A Study of Achievable Speedup in Distributed Simulation via NULL Messages

Devendra Kumar and Saad Harous

Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, Ohio 44106

Abstract: Performance study of modern computer and communication systems critically depends on our ability to simulate them with reasonable speed, since these systems are often mathematically intractable. Simulation on a uniprocessor is often unacceptably slow. A promising alternative is distributed simulation, i.e., simulating the system on a distributed system of processors that communicate with each other via messages. Several distributed simulation schemes have been proposed in the literature; but their performance is not well understood. In this paper, we report the results of an experimental study on distributed simulation of two open queuing networks. The distributed simulation scheme considered here is a simple variation of the scheme given by Chandy and Misra using NULL messages. This work presents a new approach to study the relationship between the overhead and performance of a distributed simulator, and illustrates the approach by studying these two example networks. Moreover, this study defines and measures two measures of "ideal" speedup of distributed simulation over sequential simulation. These values of ideal speedup are much less than simply the number of processors, and hence provide a more realistic value for the ideal speedup.

1. Introduction

Several schemes for distributed simulation have been proposed in the literature [2,3,4,15]. The schemes utilize overhead messages to handle the potential deadlock situations that may arise during the simulation.

Unfortunately, only a few performance studies of these schemes are available, e.g., [5,6,7,16,17,18,20]. These studies have provided useful data, and have shown some positive and some negative performance results for a few combinations of the distributed simulation scheme, the system to be simulated, and the available distributed system on which the simulation is to be carried out. Unfortunately, several obvious questions have remained unanswered. For example, what is the relationship between the amount of overhead and the performance of a distributed simulation scheme. Understanding this relationship is important since this can provide useful information as to whether there is much hope in trying to improve the performance by trying to find variations of a scheme that aim at reducing the amount of overhead. For example, suppose the distributed simulation of a particular system using the scheme [2] has low performance. Would it be useful to define and study variations of the scheme that would tend to reduce the number of NULL messages? A related question is how much speedup one can expect in an ideal variation of a distributed simulation scheme that would somehow totally eliminate the effect of overhead messages. Can we expect a reasonable performance in such a simulation? Also, what kind of ideal speedup can be expected if one is considering possibilities across all the well known schemes [2,3,4,15] and their variations.

The known performance studies have not addressed these issues adequately. The usual studies have provided data on the performance of distributed simulation for individual cases, but the above issues have been largely ignored. In some cases where performance of the distributed simulation turns out to be poor, some researchers have suggested that the poor performance is *due to* large overhead [3,5,16,20]. Such a suggestion would normally mean that (i) indeed there is a large amount of overhead, and (ii) if somehow the overhead could be reduced significantly then the performance would be improved significantly. Claiming part (ii) of this statement for a specific case requires further study and such a study is reported here.

Note that it is not quite obvious as to how to

CH2859-5/91/0000/0113$01.00 © 1991 IEEE

study the relationship between performance and overhead. One simple-minded approach would be to define and study different variations of a given scheme which lead to different number of overhead messages, thereby providing some data regarding relationship between overhead and performance. Unfortunately such an approach would have several problems:

1. In coming up with the different variations, one has to ensure that deadlocks are handled adequately, otherwise the scheme would simply be incorrect. This restricts the number of different variations one can come up with.
2. It is usually not feasible to vary the number of overhead messages in a reasonable range so as to enable one to determine the relationship between this number and the performance. In particular, the number of overhead messages may remain large, in which case it is difficult to judge what the performance would be like if this number were somehow reduced to a small value. Note that varying the number of overhead messages in a controlled way, i.e., as an independent variable is nearly impossible.
3. As the different variations are being considered it is quite possible that other activities in the simulator that are not related to overhead are also changing, e.g., the algorithmic rules that determine when an event message should be computed. This makes it harder to judge whether a change in performance is simply due to a change in the number of overhead messages or whether the other factors are also influencing the result.

