Performance Predictions for Speculative, Synchronous, VLSI Logic Simulation

Bradley L. Noble
J. Cris Wade
Roger D. Chamberlain

Bradley L. Noble, J. Cris Wade, and Roger D. Chamberlain, "Performance Predictions for Speculative, Synchronous, VLSI Logic Simulation," in *Proc. of the 34th Annual Simulation Symposium*, April 2001, pp. 56-64.

Southern Illinois University Edwardsville
and
Washington University

Performance Predictions for Speculative, Synchronous, VLSI Logic Simulation

Bradley L. Noble*, J. Cris Wade*, and Roger D. Chamberlain†

*Dept. of Electrical and Computer Engineering
Southern Illinois University Edwardsville
Edwardsville, IL
{bnoble,jwade}@siue.edu

†Computer and Communications Research Center
Department of Electrical Engineering
Washington University, St. Louis, MO
roger@ccrc.wustl.edu

Abstract

VLSI logic simulation is an application area in which execution time improvements can have direct economic benefits. Here, we investigate the use of parallel simulation techniques to improve the performance of VLSI logic simulation, including the often neglected issue of sensitivity to variations in the simulation workload. Performance predictions are presented for the use of speculative computation in synchronous discrete-event simulation of VLSI systems.

1. Introduction

VLSI logic simulation is used extensively in the design verification of digital systems. With the advent of Application Specific Integrated Circuits (ASICs) into widespread commercial use, the need for design verification has increased dramatically. The ASIC designer does not have the luxury of breadboarding a design, testing for design errors, and then modifying the breadboard prototype to correct for the errors. Given the high costs associated with individual fabrication runs for integrated circuits, the designer must have a high degree of confidence in the correctness of the design prior to fabrication. Currently, logic simulation is the primary design tool that fulfills this need.

Figure 1 illustrates a typical design flow for an ASIC development. The design is originally specified in a hardware description language (HDL). HDLs in common use are Verilog and VHDL. This original description of the system is at the user level. The user level system description is simulated in the iterative design cycle on the right-hand side of the figure until the designer is satisfied with the functionality of the system. At this point, the user-level HDL description is input to the synthesis and place-and-route tools, yielding a gate-level description of the system. This gate-level system description is simulated in the iterative design cycle on the left-hand side of the figure until the designer is completely satisfied that both functionality and timing are correct. Finally, the system is fabricated and tested.

This example design flow illustrates the important role that simulation plays in the design process. It is not uncommon to have long simulation runs (measured in hours or even days) be a significant bottleneck in the design process. As a result, improvements in simulator performance can provide significant benefits to the ASIC design community.

One approach to performance improvement is the use of parallelism. The research community has been investigating the parallel execution of discrete-event simulation models in general [5] and logic simulation models in particular [1] for more than a decade. Yet, in spite of these efforts, production logic simulators today are almost exclusively executed on uniprocessor systems. We believe that the lack of adoption of parallel implementations by the commercial community has

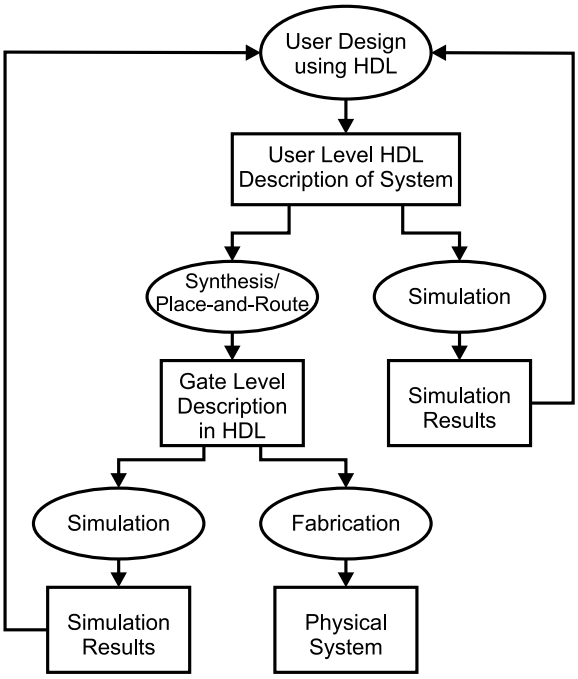

Figure 1. Example design flow for ASIC development. Simulation occurs in two iterative design cycles within the flow.

a number of causes:

1. Historically, parallel computer systems were expensive to acquire and difficult to maintain. This impediment to adoption has clearly been diminished in recent years with the availability of relatively inexpensive symmetric multiprocessor systems from a number of manufacturers.

