Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

PROFILE DRIVEN PARTITIONING OF PARALLEL SIMULATION MODELS

AJ Alt

Dept of EECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221, USA

Philip A. Wilsey

Dept of EECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221, USA

ABSTRACT

A considerable amount of research on parallel discrete event simulation has been conducted over the past few decades. However, most of this research has targeted the parallel simulation infrastructure; focusing on data structures, algorithms, and synchronization methods for parallel simulation kernels. Unfortunately, distributed environments often have high communication latencies that can reduce the potential performance of parallel simulations. Effective partitioning of the concurrent simulation objects of the real world models can have a large impact on the amount of network traffic necessary in the simulation, and consequently the overall performance. This paper presents our studies on profiling the characteristics of simulation models and using the collected data to perform partitioning of the models for concurrent execution. Our benchmarks show that Profile Guided Partitioning can result in dramatic performance gains in the parallel simulations. In some of the models, 5-fold improvements of the run time of the concurrently executed simulations were observed.

1 INTRODUCTION

Discrete Event Simulations are simulations in which the events in the system occur at specific instants in time (Law and Kelton 2000, Fujimoto 1990). The time between events is unimportant to the simulation, and so the simulator only needs to process times at which events occur. This is in contrast to continuous simulations such as physics simulations, where a fixed time step is used, requiring the simulator to perform processing at every unit time step.

Because simulations often have long run times, it is natural to attempt to distribute the work load of a single simulation across multiple processing units. Parallel Discrete Event Simulation (PDES) is an active area of research that aims to maximize the performance of these simulations (Fujimoto 1990). While spreading the computation cost of a simulation across multiple computers can reduce the run time, the effect is often much less than the ideal linear speedup.

One reason that performance does not scale perfectly is that there is a large cost to communicating between processes, especially if the processes are on different computers (Carothers et al. 1994). Sending a message over the network is significantly slower than sending a message between processes on the same machine. This problem is especially problematic in PDES where event processing is mostly a lightweight process and where events are exchanged frequently. While higher performance hardware such as Infiniband (Rashti and Afsahi 2007) and lightweight messaging layers (Subramoni et al. 2009) in the operating system can help this problem, significant performance improvements can be made by decreasing the total amount of network traffic required by the parallel simulation. More specifically substantial performance gains are possible when the event communication frequencies experienced among the nodes in the compute cluster are minimized.

In this paper, we present a method of partitioning simulation objects between processors that aims to minimize network traffic, while balancing processor load, with the goal of reducing overall simulation time. The general approach is to use profile data from a pre-simulation run of the simulation model to implement a model specific partitioning and processor assignment. The profile based approach uses a small sequential simulator that pre-simulates the model to capture information on how frequently objects within the simulation model exchange event information. This event "profile data" is then used to establish a set of partitions that attempt to minimize the amount of event information that will have to be exchanged between objects of different partitions. The identified partitions are then used by the parallel simulation kernel to assign partitions (and the simulation objects therein) to the compute nodes of the parallel Cluster. The effectiveness of this method is demonstrated by implementing profile guided partitioning in the WARPED PDES kernel (Martin et al. 1996, Radhakrishnan et al. 1998). A number of real world models are used to evaluate the effectiveness of the approach against the naive partitioning algorithm currently used by the WARPED kernel. These studies show that a significant speedups can be achieved when the studied profile based partitioning method is applied to these models. We find that in real world models, speedups of up to six times can be achieved with this method.

The remainder of this manuscript is organized as follows. Section 2 presents some background in distributed simulation and partitioning. Section 3 presents some of the related work in partitioning for parallel simulation. Section 4 presents our work on profile-based partitioning for parallel simulation. Section 5 contains a set of experiments used to evaluate the profile-based partitioning. Finally, Section 6 contains some concluding remarks.

2 BACKGROUND

A Discrete Event Simulation primarily deals with sending events between objects (Law and Kelton 2000, Fujimoto 1990). For example, in the case of a digital circuit model, the objects are the gates, and the events are logic signals sent between connected gates. The simulation is not concerned with the physical propagation of the electrical signals across wires. When the output of a gate changes, the simulator calculates the propagation time of the signal, then fast forwards its simulation clock to the time that the signal arrives at its destination. The simulator then calculates the new output of the gate that received the signal and then repeats the process. Discrete Event Simulation is used for modeling and analysis for numerous fields across a broad range of disciplines (Law and Kelton 2000).