We discuss here a simple and new approach towards determining the relationship between amount of overhead and performance. In this approach: (i) we *simulate* the distributed simulator instead of directly implementing it on a distributed system and directly measuring its performance, and (ii) in this approach we vary the amount of communication delay and the computation time for each overhead message including the case when these values are zero. Note that this approach is quite different from trying to vary the *number* of overhead messages.

Regarding the issue of "ideal speedup" in distributed simulation, the usual measure for ideal speedup used in the literature is N, which is the number of processors in the simulator. This measure is too conservative in many cases, and as such does not provide much information as to how much speedup one can hope to achieve by variations in the simulation scheme or faster communication of overhead messages. In this work we present two new measures of ideal speedup that would provide more realistic bounds on what can be expected under an ideal distributed simulation.

The above ideas have also been pointed out earlier in [13] where we studied distributed simulation of a particular open queuing network. In this paper we extend that work by further studies of two other open queuing networks. The distributed simulation scheme used is a variation of the scheme given in [2].

In section 2, we define the systems under study. In section 3, we give a brief description of the distributed simulation scheme under study. Definition of the different speedup ratios and description of our implementation are given in sections 4 and 5. The experimental results are discussed in section 6. Finally, concluding remarks are given in section 7.

2. Systems Under Study

Borrowing terminology from [2], in the following a system to be simulated is called the *physical system*. The physical system consists of a network of *physical processes* (or *pps* for short). Each physical system is simulated by a distributed simulator called the *logical system*. The logical system is a collection of *logical processes* (or *lps* for short), each one simulating a corresponding *pp*.

In this paper, we use the term *queuing networks* to refer to networks of *pps* from the five classes - *delay, fork, merge, sink,* and *source,* defined below. This definition is similar to the ones in [3,12], with the exception that in our case fork and merge processes may possibly have only one output or input line respectively. Examples of these five classes of *pps* are *pp*3, *pp*13, *pp*2, *pp*14, and *pp*1 respectively in Figure 1.

1. A *delay process* has one input line and one output line. It processes its input messages in a FCFS queuing displine, and sends each one of them out after a finite service time.
2. A *fork process* has one input line and one or more output lines. Whenever a message arrives on its input line, it sends it out along one of the output lines, after zero delay. The output line on which a message is to be sent, is chosen in a probabilistic manner, according to predefined branching probabilities for the output lines.
3. A *merge process* has one or more input lines, and one output line. Whenever a message arrives along one of its input lines, it sends that message out via its output line after zero delay. If occasionally two or more messages arrive at the same time t, they are not put together in a single message as in [2,12], but rather each one is sent out as a separate message.
4. A *sink process* has one input line, and no output lines. It simply consumes its input messages.
5. A *source process* has no input lines and one output

line. It simply produces messages and sends them out.

Queuing networks are useful in performance modeling of various kinds of service systems such as computer systems, computer-communication networks, telecommunication systems, and manufacturing systems. Here we study the two queuing networks shown in Figures 1 and 2.

3. The Distributed Simulation Scheme Under Study

In this paper we study the scheme based on NULL messages [2]. We made some minor modifications to this scheme in order to improve the degree of concurrency.

3.1 The Original Scheme of Chandy and Misra [2]

The scheme [2] requires that a process sends out NULL messages along the output lines indicating a lower bound on the t-value of its next event message along that line. These overhead messages allow the receiver processes to advance their clocks, thereby preventing deadlock from occurring. These messages do not correspond to any events that take place in the system being simulated.

The scheme [2] is based on the synchronous communication of Hoare's CSP. This can cause a minor performance degradation when implemented on an asynchronous system. Also, in [2], there are unnecessarily rigid rules as to when an *lp* is allowed to send or receive tuples. In particular, an *lp* sends out a tuple on a line only when the line clock for that line becomes minimum among all clock values of all the lines adjacent to the *lp*. This can obviously affect the degree of concurrency. For example, consider a delay *pp* which received a tuple (10,m) and sent out (15,m). Suppose next it receives a tuple (13,m). Now this *lp* is not allowed to process and send this job until the input clock becomes 15. This affects the degree of concurrency between this *lp* and the *lp* connected to its output.