2. Parallel simulation performance results have often been inconsistent. Lin and Lazowska [11] coined the term "S phenomenon" to describe the observation that speedup curves for optimistic, asynchronous simulators often have several local minima and maxima. This observation was made over a large set of different simulation applications [4, 8, 10, 20].

In this paper, we describe an approach that is aimed at addressing the second cause listed above. We are proponents of the use of synchronous algorithms for parallel simulation, specifically aimed at the need for consistent performance across a range of individual circumstances. This paper provides a performance analysis of the use of speculative computation in a synchronous simulation environment, with the target simulation application of VLSI logic design.

In the section that follows, we describe what is meant by speculative, synchronous simulation. Section 3 describes the performance evaluation methodology, based upon trace-driven simulation using data derived from the ISCAS-89 benchmark set. Section 4 provides the performance predictions and Section 5 concludes the paper.

2. Speculative, Synchronous Simulation

Speculative computation has received a great deal of attention in the parallel computing community as a technique for balancing computational load and masking latencies in interprocessor communications [6]. It has even found its way into processor design at the instruction level, with proposals for *value prediction*, speculating the results of individual instructions or basic code blocks [9]. In discrete-event simulation, parallel algorithms that perform computation in a speculative manner are generally referred to as *optimistic* algorithms.

While both synchronous and asynchronous optimistic algorithms exist, our interest is in synchronous algorithms. This is due to a desire to avoid the inconsistent (and sometimes inexplicable) performance associated with many asynchronous protocols. In addition, synchronous algorithms have an inherent simplicity and ease of implementation that is not present in asynchronous techniques.

As with many other algorithms, there is a tradeoff between simplicity and performance; the simplicity of the synchronous algorithm comes with a potential cost in performance. If frequent synchronizations are required, the algorithm becomes more fine grained. Since the critical path lies with the slowest processor at each iteration, idle time can accumulate at the other processors and the total execution time is lower bounded by the execution time of the slowest processor in each iteration. In an attempt to alleviate these performance concerns for synchronous discrete-event simulation, techniques used in asynchronous simulation algorithms (e.g., speculative computation) can be applied to the synchronous algorithm, while retaining the iterative nature of the algorithm.

Figure 2 illustrates a typical set of iterations of a synchronous iterative algorithm executing on four processors (labeled 1 through 4). An iteration can be seen as consisting of 3 phases:

1. Computation – performing the computational tasks associated with the application.

2. Idle – time between first and last processor to complete work in an iteration.

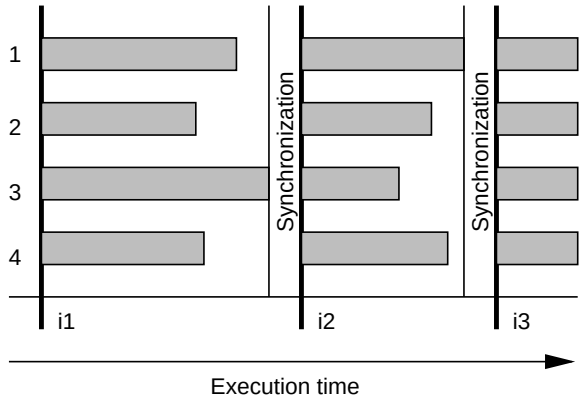

Figure 2. Synchronous iterative algorithm execution. The horizontal bars represent computation during each iteration.

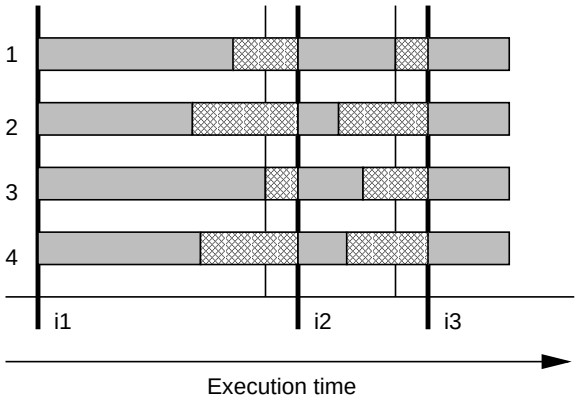

Figure 3. Speculative computation execution time line. The crosshatched areas represent speculative computing, potentially decreasing the computation needed during the subsequent iteration.

