Performance Model for Speculative Simulation Using Predictive Optimism

Bradley L. Noble
Roger D. Chamberlain

Bradley L. Noble and Roger D. Chamberlain, "Performance Model for Speculative Simulation Using Predictive Optimism," in *Proc. of 32nd Hawaii Int'l Conf. on System Sciences*, January 1999. (Received Best Paper Award in the Software Technology Track.)

Southern Illinois University Edwardsville
and
Washington University

Performance Model for Speculative Simulation Using Predictive Optimism

Bradley L. Noble* and Roger D. Chamberlain†

*Dept. of Electrical and Computer Engineering
Southern Illinois University Edwardsville
Edwardsville, IL

†Dept. of Electrical Engineering
Washington University
St. Louis, MO

Abstract. *Performance models exist that reliably describe the execution time and efficiency of discrete-event simulations executed in a synchronous iterative fashion. These performance models incorporate the effects of processor heterogeneity, other processor loads due to shared computational resources and application workload imbalance. We extend these models to include the effects of an optimistic technique known as speculative computation. This includes modeling the effects of predictive optimism, a technique for improving the accuracy of speculative assumptions.*

1 Introduction

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. In discrete-event simulation, algorithms that perform computation in a speculative manner are generally referred to as *optimistic* algorithms [1].

While both synchronous [2] and asynchronous [3] 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. Lin and Lazowska [4] coined the term “S phenomenon” to describe the observation that speedup curves for an optimistic asynchronous algorithm often have several local minima and maxima. This observation was made over a large set of different simulation applications [3, 5, 6, 7]. 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 sum of the execution times of the slowest processor at 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) have been applied to the synchronous algorithm, while retaining the iterative nature of the algorithm.

We have previously presented empirical evidence that speculative computation reduces the impact that shared computational resources have on simulation performance [8, 9]. In this paper we will present and validate a performance model for synchronous iterative algorithms that includes the effects of speculative computation. Included in this model is the degree to which speculative computations are correct (i.e., what is the impact of predictive optimism).

2 Speculative Computation and Predictive Optimism

The model developed here is not restricted to discrete-event simulation applications, but can be applied to any synchronous iterative algorithm. Figure 1 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.
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 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

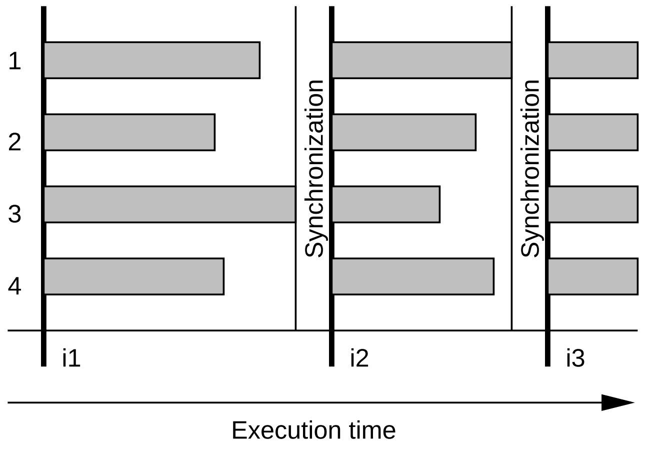

Figure 1: Synchronous iterative algorithm execution. The horizontal bars represent computation during each iteration.

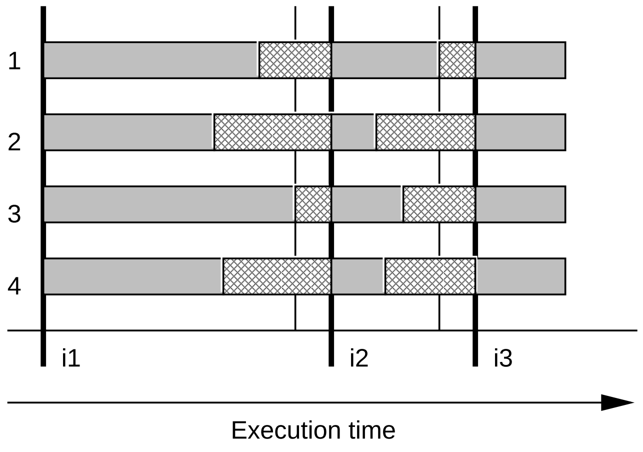

Figure 2: Speculative computation execution time line. The crosshatched areas represent speculative computing, potentially decreasing the computation needed during the following iteration.

