours machine than on the cluster.

VII. RELATED WORK

Several researchers have studied the use of partitioning to optimize the performance of PDES. In this section, we review some of the most related works.

Several efforts have targeted partitioning for logic simulation based on the static model topology. For example, Cloutier [21] studies the impact of various partitioning techniques on the performance of time-warp simulation of logic circuits. Various circuit parameters are used in partitioning including circuit topology (the netlist), the gate delays, the relative number of evaluations of the model of each circuit element, and the relative complexity of the element models evaluation. For partitioning, they represent a logic circuit by a weighted directed acyclic hypergraph. In the first approach, the computation load is distributed to the processors. The second approach is a mincut algorithm for weighted hypergraphs which minimizes the communication load of the simulation.

Another aspect of Static Partitioning is Static Analysis, obtaining the structural information from the model. This Structural information can be more easily extracted from models written with certain design methodology or higher level design languages rather than the one written in general

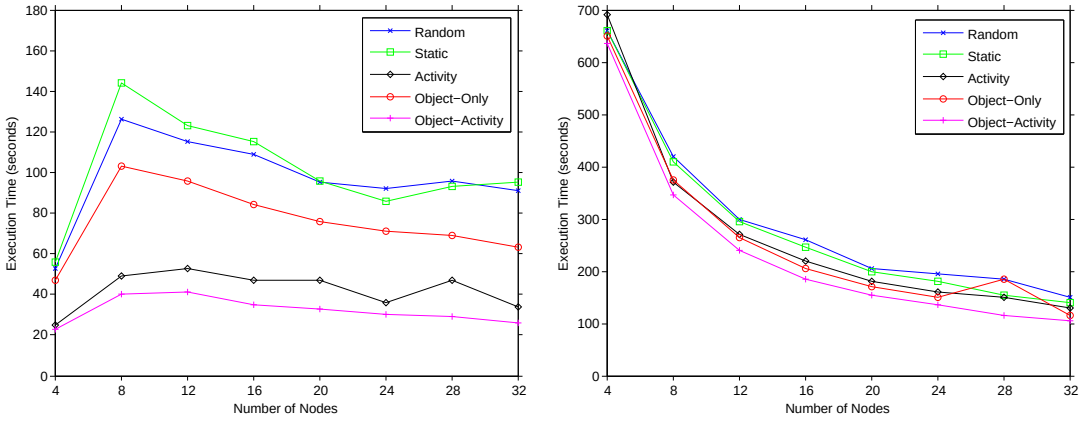


(a) Object-Activity: Communication aware Load Balancing

(b) Object-Activity: Object Deviation

(c) Object-Only: Computation Focused Load Balancing

Figure 6. Importance of Load Balancing



(a) Communication-bound Model

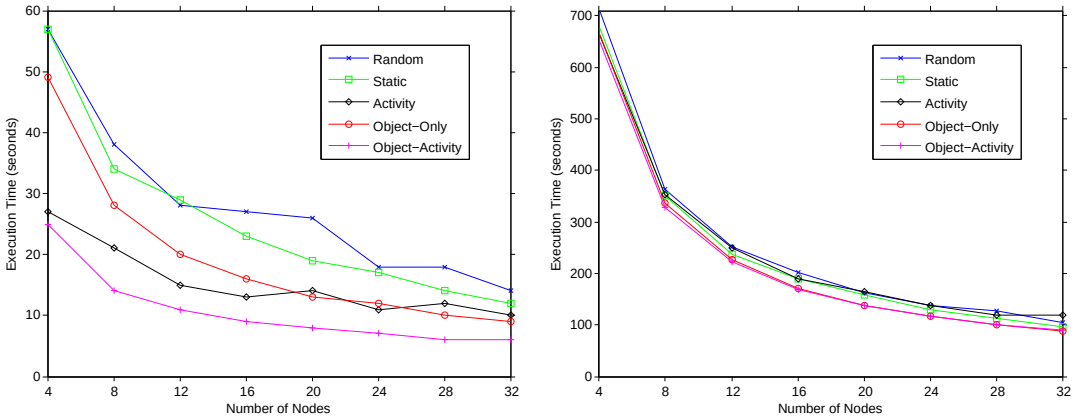
(b) Computation-bound Model

Figure 7. Cluster Performance

purpose programming languages. Reference [22] extracts such hierarchical structure information from DEVS models while [1] takes advantage of the design hierarchy of VLSI modules in Verilog. Instead of partitioning a gate-level netlist, [1] proposes a design-driven iterative partitioning algorithm which takes advantage of the design-level hierarchy embedded in VLSI module instances. A Verilog instance is represented by a vertex in the circuit hypergraph and is flattened to the gate-level if the load balancing was not achieved by instance level partitioning.

