Distributed Event-Driven Simulation of VHDL-SPICE Mixed-Signal Circuits*

Dragos Lungeanu
High Level Verification Group
Synopsys, Inc.

C.J. Richard Shi
Department of Electrical Engineering
University of Washington

Abstract

This paper presents a new framework and its prototype implementation for the distributed simulation of a mixed-signal system where some parts are modeled by differential and algebraic equations (as in SPICE) and other parts are modeled by discrete events (as in VHDL or Verilog). The work is built on top of a general-purpose framework of parallel and distributed simulation that combines both conservative and optimistic synchronization methods to extract the maximum concurrency available in such mixed analog and digital systems. We demonstrate that, the maximum speedup can be achieved with digital components using optimistic scheduling and analog components using conservative scheduling. A technique to further increase the parallelism among simultaneous events without sacrificing the simulation accuracy is proposed, by using a coarser time grain for the internal step events of analog solvers. Experimental results demonstrate a 6.2X speed-up on 8 processors for a circuit described in VHDL and SPICE.

This paper details our implementation, including how to make the numerical integration used by analog electrical-level simulation event-driven and to synchronize it with digital behavioral and gate-level simulation, analog and digital conversions, and how to resolve the non-convergence arising from straightforward analog and digital simulator integration. This paper also shows, for the first time, that optimistic synchronization for distributed analog simulation is feasible and quite efficient for small-to-medium size analog blocks.

1. Introduction

This paper addresses the *distributed* simulation of a complicated system where parts of the system are *analog* (described by differential and algebraic equations (DAE) such as in SPICE [11]) and parts of the system are *digital* (described by discrete-event semantics such as VHDL [12] or Verilog), and the entire system can be described by the new IEEE standard VHDL-AMS or emerging standard Verilog-AMS. *Mixed analog and digital (mixed-signal or mixed-mode) simulation* is an important technology for full-system (chip) verification and for intellectual-property-based design. With both the high-level behavioral modeling capacity and the ability in describing arbitrarily equations for deep-submicron effects, mixed-signal simulators and languages such as VHDL-AMS and Verilog-AMS provide mechanisms not only for functional and timing verification, but also potentially for power consumption estimation, as well as yield prediction. Furthermore, since the underlying mathematical framework of mixed-signal simulation is mixed continuous-time (DAE) and discrete-event systems, mixed-signal VHDL-AMS or Verilog-AMS simulators can play an vital role in emerging mixed-technology system design where micro-mechanical or and thermal parts can be described by differential equations, and the rest of the system may be described as discrete-event systems. With the increasing needs for complete system-on-chip verification considering deep-submicron effects and the emerging core-based design methodologies where various parts of a design may be described at various abstraction levels, this work provides the first successful attempt in exploiting the characteristics of mixed-abstraction systems for achieving maximum simulation performance.

Unfortunately, little research has been conducted on distributed simulation of such mixed analog and digital systems. Previous attempts either focused on distributed simulation at the logic-level or above it [1] or at the electrical-level [2, 9]. The only effort we know of is the work of Todesco and Meng [10] under the name of *Symphony*. Symphony limits the class of systems to those that have a register on each loop in the circuit graph. Symphony is based on the *conservative* scheduling, where local time of a process can never exceed the minimum time of its input events. However, it has been observed that for digital systems, the *optimistic* scheduling exploits more parallelism than the conservative methods. Optimistic scheduling allows a process to execute ahead of its input events and in case of inconsistence, roll-back is used.

In the paper, a new framework and a prototype simulator for distributed simulation of continuous-time and discrete-event system is proposed. We, for the first time, demonstrated that maximum speedup of a mixed-signal system parallel simulation can be achieved if some parts of the systems (typically digital) are simulated optimistically and the other parts (especially analog) are simulated conservatively. We also demonstrated, for the first time, that small and medium-size analog blocks can be simulated optimistically. The basis of our work is a new synchronization protocol based on distributed event system semantics but combining both optimistic and conservative scheduling [13].

In this paper, we first provide an overview of distributed discrete-event simulation framework. Then various issues how to integrate analog and digital simulation are described. A new technique to improve the time granularity to exploit more concurrency of analog events is described. Implementation and experimental results are presented followed by conclusions.

*Sponsored by DARPA under grant No. F33615-96-1-5601.

