Study of a Multilevel Approach to Partitioning for Parallel Logic Simulation*

Swaminathan Subramanian, Dhananjai M. Rao, and *Philip A. Wilsey*
Experimental Computing Laboratory, Cincinnati, OH 45221–0030

Abstract

Parallel simulation techniques are often employed to meet the computational requirements of large hardware simulations in order to reduce simulation time. In addition, partitioning for parallel simulations has been shown to be vital for achieving higher simulation throughput. This paper presents the results of our partitioning studies conducted on an optimistic parallel logic simulation framework based on the Time Warp synchronization protocol. The paper also presents the design and implementation of a new partitioning algorithm based on a multilevel heuristic, developed as a part of this study. The multilevel algorithm attempts to balance load, maximize concurrency, and reduce inter-processor communication in three phases to improve performance. The experimental results obtained from our benchmarks indicate that the multilevel algorithm yields better partitions than other partitioning algorithms included in the study.

1 Introduction

Parallel simulation tools are frequently used to simulate large and complex digital circuits in order to reduce time for simulation [3]. To extract better performance from parallel logic simulators, partitioning techniques are necessary [5, 19, 20]. The partitioning techniques can exploit either the parallelism inherent in (i) the simulation algorithm, or (ii) the circuit being simulated. The amount of parallelism that can be gained from the former method is limited by the algorithm used for simulation. The latter method attempts to improve performance by dividing the circuit to be simulated across processors. Hence, during simulation, the workload is distributed and the concurrency and parallelism in the circuit are exploited. Its success is bounded by the amount of parallelism inherent in the circuit and the number of processors available for simulation. The partitioning algorithms, discussed in this paper, concentrate on achieving speedup by improving concurrency, minimizing inter-processor communication, and balancing the processor workload based on the circuit being simulated [1]. The new multilevel approach to partitioning attempts to optimize the aforementioned factors by decoupling them into separate phases. The multilevel algorithm for partitioning has been studied and analyzed in [8, 12] and has been shown to produce high quality partitions (measured with respect to edges cut, *i.e.,* the number of edges that cross partition boundaries) over several partitioning algorithms such as the inertial and the spectral bisection algorithms. The complexity of the multilevel algorithm is $\mathcal{O}(\mathcal{N}_{\mathcal{E}})$, where N_E represents the number of edges in the circuit graph making the multilevel partitioning technique a fast linear time heuristic.

The remainder of the paper discusses the partitioning studies that were conducted to improve the performance of a parallel VHDL simulation framework, developed as a part of the SAVANT [21] project. The partitioning techniques were seamlessly integrated into the simulation framework in order to ease their study and use. A brief summary of the different partitioning techniques developed previously is presented in Section 2. Section 3 illustrates the multilevel partitioning algorithm developed as a part of this study. A brief description of the experimental framework along with the issues involved in the design and integration of the partitioning algorithms are presented in Section 4. The results of the experiments conducted using the framework are presented in Section 5. Section 6 presents the conclusions drawn from the study along with pointers to future work.

2 Related Work

Several techniques have been developed to partition logic circuits for parallel simulation. The algorithms address various issues related to concurrency, communication and load balancing. Prathima [1] presents a technique for partitioning using element strings. The algorithm assigns chains of logic gates to each partition, to encourage concurrency, and maintains gates with different delays on a fanout together, to minimize communication. Patil *et al* [17] employed several heuristics, such as the greedy and simulated annealing techniques, for partitioning, based on a cost function to estimate the execution time for parallel simu-

*Support for this work was provided in part by the Defense Advanced Research Projects Agency under contract DABT63–96–C–0055.

lation, given the processor assignment and the underlying architecture. The two phase corolla approach to partitioning for Time Warp simulation was studied by Sporrer *et al* [20]. In this approach, a fine-grained clustering step initially identifies strongly connected regions which is followed by a coarse-grained step forming partitions. A concurrency preserving partitioning (CPP) algorithm, that employed instantaneous workload for load balancing, was presented by Hong [14]. Bagrodia *et al* [2] have illustrated the use of an acyclic multi-way partitioning scheme for gate level simulations.