2.1 Distributed Simulation

The cost of transmitting events is compounded in a distributed environment due to the long latencies inherent in network traffic. In a distributed simulation, the set of objects is divided into a number of partitions equal to the number of simulation nodes, and each node processes events from a single partition (Fujimoto 1990). Objects can communicate with other objects on the same node, which only requires a memory access; or they can communicate with objects on other nodes, in which case a network message is sent. At any instant of the simulation, there may be many pending events waiting to be processed, since an object can create multiple events in response to a single input. These events may all be scheduled to arrive at their destination at different times.

In a sequential simulation, the simulator simply processes the event with the lowest time stamp. However, in a distributed simulation, the simulation nodes must communicate to ensure that the simulation results remain correct across the entire distributed compute platform. Because the time step in a discrete event simulation is not fixed, each simulation node will advance at a different rates of time. Without synchronization between the concurrent nodes, this can result in a faster running node receiving events scheduled to happen in its past. This is a causal violation of the events in the simulation model. Thus the parallel simulator requires some synchronization mechanism to ensure execution of events in their proper time order. While a centralized synchronization mechanism is possible, a distributed mechanism is preferable. For distributed synchronization, there are two widely used strategies, namely: *conservative synchronization* or *optimistic synchronization* (Fujimoto 1990).

Conservatively Synchronized simulators deal with the causality issue by sending synchronization messages between nodes to ensure that no out-of-order event processing occurs. This will ensure the correctness of the simulation, but the synchronization has a high overhead cost, and requires that the entire simulation run at the speed of the slowest node. Another popular method to perform distributed synchronization is Optimistic Synchronization. While there are several optimistic mechanisms (Chandy and Sherman 1989, Jefferson 1985), the focus of this manuscript is with parallel simulation synchronized with the Time Warp Mechanism (Jefferson 1985). Following the Time Warp model, all concurrently executing nodes process events in their event queues as quickly as possible without regard to the progress of the event processing rates of other nodes. If a simulation at one node receives an event with a time stamp in its past, it rolls back its state to a time before that message, and then proceeds to process events as normal. Additional details on the Time Warp mechanism are unnecessary for the specifics of this manuscript. Interested readers can find additional information in (Fujimoto 1990, Jefferson 1985). In the remainder of this paper, we will use the terms Time Warp and optimistic synchronization interchangeably.

2.2 Partitioning and Load Balancing

Even for optimistically synchronized simulations, it is important that each simulation node advance their simulation times at rates nearly equal to the other concurrently executing nodes. If one node has an unusually large portion of the workload, there will be a high number of rollback on the other simulators, and overall performance of the simulation will be reduced. Therefore, it is desirable to divide the workload evenly between processors. For many simulation models, the time to process a single event is fairly constant, regardless of the objects sending or receiving the event. Therefore, the workload can be balanced by simply balancing the numbers of objects on each node.

A more difficult problem is determining which combination of objects to place on each node. If a simulation uses N nodes, and objects are distributed randomly to each node, the probability that an event will be sent to a different node is $\frac{N-1}{N}$. This means that a large amount of network traffic is required, even for a small number of nodes. In an environment with four nodes, approximately 75% of events in the simulation will require a network message. If possible, it is advantageous to partition the nodes is a way that minimizes network traffic. In a synthetic model in which each object sends events uniformly to all other objects, the selection of partitions would have little impact on performance. However, in real world models, the event traffic can be extremely non-uniform. Although the amount of work required to process any single event is roughly the same for all events, the number of events sent from different objects can vary widely. In this case, the selection of partitions will have a large impact on the amount of network traffic, and consequently on the overall performance of the simulation.

3 RELATED WORK

Several studies have been published that attempt to partition specific simulation models using model-specific knowledge. Subramanian *et al* explored several strategies for partitioning Very-Large-Scale Integration (VLSI) circuit models using knowledge of the circuit netgraph (Subramanian et al. 2001). They found that certain strategies could achieve up to a doubling in performance under ideal conditions. Interestingly, this study was conducted on the an earlier version of the WARPED simulation kernel and used some of the same ISCAS'89 circuit models that were used for this work. Subramanian *et al* were not able to achieve the same level of speedup as this work, likely due to the fact that they statically partitioned the simulation objects without any knowledge of the frequency that events would be sent between objects.

Li and Tropper attempted to improve on the traditional approach to partitioning VLSI circuits, which attempt to partition on the gate level (Li and Tropper 2009). Li and Tropper instead partition on higher-lever modules using knowledge from the Hardware Description Language used to describe the circuits under study. In particular, they explored a novel approach to partitioning VLSI circuits by partitioning digital circuits on the scale of Verilog models instead of a flattened netlist. They found that they could achieve a smaller cut-size on the object graph than traditional methods, although the resulting run time was not compared to any other partitioning algorithms.

