Bounds and Approximations for Self-Initiating Distributed Simulation Without Lookahead

ROBERT E. FELDERMAN
USC / Information Sciences Institute
and
LEONARD KLEINROCK
University of California at Los Angeles

We provide upper and lower bounds and an approximation for speedup of an optimistic self-initiated distributed simulation using a very simple model. We assume an arbitrary number of processors and a uniform connection topology. By showing that the lower bound increases essentially linearly with P, the number of processors, we find that the optimistic approach scales well as P increases. The model tracks the progress of Global Virtual Time (GVT) and eliminates the need to know the virtual time positions of all processors, thus making the analysis quite straightforward.

Categories and Subject Descriptors: C.4 [**Computer System Organization**]: Performance of Systems—*modeling techniques, performance attributes*; I.6.0 [**Simulation and Modeling**]: General

General Terms: Performance

Additional Key Words and Phrases: Bounds, discrete-event simulation, distributed processing, Global Virtual Time, optimal, optimistic simulation, parallel processing, performance analysis, speedup, Time Warp, virtual time

1. INTRODUCTION

Distributed simulation has become an important technique in the evaluation of complex systems. Various algorithms have been developed to correctly execute a simulation task on multiple processors [5]. Our interest in this work is to examine the performance of one of these algorithms, Time Warp [7], when it runs on many processors.

This work was supported by the Defense Advanced Research Projects Agency under contract MDA 903-87-C0663, Parallel Systems Laboratory

Authors' addresses: R. E. Felderman, University of Southern California, Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292-6695; email: feldy@isi.edu, L. Kleinrock, University of California—Los Angeles, Computer Science Department, 3732L Boelter Hall, Los Angeles, CA 90024-1596; email lk@cs.ucla.edu.

Discrete-Event Simulation (DES) is generally accomplished by partitioning a physical system into logical processes (LP), which communicate by sending and receiving timestamped messages. Each LP might be placed on a separate processor in an attempt to gain speedup proportional to the number of processors used. Unfortunately, ideal speedup is often not attained as the synchronization algorithms that maintain the correctness of a simulation do not come without a performance cost. Early algorithms were dubbed "conservative" techniques due to their use of blocking to maintain correct synchronization between Logical Processes [22]. In an attempt to limit the cost of synchronization, a protocol called Time Warp (TW) was developed by Jefferson. The Time Warp mechanism allows LPs to process messages as they are received; though if more than one message is in the message queue, the one with the minimum timestamp is processed first. If a message arrives that has a lower timestamp than the value of the LP's clock, the LP is rolled back to the time of this message. This is able to be accomplished because the system periodically saves the state of the LP. Any effects of having advanced too far (i.e., erroneous messages) are canceled through an elegant technique using "antimessages."

2. TYPES OF MODELS

Analysis of distributed simulation has lagged the practice for many years. Recently, much important work has appeared in this area. Before discussing this work, we must briefly describe the types of models used.

Message-initiating models. A message-initiating model is one in which a logical process performs no work unless it receives a message from another logical process. The best examples are queueing networks where each server/queue has nothing to do until the arrival of a message from another server. The messages in the simulation correspond to the customers in the queueing system. Messages carry the work in this class of systems. Generally, the system starts with a set of messages "preplaced" in certain queues. The system progresses by processing these messages and generating new ones. This type of system is not unlike a data flow model and most accurately models the original definition of the Time Warp algorithm [7].

Self-initiating models. Another type of simulation is the *self-initiating* model [23]. This system is one where each logical process performs work independent of whether it has received any messages from other logical processes. Here, we find that messages do not carry work; rather, they merely provide some sort of state information. This information is necessary to allow the receiver to accurately execute some event, but does not cause the event to be executed. The example used in [23] is the Ising Spin model [17]. In this system, each logical process models a particle, which randomly and independently decides to modify its state (say at simulated time v). Its new state is a function of the states of neighboring particles at time v. Messages passed between LPs convey state information; they do not cause the processor to reevaluate its state.

Hybrid models. Some systems contain elements of both message-initiating and self-initiating models. A good example of this is a trace-driven multiprocessor cache simulation [16, 23]. Each logical process simulates operations (reads and writes) of one physical processor's cache, and each proceeds independently of the other LPs. A message is sent to other logical processes whenever a reference is made to global memory. The arrival of a message at a logical processor does not cause reads or writes; rather the state of the cache might have to be updated (invalidation of entries). Processors can proceed without the receipt of any messages. When a message does arrive, the LP must perform some work. This differentiation between models becomes important when discussing analytical work.

3. PREVIOUS WORK

Our work focuses on the performance evaluation of Time Warp for self-initiating models. Lavenberg, Muntz, and Samadi [11] provided the first analysis for a TW system using a two-processor self-initiating model. They developed an approximation for speedup, which was valid if the interaction between the processors was small. Mitra and Mitrani [21] also examined a self-initiating two-processor model and developed an exact formula for the distribution of separation in virtual time between the two processors and for the rate of progress in simulated time per unit real time of the two-processor system. Kleinrock and Felderman [4, 9, 10] unify the previous two models and extend their own to include costs for state saving, rollback, and message queueing in [3]. Madisetti [19, 20] provides bounds on the performance of a two-processor self-initiating system where the processors must have different speeds of processing and move at constant (deterministic) rates. Madisetti extends his model to multiple processors, something not done in any of the previously mentioned work.

Lin and Lazowska [15] have examined Time Warp and conservative methods by using critical-path analysis. Also, they have examined TW itself [13, 14, 16] to understand better the state-saving overhead, rollback mechanisms, and processor-scheduling when running the TW algorithm.

Multiple processor Time Warp analysis is appearing only recently. Gupta et al. [6] solve for the performance of Time Warp with multiple, homogeneous, message-initiating processors. The authors use a Markov chain approach where the state variable is the number of completed, but uncommitted, events at a processor. An approximate solution is obtained and is shown to be quite accurate over a wide range of parameters.

Complementary work on a self-initiating (rather than message-initiating) model was done by Nicol [23]. He provides upper and lower bounds on the optimal performance of multiprocessor self-initiating models, an upper bound on Time Warp performance, and a lower bound on the performance of a new conservative protocol. This work is most akin to ours and uses a similar model. A more thorough discussion of the relationship between Nicol's model and ours appears at the end of Section 4.

Chang and Nelson develop extensive analysis of a model similar to ours in [1]. Their model uses blocking synchronization instead of rollback synchro-

nization, but since neither our model nor theirs includes extra costs for the synchronization, the results for efficiency/speedup will be the same. Their work concentrates on showing that efficiency of the processors is lower bounded away from zero as the number of processors increases. They are interested most in understanding how the interconnection topology affects performance. They develop an excellent lower bound for systems with general interconnection topologies. The best lower bound we develop is derived using similar techniques to those in their paper. We are able to provide tight upper bounds and excellent approximations in our work, something not addressed in the Chang and Nelson paper. Finally, since they analyze a more general system their analysis is quite involved. A major contribution of this paper is its mathematical and analytical simplicity.