A partitioning scheme based on fanout/fanin cone clustering starting from the input gates was studied by Smith [19]. A random partitioning scheme that assigns nodes to partitions in a random and load balanced manner was reported in [15]. A major bottleneck for the random partitioner is communication. A Depth-First traversal of the circuit graph can also be utilized for partitioning by assigning nodes to partitions in the order traversed [11]. Cloutier [5] and Smith [19] utilize a topological (or level) algorithm for partitioning. This technique proceeds by first levelizing the circuit graph and then assigning nodes at the same topological level to a partition. Detailed analyses of this partitioning algorithm for Time Warp based simulations is available in literature [5, 19]. The fanout/fanin cone clustering, random partitioning, the depth first search partitioner and the topological partitioner have been included in this study.

3 The Multilevel Approach

The partitioning algorithms included in this study use a directed graph representation of the input circuit. The circuit graph is represented as a directed graph $G = (V, E)$ where V forms the vertex set and E the edge set of the graph. The vertices denote logic gates and edges represent *signals* [9] that interconnect these logic gates. An ideal partitioning of a circuit graph implies that the load is perfectly balanced and an equal number of gates are active in each partition at any simulation instance. A particular partitioning algorithm cannot provide ideal partitioning for all circuit graphs because each circuit has its own structure and pattern of communication. The multilevel algorithm attempts to satisfy such constraints by separating out these concerns in three phases; namely (i) the coarsening phase; (ii) the initial partitioning phase; and (iii) the refinement phase. Unlike other algorithms, the strength of the multilevel approach stems from the fact that it allows refining the circuit graph at several intermediate levels instead of at only the original circuit graph level. A detailed description of each of phase is presented in the following paragraphs.

Coarsening: The various stages involved in the coarsening phase are shown in Figure 1. The coarsening phase proceeds from the primary input nodes in the graph. The graph of the initial set of processes (gates or nodes) constituting the circuit to be simulated is represented by G_0. The coarsening phase produces a hierarchical sequence of smaller graphs, say $G_1, G_2, ..., G_m$ from the original graph G_0. Each vertex (also called globule) in a lower level graph, say G_1, represents a set of connected vertices in its immediate higher level graph G_0. Each stage in this phase coarsens (or subsumes) a set of inter-connected vertices to yield a single vertex in the next stage. Emphasis on concurrency is stressed in this phase. In essence, distributing the objects in a concurrent manner reduces the number of *rollbacks* [7], that occur during optimistic simulations, and improves performance.

Coarsening of the graphs can be achieved by using different schemes based on various parameters. Coarsening (irrespective of the scheme used) produces a sequence of smaller graphs derived from the original graph G_0. Any scheme simply defines the manner in which coarsening proceeds. As shown in Figure 1, coarsening starts from input vertices and combines a set of vertices from a higher level to form new vertices or globules in the next lower level. At each level, a vertex is allowed to be coarsened only once and vertices that contain a primary input vertex, are not allowed to be combined together. The restriction is placed in order to maintain concurrency. The coarsening procedure halts when the number of globules fall below a threshold or if all the globules are input globules preventing further combination. The resulting graphs from coarsening satisfy the following relation. Given G_0 is the original graph, $G_i = (V_i, E_i)$, and $G_{i+1} = (V_{i+1}, E_{i+1})$ are graphs at levels i and $(i+1)$, then, $V_{i+1} = \{V_{(i+1),0}, V_{(i+1),1}, ..., V_{(i+1),n}\}$. Each $V_{(i+1),k} \subset V_i$ and $V_{(i+1),k} \cap V_{(i+1),l} = \phi$. The edge set of a vertex at level $(i+1)$ then becomes the union of the edges of the vertices at level i from which it was originally composed. In the current implementation a fanout coarsening scheme was employed in order to improve concurrency of parallel simulations. Graphs with a number of vertices on their signals/edges tend to increase the number of *rollbacks* [7] in optimistic simulations if vertices on interconnecting signals are split across partitions. Fanout coarsening avoids this by maintaining vertices on a signal together in a partition; this reduces communication across partitions. In this technique, coarsening begins from primary input vertices and proceeds in a depth-first manner. When a vertex is chosen for coarsening in this scheme, it is combined with all other vertices on its fanout. A vertex could be connected to several signals, however, only one of them is considered for coarsening. At each level, other than the first, coarsening starts from vertices that were just added to a globule in the previous level, thereby increasing concurrency with linear chains.