Bahulkar *et al* perform profiling of dynamic behavior of parallel simulations, and compare the performance of their algorithm across various model types in order to determine where dynamic profiling is most effective (Bahulkar et al. 2012). They created a synthetic simulation model in order to explore the effect of dynamic partitioning on various model topologies and compared the effectiveness of several different partitioning schemes, including a static algorithm that partitions based on a weight assigned to objects, and a dynamic algorithm that takes message activity for objects into account. They found that in a high latency environment such as a Beowulf cluster, dynamic partitioning can perform up to $4\times$ better than static partitioning.

Peschlow *et al* define a method of classifying processing requirements of simulation objects in a platform-independent manner, and integrate this model into a strategy of dynamic partitioning (Peschlow et al. 2007). Guo and Hu created an algorithm to partition a simulation model of a wildfire using profiling data (Guo and Hu 2011). They collected data from a low resolution simulation which they used to partition a more fine grained simulation. They found that their profile guided algorithm could achieve speedups of up to $1.5\times$ over traditional uniform spatial partitioning. While related to this work by virtue of using a profile based approach. Their profile data is drawn from a very specific knowledge of the simulation model under consideration (wildfires). Our work is entirely independent of the particular simulation model and can work for an simulation model.

4 PROFILE BASED PARTITIONING FOR TIME WARP

Although it is possible to achieve good results by using model specific knowledge to partition objects, a more general method is desirable. We hypothesize that the sequential characteristics of a model are representative of its performance in a distributed setting. Therefore, we collect data about the model in a sequential setting, and use this data to drive the partitioning algorithm. The ultimate goal is to minimize the amount of cross-partition messages, thereby reducing the amount of network traffic necessary in the distributed simulation.

Profiling is performed by running a model sequentially and recording the number of events sent between each pair of objects. These data are represented by an undirected graph in which each vertex represents a simulation object and each edge signifies that the vertices joined by that edge communicated during the simulation. We then weight each edge by the number of events sent between the pair of objects joined by that edge. Figure 1 shows the data collected from a run of an ISCAS 89 model, which is a gate-level logic circuit. Each vertex is a single logic component (such as an XOR gate), and the edges are a heat map showing the amount of event traffic between objects. Larger, red lines indicate a large number of events were sent between the corresponding nodes, while smaller green and blue lines show a small number of events. For this model, the most active objects communicated with each other several orders of magnitude more than average, as can be seen in Figure 2. This graph shows a power-law distribution that is common in real-world models.

Once we have this weighted graph, we can partition the simulation objects by partitioning the graph and mapping the graph partitions back to the simulation objects. Since the weight of each edge represents the number of events passed between objects, performing a weighted minimum cut of the graph will minimize the amount of events passing between nodes.

The problem of partitioning the graph G into k partitions is referred to as *k-way partitioning*. The goal is to partition the set of vertices V into the subsets $V_1, V_2, \ldots, V_k$, with the constraint that $V_i \cap V_j = \emptyset$ when $i \neq j$. In order to balance load effectively, it is desirable to choose partitions such that $|V_i| = n/k$. Additionally, the number of edges with incident vertices in different partitions should be minimized.

The library chosen to perform this partitioning is METIS, which is capable of performing multilevel recursive bisection and multilevel k-way partitioning on graphs (Karypis and Kumar 1998). Several other partitioning libraries were considered, including: PaToH (Çatalyürek and Aykanat 2011), Chaco