0-7695-1200-3/01$10.00 © 2001 IEEE

2. Parallel Discrete Event Simulation (PDES)

We provide a brief overview of parallel discrete-event simulation (PDES). Excellent surveys can be found in [5, 6, 7]. The physical system under simulation is partitioned into entities that communicate via message passing. Each entity is modeled by a *logical process* (LP). The model of distributed simulation is a graph of logical processes exchanging *timestamped events* ($event@time$) over *channels*. Each LP has a *state* and a *simulate()* function. A *simulation step* of an LP calls the simulate() function with the next input event and current state, modifies the state and sends output events. The distributed simulation is correct if each LP processes its input events in chronological order of their timestamps (*local causality constraint* or *lcc*). The two major methods to ensure the *lcc* are: optimistic and conservative.

In the *conservative* methods [3], an LP **blocks** until it has a *safe* event to process. An event is safe if the LP will never receive another event with a lower timestamp. Blocking may cause *deadlock*, avoided or detected and recovered by global synchronization.

The *optimistic* methods [4] assume that all the events are safe. If there is a later event with a lower timestamp (*straggler*), it **rollbacks** by *time warping*. During the rollback the LP restores the state previous to the straggler and sends *negative events* to cancel all of the events sent during the wrong simulation. These negative events may cause rollback at their destinations. Rollback involves states and events saving and restoring, which may cause memory overflow. *Fossil collection* is the process in which unneeded memory cells are freed. Again, global synchronization is used to establish if a memory cell is old enough to be reused.

Global synchronization may be performed using *null message* or *global virtual time (gvt)* protocols. *gvt* is the smallest timestamp of an unprocessed event in the entire system. It is monotonically increasing over the simulation. A null message is an empty event with a timestamp. It is sent by an LP on the output channels to inform those LPs about the smallest timestamp of a future real event. This promise involves the knowledge of *lookahead*, a kind of delay from inputs to outputs. Lookahead is application-dependent and is sometimes zero, too expensive or impossible to compute. Without it, the null messages may not avoid deadlock.

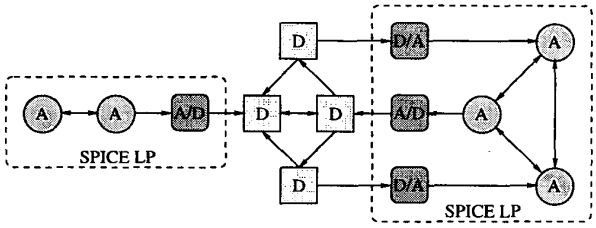

Figure 1. PDES of Mixed-Signal Circuits.

3. The SPICE Logical Process

Suppose that we want to simulate a mixed-signal circuit at the mixed-mode level. The circuit may be described in VHDL or other hardware description languages, with foreign SPICE architectures for analog blocks. The analog components are identified and isolated from the digital surrounding parts. Each analog block together with the analog to digital(A/D) and digital to analog(D/A) interfaces is mapped as a logical process (LP) in the PDES model, and simulated at the electrical level using SPICE. We call this block the Spice LP. The digital hierarchy is flattened into a graph of processes interconnected by nets, and each process is mapped as a PDES LP and simulated at the behavioral or gate level (see Figure 1).

The first problem that occurs when simulating an analog block is the communication and interface with the digital neighbors. In SPICE, the input sources must have predefined waveforms for the entire time-domain simulation. This is not possible in the PDES setup since the inputs change dynamically during the simulation according to the activity in the entire circuit. In our implementation, we model each input port as a piece-wise linear voltage source (PWL) with two time-value pairs $(t_1, v_1), (t_2, v_2)$. When a digital input event $alter@t$ arrives, the coefficients of the PWL source are altered correspondingly $((t, gnd), (t + d, vdd))$ or $((t, vdd), (t + d, gnd))$, where d is a small delay (see Figure 2). For the output ports, each waveform is monitored at each step, and if the value at current time (ct) is below a threshold VL, but the value at previous step is above VL, then a $fall@ct$ event is generated. Similarly for raising waveforms, but with a different $VH > VL$ threshold.