Initial Partitioning: The initial partitioning phase forms

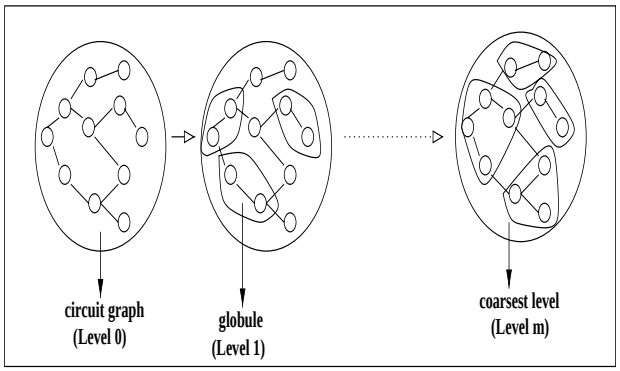

Figure 1. Coarsening Procedure

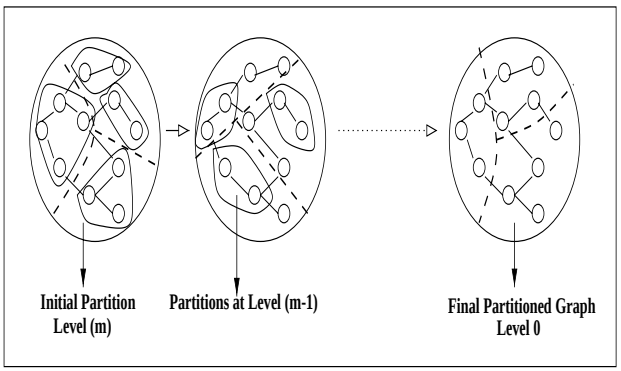

Figure 2. Refining Procedure

the second stage of the multilevel partitioning approach. Initial partitioning at the coarsest level provides a "k-way" partitioning of the original graph. The value of k is determined by the number of partitions desired. This phase attempts to load balance by distributing equal number of vertices across partitions while preserving concurrency. This phase is responsible for assigning vertices to partitions. The partitions generated in this phase are further refined in the next phase of the multilevel algorithm. The initial partition before refinement is shown in Figure 2 using dotted lines.

Initially, all the input globules in the coarsest level are split equally across the partitions such that the load is sufficiently balanced. Any remaining globules are assigned to partitions in a random manner, maintaining load balance. Let the lowest level be m, V_{ij} be the j^{th} vertex at level i $(0 \leq i \leq m)$, and $P[V_{ij}]$ be partition to which vertex V_{ij} was assigned. It can be shown that $\forall_{(v \in V_{ij})} P[v] = P[V_{ij}]$, where v corresponds to a vertex from level $(i-1)$ that has been combined with other vertices from level $(i-1)$ to form V_{ij} at level i.

Refinement: The coarsening and the initial partitioning phases concentrate on improving concurrency and load balance. The refinement phase, attempts to reduce communication by placing together strongly connected globules in a partition. Starting from the lowest coarse-grained level, this phase tries to load balance and reduce communication through "k-way" refinement at each intermediate level. The "k-way" partition of the original graph generated by the initial partitioning phase is utilized. The greedy algorithm for local refinement was used [12]. The greedy algorithm converges in a few iterations reducing the time needed for partitioning. The greedy technique has also been shown to yield better partitions [12] with reduced edge-cut compared to other refinement algorithms (*e.g.,* Kernighan-Lin [13] and Fiduccia-Mattheyses [6]).