3. Synchronization – time to complete the barrier synchronization operation.

Computation starts on all processors immediately following the barrier synchronization. During this phase, each processor executes all the tasks assigned to it that iteration. For discrete-event simulation, this consists of processing simulation events. Interprocessor data communication may be concurrent with computation. At the end of the computation phase, each processor enters a barrier and waits for its completion. The *idle* phase is a result of variation in computation times between processors due to imbalances in workload as the algorithm progresses, multitasking other unrelated processes (background load), or processor heterogeneity. *Synchronization* time is determined by the communication performance of the parallel platform in completing the barrier synchronization. After the barrier synchronization completes, the processors proceed to the next iteration, repeating the cycle until the algorithm completes.

Speculative computation utilizes the idle phase of the above algorithm by allowing processing to proceed into future iterations. While waiting for the barrier synchronization to complete, computation progresses speculatively, with the hope that a message arrival from a remote processor does not subsequently invalidate the computation. Once the barrier synchronization is complete, the speculated computation is tested for correctness and either committed or discarded.

To support processing during the execution of the barrier synchronization, a fuzzy barrier implementation is used [7]. Processors signal their willingness to complete the barrier and, rather than blocking, proceed to compute speculatively. A "barrier complete" signal indicates the end of the current iteration. The execution time line, illustrated in Figure 3, shows the speculative computation occurring during the idle times and while waiting for the barrier to complete. Note that the time required to complete iteration 2 is less than in the previous figure, since some of the computation has been completed during the otherwise idle time of iteration 1.

Mehl [12] proposed essentially this technique in the context of a conservative asynchronous algorithm, but did not report on its performance. We previously reported a set of empirical performance results for queuing network simulation in [14, 17]. Dickens et al. [3] present a performance model for a similar algorithm that predicts performance gains over a purely conservative synchronous algorithm. Steinman's Breathing Time Buckets algorithm [19] has been implemented in the SPEEDES environment and exhibits good performance on a pair of simulation models (queueing networks and proximity detection).

3. Performance Evaluation Methodology

The set of variables used in the performance model is summarized in Table 1. A more complete definition of each variable is given in the text near the first use of the variable.

The execution time of a synchronous iterative algorithm requiring I iterations and running on a set of P processors can be modeled by a function consisting of three distinct terms. In any particular iteration i,

Table 1. Parameters for performance model.

Param.	Definition
R_P	application run time with P processors
P	number of processors
I	number of iterations
t_s	serial computation time
t_p	parallel computation time
t_{ov}	parallelism overhead time
$w_{i,j}$	parallel work during iteration i on processor j, no speculation
$v_{i,j}$	parallel work during iteration i on processor j, with speculation
$s_{i,j}$	work available to speculate during iteration i on processor j
r	speculation success ratio: fraction of speculative computation that is correct

there is a portion of work which is serial in nature. The first term represents the time required to complete the portion which cannot be parallelized, which we denote as $t_{s,i}$. Each processor j has some assigned work to be performed during the parallel portion of each iteration. The time for each processor to complete this work is denoted by $t_{p,i,j}$. However, the parallel portion of each iteration is not complete until the last processor completes its assigned work. This gives the second term as $\max_{1\leq j\leq P}(t_{p,i,j})$. Finally, the time required for the overheads associated with the parallel algorithm itself during iteration i are denoted by $t_{ov,i}$.

Combining these terms gives a model for the execution time as

$$R_P = \sum_{i=1}^{I}\left[t_{s,i} + \max_{1\leq j\leq P} t_{p,i,j} + t_{ov,i}\right] \quad (1)$$

Although this equation effectively models the execution time, it requires specific knowledge of each individual iteration. By treating each of the terms as an i.i.d.[1] random process and taking the expected value, the references to a specific iteration are eliminated. A new expression for run-time is given by

$$\begin{aligned} R_P &= \sum_{i=1}^{I}\left(E\left[t_{s,i}\right] + E\left[\max_{1\leq j\leq P} t_{p,i,j}\right] + E\left[t_{ov,i}\right]\right) \\ &= I\left(t_s + E\left[\max_{1\leq j\leq P} t_{p,j}\right] + t_{ov}\right) \end{aligned} \quad (2)$$

[1]Independent and identically distributed.

Previous work has shown this model to be effective for estimating run time for several different types of synchronous iterative algorithms [18], including quantifying the performance impact of speculative computation in discrete-event simulation both with uncorrelated [15] and correlated [16] parallel workloads.

3.1. Speculative Workload Characterization

The expressions in this subsection were originally presented in [15]. The development is repeated here for clarity of understanding in the performance predictions that follow.

