An Asynchronous Distributed Discrete Event Simulation Algorithm for Cyclic Circuits using a Data-Flow Network

Sumit Ghosh
LEMS, Division of Engineering
Brown University
Providence, R.I. 02912

Erik Debenedictis
nCUBE Corporation
Belmont, CA 94002

Meng-Lin Yu, AT&T Bell Laboratories, Holmdel, NJ 07733

Abstract

The discipline of discrete event simulation may be applied to many physical systems such as digital hardware, queueing networks, telephone networks, simulated warfare, and banking transactions. Where the processes of a physical system interact asynchronously, an asynchronous distributed event-driven simulation algorithm may enable the simulation to execute on a parallel processor. This has the potential to significantly reduce the total simulation time. Until now none of the algorithms reported in the literature could offer a solution with the following characteristics - acceptable performance, freedom from deadlock, and provably correct, for circuits where the process interactions form a cyclic graph. Directed cyclic graphs appear frequently and may constitute a significant part of a physical system. For example, oscillators and industrial control mechanisms with positive or negative feedback result in cyclic graphs. This paper proposes a mathematically-proven algorithm, referred to as YADDES, for asynchronous, distributed, discrete-event simulation of circuits containing feedback loops. In this approach, every component of the circuit is represented through an executable model and the flow of information is expressed through message communication. There is no notion of global simulation time. Instead, simulation time is encapsulated in the model description and encoded in the messages. In addition, a data-flow network is synthesized based on the connectivity of the components in the circuit that computes a quantity "time of next event" for every component. This quantity permits the corresponding model to execute asynchronously as far ahead in simulation time as possible yet guaranteeing correctness. Thus, the network assures that any simulation process executing on a distributed processing environment that has sufficient information to simulate can execute while also avoiding deadlock. The algorithm has been verified through an implementation on the loosely-coupled parallel processor ARMSTRONG at Brown University.

1. A New Approach to Asynchronous Distributed Discrete-Event Simulation

In the YADDES approach (an acronym for yet another asynchronous distributed discrete-event simulation algorithm), for a given system such as a digital design only those subcircuits that constitute cyclic directed graphs are identified and preprocessed and the entities of such subcircuits are simulated as described in this section. Other entities of the system that constitute acyclic graphs are simulated as described in [3]. The approach is described for the discipline of digital hardware and applies equally to queueing networks and banking transactions.

1.1 An Intuitive Overview

The principal cause of deadlock in the traditional asynchronous distributed discrete-event simulation system is the presence of feedback loops. That is, the simulation environment represented through models connected by feedback loops is unable to accurately decide the precise execution of events. The YADDES approach proposes the synthesis of an acyclic circuit of pseudo components based on the original simulation circuit whose purpose is to enable the execution of the circuit in a deadlock free environment.

To preserve the asynchronous and concurrent nature of the algorithm, each pseudo component represents a decision-making entity whose sole function is to determine when the corresponding simulation model may correctly execute an input event. An event refers to a signal transition at an input port. YADDES requires that each pseudo component compute a quantity "time of next event" (W') at its output port through the application of a minimum operator over the W' values at its input ports and the simulation time of the event of the corresponding simulation model. Thus, the pseudo component must necessarily access the simulation time of the event from the related simulation model. This quantity reflects a measure of the time at which the next event is expected at that path. Furthermore, it may be utilized in deciding whether a model may safely execute an event. The use of the minimum function signifies the conservative nature of the YADDES algorithm.

Corresponding to each of those inputs of the acyclic circuit that represent the primary inputs, the W' value is defined equal to the assertion time of the most recent transition. The remaining inputs of the acyclic circuit are unconnected signifying that they are not influenced by any events in the circuit. Their W' values are assumed to be permanently held at a very large number expressed through the symbol ∞ so that they may not influence the W' computations of the pseudo components.