4. DEFINITION OF THE MULTIPLE PROCESSOR SYSTEM

The multiple processor model for Time Warp is a straightforward extension to the model we developed in [4] and is similar to the model introduced by Nicol [23]. There are P processors in our system with each processor maintaining its own local clock. Processors communicate by sending timestamped messages. Each processor will take an exponentially distributed amount of time (with mean one) to complete processing an event. After completion of the event, the processor will advance its local clock (virtual time) by exactly one unit. Processors make single steps forward in virtual time at each advance, and virtual times are restricted to the integers. After a processor advances, it sends a Message to exactly K processors uniformly chosen from the other $P - 1$ processors; K is defined as the fan-out of the model [23]. The message is timestamped with the virtual time of the sender *after* it advances. Therefore, the system has no lookahead which means that a processor/process is unable to predict its future behavior or output. Messages are used only for synchronization and do not carry work. Since our interest is in the cost of synchronization, messages that arrive in the virtual time future of a processor are ignored. Messages that arrive in the past cause a rollback. With more than two processors comes the possibility for "cascading" rollbacks. Each processor must maintain a queue of messages that it has sent to other processors. If processor P_1 is forced to rollback to virtual time v, then it must "cancel" any messages it sent to other processors with virtual time greater than v. These cancellation messages may potentially cause a rollback at the receiving processor, forcing it to send its own cancellation messages, etc., Receivers only rollback to the time of the erroneous message; they do not necessarily rollback all the way to virtual time $v + 1$. A fairly complex analysis of the potential performance degradation due to cascading rollbacks may be found in [18]. We assume that messages are transmitted instantaneously in real time; there is no communication delay between sending and receiving a message.

The performance measure of interest is speedup and is measured as the rate of virtual time progress of the P processor system divided by the rate of a single processor. We assume that all processors take an exponential amount

of time with mean one to complete an event. The single processor moves at rate one. In the next few sections we calculate bounds on the rate of virtual time progress of this system. We note here that since our model for Time Warp does not include costs for state saving and rollback our solutions are actually bounds on optimal execution of this distributed simulation. If Time Warp does not pay any penalty for its optimistic execution, it will be optimal since it never blocks unnecessarily.

In comparison to our model, Nicol's model has a deterministic real time to complete an event and a distribution on the advance in virtual time after finishing the event. This is exactly complementary to our model, which has a distribution on the real time it takes to complete processing an event and a deterministic single step forward in virtual time after completing the event. Further, Nicol's model for Time Warp includes costs for state saving and rollback; our model neglects these costs.

The model is motivated by several applications that would behave like the self-initiated system we are studying. One example [24] is the optimistic simulation of an SIMD architecture. The instruction counter is the virtual time of a given processor. Each processor can advance forward executing instructions by making guesses for the value of any variables requested from processors that have not yet advanced as far. When the lagging processor finally generates the needed value, it sends messages to all processors that needed that value. They will all be rolled back to the instruction after the generation of the needed value. A second example, that comes from [1], is the execution of an iterative algorithm (e.g., a coupled set of equations) on a parallel processor. An equation may need values from a previous iteration of other equations. If we allow processors to guess these values, then roll back once the values are available, we produce precisely the model studied in this paper. In the two examples just given, one may allow the processors to block instead of optimistically executing forward in time. Since our model includes no cost for rollback, the processor will eventually return to this virtual time when the lagging processor sends it a message, thus, blocking and optimistically executing result in the same speedup. It is easier to analyze the model when thinking of the optimistic approach. Finally, the aforementioned Ising Spin model described in detail in [17, 23] is also an appropriate example.

5. TRACKING GLOBAL VIRTUAL TIME ADVANCEMENT

One technique for measuring the progress of this system is to track the progress of Global Virtual Time (GVT) [7]. GVT is defined as the minimum virtual time of all local clocks in the system and of any messages in transit. Since our model assumes that there is no real-time communication delay in sending a message, there will never be any messages in transit, and GVT is simply the minimum timestamp in the system. Since all virtual-time advances are discrete (i.e., the processors lie on discrete points on a virtual-time axis), there may be several processors sitting at GVT at some point in real time. For example, Figure 1 shows a virtual time snapshot of a ten-processor system at real time t. We can see that processors four and six are sitting at

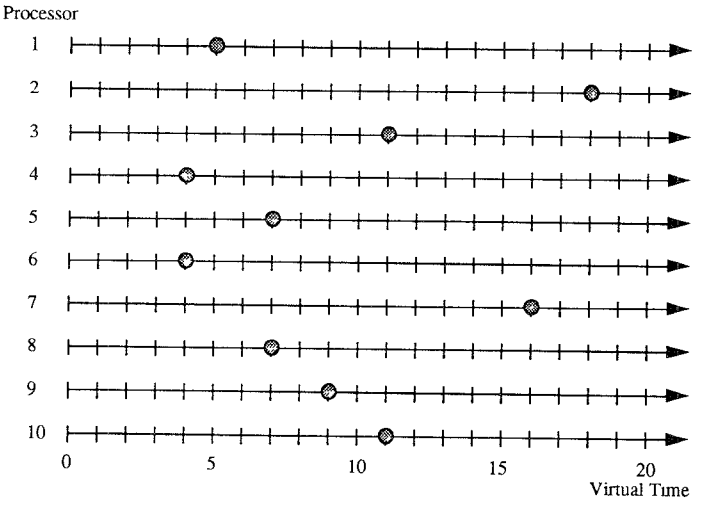

Fig. 1. Snapshot of the virtual-time position of processors at real time t.

GVT = 4. In general, there can be anywhere from one to P processors sitting at GVT at any point in real time. Any processors executing events ahead of GVT are performing speculative computations. The only tasks actually guaranteed to be correct are those with timestamps less than or equal to GVT.

This notion of focusing on GVT is the key to the simple analysis that follows. We can ignore the complicated advances/rollbacks of all the processors that are ahead of GVT. The progress of the furthest behind processors completely determines the progress of the entire system.

If we know the exact number of processors at GVT at any point in real time, we may easily calculate the "instantaneous" rate of progress of the system. With G processors at GVT, we know the distribution of the amount of real time before GVT advances one step (all G processors have to move). When the event-processing time is exponentially distributed at rate one, the expected time to advance GVT is simply the time it takes for the maximum of G exponentials to complete. This is $H[G]$ by first principles. $H[j]$ is the harmonic series $H[j] = \sum_{i=1}^{j} \frac{1}{i}$. If we were able to calculate the distribution of the number of processors sitting at GVT *after* GVT advances, we could derive the average rate of progress of the P processor system and its speedup over a single processor.