For the VLSI logic simulation applications we are interested in, both the serial term $t_{s,i}$ and the parallel overhead term $t_{ov,i}$ in (1) do not vary significantly between iterations. As such, these terms can be treated as constants. We focus on characterizing $E\left[\max_{1\leq j\leq P} t_{p,i,j}\right]$ for the both the initial workload without speculative computation and for the resulting workload with speculative computation.

Let us define $w_{i,j}$ as the work to be completed on processor j during iteration i without speculative computation. Assuming the units of work are relatively constant in time (e.g., event evaluations with similar computational complexity), $t_{p,i,j}$ will be proportional to $w_{i,j}$. When speculative computation is performed, it will take work away from future iterations whenever a processor completes before the barrier synchronization. We will define this new workload as $v_{i,j}$, the work to be completed on processor j during iteration i with speculation. In this case, $t_{p,i,j}$ is proportional to $v_{i,j}$ which, by definition, must be less than or equal to $w_{i,j}$.

The development of $v_{i,j}$ from $w_{i,j}$ can be made by examining a specific iteration, i, of a synchronous algorithm that incorporates speculative computation. Examining Figure 3, we determine that the work to be completed during iteration i is $\max_{1\leq j\leq P} v_{i,j}$. Therefore, the amount of work that can be speculated on processor j during iteration i is given by

$$s_{i,j} = \max_{1\leq j\leq P}(v_{i,j}) - v_{i,j}, \quad (3)$$

provided we limit speculation to one iteration into the future. This yields a recursive formulation that relates $v_{i+1,j}$ to $v_{i,j}$:

$$v_{i+1,j} = w_{i+1,j} - r s_{i,j} \quad (4)$$

with initial condition

$$v_{0,j} = w_{0,j}. \quad (5)$$

The scalar r that is introduced in (4) represents the speculation success ratio, or the fraction of the spec-

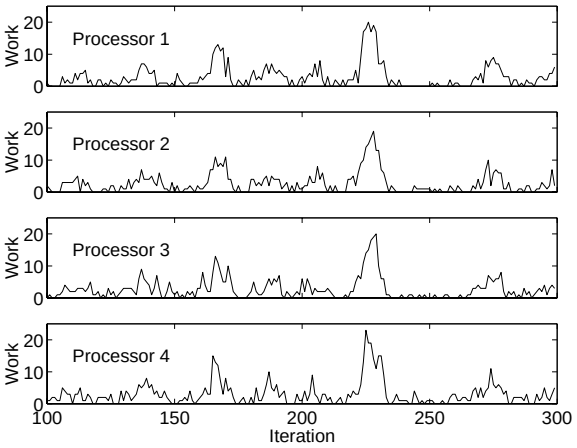

Figure 4. Example trace data for VLSI logic simulation. This represents the computational workload without speculation.

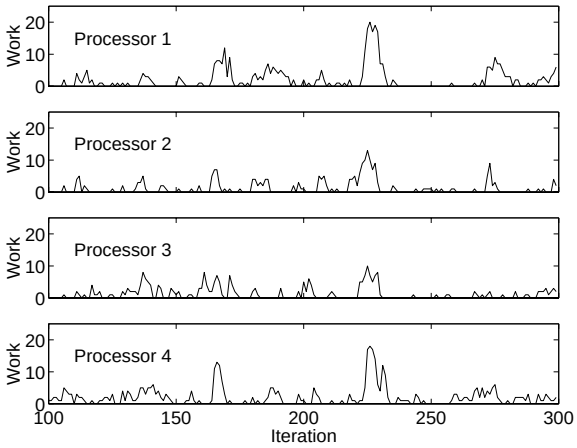

Figure 5. Execution trace for speculative VLSI logic simulation. This represents the computational workload when speculated events are successfully committed 100% of the time (i.e., $r = 1.0$).

ulated work that was successfully committed. Substituting (3) into (4) gives:

$$v_{i+1,j} \quad = \quad w_{i+1,j} - r \left(\max_{1 \le j \le P} (v_{i,j}) - v_{i,j} \right) \qquad (6)$$

These expressions can be used to empirically evaluate $v_{i,j}$ for specific instances where $w_{i,j}$ is known. Essentially, the evaluation is a form of trace-driven simulation, where trace data (which can be collected from a serial simulation execution) provides the information for $w_{i,j}$ and the repeated evaluation of (6) provides $v_{i,j}$.

3.2. Model Evaluation

In [15] and [16], the above expressions were used to guide the development of a stochastic workload model that describes the steady state distribution of the computational workload both with and without speculation. Here, we are interested in using (2) and (6) directly to develop an understanding of the performance of VLSI logic simulation. The empirical data is derived from a simulation of several of the ISCAS-89 sequential benchmark circuits [2]. For each circuit, a gate-level simulation is executed using a unit delay timing model driven with random input vectors.