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 limitations 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.

Mehl [10] proposed 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 in [8, 9]. Dickens et al. [11] present a performance model for a similar algorithm that predicts performance gains over a purely conservative synchronous algorithm. Steinman's Breathing Time Buckets [2], another synchronous optimistic algorithm, has been implemented in the SPEEDES environment and exhibits good performance on a pair of simulation models (queueing networks and proximity detection).

To support processing during the execution of the barrier synchronization, a fuzzy barrier implementation is used [12]. 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 2, 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. Quantifying this performance improvement is the goal of the model presented in this paper.

Predictive optimism is a technique for improving the accuracy of the guesses used to guide speculative computation. In traditional optimistic discrete-event simulation algorithms, the standard optimistic assumption is that if a message has not arrived on an input channel, none will arrive, and processing can continue assuming the channel is unchanged.

In predictive optimism, information theoretic techniques are used to improve the accuracy of this assumption. A predictor is placed an each input channel, and the predictor retains historical information about the messages on the channel. If a message has not arrived on a channel, the predictor can be interrogated to determine if (and when) a message is likely to arrive. If the answer is no, processing proceeds as before. If the answer is yes, speculative computation is suspended, since the results are likely to be discarded, and the bookkeeping overhead of managing the speculation can be diminished.

We have previously investigated predictive optimism in the context of VLSI systems simulation [13]. In that study, the "no message arrival" assumption was compared to a first order finite state predictor and an incremental parsing predictor based on Lempel-Ziv data compression techniques [14]. Both predictors significantly decreased (or eliminated) cases where the standard assumption performed poorly (i.e., was wrong most of the time). The analytic model presented here includes mechanisms for evaluating the performance implications of variations in prediction accuracy, in order to investigate the usefulness of predictive optimism techniques.

3 Definition of Variables

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.

Table 1: Parameters for performance model

Parameter	Definition
R_P	application run time with P processors
P	number of processors
I	number of iterations
t_s	serial computation per iteration
t_p	parallel computation per iteration
t_{ov}	parallelism overhead per iteration
$w_{i,j}$	time to complete parallel work during iter. i on proc. j, no spec.
$\hat{w}_j$	random variable representing work on proc. j each iter., no spec.
$v_{i,j}$	time to complete parallel work during iter. i on proc. j, with spec.
$\hat{v}_j$	random variable representing work on proc. j each iter., with spec.
$s_{i,j}$	time available to speculate during iter. i on proc. j
$\hat{s}_j$	random variable representing available spec. time on proc. j each iter.
r	speculation success ratio: fraction of spec. comp. that is correct

4 Model Development

The time required to complete an iteration in a synchronous iterative algorithm is a function of three distinct parts. Given any iteration i, a synchronous algorithm has some serial work to be completed (i.e. work that cannot be parallelized). We will denote the time required to complete this work as $t_{s,i}$. Given a set of P processors, each processor will have some assigned workload to be performed in parallel at each iteration. We will denote the time for processor j to complete assigned parallel work at iteration i as $t_{p,i,j}$. The time required to complete the iteration is equal to the time required for the last processor to complete its assigned work during that iteration. This is given by $\max_{1\le j\le P}(t_{p,i,j})$. The last part is the time required to implement the overheads associated with a parallel algorithm. This will be denoted by $t_{ov,i}$. Given I iterations to complete, the run time is expressed by

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

In an effort to remove references to a specific iteration i, a "typical" iteration can be characterized by treating each term as a random variable and using the expected value. The resulting run time is modeled by

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

This model has been shown to be effective for estimating run time for several different types of synchronous iterative algorithms [15]. We extend this model to incorporate both speculative computation and the impact that the speculation success rate has on R_P.