The bounds derived in the next sections are created by bounding the number of processors at GVT after GVT advances. If we provide a *lower* bound on the number of processors at GVT, we create an *upper* bound on speedup. Conversely, an upper bound on the number of processors creates a lower bound on speedup. This technique allows our analysis to be extremely straightforward. We do not need to concern ourselves with the location of every processor, only those sitting at (or near) Global Virtual Time. This is a tremendous advantage over more complicated models and analyses.

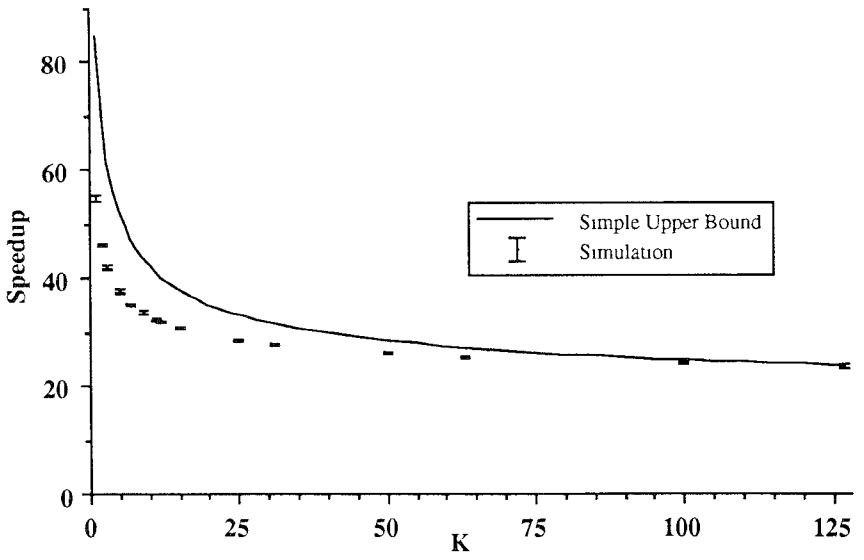

Fig. 2 Normalized speedup versus K for 128 processors.

The derivation of each bound requires a different restriction on the values of P and K. These restrictions will be discussed with each bound.

6. A SIMPLE UPPER BOUND

Certainly, the minimum number of processors at GVT, after GVT advances, will be $K + 1$ since when the last processor moves forward to allow GVT to advance, it will send messages to K processors and pull them back to the new GVT. In general, there will be more than $K + 1$ processors at GVT after GVT advances so our speedup will be less. If there were always $K + 1$ processors at GVT after a GVT advance, then the rate of progress of the system would be

$$\text{Rate} = \frac{1}{H[K+1]};$$

and an upper bound on speedup (S) is thus

$$S \leq S_u = \frac{P}{H[K+1]} \tag{1}$$

An excellent approximation for $H[j]$ which removes the summation and allows for noninteger values of j is [8]

$$H[j] \approx E_c + \ln(j) + \frac{1}{2j} - \frac{1/12}{j(j+1)}$$

where E_c = Euler's Constant ≈ 0.57722.

In Figure 2, we show this analytic bound on speedup versus K for 128 processors as well as the speedup values derived from a simulation of our

model. We simulate our P processor model since we are merely trying to verify how well our approximations match our model. We are not comparing the approximations to an actual implementation of Time Warp or any other simulation mechanism. As might be expected, the upper bound is not very tight when K is small, but is exact for $K = P - 1$ and quite tight for $K/P > 1/2$.

When the last processor advances, there must be at least K processors that already jumped. Therefore, $P - 1 \geq K$ or $P \geq K + 1$. Also, we must have $P \geq 2$ or the model does not make any sense.

7. A BETTER UPPER BOUND

Using the technique above, we can "iteratively" improve the upper bound. Instead of looking only after the last processor advances, we can concentrate on what happens as the last two processors advance from GVT. After the penultimate processor advances there will be at least K processors sitting at GVT + 1. Often, there will be $K + 1$ processors at GVT + 1 unless the penultimate processor sends a message to the one processor still sitting at GVT. We can calculate how many of these processors are still at GVT + 1 when the final processor advances.

$$\text{Prob}[\, i \text{ of the } K \text{ procs. remain when last one advances GVT}]$$
$$= \binom{K}{i}\left(\frac{1}{2}\right)^{i}\left(\frac{1}{2}\right)^{K-i}$$
$$= \binom{K}{i}\left(\frac{1}{2}\right)^{K}$$

since each one independently has probability 1/2 of not advancing prior to when the last processor advances. This is due to the exponential assumption on the time for each processor to execute an event. The GVT processor and the K processors at GVT + 1 each have the same distribution for the time to complete processing an event. Therefore, each of the K processors has the same probability 1/2 of still being at GVT + 1 when the last processor advances.

For future reference let us call these remaining processors the *red* processors. In addition to the i *red* processors that remain when the last one advances, the last processor will send a message to K randomly chosen processors. Some of these processors will be part of the red set; others will be forced to rollback. In fact

$$\text{Prob}[\, j \text{ of } K \text{ msg rcvrs. are in the red set} | i \text{ in red set}]$$
$$= \frac{\binom{i}{j}\binom{P-1-i}{K-j}}{\binom{P-1}{K}}.$$

The above derivations had K processors at GVT + 1 after the penultimate processor advanced. This will occur if that advancing processor **does** send a message to the one processor sitting at GVT. This occurs with probability given below:

$$\text{Prob[advancing processor sends a message to proc. at GVT]} = 1 - \frac{\binom{P-2}{K}}{\binom{P-1}{K}} = \frac{K}{P-1}.$$

Therefore, using conditional probability, the final equation for the improved upper bound on speedup is

$$S \le S_{u_2} = \left(\frac{K}{P-1}\right) P\left(\sum_{i=0}^{K}\sum_{j=0}^{i} \frac{\binom{K}{i}\left(\frac{1}{2}\right)^{K}\binom{i}{j}\binom{P-1-i}{K-j}}{\binom{P-1}{K} H[K+i-j]}\right) + \left(1 - \frac{K}{P-1}\right) P\left(\sum_{i=0}^{K+1}\sum_{j=0}^{i} \frac{\binom{K+1}{i}\left(\frac{1}{2}\right)^{K+1}\binom{i}{j}\binom{P-1-i}{K-j}}{\binom{P-1}{K} H[K+1+i-j]}\right) \tag{2}$$

This bound requires that there be K processors that advanced before the penultimate (second-to-last processor). Therefore we have $P - 2 \ge K$ or $P \ge K + 2$.

This technique of looking several processors back before the final one advances from GVT can be extended to more than two processors into the past if K and P satisfy more stringent conditions. In general, we must have $P - i \ge K$ where i is the number of processors used for calculating the bound.