3.2 Modifications

In order to achieve higher level of concurrency we made the following modifications: (i) An *lp* is allowed to send messages whenever possible, rather than having to wait for inputs even when it is possible to generate the next output, (ii) An *lp* can receive inputs any time, except when it is able to send out more outputs, (iii) Message communication in the logical system is assumed asynchronous, (iv) We assume infinite input buffers at the input port of any *lp*.

4. Overall Design of Our Experiments

In our experiments we have a two level simulation. A sequential simulator simulates the behavior of the distributed simulator defined above while the distributed simulator is simulating the physical system (i.e., the queuing network being simulated).

4.1 Simulation Parameters

We first list below the various simulation parameters that were used in our experiments. Any items not listed here should be assumed to have their obvious default values (e.g., the time that an *lp* takes to receive an input message should be assumed to be zero).

- Time up to which the physical system is to be simulated (Z).
- Mean interarrival times of messages at source *pp* (IT).
- Service time at a delay *pp*.
- Branching probabilities at a fork *pp*.
- Communication delay for NONNULL messages (NNCOMDEL).
- Communication delay for NULL messages (NCOMDEL).
- Time to compute a NONNULL message by an *lp*.
- Time to compute a NULL message by an *lp* (NULLTM).

4.2 Measures of Importance

The most important measure of performance is the actual speedup obtained by distributed simulation over sequential simulation, i.e., the ratio ASR = SST/DST of total elapsed times in the two methods of simulation:

SST = Sequential simulation time, i.e., the time taken by a sequential simulator to simulate some physical systems up to time Z.
DST = Distributed simulation time, i.e., the time taken by a logical system to simulate the same physical system up to time Z.

Next we define some terms to capture various notions of "ideal speedup ratio". The first ideal speedup ratio, called ISR1, is defined to be simply N where N is the number of processors in the distributed simulator. This is the usual notion of ideal speedup ratio commonly used in the literature. Next we define more refined notions of ideal speedup ratio.

IDST2 (ideal distributed simulation time) = the time taken by a logical system to simulate the same physical system up to time Z, assuming that: (1) any *lp* has all input NONNULL messages available whenever it needs them in its computations; equivalently, any *lp* has all its input NONNULL messages available to it when simulation starts, (2) Communication de-

lay is zero for both NULL and NONNULL messages (NCOMDEL=NNCOMDEL=0), and time spent in any activity other than computing NONNULL message is zero(thus, NULLTM is assumed to be zero for each *lp*). Then we define the ideal speedup ratio, ISR2, to be SST/IDST2.

Define a distributed simulation scheme to be a *BASIC-related* scheme if it involves simulating each *pp* by an *lp*. The well-known schemes of [2,3,4,15] are *BASIC-related* schemes. Note that the ideal speedup ratio ISR2 indicates how much speedup can be achieved in simulating a given physical system by *any Basic-related* scheme. Also note that in the ideal concurrent simulation corresponding to the ISR2, if each *lp* spends the same amount of total time in processing NONNULL messages, then the ISR2 becomes the same as ISR1.

Next we define ISR3 to capture an even more refined notion of "ideal speedup ratio".
IDST3 (ideal distributed simulation time) = the time taken by a logical system to simulate the physical system up to time Z with the assumption that NULLTM=0 and NCOMDEL=0.
Then we define the ideal speedup ration ISR3 by ISR3 = SST/IDST3. Thus, ISR3 is the same as ASR when communication delay for overhead messages is set to zero and NULLTM is zero for each *lp*.

Note that ISR3 is an upper bound on the speedup ratio only for the given distributed simulation scheme under consideration. It may be possible to achieve higher speedup ratio than ISR3 via some other scheme. The reason is that a different scheme might produce input NONNULL messages earlier than the given scheme under consideration.