This phase starts from the lowest level of the hierarchical sequence of graphs, namely level m. The cut-set, that represents the number of edges that cross over partitions, is used as the parameter to be minimized by the greedy algorithm. The greedy refinement algorithm selects a vertex at random and computes the gain in the cut-set (reduction in edge-cut) for every partition that the vertex can be moved to. The partition with maximum gain is then selected for the move. A move is feasible if it reduces the cut-set and preserves load balance. Reducing the cut-set in turn reduces communication overheads. Once a vertex is selected for a move, it is "locked", preventing its move until an iteration of the greedy algorithm finishes. The greedy algorithm was found to converge in a few iterations. As illustrated in Figure 2, the graphs are recursively projected to the next higher level, preserving the partitioning information, while refining across levels.

4 Experimental Framework

The simulation framework used for the partitioning studies (illustrated in Figure 3) consists of three primary components [21]: (i) SAVANT, (ii) TyVIS and (iii) WARPED. The primary input to the framework is the description of the hardware component in VHDL [9]. The input VHDL is analyzed into an Internal Intermediate Representation (IIR) called the Advanced Intermediate Representation with Extensibility [22] (AIRE) using `scram`; the VHDL parser and code-generator developed as a part of the SAVANT project. The intermediate form is used to generate C++ code compliant with the TyVIS interface. The generated code is complied along with TyVIS libraries to obtain the final simulation executable.

TyVIS is a VHDL kernel that provides necessary runtime support for simulation of VHDL designs. It provides the basic data structures and methods necessary to interface the generated C++ code from `scram` with the WARPED [18] parallel simulation kernel. WARPED is an optimistic parallel discrete event simulator developed at the University of Cincinnati. It uses the Time Warp mechanism [10] for distributed synchronization. In WARPED, the logical processes

Circuit	Inputs	Gates	Outputs
s5378	35	2779	49
s9234	36	5597	39
s15850	77	10383	150

Table 1. Characteristics of benchmarks

(LPs) that represent the physical processes being modeled are placed into groups called "clusters" that represent operating system level parallel processes. LPs within a cluster operate as classical Time Warp processes. Further details on the working of TyVIS and WARPED are available in the literature [21].

The necessary infrastructure to enable partitioning of the VHDL processes was integrated with the TyVIS kernel. A set of six different partitioning strategies, including the multilevel strategy, were incorporated. Since the framework employs a runtime elaboration technique [21], partitioning occurs at runtime, after the simulation is instantiated and initialized. The runtime support functions generated by `scram` are used by the partitioning algorithms to build the necessary data structures. The runtime partitioning technique provides the flexibility to choose from different partitioning algorithms without necessitating re-compilation of the system. The design also provides a simple technique to choose the number of partitions. Also, the object oriented techniques employed provide an efficient interface that can be used to integrate other partitioning algorithms.

5 Experiments

Three of the ISCAS '89 benchmarks [4] were used to evaluate the performance of the partitioning algorithms. The characteristics of the benchmarks used in the experiments are shown in Table 1. All the partitioning algorithms failed to provide speedup for benchmarks with less than 2500 gates, since such models were small enough for the sequential simulator to outperform the parallel version. The parallel simulation experiments were conducted on eight workstations inter-connected by fast ethernet. Each workstation consisted of dual Pentium II processors with 128 MB of RAM running Linux 2.2.12. The experiments were repeated five times and the average was used as the representative value in all the characteristics. The partitioning techniques used in the experiments included the multilevel methodology and the following five algorithms (i) Random, (ii) Topological, (iii) Depth First, (iv) Cluster (Breadth First) and (v) Fanout cone.