If we look when the third-to-last processor jumps we may create a more accurate, yet more complicated upper bound. When the third-to-last processor jumps there will be at least $K - 1$ processors at GVT + 1. There may be up to $K + 1$ depending again on whether a message is sent to processors sitting at GVT. First,

$$\text{Prob[}i\text{ of the }K-1\text{ procs. remain when penultimate one jumps]} = \binom{K-1}{i}\left(\frac{2}{3}\right)^{i}\left(\frac{1}{3}\right)^{K-1-i}.$$

Since each processor has a 2/3 chance of remaining at virtual-time one when the penultimate processor advances. This value is easily derived from the following example. There are two processors at GVT, call them "A" and "B." A third processor called "C" is sitting at GVT + 1. "C" is representative of any

of the $K-1$ processors at GVT + 1. Each of the processors A, B, and C are equally likely to advance first due to the exponential assumption. The following lists all the possible orderings of task completion time of the three processors:

ABC ACB BAC BCA CAB CBA.

The first four cases are when "C" is still at GVT + 1 as the penultimate processor advances ($4/6 = 2/3$).

Next, as before, we find the number of receivers in the red set when the penultimate processor advances, given that i were in the red set.

$$\text{Prob}[j \text{ of } K \text{ msg rcvrs. are in the red set} | i \text{ in red set}] = \frac{\binom{i}{j}\binom{P-1-i}{K-j}}{\binom{P-1}{K}}.$$

At this point we would have $K + i - j$ processors one step ahead and one at GVT. The remaining calculation is exactly the same as was done for S_{u_2} and is just a matter of conditioning and summing.

These bounds are tedious to calculate. We may simplify the expressions by approximation. The approximations allow us to derive simpler expressions for the bounds, but sacrifice accuracy. We may eliminate one sum from S_{u_2} by noting that the expected number of message receivers not in the red set is

$$E[\text{Number of rcvrs. NOT in the red set} | i \text{ in red set}] = \left(1 - \frac{i}{P-1}\right)K$$

by noting that $i/(P-1)$ of the K processors that receive a message will be in the red set. We further simplify the bound by assuming that, on average, one half of the $K + 1$ or K processors left after the penultimate processor jumps will remain as red processors when the final processor jumps. Finally, we use the average number of processors at GVT after the penultimate processor advances, which is found from

$$\begin{aligned} &E[\text{Number of procs. at GVT after penultimate proc. advances}] \\ &= (K+1)\left(1 - \frac{K}{P-1}\right) + K\left(\frac{K}{P-1}\right) \\ &= 1 + \frac{K(P-2)}{P-1}. \end{aligned}$$

Therefore, the approximate upper bound is

$$S_{u_2} \approx S_{u_2'} = \frac{P}{H\left[1 + K\left(1 - \dfrac{1 + \dfrac{K(P-2)}{P-1}}{2(P-1)}\right) + \dfrac{1 + \dfrac{K(P-2)}{P-1}}{2}\right]}. \quad (3)$$

Of course, these approximations lead to inaccuracy in the upper bound. These errors are most apparent for small K. Figure 3 lists the values of S_{u_2} and $S_{u_2'}$ for $1 \leq K \leq 10$ and $P = 256$. We see that the approximation is within one percent for $K \geq 4$ ($K/P = 4/256 \geq 0.015$). So it appears that the approximations yield results that are sufficiently close to the true values.

In order for these approximation techniques to be exact, the following would need to be true,

$$f[E[x]] = E[f[x]],$$

meaning that the expectation of a function of x is equal to the function of the expectation of x. In general this is not true, but is fairly accurate for our function $H[x]$ for large enough x. Unfortunately though, the approximate upper bound is smaller than the true upper bound. Therefore, the approximations are not *proper* upper bounds, rather approximate upper bounds. Where accurate, true upper bounds are needed, we must use the exact formulas.

Pressing on with the approximation method, we see that the "three-step" upper bound may be approximated in the same fashion. When the third-to-last processor jumps, on average, there will be approximately $1 + K((P - 3)/(P - 1))$ processors at GVT + 1. When the next processor jumps there will be, on average, 2/3 of those processors in the red set and

$$\left(1 - \frac{2K(P-3)}{3(P-1)}\right)$$

processors pulled back by the second to the last processor. Of these processors, approximately 1/2 will remain as red processors when the final processor jumps.

Generalizing this technique leads to the following set of recursive equations that generate a lower bound on the number of processors at GVT after GVT advances by calculating from the Max to last processor to jump.

$$n[i] = 1 + K\left(\frac{P - \text{Max}}{P-1}\right) \quad i \geq \text{Max} \tag{4}$$

$$n[i] = \frac{n[i+1]i}{i+1} + 1 + K\left(1 - \frac{n[i+1]i}{(i+1)(P-1)} - \frac{i-1}{P-1}\right) \quad i < \text{Max} \tag{5}$$

Equation 5 may be explained as follows. The first term is the expected number of processors remaining in the red set. The "1" is the advancing processor. Finally, the last term is the expected number of new processors rolled back by the advancing processor. At most, there would be K processors. Then we subtract the expected number of message receivers in the red set and the expected number of message receivers still sitting at GVT. If we

K	S_{u_2}	$S_{u'_2}$	% diff
1	143.27	139.59	2.57
2	118.91	117.11	1.51
3	105.76	104.67	1.03
4	97.30	96.55	0.78
5	91.29	90.72	0.62
6	86.73	86.28	0.52
7	83.12	82.76	0.44
8	80.18	79.87	0.38
9	77.71	77.44	0.34
10	75.60	75.37	0.31

Fig. 3. Approximation error for the two-step upper bound versus K for 256 processors.

set Max = 1, then we have $n[1] = (K + 1)$, our original approximation. The approximation for several values of Max is shown below.

$$\text{Max} = 1 \rightarrow n[1] = K + 1 \tag{6}$$

$$\text{Max} = 2 \rightarrow n[1] = 1 + K\left(1 - \frac{1 + \frac{K(P-2)}{P-1}}{2(P-1)}\right) + \frac{1 + \frac{K(P-2)}{P-1}}{2} \tag{7}$$

$$\text{Max} = 3 \rightarrow n[1]$$

$$= 1 + \frac{1 + K\left(1 - \frac{1}{P-1} - \frac{2\left(1 + \frac{K(P-3)}{P-1}\right)}{3(P-1)}\right) + \frac{2\left(1 + \frac{K(P-3)}{P-1}\right)}{3}}{2} \tag{8}$$

$$+K\left(1 - \frac{1+K\left(1 - \frac{1}{P-1} - \frac{2\left(1+\frac{K(P-3)}{P-1}\right)}{3(P-1)}\right) + \frac{2\left(1+\frac{K(P-3)}{P-1}\right)}{3}}{2(P-1)}\right) \tag{9}$$

Some of these stepwise approximations are plotted in Figure 4 for 256 processors, and we see the diminishing returns as we calculate further back from the last processor to advance from GVT.