A significant limitation of the synthesized acyclic circuit includes the lack of connectivity between the pseudo components of the respective feedback loops that may be otherwise dictated in the simulation circuit. This is manifested as follows. For a given feedback loop, the W' value at the output of the leftmost pseudo component does not reflect the simulation times of the events associated with other simulation models in the same loop nor those of other models that may influence the computation. As a result, the computed W' value may be inaccurate. In fact, it is probably optimistic for the following reason. Given the use of the minimum operator, the consideration of the W' values associated with other models would only imply a lower value in the computation of the W'

ISSN# 0-7803-0233-8/91 $1.00©1991 IEEE

value for a pseudo component. To address this limitation, a second identical copy of the acyclic circuit is synthesized. To distinguish between them, the first and second acyclic circuits are referred to as primed and unprimed respectively and the quantity "time of next event" is expressed through W for the unprimed circuit. All outputs of the primed circuit are connected to each input of the unprimed circuit through a minimum operator. This interconnection network is encapsulated in the concept of a crossbar switch that, in essence, expresses the dependency between the feedback loops. Where the activities of a feedback loop may not affect those of another loop, the corresponding link in the switch is considered non-existent; otherwise, a link exists. An existent link has a weight associated with itself that is equal to the computed propagation delay from the output of the pseudo primed component X' to the input of the pseudo unprimed component Y. Although the maximum capacity of the switch is N$\times$N, the actual size is defined by the circuit in question. The role of the outputs of the unprimed circuit is discussed later in section 6. The W values computed by the pseudo components of the unprimed circuit correctly include the simulation times of all appropriate events in the entire circuit. These values may be used in accurately determining when an event may be executed by a model. The primed and unprimed circuits and the switch are collectively termed "data-flow network" for the circuit in question.

In essence, the optimistic nature of the evaluation process in the primed circuit acts as a window into future events. These future events are presented to the unprimed circuit for consideration and the conservative characteristic of the unprimed circuit guarantees accuracy of the simulation process.

2. Implementation Issues

The implementation of the YADDES algorithm is complex and is described as follows. Given any complex circuit and a user specified partition, the total number of processors required for simulation equals N+2 where N is the number of partitions. While the components of every partition execute on a processor, the primary inputs of the simulation circuit and the outputs of the data-flow network are modeled as entities P0 and P1 respectively and are executed on unique processors. The entity P1 signifies the rightmost boundary of the data-flow network and participates in the propagation of acknowledgements. In the event that a circuit contains feedback loops, the user specified feedback arc set is accepted by a preprocessor that generates the data-flow network. Corresponding to every component of the simulation circuit, the final implementation consists of three entities - a simulation model that represents the functionality of the component and the primed and unprimed pseudo components. These are expressed through the C-functions "sim-component", "ppc-component", and "puc-component" respectively. Although they are conceptually concurrent entities, in the current implementation on ARMSTRONG, they are executed round-robin on a processor. When a partition includes multiple models, an interconnection between two or more models on the same processor is expressed through a data structure. When the models are located on separate processors, an interprocessor protocol represents the connection.

A significant part of the implementation consists of a kernel C description (approximately 2500 lines) that executes on every processor except those that execute the entities P0 and P1. Each processor accepts an unique input file that represents information on the models and pseudo components and their interconnection for the corresponding partition. The input files for the partitions are generated by a preprocessor that accepts a description of the circuit in a hardware description language ESL [12] and the user specified partitions and feedback arc set.

The flow of control during the execution of the algorithm may be described as follows. The simulation models i.e., the sim-component functions, corresponding to those components that receive signal transitions from the external world at their primary input ports are executed first. A sim-component, in turn, initiates the executions of the puc-component and ppc-component functions and suspends itself. When the executions of puc-component and ppc-component are complete, the sim-component is reactivated. The execution of a puc-component (or ppc-component) is considered complete when either the W (or W') value at the output is unchanged or an acknowledgement is received signifying that the change in the output W (or W') value has been propagated throughout the data-flow network. Additionally, the puc-component and ppc-component functions may be initiated for execution when a new W (or W') value is received at any of its input ports from the left. Thus, as the execution of the algorithm continues, the thread of control shifts from one entity to another. Eventually, the simulation process terminates when all events have been executed i.e., all externally supplied (usable) transitions at the primary input ports have been utilized to generate output transitions.

3. Conceptual Comparison with Previous Approaches to Asynchronous Discrete-Event Simulation

This section presents a conceptual understanding of the algorithm presented in this paper and its fundamental differences with the algorithms proposed in [5] and [3].