The simulation times for the s9234 benchmark are shown in Figure 4. It is observed that the multilevel algorithm outperforms all other partitioning algorithms when more than 4 nodes are involved in the simulation. The performance of the Cluster and DFS algorithms deteriorates with increase in number of nodes due to lack of concurrency. The lack of concurrency also increases the number of rollbacks in the simulations. The performance of the Topological algorithm is limited due to increased communication overheads; more signals are split across partitions for concurrency. The simulation execution times for all the benchmarks have been tabulated in Table 2. As illustrated in the table, the multilevel strategy performs better than other strategies when the number of processors employed, lie between 8 (4 workstations) and 16 (8 workstations). When 4 processors were employed to simulate the s15850 model, the simulations ran out of memory and hence the results are not presented in Table 2.

The messaging characteristics for the simulation experiments is presented in Figure 5. As shown in the figure, the multilevel algorithm reduces the amount of communication in the 8 to 16 processor region. The Cone partitioner performed well due to lower communication and better concurrency features. Increased communication overheads due to greater edge cut in the case of the Topological partitioner resulted in increased execution times. The Cluster and the DFS partitioning did not perform well in the 16 node case due to similar reasons.

The rollback characteristics of the s9234 model is shown in Figure 6. As illustrated by the bar chart, the multilevel algorithm greatly reduces the number of rollbacks during simulation; highlighting the equilibrium achieved between concurrency and communication. The sudden dips in the execution time graph (Figure 4) are caused by a lower number of rollbacks and lower communication for most of the partitioning algorithms. The Cluster, DFS and the Topological algorithms suffered from a number of rollbacks with more communication degrading their performance in the 16 processor case.

6 Conclusions

In this paper, we presented the partitioning studies that were conducted on a parallel VHDL simulation framework (SAVANT/TyVIS/WARPED). A new partitioning technique based on the multilevel heuristic was developed and its performance relative to existing partitioning strategies was investigated. In addition, the design and the integration of the various partitioning strategies into the simulation framework was also described. Results from the experimental analysis indicate that the multilevel technique yielded better partitions than other partitioning strategies. Parallel simulation (of all the sample applications) on 16 processors using the multilevel technique executed in less than half the time taken by a sequential simulation of the same application(s). This speedup can be attributed to the reduction in both the number of rollbacks and the amount of inter-