7.1 An Approximation

When these approximations are used to create an upper bound, the approximation is lower than the exact calculation of the upper bound. Therefore, the

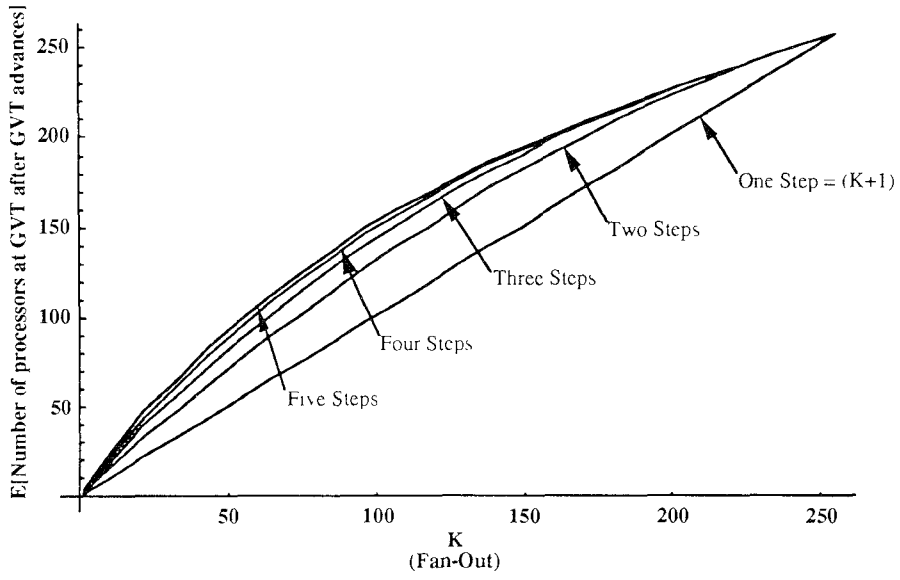

Fig. 4. Bounds on the number of processors at GVT after GVT advances for 256 processors.

approximation is not guaranteed to be an upper bound. But, by using a large enough value for Max, we generate an extremely tight (approximate) upper bound that we may use as an approximation for the exact value of speedup. The equation we will use for the approximation is when Max = 10 (arbitrary choice), the ten-step approximation. Therefore, the speedup approximation is

$$S \approx \frac{P}{H[n[1]_{|\mathrm{Max}=10}]}. \tag{10}$$

8. A LOWER BOUND ON SPEEDUP

We find a lower bound on speedup by overestimating the number of processors at GVT when GVT advances. Too many processors at GVT means the system will advance more slowly than the normal system. Imagine placing all the processors at virtual-time zero and allow them to proceed normally. When the last processor jumps from virtual-time zero to virtual-time one, there would be more processors at virtual-time one than would normally be expected if the system were in steady state. This is the basis for our lower bound on speedup. By ordering the processors by their time of advancing from virtual-time zero to virtual-time one [24], we may easily calculate the number of processors remaining at virtual-time one when the last processor advances from virtual-time zero.

The first processor to advance has probability $1/P$ of still remaining at virtual-time one until the last processor advances. This is due again to the

exponential assumption. When the first processor advances, there will be a single processor at time one and $P - 1$ at time zero. In order for this processor to have stayed at time one (without advancing) until the last processor advances from time zero, it would have to be the last of P processors to advance. The second processor to advance from time zero has probability $1/(P - 1)$ of remaining at time one; the third has $1/(P - 2)$, and so on. When each of these processors advanced each would, at most, pull back K processors. All $K + 1$ processors (K pulled back plus itself) would have the same probability (given above) of remaining until the last processor advanced from virtual-time zero to virtual-time one. Therefore, the expected number of processors remaining when the last one advances is

$$
\begin{aligned}
&E[\text{Num. procs. remaining when GVT advances one step}] \\
&\quad \le (K + 1) \sum_{i=2}^{P} \frac{1}{i} \\
&\quad = (K + 1)\left(\sum_{i=1}^{P} \frac{1}{i} - 1 \right) \\
&\quad = (K + 1)(H[P] - 1).
\end{aligned}
$$

The final processor will pull back an additional K (at most), and adding in itself we find

$$
\begin{aligned}
&E[\text{Number of processors at GVT after GVT advances 1 step}] \\
&\quad \le (K + 1)(H[P] - 1) + K + 1 \\
&\quad = (K + 1)H[P].
\end{aligned}
$$

Therefore, a lower bound on speedup is

$$
S \ge S_l = \frac{P}{H[(K + 1)H[P]]}. \tag{11}
$$

This bound also approximates the expectation of a function of x by a function of the expectation of x. Since this approximation is lower than the true lower bound, the approximation is a proper lower bound.

The bound can be improved by using a technique akin to that found in [1]. Our bound above grossly overestimates the number of processors at GVT after the single step advance. We improve it as follows. Again, order the processors by their departure from virtual-time zero. Now, look at the system when there are still $\lceil P/(K + 1) \rceil$ processors left at virtual-time zero. If we assume that each of these processors will send a message to a different set of K processors when it advances, then these remaining processors will "pull back" all P processors. We calculate how many of the processors will still be

remaining when the last processor advances to virtual-time one in the same fashion as the previous bound.

$$
\begin{aligned}
&E[\text{Num. procs. remaining when GVT advances}] \\
&\quad \le (K+1) \sum_{i=2}^{\left\lceil \frac{P}{K+1} \right\rceil} \frac{1}{i} \\
&\quad = (K+1)\left(\sum_{i=1}^{\left\lceil \frac{P}{K+1} \right\rceil} \frac{1}{i} - 1 \right) \\
&\quad = (K+1)\left(H\left[\left\lceil \frac{P}{K+1} \right\rceil \right] - 1 \right).
\end{aligned}
$$

The final processor will pull back an additional K (at most), and adding in itself we find

$$
\begin{aligned}
&E[\text{Num. procs. at GVT after GVT advances 1 step}] \\
&\quad \le (K+1)\left(H\left[\left\lceil \frac{P}{K+1} \right\rceil \right] - 1 \right) + K + 1 \\
&\quad = (K+1) H\left[\left\lceil \frac{P}{K+1} \right\rceil \right].
\end{aligned}
$$

Therefore, the revised lower bound on speedup is

$$
S \ge S_l = \frac{P}{H\left[(K+1) H\left[\left\lceil \frac{P}{K+1} \right\rceil \right] \right]}. \tag{12}
$$