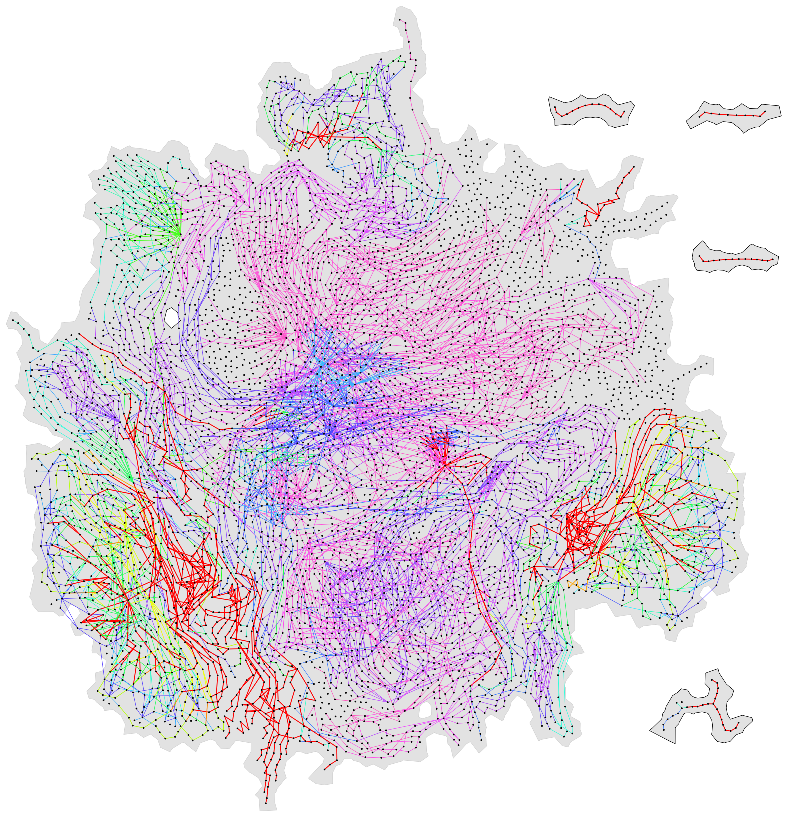

Figure 1: Heatmap of messages sent during the ISCAS'89 s9234 simulation.

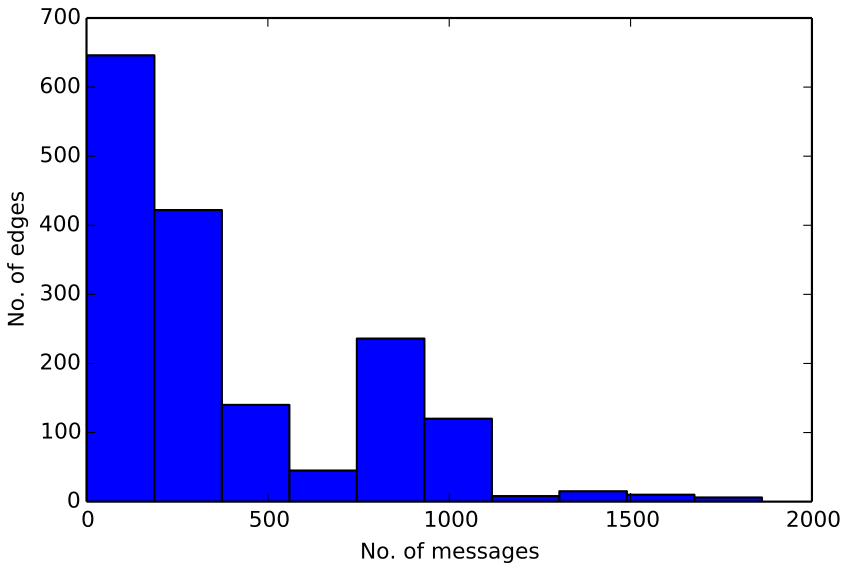

Figure 2: Histogram of edge weights for ISCAS'89 s9234 simulation.

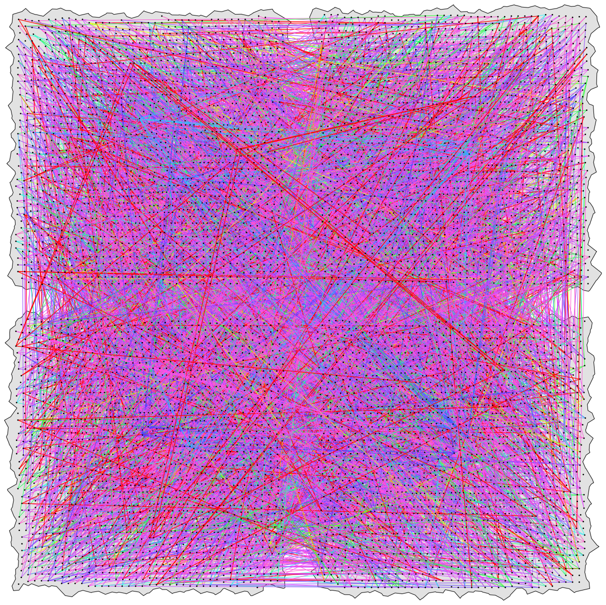

Figure 3: Heatmap of messages sent during the ISCAS'89 s9234 simulation, partitioned randomly into four partitions.

(Hendrickson and Leland 1995), and Scotch (Chevalier and Pellegrini 2008). Ultimately, METIS was chosen due to its highly compatible ANSI C implementation, its open licensing, and its impressive performance, both in speed and in quality of partitions.