Similar to our work, Nandy et al. [23] attempt to track model activity but use it for dynamic object migration. They represent the LPs as nodes with weight indicating expected execution time for activations of that LP and links represent communication channels with weight indicating the expected number of messages. Estimates are obtained by pre-simulation runs for the sake of abstracting the problem. They motivate the need for partitioning by presenting a benefit of 20% and the impact of load imbalance. Though the remaining paper focuses on a parallel partitioning scheme based on movement of nodes which yields as good a partition as a sequential scheme, the overall graph based abstraction of the simulation is a recurring idea in many later load balancing studies. Reference [24] presents a static partitioning and mapping algorithm for conservative parallel simulations. It assumes the same abstraction mentioned in [23], and presents a partitioning scheme based on Simulated Annealing.

Wilson et al. [25] is a good example of load balancing study in the context of optimistic PDES simulation. Emphasizing automation of partitioning decision they compare three possibilities of balancing workload among the processors. The first approach determines the object placement based only on the computation weight of each object. This approach concentrates on balancing the workload on the processors but ignores communication costs. The second approach, arrange the LPs in a linear chain so as to keep heavily communicating LPs close to each other in the chain, thereby increasing the chance that they will be assigned to the same subchain and hoping that computation and communication are not correlated. The third approach, modifies the linear chaining algorithm to discourage clustering of extremes: pairs of computationally heavy-weight objects or pairs of computationally light-weight objects.



(a) Communication-bound Model

(b) Computation-bound Model

Figure 8. Performance on an AMD Magnycours 48-core Machine

A more comprehensive approach to partitioning for distributed simulations is proposed in [26]. It presents a partitioning layer in JAMES II, a modelling and simulation framework. The partitioning process involves a model analyzer to extract the model properties and an infrastructure analyzer to obtain the underlying topology of computing platform. Partitioning algorithms then make use of both sets of information for mapping the model on the processors.

Thulasidasan et al. [27] argue that dynamic load balancing presents significant implementation challenges due to object migration and explores the possibility of static partitioning for conservative simulation specifically for spatially clustered models with geographic hot-spots where most of the computation and messaging occurs (e.g. urban regions in transportation networks). They argue that in such models CPU load is a greater determinant of parallel simulation than message passing overhead. In general, dynamic object migration (e.g., [28], [29]) moves objects during run-time to achieve better partitioning. Since object migration is done on-line, with limited local knowledge, it cannot achieve the effectiveness of dynamic partitioning. However, unlike partitioning, object migration can adapt to changing simulation behavior.

VIII. Concluding Remarks

In this paper, we presented a case for partitioning based on dynamic model behavior. Most existing partitioning schemes only consider the static structure of the model. When application behavior exhibits large divergence from this static model, these approaches cannot effectively reduce communication or load balance the simulation. We showed that by taking the dynamic information into account, much more effective partitioning can result. In environments with high communication latency, this leads to up to 4x improvement in run-time for some applications. Up to 2x improvement was observed in a low-latency multi-core platform. Our future work targets effective approaches for identifying the dynamic model behavior. In the current study, we collected this information through profiling. However, it is likely that static analysis, perhaps augmented with limited profiling, can provide a more attractive approach for estimating the model behavior.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-11-2-0004. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. We also gratefully acknowledge support from the National Science Foundation grants CNS-0916323 and CNS-0958501. The authors thank Jason Gallia for his help with this paper.

References

[1] L. Li and C. Tropper, "A design-driven partitioning algorithm for distributed verilog simulation," in *Proc. of 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007, pp. 211–218.

[2] P. Konas and P.-C. Yew, "Parallel discrete event simulation on shared-memory multiprocessors," in *Proc. of the 24th annual symposium on Simulation*, ser. ANSS '91, 1991, pp. 134–148.

[3] R. Ewald, C. Maus, A. Rolfs, and A. Uhrmacher, "Discrete event modelling and simulation in systems biology," *Journal of Simulation*, vol. 1, no. 2, pp. 81–96, May 2007.