In the deadlock recovery algorithm [5], a simulation model does not propagate any output signal information to other models connected to its output port when its value as a consequence of execution remains unchanged. As a result, other models whose execution depends on the output value of this model may not execute and constitute a deadlock. When such a deadlock occurs across the entire system, a distributed deadlock detection mechanism detects the situation and a central entity synchronously accesses the U and W values [5] of every model, computes their minimum, and permits execution of all models up to the minimum value. In the YADDES approach, the effect of any change in W or W' must ripple through the data-flow network as far as the effect may propagate. The crossbar switch implies transitive closure over all models in a feedback loop and guarantees that the U and W values of every simulation model that may possibly be affected by a change in the W or W' value of a model shall be updated. In addition, the minimum operator used in the computation of every W', W, and K value ensure their correctness in the presence of multiple changes. The crossbar switch is equivalent to the traversal of all relevant loops in a design and computing the minimum over all relevant U and W values.

In the algorithm proposed in [3], messages are sent with incrementally increased time values even when the logical values at the outputs are unchanged. Consequently, the simulation time up to which every model is simulated is advanced continuously. The mechanism is motivated by a model's inability to view the global picture such as an unchanged external input signal. A consequent limitation is the potentially large number of messages in the system for scenarios where external input signal is unchanged for extended periods of time relative to the cumulative propagation delays of the models in the feedback loop. In YADDES, the missing picture is substituted by the primed copy of the data-flow network. It permits optimistic jumps in the values of W' assuming that future events will be unable to influence and modify them. Normally such optimism may lead to inconsistency and error, but the presence of the crossbar switch and minimum operator ensures the correct advancement of the W value.

4 Conclusion

The issue of asynchronous distributed discrete event simulation of cyclic circuits is crucial to the field of computer simulation and has the potential of addressing problems in the domains of digital hardware design, queueing networks, and banking transactions. Until now, none of the algorithms reported in the literature could offer a solution with the characteristics of freedom from deadlock and acceptable performance. This paper has presented an algorithm for asynchronous distributed discrete event simulation of cyclic circuits. The YADDES approach opens up the possibility of modeling challenging problems from other disciplines such as banking, railway and mobile phone networks, sociological interactions, human decision-making process, aircraft simulation, oceanics, and weather forecasting as discrete-event systems. Some of these applications are under investigation. The algorithm has been mathematically proven correct and free from deadlock. The algorithm has been verified through an implementation on the ARMSTRONG parallel processor system at Brown University. Furthermore, the investigators are studying the use of this algorithm as a basis for developing more complex concepts such as (i) an algorithm for distributed fault simulation based on circuit partitioning, (ii) an algorithm for distributed real-time banking systems, and (iii) modeling large switching networks to investigate the role of overload conditions on network performance.

References

[1] Jaydev Misra, "Distributed Discrete-Event Simulation," Computing Surveys, Vol 18, No 1, March 1986, pp.39-65.

[2] David Jefferson, "Virtual Time," ACM Transactions on Programming Languages, Vol 7, No 3, July 1985, pp. 404-425.

[3] Sumit Ghosh and Meng-Lin Yu, "An Asynchronous Distributed Approach for the Simulation of Behavior-Level Models on Parallel Processors," Proceedings of the 1988 International Conference on Parallel Processing, August 15-19, 1988, St. Charles, Illinois.

[4] K.M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Communications of the ACM, Vol 24, No 4, April 1981, pp. 198-206.

[5] K.M. Chandy, L.M. Haas, and J. Misra, "Distributed Deadlock Detection," ACM Transactions on Computer Systems, Vol 1, No. 2, May 1983, pp. 144-156.

[6] Daniel A. Reed and Allen Malony, "Parallel discrete event simulation: The Chandy-Misra Approach", Proceedings of the SCS Multiconference on Distributed Simulation, 3-5 February 1988, San Diego, California, pp.8-13.

[7] Private communications with K.F. Wong, Department of Computer Science, Washington University, St. Louis, MI 63130, June 1988.

[8] Private communications with Boris Lubachevsky, AT&T Bell Laboratories, Murray Hill, NJ

[9] Narsingh Deo, "Graph Theory with Applications to Engineering and Computer Science," Prentice Hall Inc. 1974.

[10] J. T. Rayfield and H. F. Silverman, ""Operating System and Applications of the Armstrong Multiprocessor", IEEE Computer, Vol. 21, No. 6, June 1988, pp. 38-52.

[11] Erik DeBenedictis, "Multiprocessor Programming with Distributed Variables," Proceedings of the Conference on Hypercube Multiprocessor, Aug 1985.

[12] S. Davidson and J. Lewandowski, "ESIM/AFS - A Concurrent Architectural Level Fault Simulator," Proceedings of the International Test Conference, October 1986.