Our model evaluation methodology starts with a set of trace data that directly represents $w_{i,j}$, $1 \le i \le I$, $1 \le j \le P$. Workload data from the logic simulation application was recorded for a variety of logic circuits. Example trace data from logic simulation `s9234` on four processors ($P = 4$) is shown in Figure 4. The units of work shown are simulation event counts.

The empirical data, $w_{i,j}$, is evaluated using (6) to determine $v_{i,j}$. This evaluation constitutes a trace-driven simulation of the speculative execution algorithm. An example trace of $v_{i,j}$ is shown in Figure 5 for circuit `s9234` simulated on four processors ($P = 4$) assuming a success ratio of $r = 1.0$.

Once trace data is available that represents the computational workload both with and without speculation, the central term in (2) can be estimated by evaluating the expectation (across iterations) of the maximum (across processors) of the workload. These values are tabulated in Table 2 for two different values of the speculation success ratio r. Also given in the table is the percentage performance improvement predicted due to the use of speculative computation.

3.3. Speculation Success Ratio

The above techniques provide predicted performance for a speculative, synchronous simulation execution, provided that the speculation success ratio, r, is given. In this subsection, we address two issues associated with the speculation success ratio: 1) whether or not a mean value (i.e., scalar) model is sufficient, and 2) what is an appropriate value for VLSI logic simulation.

To address the first issue, we model the speculation success ratio, r, as a random variable, and investigate the sensitivity of the performance results to various distributions for r, each with the same mean. Consider the following four distributions: uniform, binomial, tri-

Table 2. Initial performance results for s9234 with $P = 4$.

	$E[\max_{1 \le j \le P} w_{i,j}]$	$E[\max_{1 \le j \le P} v_{i,j}]$	% improvement
$r = 0.5$	4.98	4.29	14%
$r = 1.0$	4.98	3.84	23%

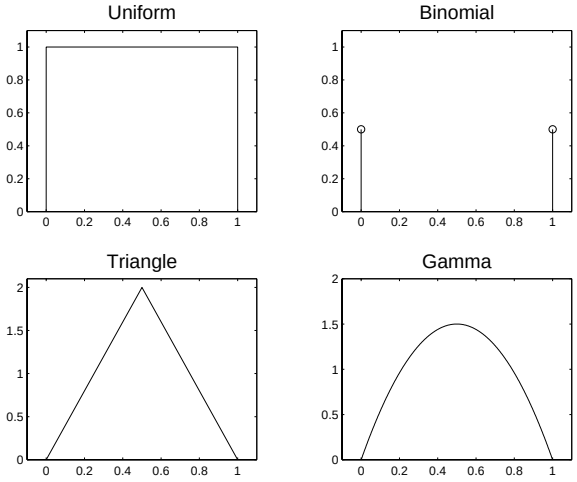

Figure 6. Density functions used to examine sensitivity of the model to distribution of speculation success ratio. These densities all have a mean value of 0.5.

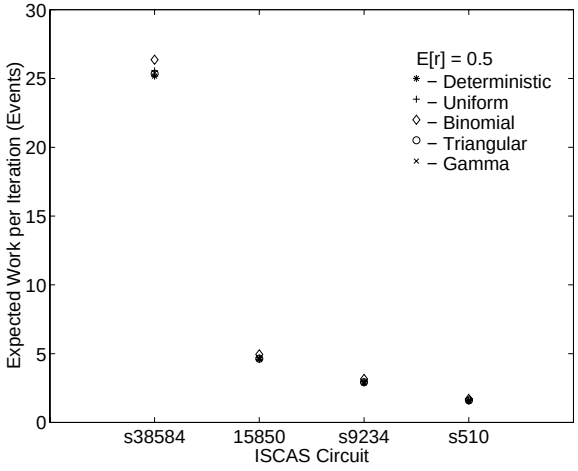

Figure 7. Performance predictions for various distributions of the speculation success ratio. These success ratio distributions all have an expected value of $E[r] = 0.5$.

angle, and Gamma. The density functions of each of these distributions are shown in Figure 6, where each distribution is shown with a mean value of 0.5.

To evaluate the sensitivity of the logic simulator performance to the distribution of the speculation success ratio, the trace-driven simulation is executed again with a probabilistic model for r. Each time (6) is evaluated, a sample is drawn for r from the desired distribution. The resulting values for $E[\max_{1 \le j \le P} v_{i,j}]$ are plotted in Figures 7 and 8. The deterministic value for r is plotted along with various distributions, each for a variety of logic circuits.