4.1 Workload Characterization

Our model development assumes and our empirical results are based on algorithms with relatively constant $t_{s,i}$ and $t_{ov,i}$ which do not vary significantly between iterations. This is definitely true for the discrete-event simulation applications of interest here. We will focus on characterizing $E\left[\max_{1\le j\le P} t_{p,i,j}\right]$ for both the initial parallel workload without speculative computation and for the resulting workload with speculative computation.

Let us define $w_{i,j}$ to be the time to complete work assigned to processor j during iteration i without speculative computation. In this context, $t_{p,i,j}$ simply equals $w_{i,j}$. With speculative computation, we define $v_{i,j}$ to be the time to complete work on processor j during iteration i. Now $t_{p,i,j}$ is equal to $v_{i,j}$.

The evolution from $w_{i,j}$ to $v_{i,j}$ can be seen by examining a specific iteration, i, of a synchronous algorithm that incorporates speculative computation. As seen in Figure 2, the time to complete parallel work during iteration i is $\max_{1\le j\le P} v_{i,j}$. The time available to speculate on processor j during iteration i is then defined by $s_{i,j} = \max_{1\le j\le P}(v_{i,j}) - v_{i,j}$ where initially, $v_{0,j} = w_{0,j}$. This sets up a recursive formula that relates $v_{i+1,j}$ to $v_{i,j}$.

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

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

Substituting (3) into (4) yields:

$$v_{i+1,j} = w_{i,j} - r\left(\max_{1\le j\le P}(v_{i,j}) - v_{i,j}\right) \qquad (5)$$

The scalar r represents the speculation success rate, or what fraction of time $s_{i,j}$ was used for useful speculations. The above expression accurately describes the case when speculation is limited to one iteration into the future. A similar expression can be developed relating $v_{i+2,j}$ to $v_{i+1,j}$ and $v_{i,j}$ for speculation two iterations ahead.

Although these expressions can be used to empirically evaluate $v_{i,j}$ for a specific instance (e.g., when $w_{i,j}$ is known), it does not provide insight into the usefulness of speculation for arbitrary workload distributions.

4.2 Stochastic Workload Model

We model $w_{i,j}$ with the random variable $\hat{w}_j$ representing the time required to complete work on processor j, independent of iteration i. This means $t_{p,j}$ is equal to $\hat{w}_j$ when speculation is not present. Likewise, we model $v_{i,j}$ with the random variable $\hat{v}_j$ representing the time to complete work on processor j with speculation. In this case, $t_{p,j}$ is equal to $\hat{v}_j$. For the model, we are assuming that $\hat{w}_j$ and $\hat{v}_j$ are i.i.d. In the next section, we discuss the implications when this assumption does not hold.

If a stationary distribution exists for $\hat{v}_j$, Equation (5) must also hold.

$$\hat{v}_j \quad = \quad \hat{w}_j - r\left(E\left[\max_{1\le j\le P}(\hat{v}_j)\right] - \hat{v}_j\right) \qquad (6)$$

We define the random variable $\hat{s}_j$ as the amount of time available on processor j for speculative computation each iteration.

$$\hat{s}_j \quad = \quad E\left[\max_{1\le j\le P}(\hat{v}_j)\right] - \hat{v}_j \qquad (7)$$

While (6) gives conditions that $\hat{v}_j$ must meet, it does not directly support the calculation of its distribution. To derive the distribution of $\hat{v}_j$, an iterative approach is used in an attempt to solve for the fixed point of (6). Using k as the iteration variable, the following expression defines an iteration.

$$\hat{v}_j^{k+1} \quad = \quad \hat{w}_j - r\left(E\left[\max_{1\le j\le P}(\hat{v}_j^k)\right] - \hat{v}_j^k\right) \qquad (8)$$

Initially, $\hat{v}_j^0 = \hat{w}_j$. Convergence is reached when the distributions of $\hat{v}_j^{k+1}$ and $\hat{v}_j^k$ are equal.

Clearly, there is no guarantee that the process described above will converge to a stationary distribution for $\hat{v}_j$. In fact, for some applications, the speculation process itself might be highly unstable, even oscillatory. In the case that it does converge, however, the solution is a viable steady state condition for the parallel computation.