Obviously, in different simulation runs to determine ASR, ISR3, and ISR2, if we have the same events in the physical system (e.g., random numbers generated do not change the events of the physical system), then we would have ASR $\leq$ ISR3 $\leq$ ISR2 $\leq$ ISR1.

Our simulation program also records total number of NULL and NONNULL messages. This would be useful in considering whether it is reasonable in a particular case to say that "the poor performance is *due to* large overhead".

4.3 How to Compute the Performance Measures

In any experiment, with appropriate values for simulation parameters, the value of DST is measured directly by our sequential simulator that simulates the distributed simulator. In any such experiment we measure the total number of NONNULL messages sent on each line in the logical system during the distributed simulation. Using these values, we directly compute the sequential simulation time SST. In this computation we ignore the processing time for event list manipulations, and simply add the times taken by *lps* to compute each of their output NONNULL messages. In this way we can compute ASR.

The values of IDST2 are computed by simply measuring, in each of the ASR experiments, the number of NONNULL messages sent out by each *lp*. From these measurements, IDST2 can be directly computed. ISR2 can be computed from IDST2 and SST. Note that as the value of NNCOMDEL is varied, the values of IDST2 and ISR2 remain the same. Similarly, note that the values of NCOMDEL and NULLTM will not affect IDST2 or ISR2 since there are no NULL messages sent out in this case (except for possibly one NULL message on any line at the end of simulation, which we ignore).

The value of IDST3 and ISR3 can be measured in the same way as DST and ASR by choosing appropriate simulation parameters as input.

4.4 Ordering of the Experiments

An "experiment" here refers to a single execution of our simulation program. We ordered our experiments in the following manner. For a given value of the physical and logical system parameters (except for NNCOMDEL), we varied NNCOMDEL from 0 to a reasonably high value. For each such value of NNCOMDEL we conducted two experiments:

- one to determine ASR and ISR2.
- one to determine ISR3 and ISR2 (i.e., NCOMDEL and NULLTM are set to zero). Note that the values of ISR2 in these two experiments would be the same.

Then we repeated the above experiments for different values of physical system parameters.

5. Further Details on Our Experimentation

Our sequential simulator simulating the distributed simulator is implemented on SUN workstations. The sequential simulator mentioned above is written in the language MAY [1]. MAY is a simulation language for simulating distributed systems. It provides facilities for defining parameterized process types, instantiating processes at run time, message communications among processes, a global clock keeping the simulation time, conditional waiting for events, and process termination.

We simulated the two queuing networks shown in Figures 1 and 2. Network 1 is a simple open queuing network. Network 2 is borrowed from [20].

5.1 Physical System Parameters

In our experimentation we varied several physical system parameters for any given network, e.g., mean service times at delay processes, mean interarrival times (IT) at *source* processes, etc.

The service times at every delay *pp*, and the interarrival times of messages at every *source pp* in these experiments were chosen to be exponentially distributed, with the following mean values.

- For a delay *pp* i, it was 3000.0 (this value is assumed in all our experiments).
- The mean interarrival time at a source *pp* is varied so as to result in varying levels of saturation at the delay *pps* in the system. In particular, we considered three cases: (i) non-saturated delay *pps* (less than 80% utilization), (ii) nearly saturated delay *pps* (nearly 100% utilization) and (iii) oversaturated delay *pps*.

The output branches of each fork *pp* have the branching probabilities shown in Figures 1 and 2.

Each physical system was simulated for the physical system time interval [0, Z] where Z is chosen such that a large number of jobs is generated at the source *pp* and a sufficiently large number of messages is sent through each line in the physical system.

5.2 Logical System Parameters

In all experiments, the time to compute one output NONNULL message for any given class of *lps* is: (i) delay : 300, (ii) fork : 90, (iii) merge : 120 and (iv) source : 60.

In Measurements of ASR: The value of NNCOMDEL is varied from 0 to 6400. The value of NCOMDEL is same as NNCOMDEL. The values of NULLTM are: (i) delay : 150, (ii) fork : 45, (iii) merge : 60 and (iv) source : 60.