The primary observation we make when observing these plots is the small impact there is when using a probabilistic model for r rather than the deterministic model. The maximum deviation is under 10% across the range of experiments. It should be noted that the deterministic model is consistently optimistic (i.e., smaller) with respect to the probabilistic models, and that the binomial distribution (likely the most realistic of the distributions) consistently gives the most pessimistic (i.e., largest) results.

Given that there is some impact on predicted performance due to the distribution of the speculation success ratio, we wish to determine a reasonable distribution for the VLSI logic simulation application. In [13], we evaluated the effectiveness of the standard optimistic assumption at predicting the future message content on a communications channel. For a two-valued logic simulation, this prediction is equivalent to whether or not a speculated event will be committed. As a result, it is reasonable to use this data to derive an estimate of the true distribution of the speculation success ratio.

Figure 9 shows a histogram of the message prediction accuracy across communications channels. The data comes from the latch outputs of all of the ISCAS-89 circuits (a total of 9696 channels). Here, we assume that this data generalizes to all of the functional evaluations in the simulation.

The trace-driven simulation now proceeds as follows. At each iteration, (3) is evaluated to determine $s_{i,j}$, the number of speculated events. For each event in $s_{i,j}$, a random sample is drawn from the distribution of Figure 9. This sample determines the probability p that the event is committed. This probability p is used to control a Bernoulli trial, the outcome of which deter-

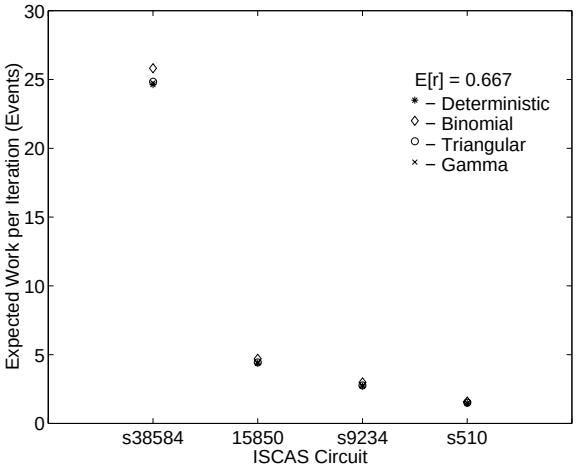

Figure 8. Performance predictions for various distributions of the speculation success ratio. These success ratio distributions all have an expected value of $E[r] = 0.667$.

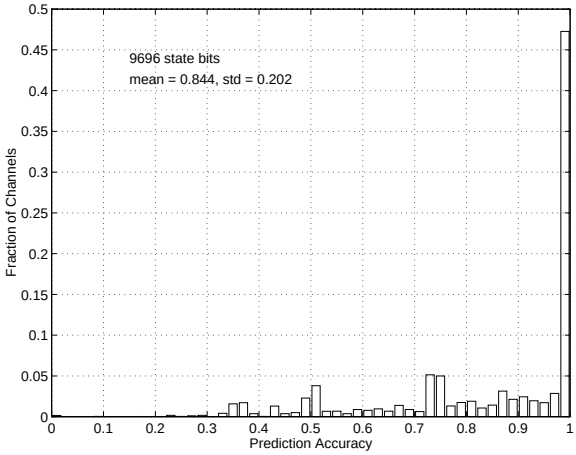

Figure 9. Distribution of successful predictions for latch output channels across the ISCAS-89 benchmarks [13].

mines whether or not the event is committed. Once it is known how many of the events are committed, this quantity is used for $rs_{i,j}$, the second term in (4). Evaluation of (4) then follows, and the simulation of one iteration is complete.

4. Performance Predictions

Here, we use the performance evaluation methodology described in the previous section and apply it to a subset of the ISCAS-89 benchmark circuits. Table 3 lists the specific circuits used, their size (component count), and the mean workload per iteration (event count) for a serial execution. These specific circuits were chosen because of their size (i.e., they represent a range across the available choices).

Table 3. Benchmark circuits and their properties.

circuit	size (components)	mean workload $E[\sum_{1<j<P} w_{i,j}]$
s38584	11448	177.1
s15850	3448	23.6
s9234	2027	13.1
s510	179	5.4

Figures 10 and 11 present performance predictions for the circuits listed above, both with speculative computing and without speculative computing. The largest circuit (s38584) is shown separately due to the differences in scale.