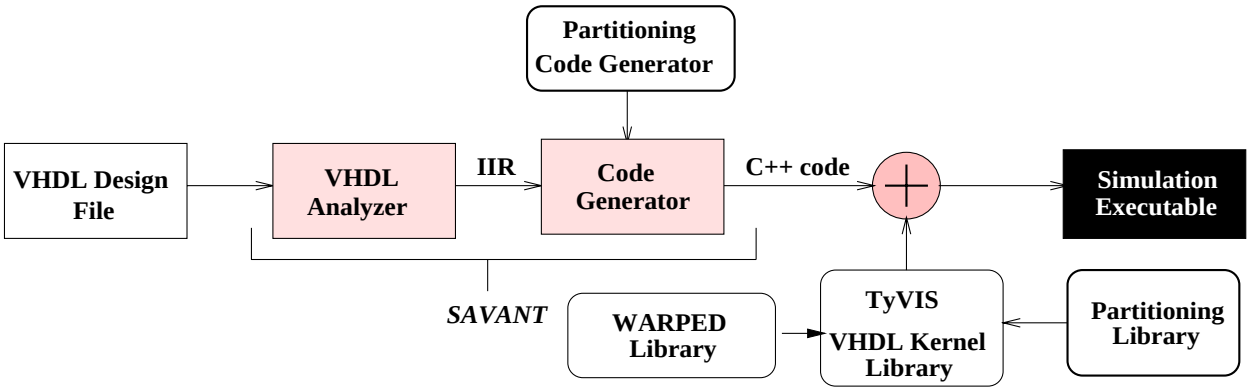

Figure 3. The Simulation Framework

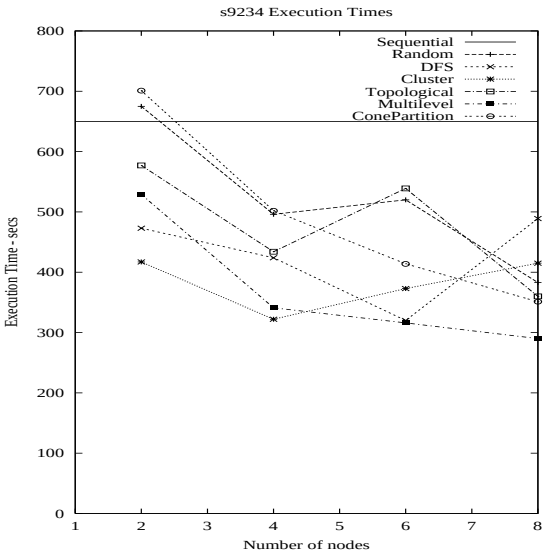

Figure 4. Execution times of s9234

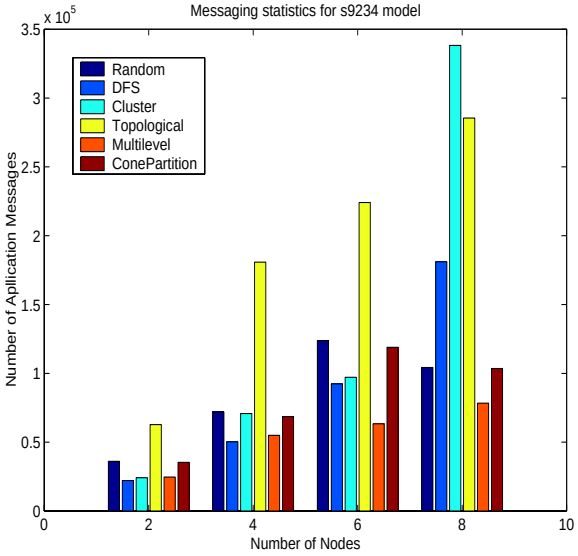

Figure 5. Messaging characteristics of s9234

Circuit	Seq Time	No. of Nodes	Random	DFS	Cluster	Topological	Multilevel	Cone
s5378	149.96	2	166.44	118.72	97.45	128.63	91.66	166.54
		4	116.11	84.80	83.28	331.45	84.07	113.11
		6	131.95	76.12	96.86	194.34	63.61	96.07
		8	101.89	81.09	78.62	152.91	52.94	76.56
s9234	651.24	2	675.07	473.90	417.63	577.14	529.39	701.10
		4	496.30	424.41	322.02	434.85	341.84	502.60
		6	520.80	320.98	373.41	539.59	316.96	414.65
		8	383.32	489.97	415.02	360.90	290.31	351.35
s15850	2154.21	4	2090.82	1279.19	1317.28	2272.62	1043.43	1832.24
		6	1434.79	906.08	1351.17	1439.99	943.91	1363.40
		8	1407.33	947.64	1215.64	2735.07	864.03	1176.36

Table 2. Simulation Time (in secs) for the different partitioning algorithm

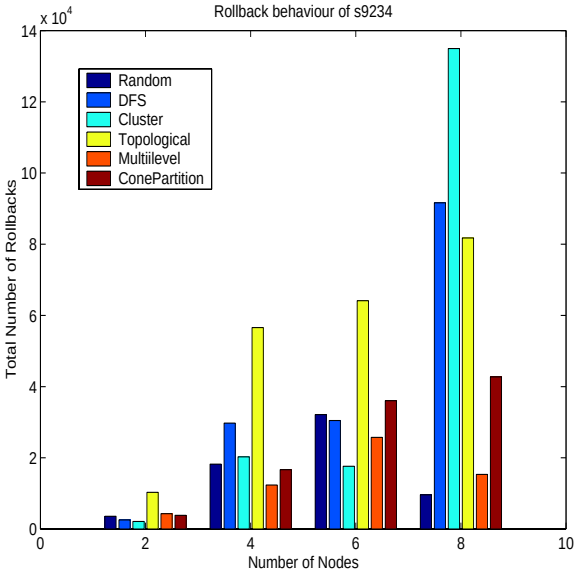

Figure 6. Rollback characteristics of s9234

processor communication. Since the multilevel technique is a linear time heuristic, it can be easily scaled to partition for a large number of processors. Research is currently ongoing to incorporate several enhancements to the multilevel heuristic. For example, we are currently investigating the use of activity levels of communication to make better decisions while coarsening. In addition, different schemes for coarsening and refinement are also being studied.