In Measurements of ISR3: NNCOMDEL here is varied as in case of measuring ASR. By definition, NCOMDEL and NULLTM are zero.

In Measurements of ISR2: In each experiment of ASR or ISR3, we compute ISR2. But note, as mentioned before, that the value of ISR2 is unaffected by changes in NNCOMDEL, NCOMDEL or NULLTM.

For simplicity no times are associated with other activities in the logical system. In particular, no time is associated with receiving a NONNULL message.

6. Simulation Results

Tables 1 and 2 show the main experimental results regarding networks 1 and 2 respectively. We have also conducted additional experiments by varying branching probabilities at fork *pps* in these networks. For the sake of brevity, those results are not shown or discussed here. The interested reader is referred to [9].

Next we state our main conclusions from the data shown in Tables 1 and 2 respectively.

Queuing Network 1:

The three values (in increasing order) of IT in Table 1 correspond, respectively, to oversaturated, nearly saturated, and non-saturated conditions of delay *pps* in the system.

1. For any value of IT, at NNCOMDEL= 0 the value of ASR is fairly high – roughly one less than the number of delays *lps* in the system.

   As NNCOMDEL is increased, the value of ASR goes down - and would become <1 for sufficiently large value of NNCOMDEL.

   Thus, for any given value of IT, there is a range of NNCOMDEL values between 0 and a fairly large value (which depends on IT), such that in this range the value of ASR is high.

   Note that in the experiments to determine ASR, the value of NCOMDEL is the same as the value of NNCOMDEL.
2. As IT is increased, the range of values of NNCOMDEL which gives high ASR values is somewhat decreased.
3. If the value of NNCOMDEL is small, (say between 0 and the time it takes a delay *lp* to process a NONNULL message), then trying to reduce NNCOMDEL further won't significantly improve ASR value.
4. Interestingly, for each entry of the Table the difference between ASR and ISR3 values is negligible. [This is true irrespective of whether ASR values are low or high]. This shows that:
   (a) Reducing NCOMDEL or NULLTM would not have any significant effect on ASR.
   (b) Consider variations of this distributed simulation scheme where one tries to reduce the number of overhead messages. Such variations are unlikely to significantly improve the performance.
5. Consider the cases where performance is poor, i.e., ASR values are near or less than 1. (As indicated above, this happens when NNCOMDEL value is sufficiently high for any given IT value). It would be wrong in these cases to claim that the poor performance is "due to a large number of overhead messages":

   since in these cases ISR3 is also near or less than 1, this indicates that even if, somehow, one could reduce these overhead messages to zero, the ASR value is unlikely to be signifi-

cantly higher than 1.

6. In many cases ISR3 is significantly less than ISR2 or ISR1. This shows that in studying variations of this scheme which try to reduce overhead, ISR3 captures the ideal speedup ratio in a more realistic manner than ISR1 or ISR2.
7. For a large number of cases (with low values of NNCOMDEL), the values of ASR are close to ISR2. Thus, for these cases, the distributed simulation scheme considered is nearly optimal with respect to all *BASIC-related* schemes.
8. The values of ISR2 are less than ISR1 in these experiments. This clearly shows that in studying variations of *BASIC-related* schemes, ISR2 captures the ideal speedup ratio in a more realistic manner than ISR1.

Queuing Network 2:

The three values (in increasing order) of IT in Table 2 correspond, respectively, to oversaturated, nearly saturated, and non-saturated conditions of $pp4$ and $pp5$.

1. For any value of IT, at NNCOMDEL= 0 the value of ASR is fairly high.

   As NNCOMDEL is increased, the value of ASR goes down - and would become <1 for sufficiently large value of NNCOMDEL.

   Thus, for any given value of IT, there is a range of NNCOMDEL values between 0 and a positive value (which depends on IT), such that in this range the value of ASR is high.
2. As IT is increased, the range of values of NNCOMDEL which gives high ASR values is decreased.
3. Obviously in several cases NNCOMDEL has a significant impact on ASR (the impact is higher when NNCOMDEL is large or when IT is large). Therefore in these cases a good way to improve performance is to cut down on NNCOMDEL.