Of the various partitioning algorithms supported by METIS, its multilevel k-way partitioning algorithm was chosen due the high quality partitions produced. This algorithm works by *coarsening* the graph by collapsing vertices and edges together to successively reduce its size. Once the size of the graph has been reduced to a small enough size, the small graph is partitioned and then uncoarsened into a partition of the original graph (Karypis and Kumar 1998).

Multilevel k-way partitioning can achieve very good results. Figure 3 shows the above graph partitioned into four partitions according to a random algorithm. The random algorithm naively places simulation objects into partitions in a round-robin manner without using any information about the objects themselves. As expected, 75.83% of messages cross between partitions. In contrast, Figure 4 shows the same graph partitioned using METIS to produce a minimum cut in which only 1.38% of messages cross between partitions. Figure 5 shows the same graph partitioned into 8 partitions, with 1.75% of messages crossing partitions.

5 EXPERIMENTAL ANALYSIS

We evaluated the performance of this algorithm using a number of real world simulation models using the WARPED time warp synchronized parallel simulation engine (Martin et al. 1996, Radhakrishnan et al. 1998). All tests were run on a Beowulf cluster with each node composed of a quad-core, hyperthreaded Intel Xeon processors running at 2.33GHz. The testing occurred with 2, 4, and 8 nodes (and, correspondingly, partitions) on the Beowulf cluster. The following simulation models are used in this study.

5.1 ISCAS89

The ISCAS89 benchmark is a standard set of circuits that make use of both combinatorial and stateful logic. Each circuit in the benchmark contains a number of inputs and outputs, logic gates, and D-Flip Flops. We

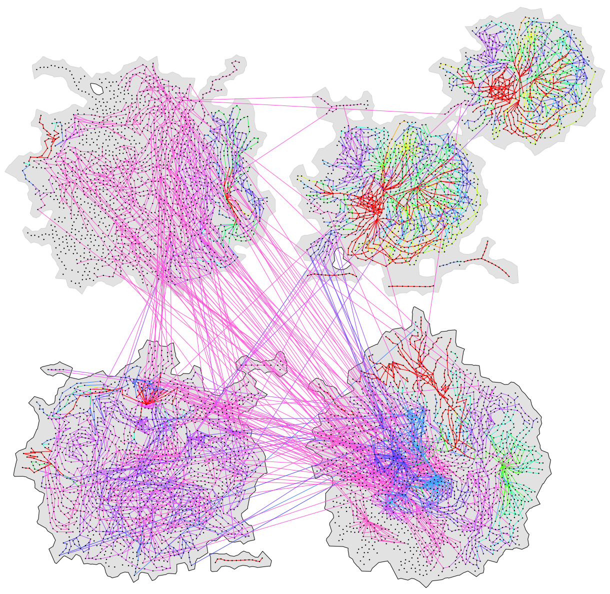

Figure 4: Heatmap of messages sent during the ISCAS'89 s9234 simulation, partitioned into four partitions using the profile guided algorithm.

study three separate circuits from this benchmark: s5378, s9234, and s38584.1. The characteristics of each can be found in Table 1.

Table 1: Characteristics of the ISCAS'89 benchmark circuits examined in this paper.

Circuit	Flip Flops	Inverters	Gates
s5374	179	1775	1004
s9234	228	3570	2027
s38584.1	1426	7805	11448

5.2 RAID

The RAID model is a simulation of a RAID-5 disk array. It models the characteristics of the array during reading and writing, including the calculation of parity information across the array. The model used in this evaluation contained 32 disks, 8 RAID controllers and 96 Input Processes generating disk activity.

5.3 Results

The benchmarks were run with the Profile Guided Partitioning algorithm described above, and with a random partitioning algorithm. The simulation times for the benchmarks are shown in Table 2 and graphed in Figures 6-9. The Profile Guided Partitioning algorithm outperforms naive partitioning by a factor of $1.74\times$ to $6.04\times$, depending on the simulation model and number of nodes.

Although significant speedup is observed in all models, the amount and scaling varies. The speedup for the RAID model remains roughly constant as the number of nodes increases. This model has a regular structure, which would allow the partitioning algorithm to work effectively over the range of partitions. It is likely that this model is compute bound at any number of nodes when partitioned correctly.