5 Model Validation

The high-level performance model (Equation (2)) has been previously proposed and validated in [15]. Here, we are interested in verifying the correctness of the distribution of the random variable $\hat{v}_j$ and $E[\max_{1\le j\le P}\hat{v}_j]$.

To perform the validation we will compare model results with empirical data from two discrete-event simulation applications, one that closely matches the assumptions made during the model development and one that includes some deviation from those assumptions. In this way, we can both validate the effectiveness of the model and check its sensitivity to the modeling assumptions. All of the results presented here (both modeled and empirical) represent four processor executions ($P = 4$).

The first application is a closed queueing network simulation. The topology is a regular network of the style used in [16] and [17] to investigate the performance of asynchronous algorithms. The queueing discipline is FCFS, the service requirements are exponentially distributed with a specified minimum service time, and the routing probabilities are uniformly distributed to each of the neighboring queueing stations. The second application is a VLSI logic simulation. Here, a gate-level simulation of one of the ISCAS-89 sequential benchmark circuits (s9234) [18] is executed using a unit delay timing model driven with random input vectors.

The validation methodology is as follows. We start with a set of trace data that directly represents $w_{i,j}$, $1 \le i \le I$, $1 \le j \le P$. For the queueing network simulation application, this is synthetically generated to match the known statistics from the application [19], and for the VLSI logic simulation application, it is recorded from a parallel logic simulation execution [20]. Example trace data for the two applications is shown in Figures 3 and 4. The units of work on these plots are simulation events. A normalized histogram of $w_{i,j}$ is presented in Figures 5 and 6. Note, in all of the histogram plots and distribution plots below, the plot is for $j = 1$. The data for the remaining processors is not significantly different. Normalization fixes the sum to one.

To apply the model, we use the empirical histogram data to represent the distribution of the random variable $\hat{w}_j$. Equation (8) is then evaluated numerically (drawing samples from $\hat{v}_j^k$ to develop a histogram approximating the distribution of $\hat{v}_j^{k+1}$) repeatedly until convergence. The resulting distributions, $\hat{v}_j$, are shown in Figures 7 to 10 for two different values for the speculation success ratio, r. For comparison purposes, the empirical data is evaluated using Equation (5) to determine $v_{i,j}$, $1 \le i \le I$, $1 \le j \le P$. Histograms of the resulting workloads are presented in Figures 11 to 14 for two values of the success ratio, r.

Given the above methodology, we can now examine both the effectiveness of the model and the conclusions that can be drawn from the model. First, Figures 3 and 4 illustrate an important distinction between the two applications. The analytic model assumes $\hat{w}_j$ is i.i.d., and $w_{i,j}$ for

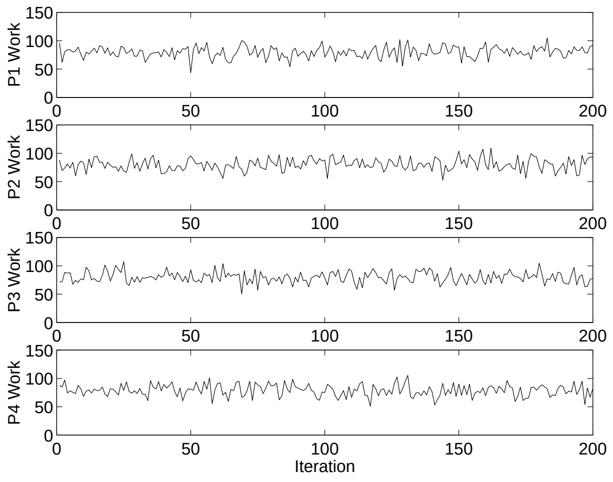

Figure 3: Example trace data for queueing network simulation application (QNS).