In Figure 5, we plot simulation data for 128 processors versus the two-step upper bound (Equation 2), the approximation (Equation 10), and the lower bound (Equation 12). Figure 6 shows a close up of the region where K is small ($1 \le K \le 25$). In Figure 7, we show simulation speedup versus number of processors for various values of K. Also included on this plot are the upper bound, approximation, and lower bound. In Figure 8, we show how the bounds scale with an increasing number of processors (P) for various values of K. The important point of this plot is to see that the lower bound increases linearly as P increases. The performance of the system continues to increase significantly as P increases. We can see why this is true by examining the lower bound more closely. Remembering that

$$
H[j] \approx 0.57722 + \ln j + \frac{1}{2j} - \frac{1/12}{j(j+1)}
$$

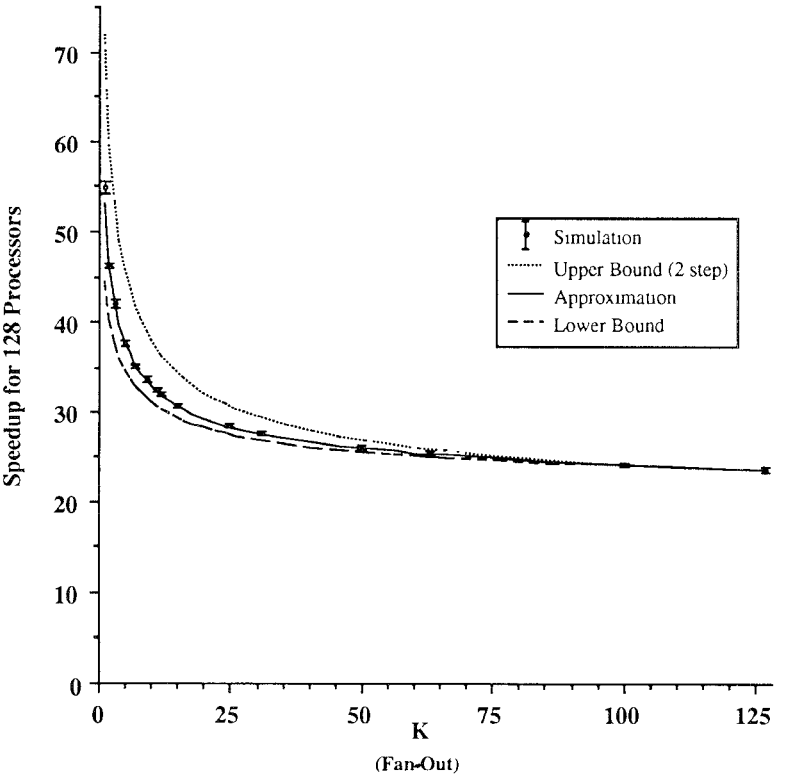

Fig. 5. Bounds on speedup versus K for 128 processors.

we see that for large P the lower bound is essentially

$$S_l \approx \frac{P}{\ln\left((K+1)\ln\left(\frac{P}{K+1}\right)\right)} = \frac{P}{\ln(K+1) + \ln\ln\left(\frac{P}{K+1}\right)}.$$

For K fixed, as P increases S_l is effectively equal to $P/(\text{constant})$. One over that constant is the efficiency. For example, if $P = 65536$ and $K = 10$, we find that $\ln\ln(\frac{P}{K+1}) \approx 2.16$ and $\ln(K+1) \approx 2.40$, the sum is 4.56. Therefore the efficiency is approximately 22%. Lastly, we show the lower bound on efficiency (S/P) versus P and K in Figure 9 and note that efficiency decays fairly rapidly for small K, but flattens out quickly and stays well above zero for large P and K.

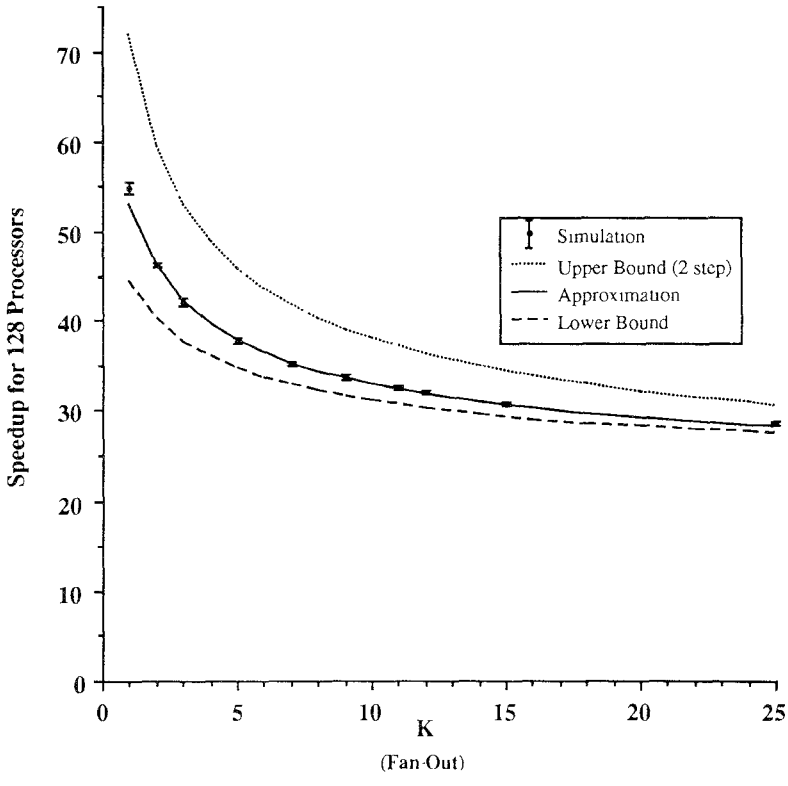

Fig. 6 Bounds on speedup versus K for 128 processors ($1 \leq K \leq 25$).

As noted in [1 (Corollary 2.6)] and [23 (Section 3.1.1)], the lower bound using an exponential distribution is also a lower bound for any distribution that is New Better than Used in Expectation (NBUE). An NBUE distribution is one where the expected residual life is less than or equal to the expectation of the distribution.

9. COMPARISON TO PREVIOUS WORK

9.1 Two Processors

For the limiting case of $P = 2$ and $K = 1$, in previous work [2, 4] we have derived exact solutions for speedup. For the present model the upper bound and lower bound both equal $P/H[2] = 4/3$. As expected, this is the exact value for speedup derived for the two-processor case.