References

[1] P. Agrawal. Concurrency and communication in hardware simulators. *IEEE Transactions on Computer-Aided Design*, Oct. 1986.

[2] R. L. Bagrodia and W. Liao. Maisie: A language for the design of efficient discrete-event simulations. *IEEE Transactions on Software Engineering*, 20(4):225–238, Apr. 1994.

[3] M. L. Bailey, J. V. Briner, Jr., and R. D. Chamberlain. Parallel logic simulation of VLSI systems. *ACM Computing Surveys*, 26(3):255–294, Sept. 1994.

[4] CAD Benchmarking Lab , NCSU. *ISCAS'89 Benchmark Information*. (available at `http://www.cbl.ncsu.edu/www/CBL_Docs/iscas89.html`).

[5] J. Cloutier, E. Cerny, and F. Guertin. Model partitioning and the performance of distributed time warp simulation of logic circuits. In *Simulation Practice and Theory*, pages 83–99, 1997.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[7] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[8] B. Hendrickson and R. Leland. A multi-level algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, Dec. 1995.

[9] *IEEE Standard VHDL Language Reference Manual*. New York, NY, 1993.

[10] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):405–425, July 1985.

[11] K. L. Kapp, T. C. Hartrum, and T. S. Wailes. An improved cost function for static partitioning of parallel circuit simulations using a conservative synchronization protocol. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 78–85, 1995.

[12] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. Technical Report TR 95-055, University of Minnesota, Computer Science Department, Minneapolis, MN 55414, Aug. 1995.

[13] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell Systems Technical Journal*, pages 291–307, Feb. 1970.

[14] H. K. Kim and J. Jean. Concurrency preserving partitioning (CPP) for parallel logic simulation. In *Proceedings of the Tenth Workshop on Parallel and Distributed Simulation*, pages 98–105, May 22–24 1996.

[15] S. A. Kravitz and B. D. Ackland. Static vs. dynamic portioning of circuits for a MOS timing simulator on a message-based multiprocessor. In *Proceedings of the SCS Multiconference on Distributed Simulation*, 1988.

[16] N. Manjikian and W. M. Loucks. High performance parallel logic simulation on a network of workstations. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pages 76–84, May 1993.

[17] S. Patil, P. Banerjee, and C. D. Polychronopoulos. Efficient circuit partitioning algorithms for parallel logic simulation. In *Proceedings, Supercomputing '89*, pages 361–370, Nov. 1989.

[18] R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. An Object-Oriented Time Warp Simulation Kernel. In D. Caromel, R. R. Oldehoeft, and M. Tholburn, editors, *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, volume LNCS 1505, pages 13–23. Springer-Verlag, Dec. 1998.

[19] S. P. Smith, B. Underwood, and M. R. Mercer. An analysis of several approaches to circuit partitioning for paral lel logic simulation. In *In Proceedings of the 1987 International Conference on Computer Design.*, pages 664–667. IEEE, NewYork, 1987.

[20] C. Sporrer and H. Bauer. Corolla partitioning for distributed logic simulation of VLSI-circuits. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pages 85–92, May 1993.

[21] K. Subramani, D. E. Martin, and P. A. Wilsey. SAVANT/TyVIS/WARPED: Components for the analysis and simulation of vhdl. *VHDL User's Group*, pages 195–201, 1998.

[22] J. C. Willis, P. A. Wilsey, G. D. Peterson, J. Hines, A. Zamfriescu, D. E. Martin, and R. N. Newshutz. Advanced intermediate representation with extensibility (AIRE). In *VHDL Users' Group Fall 1996 Conference*, pages 33–40, Oct. 1996.