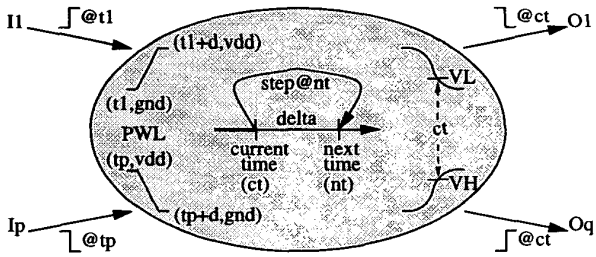

Figure 2. The SPICE Logical Process.

Any numerical application faces the problem of non-convergence. It may be caused by discontinuities or by unstable methods. A first attempt to model the input ports in the Spice LP would be as fixed voltage sources (DC). When an input digital event $alter@t$ arrives, it will change the source value to the value of the discrete event. This method will not work, because it changes too abruptly the parameters and causes non-convergence. This is the reason why the input sources are modeled as PWL, to have continuous values. However, the PWL voltage sources still have discontinuities in the first derivative, and they may still cause non-convergence. When an input source is altered by an external event $alter@t$, we insert two breakpoints at the derivative discontinuities points t and $t + d$. At each breakpoint, SPICE decreases the time-step and cuts the integration order, making the method more stable.

4. Analog - Digital Synchronization

The most important problem in mixed-mode simulation is the synchronization between the event-driven digital simulation and the numerical integration in the electrical simulation. The numerical integration computes at discrete points in time a solution to a system of differential and algebraic equations. At the current time point ct a new time-step $delta$ is computed such that the system converges at next time point $nt = ct + delta$. We need to synchronize the numerical integration time-stepping with the external digital events that alter the input sources. The idea is to transform the time-stepping engine to fit into the event-driven paradigm. The synchronization is then accomplished by the event-driven simulator (sequential or distributed), which processes the events in the chronological order of their timestamps.

The most common mixed-signal timing synchronization methods are: lock-step, digital-controlled, analog-controlled, and roll-back. The **roll-back** scheme seems to be the most promising, since analog and digital simulators use their own time steps, and the latency in each portion can be exploited. The digital and analog simulators alternate taking time steps, and when synchronization is needed, the simulator ahead in time is rolled back to the synchronization point and re-evaluated. In high feedback systems the rollback is performed too often, and the overall simulation speed may be degraded. An improvement to roll-back scheme is the **Calaveras** algorithm used in the Saber mixed-signal simulator from Analogy Inc. When data needs to be exchanged, if the analog simulator is ahead in time, it is again rolled back in time, but the analog matrix instead of being re-evaluated, it is interpolated at the synchronization point.

These methods are used in sequential simulators, where the digital and analog engines share a common address space, and alternate taking turns in execution. Both schemes force the analog solver to return in time after taking a step beyond a synchronization point with the digital algorithm. Both methods will reject the step that passed the synchronization point after it was accepted and completely solved. The following sections propose two other methods, similar to the original roll-back scheme, with the improvement that the next time point is not completely accepted unless there is no external synchronization digital event with a lower timestamp. Instead it is only checked for convergence. The second method proposed will further improve the performance for distributed simulation by increasing the parallelism among analog solvers.

4.1. Method I: Internal Step Event

This section presents the first method we propose for the analog - digital timing synchronizations. At current time ct, after computing the new $delta$, but before accepting the solution at next time $nt = ct + delta$, the Spice LP schedules an internal event $step@nt$ (see Figure 2). If there is no external event $alter@t$ with $t < nt$, then the solution is accepted at nt, the current time becomes $ct = nt$, a new $delta$ is computed and a new $step@nt$ is scheduled. Otherwise, if there is a digital event $alter@t$ with $t < nt$, then the corresponding input source is updated, the pending internal $step@nt$ event is canceled, a new $delta$ is computed based on the new input values, and another $step@nt$ is scheduled.

This method is similar to the roll-back scheme for the sequential mixed-mode timing control. The difference is that after computing the variable time step $delta$, such that the solution converges at $nt = ct + delta$, the analog solver does not immediately advance the time to nt, and does not accept the new solution at nt. Instead, the current time is kept at ct and an internal event is scheduled to manifest the intent to advance the time at nt. If there is no external digital synchronization event in between the current time ct and the new time point nt, then the new solution is accepted and the current time advanced. If a digital event occurs between the current time ct and the next manifested time point nt, then the time step is canceled and recomputed based on the updated inputs. Computationally, this is similar to trying a smaller step $delta$ because the bigger step was rejected due to non-convergence. There is no roll-back in time.