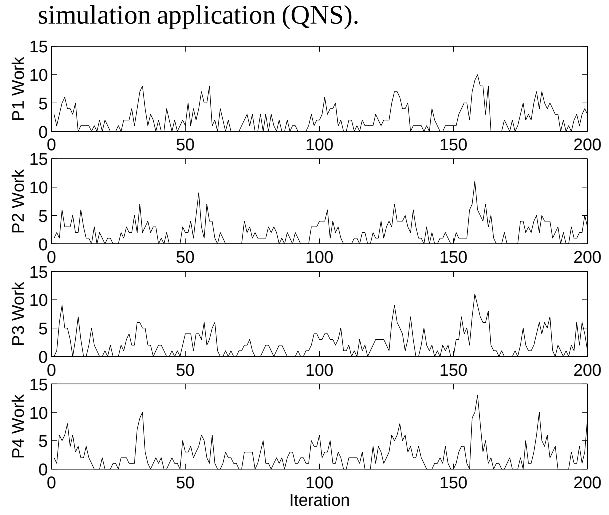

Figure 4: Example trace data for VLSI logic simulation application (VLS).

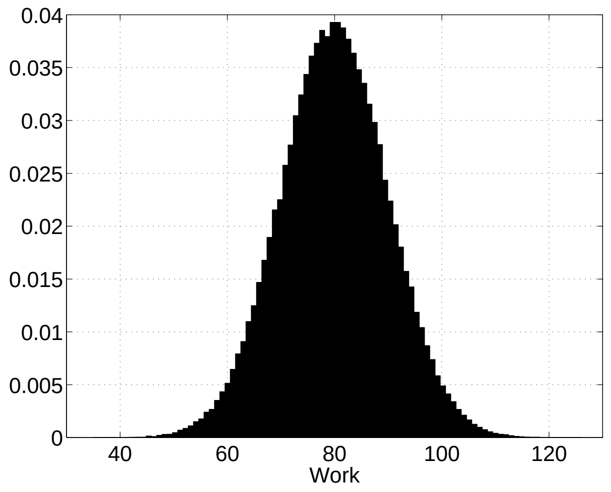

Figure 5: Normalized hist. of $w_{i,j}$ for QNS.

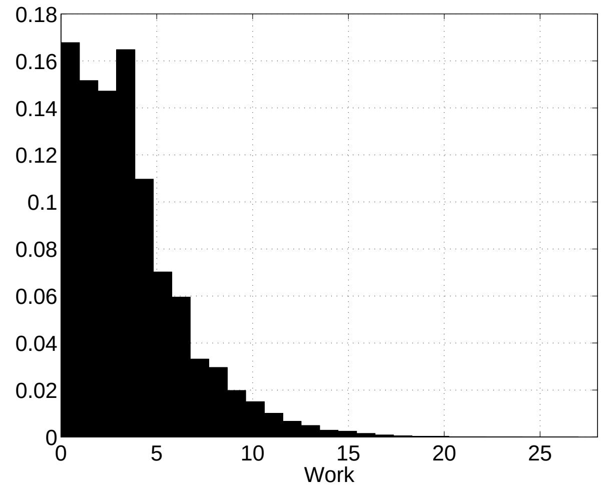

Figure 6: Normalized hist. of $w_{i,j}$ for VLS.

the logic simulation application is clearly highly correlated. Iterations immediately following the clock signal have a high level of simulation activity, while iterations later in the clock period have a lower activity level. This will impact how accurate the model can be for this application.

Second, comparing the initial workload data (Figures 5 and 6) with the model results (Figures 7 to 10) we can see that, in every case, the distribution of the resulting workload moves to the left (representing less work to complete) when speculation is present. The degree of movement can be dramatically affected by the speculation success ratio, r, indicating that techniques such as predictive optimism that attempt to increase r can have a significant performance impact.

Third, the model results match quite well with the empirical results for the queueing network simulation application. (See Figures 7, 9, 11 and 13.) We see a dramatic shift to the left in $\hat{v}_j$ for $r = 1.0$ and a modest shift for $r = 0.5$, with nearly identical results shown in the empirical data. This is our strongest evidence of the effectiveness of the model.

Fourth, the model gives somewhat optimistic results for the logic simulation application. The distribution of $\hat{v}_j$, for both $r = 1.0$ (Figure 8) and $r = 0.5$ (Figure 10), is lower than the empirical results (Figures 12 and 14). The model overestimates the application performance in this case. The cause for this is the fact that the model assumes that the distributions of $\hat{w}_j$ and, therefore, $\hat{v}_j$ are independent across the processors. As can be seen in the trace data (Figure 4), this is not the case for the logic simulation application.