[4] H. Kitano, "Systems biology: A brief overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.

[5] M. McDowall, M. Scott, and G. Barton, "Pips: human protein–protein interaction prediction database," *Nucleic acids research*, vol. 37, no. suppl 1, pp. D651–D656, 2009.

[6] R. Govindan and H. Tangmunarkunkit, "Heuristics for internet map discovery," in *Proc. of Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2000.

[7] A. Medina, I. Matta, and J. Byers, "On the origin of power laws in internet topologies," *ACM SIGCOMM Computer Communications Review*, vol. 30, no. 2, 2000.

[8] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Brite: An approach to universal topology generation," in *Proc. of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001.

[9] L. Li, D. Alderson, W. Willinger, and J. Doyle, "A first-principles approach to understanding the internet's router-level topology," in *Proc. of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM)*, 2004.

[10] G. Karypis and V. Kumar., "hmetis: a hypergraph partitioning package." available on WWW at URL: http://www.cs.umn.edu/ karypis/metis/hmetis.

[11] R. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[12] K. Perumalla, "Scaling time warp-based discrete event execution to 1024 processors on a blue gene supercomputer," in *Proc. Computing frontiers*, 2007, pp. 69–76.

[13] A. Holder and C. Carothers, "Analysis of time warp on a 32,768 processor ibm blue gene/l supercomputer," in *European Modeling and Simulation Symposium*, 2008.

[14] V. Balakrishnan, P. Frey, N. Abu-Ghazaleh, and P. Wilsey, "A framework for performance analysis of parallel discrete event simulators," in *Proc. of the Winter Simulation Symposium*, 1997.

[15] P. Reynolds and P. Dickens, "SPECTRUM: A parallel simulation testbed," in *The Hypercube Conference*, 1989.

[16] D. Martin, T. McBrayer, and P. Wilsey, "Warped: a time warp simulation kernel for analysis and application development," in *Proc. of the 29th Hawaii International Conference on System Sciences*, 1996.

[17] Metron, Inc., "The SPEEDES parallel environment for emulation and discrete event simulation," 2008, available from http://www.speedes.com.

[18] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "Gtw: a time warp system for shared memory multiprocessors," in *Proceedings of the 26th conference on Winter simulation*, ser. WSC '94, 1994.

[19] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648 – 1669, 2002.

[20] WarpIV Technologies (J. Steinman et al), "The warpiv parallel simulation kernel version 1.5.2," 2008, software available from http://www.warpiv.com/.

[21] J. Cloutier, "Model partitioning and the performance of distributed timewarp simulation of logic circuits," *Simulation Practice and Theory*, no. 1, pp. 83–99, 1997.

[22] K. Kim, T. Kim, and K. Park, *Journal of Systems Architecture*, vol. 44, no. 6-7, pp. 433–455, March 1998.

[23] B. Nandy and W. M. Loucks, "On a parallel partitioning technique for use with conservative parallel simulation," *SIGSIM Simul. Dig.*, pp. 43–51, July 1993.

[24] A. Boukerche and C. Tropper, "A static partitioning and mapping algorithm for conservative parallel simulations," in *Proc. of the eighth workshop on Parallel and distributed simulation (PADS)*, 1994, pp. 164–172.

[25] L. Wilson and D. Nicol, "Experiments in automated load balancing," in *Proc. of the tenth workshop on Parallel and distributed simulation (PADS)*, 1996, pp. 4–11.

[26] R. Ewald, J. Himmelspach, and A. M. Uhrmacher, "A non-fragmenting partitioning algorithm for hierarchical models," in *Proc WSC*, 2006, pp. 848–855.

[27] S. Thulasidasan, S. P. Kasiviswanathan, S. Eidenbenz, and P. Romero, "Explicit spatial scattering for load balancing in conservatively synchronized parallel discrete event simulations," in *Proc. of 24th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, 2010.

[28] P. Peschlow, T. Honecker, and P. Martini, "A flexible dynamic partitioning algorithm for optimistic distributed simulation," in *Proc. of 21st International Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 2007.

[29] A. Boukerche and S. Das, "Dynamic load balancing strategies for conservative parallel simulation," in *Proc. of 11th Workshop on Parallel and Distributed Simulation (PADS)*, 1997.