4-6. These points are the same as points 4-6 in the case of network 1.

7. For some cases (when values of NNCOMDEL and IT are both low), the values of ASR are close to ISR2. Thus, for these cases, the distributed simulation scheme considered is nearly optimal with respect to all *BASIC-related* schemes.
8. This point is the same as point 8 in the case of network 1.

7. Conclusion

In this paper we have discussed a new approach to study performance of distributed simulation. This approach aims to answer several questions that have remained unanswered previously – relationship between overhead and performance, ideal speedups, and relationship between the logical system parameters and performance. Also, it clarifies some statements previously made by researchers regarding the relationship between overhead and performance of distributed simulation.

The essential elements of the approach are: (i) simulate the distributed simulator, which makes it easy to vary parameters such as communication delays, computation times etc. (ii) study the relationship between overhead and performance by varying computation and communication times for overhead messages, and (iii) define and measure certain notions of ideal speedup (i.e., ISR2 and ISR3) that are more realistic than simply the number of processors.

We have illustrated the usefulness of our approach by distributed simulation of two open queuing networks.

In our simulation study of the two networks, we identified several cases where both ASR and ISR3 are near or less than 1 – thus the performance is poor, but it would remain poor even if the effect of overhead were to be eliminated. In other words, trying to reduce the number of overhead messages would not be enough to significantly improve performance. Thus it would be incorrect in these cases to say that the poor performance is *due to* large overhead.

By using our approach, we also showed that in many cases, the scheme considered is nearly optimal with respect to all *BASIC-related* schemes.

Also, our experimental study pointed out cases where trying to reduce NNCOMDEL would be particularly helpful and where it won't.

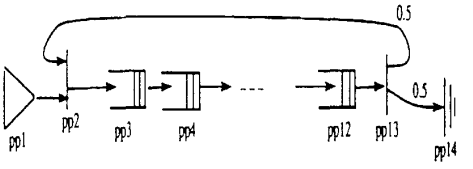

Figure 1: Queuing Network 1.

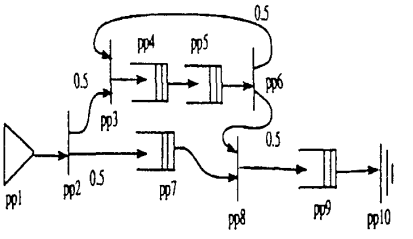

Figure 2: Queuing Network 2.

IT	NSCOMDEL→	0	200	400	800	1600	3200	6400	ISR2
5000	ASR	8.9575	8.6601	8.1991	7.0784	5.0626	2.8578	1.4980	9.1213
	ISR3	9.0015	8.7527	8.3559	7.2658	5.2355	2.9631	1.5521	
6000	ASR	9.4554	9.0382	8.3086	6.6750	4.3297	2.3548	1.2314	9.6788
	ISR3	9.5064	9.1280	8.4719	6.8537	4.4763	2.4303	1.2697	
8600	ASR	9.6445	8.1721	6.4826	4.2254	2.4665	1.3378	0.6972	10.2013
	ISR3	9.7071	8.2682	6.5969	4.2965	2.4665	1.3697	0.7147	

Table 1: Simulation Results For Queuing Network 1.

IT	NSCOMDEL→	0	200	400	800	1600	3200	6400	ISR2
2000	ASR	5.5704	5.3779	5.1753	4.7162	3.8064	2.5991	1.4929	5.6780
	ISR3	5.5980	5.4690	5.3318	4.9585	4.1422	2.9345	1.7080	
3000	ASR	4.7987	4.3459	3.7355	2.7131	1.5787	0.8469	0.4394	5.0253
	ISR3	4.8906	4.5593	4.0209	2.9824	1.7539	0.9436	0.4906	
4300	ASR	3.2679	1.8972	1.2998	0.7911	0.4436	0.2362	0.1220	5.0142
	ISR3	3.6197	2.2788	1.6036	0.9971	0.5675	0.3048	0.1583	

Table 2: Simulation Results For Queuing Network 2.