Fifth, the modeled and empirical times to complete par-

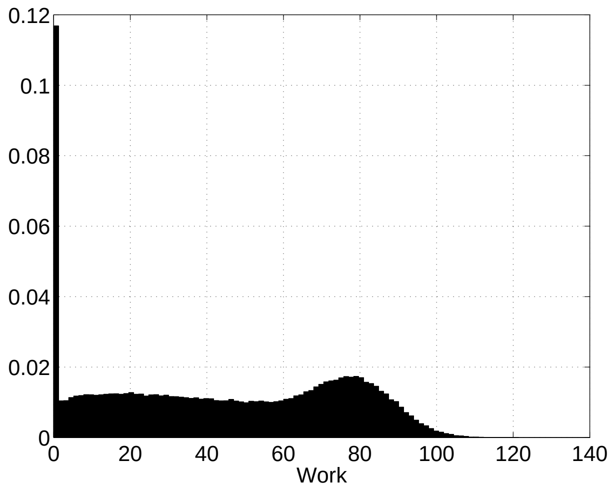

Figure 7: Dist. of $\hat{v}_j$, $r = 1.0$, for QNS.

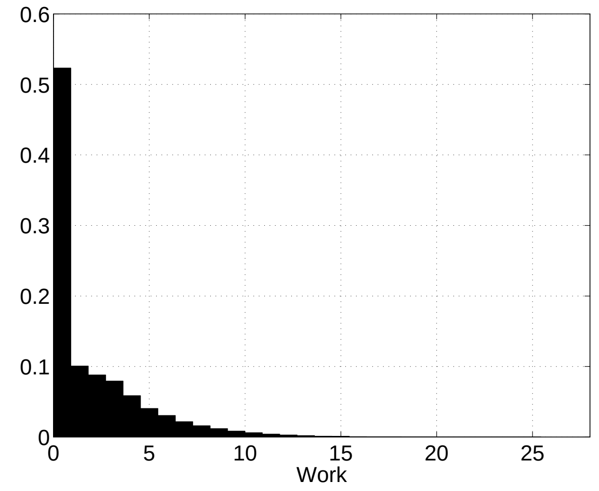

Figure 8: Dist. of $\hat{v}_j$, $r = 1.0$, for VLS.

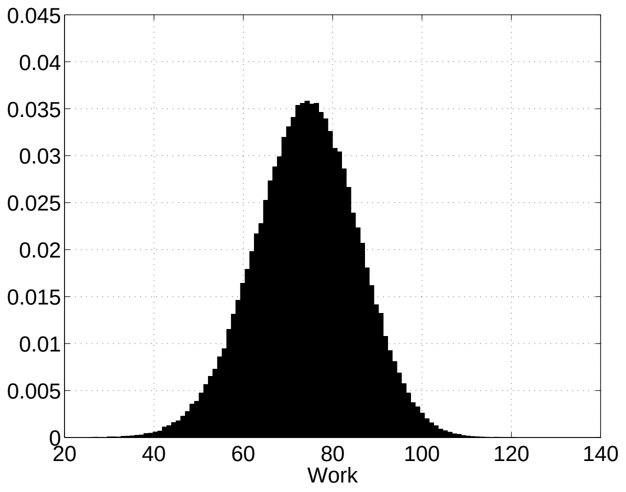

Figure 9: Dist. of $\hat{v}_j$, $r = 0.5$, for QNS.