Table 4 gives the percentage improvement (i.e., decrease in expected workload per iteration) due to speculative computation relative to the synchronous algorithm without speculation. As one would predict, the performance improvement is greater for larger numbers of processors, due to the increased variability in the original workload and, therefore, the increased opportunity for speculative computation. The smaller percentage increases associated with the largest circuit (s38585) will be understood upon examination of the table that follows (Table 5).

Table 4. Performance gain due to speculative computation.

circuit	$P = 2$	$P = 4$	$P = 8$
s38584	2.5%	6.4%	12.1%
s15850	7.8%	17.2%	23.0%
s9234	11.7%	20.2%	27.9%
s510	11.0%	19.9%	30.3%

In order to evaluate the overall gains due to parallelism, one must estimate the remaining terms in (2), the serial processing time and the parallel overhead. Table 5 gives the predicted speedup vs. a serial im-

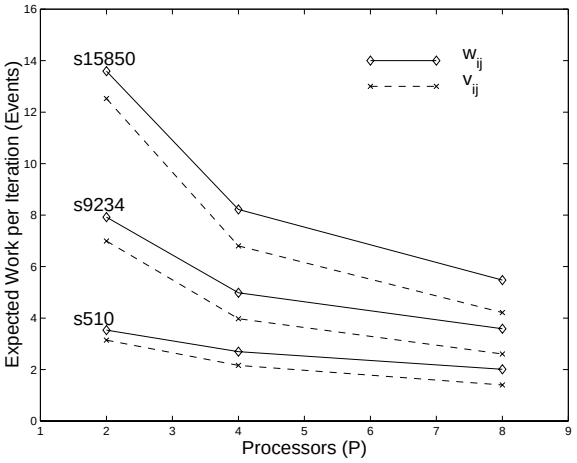

Figure 10. Predicted performance for synchronous simulation without speculation ($E[\max_{1\leq j\leq P} w_{i,j}]$) and with speculation ($E[\max_{1\leq j\leq P} v_{i,j}]$).

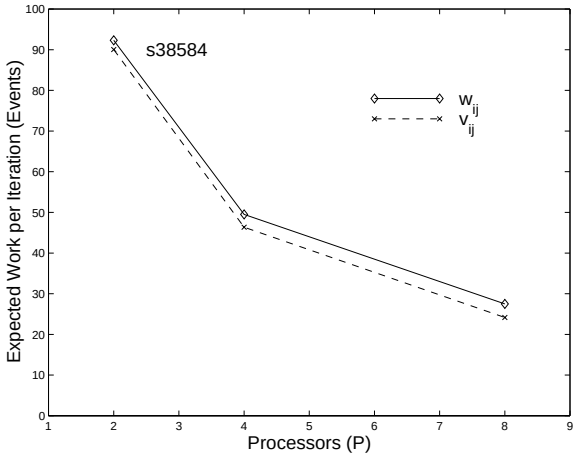

Figure 11. Predicted performance for `s38584` without speculation ($E[\max_{1\leq j\leq P} w_{i,j}]$) and with speculation ($E[\max_{1\leq j\leq P} v_{i,j}]$).

plementation assuming that the sum of the serial and parallel overhead processing each iteration is equivalent to a single event evaluation. Clearly this assumption is not valid for every parallel execution platform, but it is reasonable for tightly-coupled, shared-memory architectures on which a barrier synchronization is relatively inexpensive. Examining the data in the table, it is clear that the overall performance of the speculative computing technique is quite favorable, with parallel efficiencies of about 50% or higher across the board. The lowest performance is associated with the smallest circuits, and the highest performance is associated with the largest circuits.

Table 5. Predicted speedup of speculative, synchronous simulation over serial execution.

circuit	$P = 2$	$P = 4$	$P = 8$
`s38584`	1.9	3.7	7.0
`s15850`	1.7	3.0	4.5
`s9234`	1.6	2.6	3.6
`s510`	1.3	1.7	3.9

The lower than average percentage improvements due to speculative computation shown in Table 4 can now be understood in the appropriate context. With speedup values so high (e.g., 3.7 using 4 processors, a parallel efficiency of 93%), the performance gain due to parallelism is near the maximum achievable.

5. Summary and Conclusions

This paper has presented performance predictions for a speculative, synchronous simulation algorithm applied to the application domain of parallel VLSI logic simulation. The performance results show that there is clear benefit to be gained by developing parallel implementations of logic simulation design tools.

An important piece of this investigation is the uniformity of the results. The performance predictions were explicitly tested for sensitivity to the distribution of the speculation success ratio, and the shape of the distribution did not have a significant impact on the results.