References

[1] R. Bagrodia, "May: A Process Based Simulation Language", *Master's Report*, Dept. of Computer Sciences, University of Texas at Austin, Austin, Texas, 1983.

[2] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study In Design And Verification of Distributed Programs", *IEEE Transactions on Software Engg.*, SE-5(5):440-452, September 1979.

[3] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations", *Communications of the ACM*, Vol. 24, No. 4, pp.198-205, April 1981.

[4] D. R. Jefferson and H. A. Sowizral, "Fast Concurrent Simulation Using The Time Warp Mechanism, Part I: Local Control", *Technical Report*, The Rand Corporation, Santa Monica, California, December 1982.

[5] R. M. Fujimoto, "Performance Measurements of Distributed Simulation Strategies", *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 14-20, Feb. 3-5, 1988.

[6] R. M. Fujimoto, "Lookahead in Parallel Discrete Event Simulation", *Proceedings of the 1988 International Conference on Parallel Processing*, pp. 34-41, August 1988.

[7] R. M. Fujimoto, "Time Warp on a Shared Memory Multiprocessor", *Proceeding of the 1989 International Conference on Parallel Processing*, pp. 242-249, August 8-12, 1989.

[8] B. Groselj and C. Tropper, "Pseudosimulation: An algorithm for distributed simulation with limited memory", *Int. J. Parallel Programming*, 15, 5, pp. 413-456, 1987.

[9] S. Harous, "Studies in Distributed Simulation", *Ph.D. Dissertation*, Department of Computer Engineering and Science, Case Western Reserve University, Cleveland, Ohio.

[10] M. Krishnamurthy, U. Chandra, and S. Sheppard, "Two approaches to the implementations of a distributed simulation system", *Proceedings of the 1985 Winter Simulation Conference*, San Francisco, California, 435-443, December 1985.

[11] D. Kumar, "Simulating Feedforward Systems Using a Network of Processors",*Nineteenth Annual Simulation Symposium*, Tampa, Florida, March 12-14, 1986.

[12] D. Kumar, "An Approximate Method to Predict Performance of a Distributed Simulation Scheme", *18th International Conference on Parallel Processing*, St. Charles, Illinois, pp. 259-262, August 8-12, 1989.

[13] D. Kumar, and S. Harous, "An Approach to Study Performance Properties of Distributed Simulation", *The Second IEEE Symposium on Parallel and Distributed Processing*, Dallas, Texas, December 9-13, 1990.

[14] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks", *Commun. ACM*, Vol.32, No. 1, pp. 111-123, Jan. 1989.

[15] J. K. Peacock, J. W. Wong, and E. G. Manning, "Distributed Simulation Using A Network of Processors", *Computer Networks* 3(1):44-56, Feb. 1979.

[16] D. A. Reed, " A Simulation Study of Multimicrocomputer Networks", *Proceedings of International Conf. on Parallel Processing*, pp. 161-163, August 1983.

[17] D. A. Reed, "Parallel Discrete Event Simulation: A Case Study", *Record of Proceedings: 18th Annual Simulation Symposium*, pp. 95-107 , March 1985 (invited paper).

[18] D. A. Reed, A. D. Malony, and B. D. McCredie, "Parallel Discrete Event Simulation Using Shared Memory", *IEEE Transactions on Software Engineering*, 14, 4, pp. 541-553, April 1988.

[19] P. F. Reynolds, "A Sepctrum of Options for Parallel Simulation", *Proceedings of 1988 Winter Simulation Conference*, pp. 325-332, December 1988.

[20] M. Seethalakshmi, "A Study And Analysis of Performance of Distributed Simulation", *Master's Report*, Dept. of Computer Sciences, University of Texas, Austin, Texas 78712, May 1979.