In parallel and distributed event simulation (PDES) each logical process (LP) has a virtual $simulate()$ function defined by the application. The function takes as parameters the $state$ and the next event $event@time$ for the LP, and it is called by the simulator in the chronological order of the event timestamps. The Spice LP simulate() function for this method is sketched in Figure 3.

```
Spice_LP::simulate(state, event@time)
1    if(event = alter) // ct ≤ time < nt
3      alter the PWL source at time and at time + d
4      set breakpoint at time and at time + d
5      preempt the internal event step@nt
6      compute new delta
7      schedule new event step@nt = ct + delta
8    else // event = step, time = nt
9      accept solution at new current time ct = nt
10     compute new delta
11     schedule new event step@nt = ct + delta
12     check output waveforms relative to VL, VH
13     send output events @ct if any
```

Figure 3. SpiceLP::simulate() method I.

In PDES, an LP may be conservative or optimistic. In the second case, it needs to define a $state$ which is saved after each simulation step and restored in case of rollback. For the Spice LP, the state contains everything that may change during a simulation step like the values of input and output ports, the internal event already scheduled, and the following informations stored in the SPICE [11] circuit CKTcircuit data structure: all the scalars (including CKTtime, CKTdelta, CKTtemp, CKTorder, and others), the CKTstates vectors for each integration order, the vectors with current and old solutions of the system, the vector of breakpoints. This state is proportional to the complexity of the sub-circuit, and saving and restoring it may be a huge overhead for the optimistic LP.

4.2. Method II: Truncated Internal Timestamp

This section proposes the second method for analog-digital synchronization. Time in SPICE, as in any numerical integration system, is represented as floating point numbers. Therefore the events generated by the Spice LP do not have integer number, but real number timestamps: nt for the internal events $step@nt$, and ct for the external events. In this case, the chance to have two or more events with the same timestamp is very small. This is detrimental for synchronous distributed simulation (like conservative without lookahead), where the parallelism is measured by the number of events with the same timestamp available for different LPs. This means that synchronous distributed simulation will perform very poor, leaving only asynchronous methods (optimistic and/or conservative with lookahead) as an option. The asynchronous methods add overhead to distributed synchronization in the form of state saving and rollback for optimistic methods and lookahead prediction and additional null messages for conservative with lookahead methods.

We propose to modify method I such that the parallelism among Spice LPs at the same time is increased and they can run conservatively without lookahead, eliminating the overhead of saving large states or of using null messages. If two or more Spice LPs have scheduled internal step events at close times, for example LP1 has a $step@12.2$ and LP2 a $step@12.3$, we can consider these LPs as running in parallel with concurrent activity at the same time, for example $step@12$. The number of events with equal timestamp may be increased if the time granularity is coarsened to, for example, 1 ns, i.e. the timestamps of Spice LP events are truncated to 1ns multiples. If only the timestamps of internal events $step@nt$ are truncated to $\lfloor nt \rfloor$ in ns, leaving external events with original floating number ct timestamps, then the accuracy of external events is preserved, and the parallelism is still improved since most of the events are internal.

```
Spice_LP::simulate(state, event@time)
1    if(event = alter) // ⌊ct⌋ ≤ time < ⌊nt⌋ ≤ nt
2      set breakpoint at time;
3      simulate until ct ≥ time // zero or more steps
4      alter the PWL source at ct and at time + d
5      set breakpoint at ct and at time + d;
6      preempt the internal event step@⌊nt⌋
7      compute new delta
8      schedule new event step@⌊nt = ct + delta⌋
9    else // event = step, time = ⌊nt⌋
10     accept solution at new current time ct = nt
11     compute new delta
12     schedule new event step@⌊nt = ct + delta⌋
13     check output waveforms relative to VL, VH
14     send output events @ct if any
```

Figure 4. SpiceLP::simulate() method II.

The choice of truncation value for analog internal step events is technology, model and circuit activity dependent. For faster technologies or more active circuits, a smaller value (like 1 ps or 1 fs) may be enough to characterize the simultaneity among different analog solvers.

Figure 4 illustrates the SpiceLP $simulate()$ function for this second method. This technique affects slightly the timing of the D/A input interface. If the input events have timestamps at nanosecond multiples, which is the case with most of the digital circuits, then method II behaves exactly like method I, with increased internal parallelism.