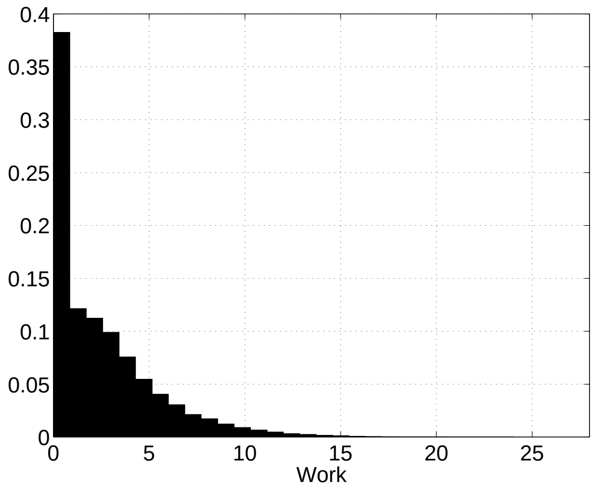

Figure 10: Dist. of $\hat{v}_j$, $r = 0.5$, for VLS.

allel work during an iteration are compared in Table 2. This corroborates the conclusions drawn earlier. The modeled and empirical results match quite closely for the queueing simulation application and the model overestimates the performance for the logic simulation application. One point to note, however, is that even though the model results are consistently optimistic for the logic simulation case, the percent improvement when speculation is present is similar for the model results (37% for $r = 1.0$, 22% for $r = 0.5$) and the empirical results (31% for $r = 1.0$, 17% for $r = 0.5$).

6 Summary and Conclusions

This paper has presented a performance model for synchronous iterative algorithms that includes speculative computation. Discrete-event simulation has been modeled in this fashion, and empirical data from two different simulation applications are used to validate the effectiveness of the model.

One conclusion that can be drawn from the model is the fact that the application performance when using speculative computation can depend heavily on the success rate of the speculation. This lends weight to the need for improved speculation functions, such as predictive optimism.

There are a number of improvements we would like to make to this model. First, the scalar form of the speculation success ratio is somewhat limiting. A random variable model for r could more accurately reflect changes in the distribution of the speculation success. Second, the i.i.d. assumption on the random variables $\hat{w}_j$ and $\hat{v}_j$ is clearly not true in some circumstances. We are currently working on a revised model to predict $\hat{s}_j$ in the case where $\hat{w}_j$ is highly

Application	r	$E\left[\max_{1\le j\le P} \hat{w}_j\right]$	$E\left[\max_{1\le j\le P} \hat{v}_j\right]$	mean of $\max_{1\le j\le P} w_{i,j}$	mean of $\max_{1\le j\le P} v_{i,j}$
Queueing	1.0	90.28	81.10	90.25	81.09
simulation	0.5	90.30	85.95	90.25	85.94
Logic	1.0	6.86	5.01	4.67	3.57
simulation	0.5	6.85	5.63	4.67	3.99

Table 2: Time to complete parallel work

correlated. Finally, additional validation efforts are necessary before the model can truly be counted on to reliably predict performance in cases where empirical data is not available for comparison purposes. We are currently pursuing each of the above items.

Acknowledgements

The research described here was supported in part by the National Institutes of Health under grant GM28719 and the National Science Foundation under grant CTS-9612499.

References

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

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[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.

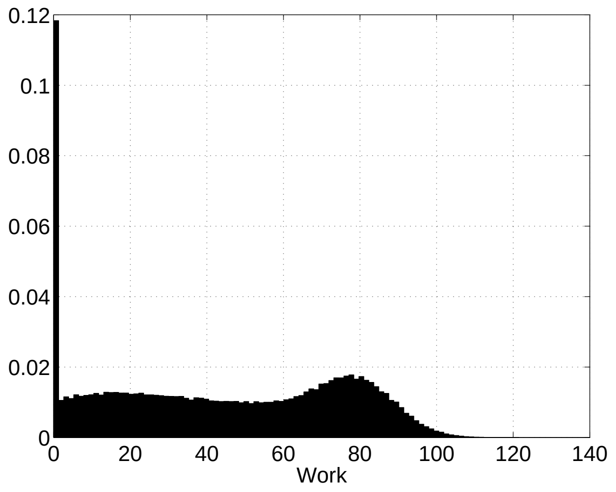

Figure 11: Hist. of $v_{i,j}$, $r = 1.0$, for QNS.