In addition, the overall performance figures are quite consistent across a range of circuits and processor populations. This is quite different than the typical circumstance encountered when executing asynchronous parallel simulations.

Given this level of potential performance gain, we believe that commercial implementations of these techniques will be successful in improving the design environment for ASIC development, consistently reducing the time required to perform design verification.

References

[1] M. Bailey, J. Briner, and R. Chamberlain. Parallel Logic Simulation of VLSI Systems. *ACM Comput-*

ing Surveys, 26(3):255-294, September 1994.

[2] F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proc of the Int'l Symp. on Circuits and Systems*, pages 1929–1934, May 1989.

[3] P. M. Dickens, D. M. Nicol, P. F. Reynolds, Jr., and J. M. Duva. The Impact of Adding Aggressiveness to a Non-Aggressive Windowing Protocol. In *Proc. of the 1993 Winter Simulation Conf.*, pages 731–739, December 1993.

[4] M. Ebling, M. Di Loreto, M. Presley, F. Wieland, and D. Jefferson. An Ant Foraging Model Implemented on the Time Warp Operating System. In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 21–28, March 1989.

[5] R. M. Fujimoto. Parallel Discrete-Event Simulation. *Communications of the ACM*, 33(10):30-53, October 1990.

[6] V. Govindan and M. Franklin. Speculative Computation: Overcoming Communication Delays. In *Proc. of the 1994 Int'l Conf. on Parallel Processing*, volume III, pages 12–16, August 1994.

[7] R. Gupta. The Fuzzy Barrier: A Mechanism for the High Speed Synchronization of Processors. In *Proc. of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 54–63, April 1989.

[8] P. Hontalas, B. Beckman, M. Di Loreto, L. Blume, P. Reiher, K. Sturdevant, L. Van Warren, J. Wedel, F. Wieland, and D. Jefferson. Performance of the Colliding Pucks Simulation on the Time Warp Operating System (Part 1: Asynchronous Behavior and Sectoring). In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 3–7, March 1989.

[9] J. Huang and D. Lilja. Exploiting Basic Block Value Locality with Block Reuse. In *Proc. of 5th Int'l Symp. on High Performance Computer Architecture*, pages 106–114, January 1999.

[10] D. Jefferson et al. Distributed Simulation and the Time Warp Operating System. In *Proc. of the 11th ACM Symp. on Operating Systems Principles*, 1987.

[11] Y.-B. Lin and E. D. Lazowska. Processor Scheduling for Time Warp Parallel Simulation. In *Proc. of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 11–14, January 1991.

[12] H. Mehl. Speedup of Conservative Distributed Discrete Event Simulation Methods by Speculative Computing. In *Proc. of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 163–166, January 1991.

[13] B. L. Noble and R. D. Chamberlain. Predicting the Future: Resource Requirements and Predictive Optimism. In *Proc. of 9th Workshop on Parallel and Distributed Simulation*, pages 157–164, June 1995.

[14] B. L. Noble and R. D. Chamberlain. Performance of Speculative Computation in Synchronous Parallel Discrete-Event Simulation on Multiuser Execution Platforms. In *Proc. of the 8th IASTED Int'l Conf. on Parallel and Distributed Computing and Systems*, pages 489–494, October 1996.

[15] B. L. Noble and R. D. Chamberlain. Performance Model for Speculative Simulation Using Predictive Optimism. In *Proc. of 32nd Hawaii Int'l Conf. on System Sciences*, January 1999.

[16] B. L. Noble and R. D. Chamberlain. Analytic Performance Model for Speculative, Synchronous, Discrete-Event Simulation. In *Proc. of 14th Workshop on Parallel and Distributed Simulation*, May 2000.

[17] B. L. Noble, G. D. Peterson and R. D. Chamberlain. Performance of Synchronous Parallel Discrete-Event Simulation. In *Proc. of 28th Hawaii Int'l Conf. on System Sciences*, Vol. II, pages 185–186, January 1995.

[18] G. D. Peterson and R. D. Chamberlain. Parallel Application Performance in a Shared Resource Environment. *Distributed Systems Engineering*, 3:9-19, 1996.

[19] J. S. Steinman. SPEEDES: A Multiple-Synchronization Environment for Parallel Discrete-Event Simulation. *Int'l Journal in Computer Simulation*, 2:251–286, 1992.

[20] F. Wieland, L. Hawley, A. Feinberg, M. Di Loreto, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. Jefferson. Distributed Combat Simulation and Time Warp: The Model and Its Performance. In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 14–20, March 1989.