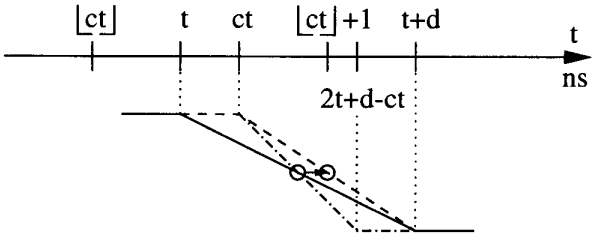

Figure 5. Modification of the D/A interface.

If the input events $alter@t$ have a finer timing than nanosecond, then at the moment they are simulated we have $t < \lfloor nt \rfloor$. Also, $\lfloor ct \rfloor \leq t$, otherwise $alter@t$ would have been simulated in the previous step, when $t < \lfloor \text{old}nt \rfloor$ and old$nt = ct$. So $\lfloor ct \rfloor \leq t < \lfloor nt \rfloor$. Also, $\lfloor ct \rfloor \leq ct < \lfloor ct \rfloor + 1$. If $ct \leq t$, then again, method II behaves exactly like method I, with increased internal parallelism. There is a 50% chance though, to have $t < ct$. Then $\lfloor ct \rfloor \leq t < ct < \lfloor ct \rfloor + 1$. So $ct - t < 1$ ns. By choosing the delay $d = 1$ ns, we have $ct < t+1 = t+d$, which means that only the first time-point of the PWL source is delayed by maximum 1 ns, the second one being the same (Figure 5 dashed line). The middle of the edge $((ct, vdd), (t + d, gnd))$ is delayed by max. 0.5 ns and only 50% of the time of the rare cases when the digital part has a timing finer than nanosecond. This delay can be compensated by doubling the edge delay $d = 2$ ns. Then, if $t < ct$, the PWL timepoints are adjusted to ct and $t+d-(ct-t) = 2t+d-ct$ (Figure 5 dash-dotted line). We still have a positive edge: $(2t+d-ct)-ct = 2t + 2 - 2ct = 2(1 - (ct - t)) > 2(1 - 1) = 0$.

5. Implementation and Results

The discrete cosine transform processor (DCT) in Figure 6 was described in VHDL with all 16 8-bit adders as SPICE foreign architectures. This circuit is dedicated for an 8x8 pixel block and is organized as a two stage pipeline. Each stage contains 8 similar cells. The first stage cell is a multiplier - adder - accumulator, and the bottom stage cell is a multiplier - adder - shift-register accumulators. The VHDL processes and signals were translated by our VHDL compiler into C++ classes derived from the VHDL_Process and VHDL_Signal classes that form our distributed VHDL kernel [14]. For each analog adder there is an instance of the Spice_LP class, which is derived (as well as VHDL_Process and VHDL_Signal classes) from the base LP class, the core of our distributed simulator [13]. We have modified the SPICE source code, especially the transient analysis functions to interrupt the numerical integration loop after the new time step $delta$ is computed and tested for convergence, and to continue

the step by accepting and committing it in case no external synchronization event arrives, or to reject current time step *delta* and recompute it if the input source parameters have changed.

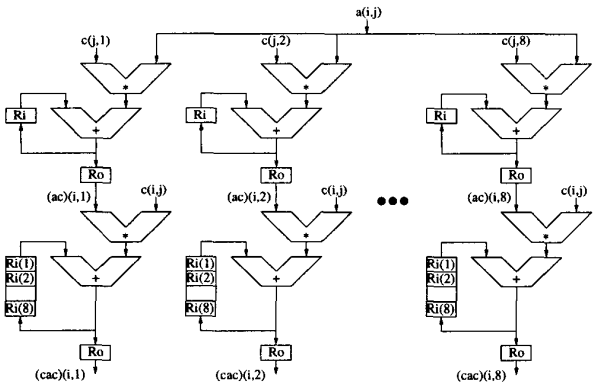

Figure 6. The Architecture of DCT.