The various circuits of the ISCAS'89 benchmark each exhibit different speedup scaling behaviors. The speedup obtained with Profile Guided Partitioning for s5378 circuit decreases slightly for the s5378 circuit

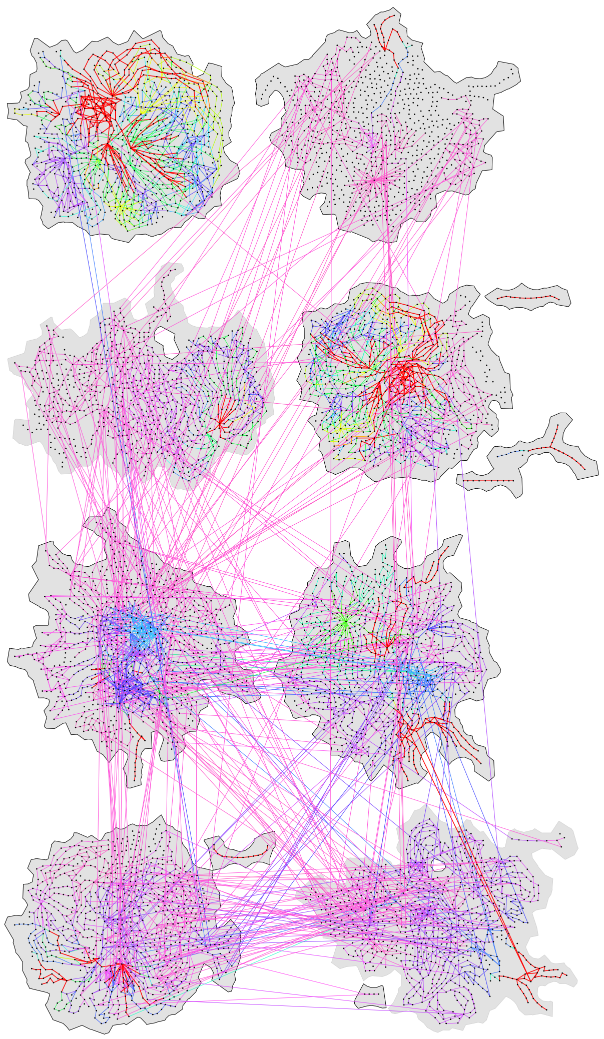

Figure 5: Heatmap of messages sent during the ISCAS'89 s9234 simulation, partitioned into eight partitions using the profile guided algorithm.

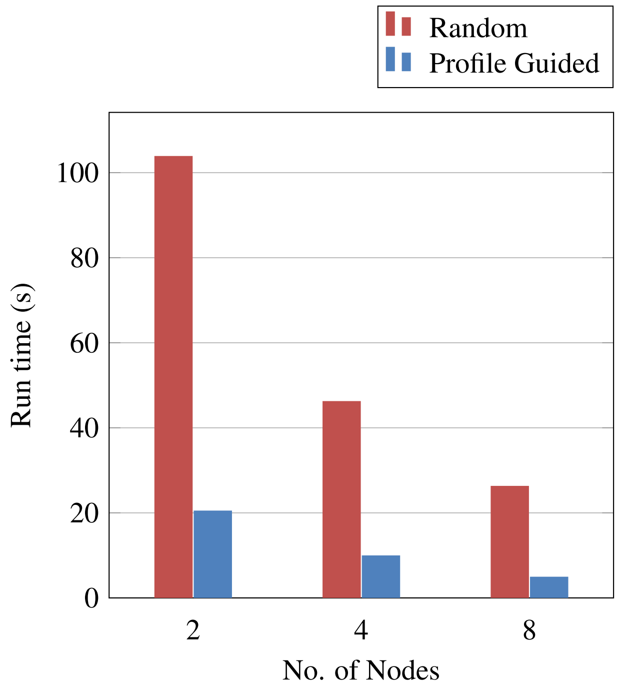

Figure 6: Run time of RAID simulation.

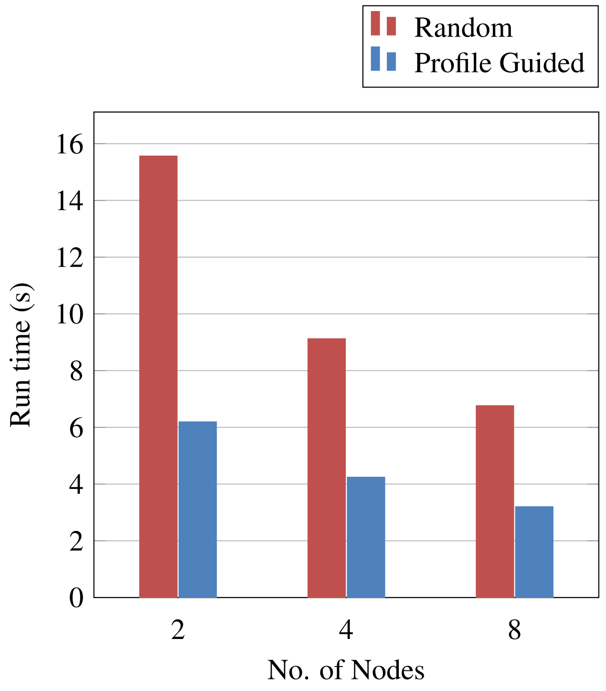

Figure 7: Run time of ISCAS'89 simulation using the s5378 circuit.