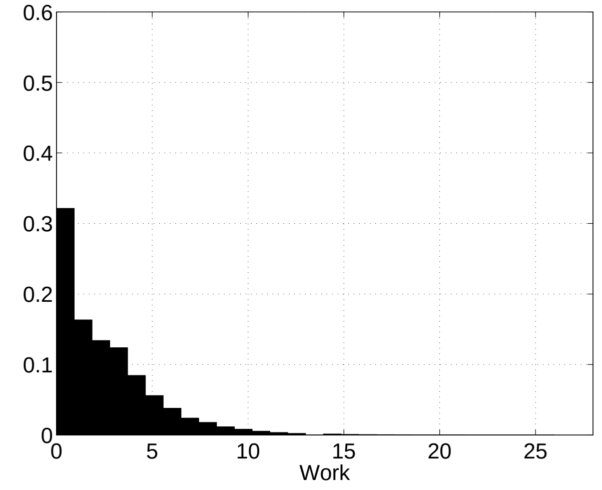

Figure 12: Hist. of $v_{i,j}$, $r = 1.0$, for VLS.

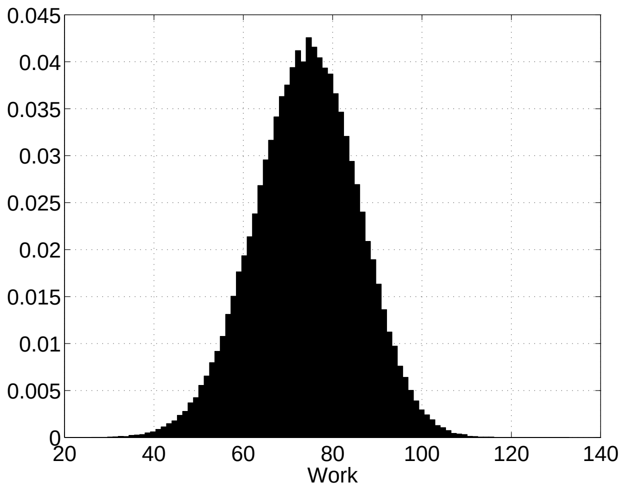

Figure 13: Hist. of $v_{i,j}$, $r = 0.5$, for QNS.

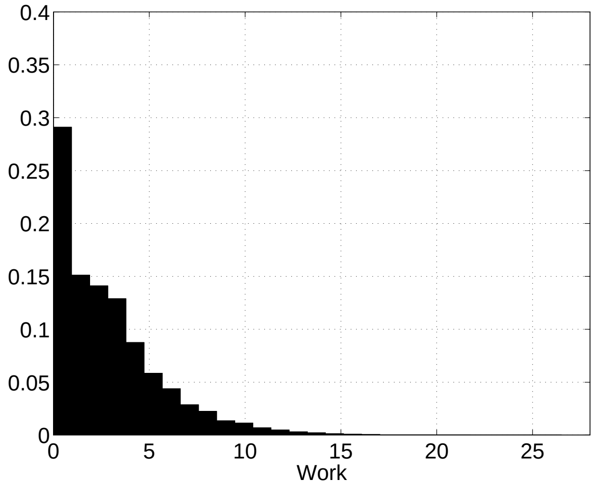

Figure 14: Hist. of $v_{i,j}$, $r = 0.5$, for VLS.

[14] M. Feder, N. Merhav, and M. Gutman. Universal Prediction of Individual Sequences. *IEEE Trans. on Information Theory*, 38(4):1258–1270, July 1991.

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

[16] R. M. Fujimoto. Performance Measurements of Distributed Simulation Strategies. *Trans. of Society for Computer Simulation*, 6:89–132, 1989.

[17] D. M. Nicol. High Performance Parallelized Discrete-Event Simulation of Stochastic Queueing Networks. In *Proc. of the 1988 Winter Simulation Conf.*, 1988.

[18] 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.

[19] G.D. Peterson and R.D. Chamberlain. Beyond Execution Time: Expanding the Use of Performance Models. *IEEE Parallel and Distributed Technology*, 2(2):37-49, Summer 1994.

[20] Y. Chen, B. L. Noble, and R. D. Chamberlain. Comparing Edge-cuts to Communications Volume in Parallel VLSI Logic Simulation. In *Proc. of the 8th IASTED Int'l Conf. on Parallel and Distributed Computing and Systems*, pages 481–484, October 1996.