For the distributed simulation, we used an SGI parallel machine with 16 processors out of which only 11 were available for our tests, and MPI for the communication system. Since our distributed simulator is very flexible, we tested various configurations for both methods I and II:

C1: sequential;
C2: all LPs conservative with lookahead;
C3: all LPs conservative without lookahead;
C4: all LPs optimistic;
C5: digital LPs optimistic, analog LPs conservative with lookahead;
C6: digital LPs optimistic, analog LPs conservative without lookahead;
C7: digital LPs conservative, analog LPs optimistic;
C8: all LPs optimistic, but analog ones with a doubled state (as they were 16-bit adders instead of 8-bit).

Figure 7 illustrates the speed-ups for the method I, Figure 8 the speed-ups for the improved method II, and Figure 9 gathers all curves for comparison. As expected, the method I with configuration C3 gives no speedup, since there are few events with equal timestamps available for the Spice LPs and they are the only ones that can be simulated in parallel by the conservative LPs without lookahead.

Adding lookahead support to conservative synchronization gives each LP a time horizon up to which all the events are safe, so configuration C2 can simulate in parallel not only the events with the same minimum timestamp, but also all the events with timestamps up to the time horizon. Computing the lookahead is a big overhead as can be proved by comparing the speedup difference between C2 and C3 in method I with the same difference in method II. C3 still simulates in parallel only events with equal timestamp, but their number increased and there is no lookahead overhead.

The optimistic synchronization exploits better the parallelism inside the application and is very suitable for light-state digital circuits. Also, it does not need lookahead. The analog Spice LP has a large state which has to be saved at each step. For small to medium analog sub-circuits, the optimistic C4 outperforms the conservative with lookahead C2. Doubling the state of the analog adder slows down the optimistic simulation as shown by the optimistic C8.

Distributed simulation of a mixed-signal circuit must be performed using a mixed synchronization. Making the digital LPs optimistic avoids unneeded blocking at a small cost, because the state is small, and the overhead of saving heavy analog states can be avoided by mapping the analog LPs as conservative. The mixed configurations C5 and C6 clearly demonstrate that mixed-signal parallel simulation gives the best results when using mixed synchronization methods with digital parts optimistic and analog parts conservative. The difference between the mixed C5 and the pure conservative C2 or optimistic C4 is not as big as the difference between mixed C6(improved) and conservative C3(improved) or optimistic C4 because the gain in using optimism for digital parts is shaded by the more expensive lookahead computation in an optimistic LP (needed by coexisting conservative LPs). By using the improved method II, C6 eliminates the need of the lookahead computation, detaching itself as a clear winner. C6 speeds-up the C1 sequential simulation from more than 2 hours and 30 minutes to less than 24 minutes (and from 4 hours to 1 hour on a slower network of 16 workstations).

We also tested the reverse mapping (digital LPs - conservative and analog LPs - optimistic), but as can be seen in Figure 9, mixed C7 is slower than pure conservative C2 or optimistic C4.

In general it can be observed a big step in speedup at 8 processors, also expected at 16, since there are 16 analog adders. At 6 processors there is a dip in the curves caused by the naive partitioning (equal LPs for each processor) that we used. The communication and global synchronization overheads are more noticeable as the number of processors increases.

6. Conclusion

This paper presented a methodology for parallel and distributed mixed-mode simulation of mixed-signal circuits. Mixed-signal systems are best simulated in parallel by using a mixed synchronization. Making the digital components optimistic, the overhead of unnecessary blocking is avoided, while mapping the analog components conservative, the overhead of saving their heavy state is eliminated. By using a coarser time grain only for the internal step events of the analog solver, the parallelism among simultaneous events is increased without sacrificing timing accuracy and the overhead of null messages and lookahead is eliminated too, achieving a 6.4X speed-up on 11 processors for a DCT processor circuit described in VHDL and SPICE. This paper also demonstrated, for the first time, that optimistic synchronization for distributed analog simulation is feasible and quite efficient for small-to-medium size analog blocks.

With the increasing needs for complete systems-on-chip verification considering deep-submicron effects and the emerging core-based design methodologies where various parts of a design may be described at various abstraction levels, this work provides a successful attempt in exploiting the concurrency of mixed-abstraction systems to achieve maximum simulation performance by parallel and distributed computation.