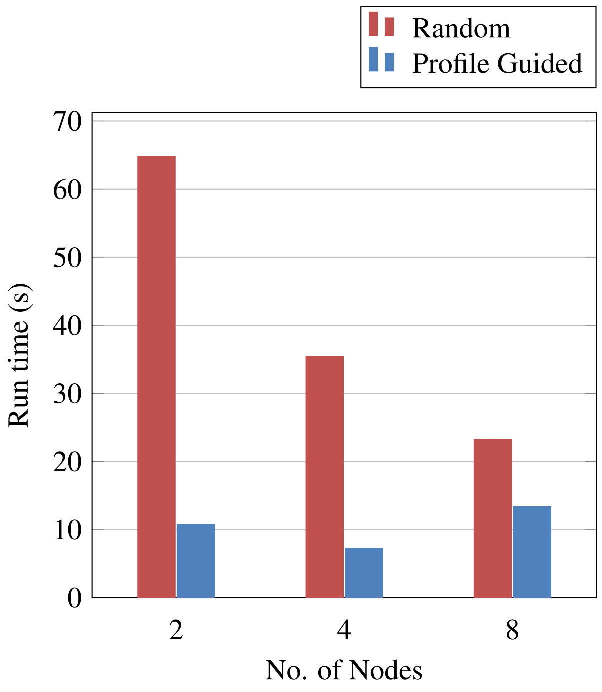

Figure 8: Run time of ISCAS'89 simulation using the s9234 circuit.

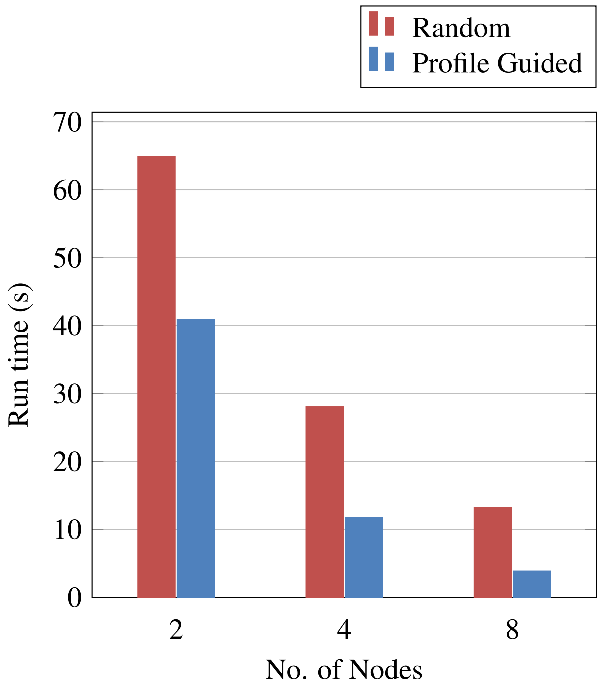

Figure 9: Run time of ISCAS'89 simulation using the s38584.1 circuit.

Table 2: Simulation Run Time (in seconds) and Speedup for the different partitioning algorithms.

Model	No. of Nodes	Random	Profile Guided	Speedup
ISCAS89: s5378	2	15.57	6.19	2.51×
	4	9.12	4.25	2.15×
	8	6.77	3.20	2.11×
ISCAS89: s9234	2	64.76	10.73	6.04×
	4	35.39	7.23	4.89×
	8	23.23	13.37	1.74×
ISCAS89: s38584.1	2	64.92	40.92	1.59×
	4	28.05	11.76	2.39×
	8	13.25	3.87	3.42×
RAID	2	103.79	20.41	5.09×
	4	46.16	9.78	4.72×
	8	26.22	4.86	5.40×

as the number of nodes increases. The speedup decreases sharply with the number of nodes for the s9234 circuit, and increases with the number of nodes for the s38584.1 circuit.

As can be seen in Figure 4, the bulk of the messages sent during the simulation are contained in a subset of the objects. For small numbers of partitions, it is possible to select partitions in such a way that all of the high traffic objects are contained in the same partition. As the number of partitions increases, it becomes necessary to split these high traffic objects across partitions, which may explain the narrowing performance gap between Profile Guided Partitioning and the Random Partitioning.