9.2 Nicol's Bounds

It is interesting to compare these bounds to the results developed by Nicol [23]. The difference between the models is that the deterministic and random portions are reversed. His model has deterministic real-time and random

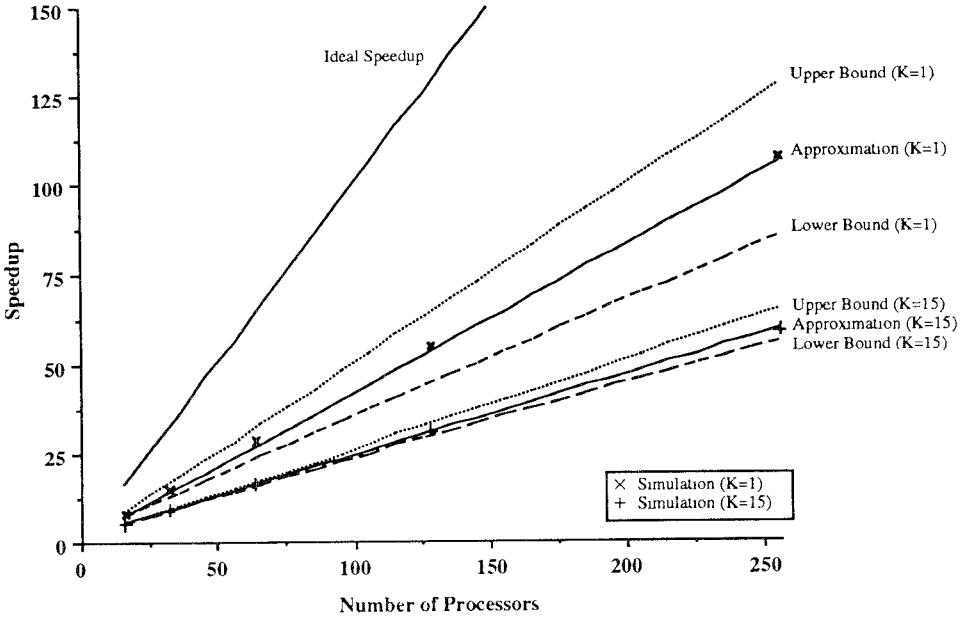

Fig. 7. Simulation speedup with lower and upper bounds.

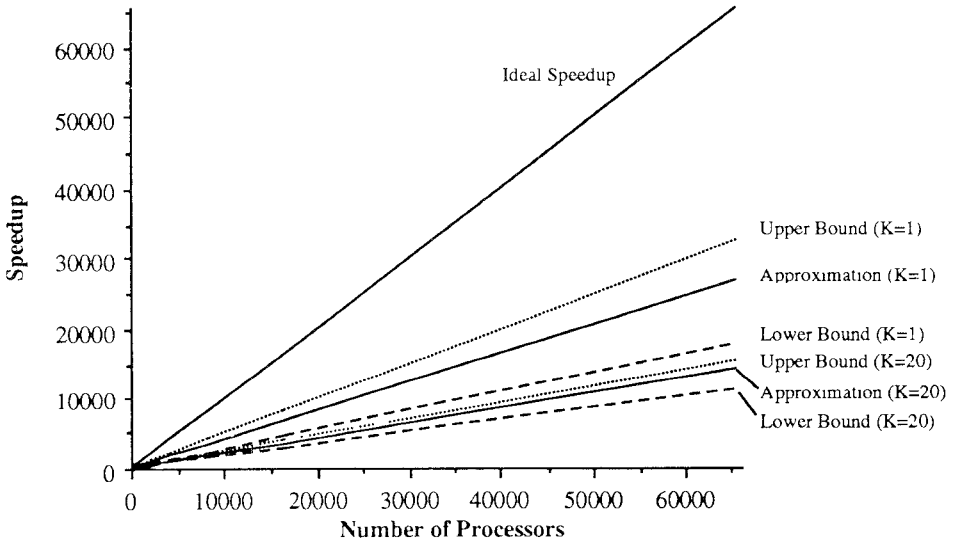

Fig. 8. Bounds on speedup versus number of processors for $K = 1, 20$.

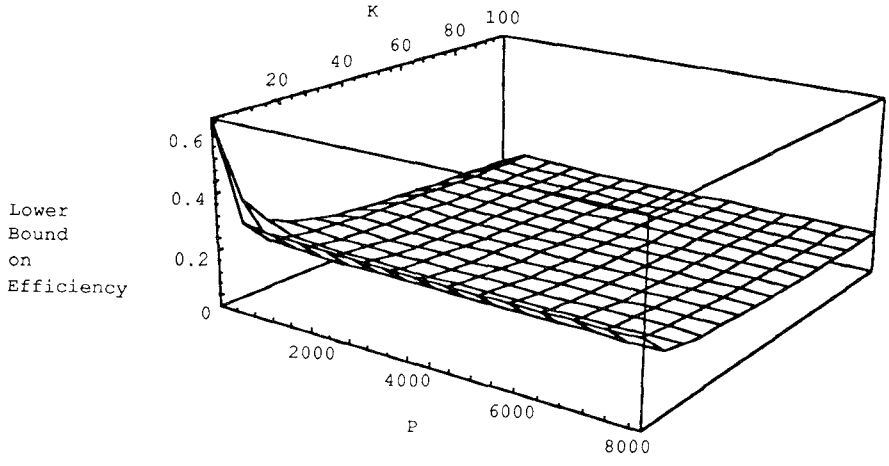

Fig. 9. Lower bound on efficiency versus P and K.

virtual-time increments while our model has random (exponential) real time to process an event and deterministic advances in virtual time. The coarse bounds we generated for speedup are

$$\frac{P}{H\left[(K+1)H\left[\frac{P}{\lceil K+1\rceil}\right]\right]} \leq \text{Speedup} \leq \frac{P}{H[K+1]}.$$

Nicol calculated an upper bound on optimal efficiency of $1/K$ for an exponential distribution on the virtual-time increment. Unfortunately, his lower-bound calculation requires some lookahead in the system whereas ours does not. A comparison is not warranted. Our upper bound on efficiency scales as approximately $1/\ln K$ while Nicol's scales as $1/K$. We show these two upper bounds and our lower bound for $P = 1024$ in Figure 10. Notice that Nicol's upper bound lies below our lower bound.

The basic conclusion is that modelers need to be very explicit about their assumptions and modeling technique before making any claims about the performance of these systems. One might have guessed that our results and those of Nicol would be very similar. That is clearly not the case.