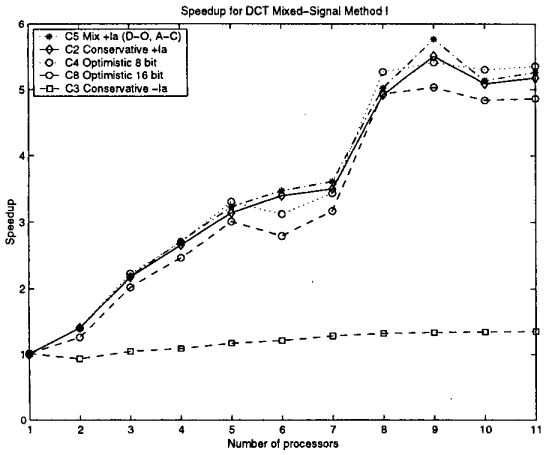

Figure 7. Speedup for Method I.

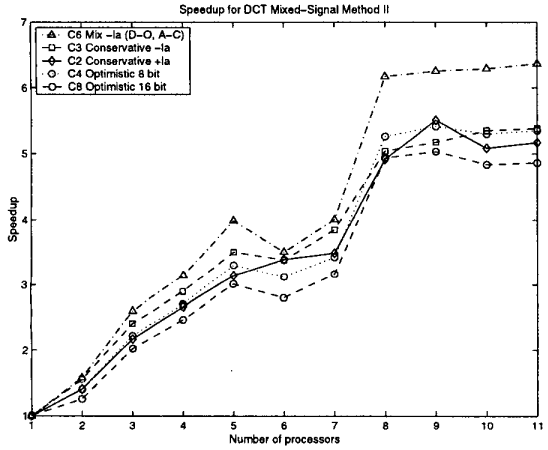

Figure 8. Speedup for Method II.

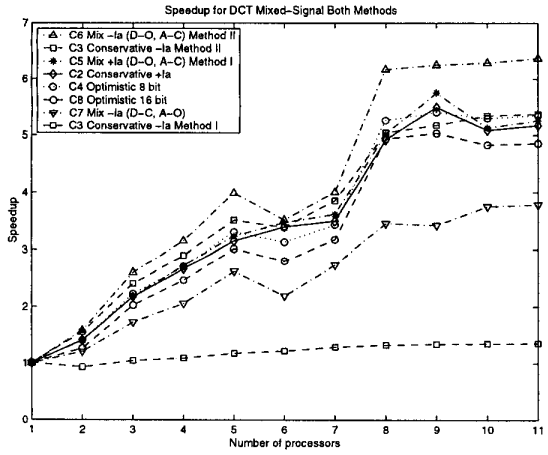

Figure 9. Speedup for both Methods.

References

[1] R.D. Chamberlain. Parallel logic simulation of VLSI systems. *Proc. 32rd IEEE/ACM Design Automation Conference*, pages 139–143, June 1995.

[2] R. Saleh et al. Parallel circuit simulation on supercomputers. *Proc. of IEEE*, 77(2m), December 1989.

[3] K.M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, November 1981.

[4] D.A. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[5] A. Ferscha. *Parallel and distributed simulation of discrete event systems.* Parallel and Distributed Computing Handbook. McGraw-Hill, 1995.

[6] R.M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30+, October 1990.

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

[8] L. Snyder and M. Bailey. An empirical study of on-chip parallelism. *Proc. IEEE/ACM Design Automation Conference*, June 1988.

[9] C.-P. Wen and K. A. Yelick. Parallel timing simulation on a distributed memory multiprocessor. *Proc. ICCAD*, pages 130–135, November 1994.

[10] A.R.W. Todesco and T.H.-Y. Meng. Symphony: A simulation backplane for parallel mixed-mode co-simulation of VLSI systems. *Proc. 33rd IEEE/ACM Design Automation Conference*, 6:149–154, June 1996.

[11] L.W. Nagel. *SPICE2: A computer program to simulate semiconductor circuits.* PhD thesis, U.C. Berkeley, Electronics Research Laboratory Rep. No. ERL-M520, May 1975.

[12] IEEE. VHDL LRM. *IEEE Std. 1076-1993*, 1994.

[13] D. Lungeanu and C.-J. Richard Shi. Distributed simulation of VLSI circuits via lookahead-free self-adaptive optimistic and conservative synchronization. *Proc. ICCAD*, pages 500–504, November 1999.

[14] D. Lungeanu and C.-J. Richard Shi. Parallel and distributed VHDL simulation. *Proc. DATE*, pages 658–662, March 2000.