6 CONCLUSIONS

In this paper, we presented a method of performing partitioning on arbitrary Discrete Event Simulation models. We demonstrated that the event characteristics of a model in a sequential environment are capable of directing a partitioning strategy for that model in a parallel simulation. We found that this algorithm can result in up to a 6× speedup over naive partitioning. The speedup obtained varies between models, indicating that the model topology can have a significant effect on the performance of partitioning. Future work could attempt to identify model topologies that are most amenable to Profile Guided Partitioning, and perhaps perform graph minimum cuts that take additional behavior other than message quantity into account.

ACKNOWLEDGMENTS

Support for this work was provided in part by the National Science Foundation under grant CNS–0915337.

REFERENCES

Bahulkar, K. et al. 2012. “Partitioning on Dynamic Behavior for Parallel Discrete Event Simulation”. In *Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, PADS ’12, 221–230.

Carothers, C. D. et al. 1994, July. “Effect of Communication overheads on Time Warp Performance: An Experimental Study”. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation (PADS 94)*, 118–125. Society for Computer Simulation.

Çatalyürek, Ü., and C. Aykanat. 2011. “PaToH (Partitioning Tool for Hypergraphs)”. In *Encyclopedia of Parallel Computing*, 1479–1487. Springer.

Chandy, K. M., and R. Sherman. 1989. "Space-Time and Simulation". In *Distributed Simulation*, 53–57. Society for Computer Simulation.

Chevalier, C., and F. Pellegrini. 2008. "PT-Scotch: A tool for efficient parallel graph ordering". *Parallel Computing* 34 (6): 318–331.

Fujimoto, R. 1990, October. "Parallel Discrete Event Simulation". *Communications of the ACM* 33 (10): 30–53.

Guo, S., and X. Hu. 2011. "Profile-based spatial partitioning for parallel simulation of large-scale wildfires". *Simulation Modelling Practice and Theory* 19 (10): 2206–2225.

Hendrickson, B., and R. Leland. 1995. "The Chaco users guide: Version 2.0". Technical report, Technical Report SAND95-2344, Sandia National Laboratories.

Jefferson, D. 1985, July. "Virtual Time". *ACM Transactions on Programming Languages and Systems* 7 (3): 405–425.

Karypis, G., and V. Kumar. 1998. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". *SIAM Journal on Scientific Computing* 20 (1): 359–34.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling and Analysis*. 3rd ed. McGraw-Hill.

Li, L., and C. Tropper. 2009, April. "A Multiway Design-driven Partitioning Algorithm for Distributed Verilog Simulation". *Simulation* 85 (4): 257–270.

Martin, D. E. et al. 1996, January. "WARPED: A Time Warp Simulation Kernel for Analysis and Application Development". In *29th Hawaii International Conference on System Sciences (HICSS-29)*, edited by H. El-Rewini and B. D. Shriver, Volume Volume I, 383–386.

Peschlow, P. et al. 2007, June. "A Flexible Dynamic Partitioning Algorithm for Optimistic Distributed Simulation". In *Principles of Advanced and Distributed Simulation, 2007. PADS '07. 21st International Workshop on*, 219–228.

Radhakrishnan, R. et al. 1998, December. "An Object-Oriented Time Warp Simulation Kernel". In *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, edited by D. Caromel, R. R. Oldehoeft, and M. Tholburn, Volume LNCS 1505, 13–23. Springer-Verlag.

Rashti, M., and A. Afsahi. 2007, March. "10-Gigabit iWARP Ethernet: Comparative Performance Analysis with InfiniBand and Myrinet-10G". In *Parallel and Distributed Processing Symposium*, 1–8.

Subramanian, S. et al. 2001, March. "Applying Multilevel Partitioning to Parallel Logic Simulation". *Parallel and Distributed Computing Practices* 4 (1): 37–59.

Subramoni, H. et al. 2009, August. "RDMA over Ethernet — A Preliminary Study". In *Cluster Computing and Workshops*, 1–9.

AUTHOR BIOGRAPHIES

AJ ALT is a graduate student in the Department of Electrical Engineering and Computing Systems at the University of Cincinnati. His email address is altaj@mail.uc.edu.

PHILIP A. WILSEY is a professor in the Department of Electrical Engineering and Computing Systems at the University of Cincinnati. His is an experimentalist working in parallel and distributed systems, embedded system, and point-of-care medical devices. He is currently studying the challenges of parallelism in multi-core and many-core platforms and is studying the optimization of Beowulf clusters composed of multi-/many-core processors to support efficient parallel execution of fine grained applications. His email address is wilseypa@gmail.com and his web pages are http://secs.ceas.uc.edu/~paw and http://github.com/wilseypa.