10. CONCLUSIONS

This paper provided upper and lower bounds on speedup for optimistic self-initiating simulation using a new, continuous-time, discrete-state model that does not exploit lookahead. The bounds were derived from a simple understanding of the behavior of the processors determining GVT. To evaluate speedup, we need not be concerned with the advances and rollbacks of processors that are ahead of GVT, thus simplifying the analysis. We also created an approximation for speedup by iteratively improving the upper

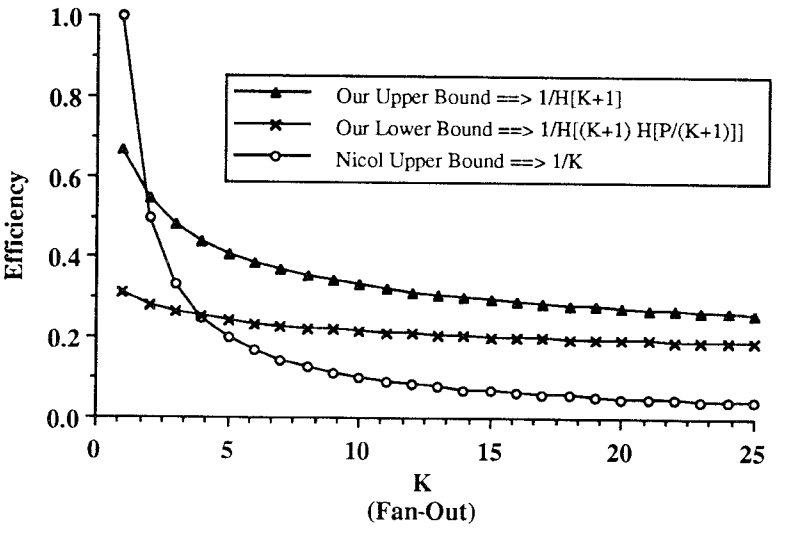

Fig. 10. Bounds on efficiency versus K for $P = 1024$.

bound. The basic bounds on speedup derived in this paper are

$$\frac{P}{H\left[(K+1)H\left[\left[\frac{P}{K+1}\right]\right]\right]} \leq \text{Speedup} \leq \frac{P}{H[K+1]},$$

where P is the number of processors, K is the message fan-out from each processor, and $H[j]$ is the harmonic series $H[j] = \sum_{i=1}^{j} \frac{1}{i}$. The upper bound may be improved in an iterative fashion, and a suitably tight upper bound was used as an approximation. The lower bound shows that speedup will essentially increase linearly as P increases for a fixed fan-out. This implies that the optimistic approach should be able to gain excellent speedup assuming costs for rollback can be kept small.

REFERENCES

1. Chang, C. S., and Nelson, R. Bounds on the speedup and efficiency of partial synchronization in parallel processing systems. Tech. Rep. RC 16474, IBM T. J. Watson Research Center, Yorktown Heights, N.Y., Jan. 1991.
2. Felderman, R. E. Performance analysis of distributed processing synchronization algorithms. Tech. Rep. 910019, Univ. of California, Computer Science Dept., Los Angeles, 1991, Ph.D. dissertation.
3. Felderman, R. E., and Kleinrock, L. Two processor time warp analysis: Capturing the effects of message queueing and rollback/state saving costs. Tech. Rep. 920035, Dept. of Computer Science, Univ. of California, Los Angeles, 1992.
4. Felderman, R. E., and Kleinrock, L. Two processor time warp analysis: Some results on a unifying approach. In *Proceedings of the 5th Workshop on Parallel and Distributed Simulation (PADS91)*, vol. *23*, Society for Computer Simulation, Jan. 1991, pp. 3–10.

5. FUJIMOTO, R M. Parallel discrete event simulation. *Commun. ACM 33*, 10 (Oct. 1990), 30–53.
6 GUPTA, A., AKYILDIZ, I. F., AND FUJIMOTO, R. M. Performance analysis of time warp with multiple homogeneous processors. *IEEE Trans. Softw. Eng 17*, 10 (Oct. 1991), 1013–1027.
7. JEFFERSON, D. R. Virtual time. *ACM Trans. Prog. Lang. Syst. 7*, 3 (July 1985), 404–425.
8. JOLLEY, L. B. W. *Summation of Series*. 2d ed Dover, 1961.
9. KLEINROCK, L. On distributed systems performance. *Comput. Netw. ISDN Syst. 20*, 1–5 (Dec. 1990), 206–215.
10. KLEINROCK, L., AND FELDERMAN, R. E. Two processor time warp analysis: A unifying approach. Tech Rep. 920034, Computer Science Dept , Univ of California, Los Angeles Also *Int J. Comput. Simul.*
11. LAVENBERG, S., MUNTZ, R., AND SAMADI, B. Performance analysis of a rollback method for distributed simulation. In *Performance '83*. North Holland, Amsterdam, 1983, pp. 117–132.
12. LIN, YI-B , AND LAZOWSKA, E. D. A study of time warp rollback mechanisms. *ACM Trans. Model. Comput Simul.* (Jan. 1991), 51–72.
13. LIN, YI-B., AND LAZOWSKA, E. D. Effect of process scheduling in parallel simulation. *Int. J Comput. Simul. 2*, 1 (1992), 107–121.
14. LIN, YI-B., AND LAZOWSKA, E. D. Reducing the state saving overhead for time warp parallel simulation. Tech. Rep. 90-02-03, Dept. of Computer Science and Engineering, Univ. of Washington, Feb. 1990.
15. LIN, YI-B., AND LAZOWSKA, E D. Optimality considerations for "time warp" parallel simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. *22*, 1, Society for Computer Simulation, Jan. 1990, pp. 29–34.
16. LIN, YI-B., LAZOWSKA, E. D., AND BAER, J.-L. Parallel trace-driven simulation of multiprocessor cache performance: Algorithms and analysis. Tech. Rep. 89-07-06, Dept. of Computer Science and Engineering, Univ. of Washington, Seattle, July 1989.
17 LUBACHEVSKY, B. D. Efficient parallel simulations of asynchronous cellular arrays. *Complex Syst 1* (1987), 1099–1123.
18. LUBACHEVSKY, B., SHWARTZ, A., AND WEISS, A Rollback sometimes works . . . if filtered. In *Proceedings of the 1989 Winter Simulation Conference*. 1989, pp. 630–639.
19. MADISETTI, V. K. Self-synchronizing concurrent computing systems Tech. Rep. UCB/ERL M89/122, Electronics Research Laboratory, College of Engineering, Univ. of California, Berkeley, Oct. 1989.
20. MADISETTI, V , WALRAND, J., AND MESSERSCHMITT, D. Synchronization in message-passing computers: Models, algorithms, and analysis. In *Proceedings of the SCS Multiconference on Distributed Simulation*, vol. *22*, 1, Society for Computer Simulation, Jan. 1990, pp. 35–48.
21 MITRA, D., AND MITRANI, I. Analysis and optimum performance of two message-passing parallel processors synchronized by rollback. In *Performance '84*. Elsevier Science, North-Holland, Amsterdam, 1984, pp. 35–50.
22. MISRA, J. Distributed discrete-event simulation. *ACM Comput. Surv. 18*, 1 (Mar. 1986), 39–65
23. NICOL, D. M. Parallel self-initiating discrete-event simulations *ACM Trans Model. Comput. Simul. 1*, 1 (Jan. 1991), 24–50.
24. SHEN, S. The Virtual Time Data Parallel Machine. Ph D. thesis, Univ. of California, Computer Science Dept., Los Angeles, Sept. 1991.

Received September 1991; revised March 1992; accepted May 1992