Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

RESTRUCTURING NETWORKS TO AVOID DEADLOCKS IN PARALLEL SIMULATIONS

Christopher H. Nevison

Department of Computer Science
Colgate University
Hamilton, New York 13346

ABSTRACT

We introduce a new approach for parallel discrete event simulation which reduces or eliminates the possibility of deadlock without complex time synchronization or deadlock detection protocols. Deadlock is eliminated by restructuring the logical network representing the physical system to an equivalent network which is deadlock free. For large networks, a full elimination of deadlock can concentrate messages too much, so an intermediate restructuring is used to reduce and control the possibilities of deadlock. Our empirical studies show that small to medium size queuing networks for which previous conservative methods have failed can be simulated in parallel with efficiencies ranging from 30 to 80 percent.

1. INTRODUCTION

Studies of conservative methods of parallel discrete event simulation have shown that the Chandy-Misra approach of either using null messages to avoid deadlock or using deadlock detection and recovery methods do not work well for small-scale simulations (Reed and Maloney, 1988; Fujimoto, 1988; Wagner, Lazowska, and Bershad, 1989). Other studies have shown that by using look-ahead information on the logical nodes, for example by precomputing service times on a FIFO non-preemptive queuing model, and with large systems where many logical nodes of the network are assigned to each processor, good speedup results can be achieved (Nicol, 1988). We have developed a method for restructuring the logical network of a message-passing simulation which eliminates any possibility for deadlock. This method works well for small to medium scale simulation models which have some inherent structure, models such as are found in manufacturing applications and communications networks. For larger models, we show how our method can be applied to reduce the scope of deadlock, thereby making more efficient simulations possible. In some cases, the model itself naturally has the structure required for our methods to apply.

This paper is organized as follows. First some theorems about deadlock free networks, first presented in (Nevison, 1990), are discussed. In the third section, these results are applied to create deadlock free networks. In the fourth section we review prior results from two simple examples which have been studied by different investigators with disappointing results for the Chandy-Misra approach to conservative parallel simulation. In the fifth section, our methods are applied to these examples. The empirical results of our study of these two example are given in the sixth section, with comparison to the previous work. In section seven we summarize our conclusions and discuss future directions of this research.

2. CLOSED LOOP SYSTEMS

Nevison (1990) proves that certain simulation networks can be made deadlock free. In this section those results are reviewed and we show how any network can be restructured to be deadlock free but provide an equivalent simulation. First some definitions.

We represent a phenomenon to be simulated in terms of physical processes and communications between those processes. For example, in a manufacturing system, the processes could be different steps in the manufacturing process, stages of transport of items or parts between manufacturing cells, or storage facilities. The communications between those processes would represent the parts or items which move through the system or communications such as orders for parts which also move through the system. The simulation model is based on a logical network of processes which mirror the physical system but which also may include additional structure for coordinating the simulation.

The types of process node which we work with in this paper include the following (some of these are from Reed and Maloney, 1988):

Server	Holds a part or parts for a simulated delay representing a step in manufacturing or other process. May include routing calculation. Always single input, single output.
Match	Receives two or more different kinds of message and holds them until a complete set is matched, before sending out the group or a single message in their stead. Used to simulate the manufacture of an item from two or more parts.
Linear Node	Refers to any node with a single input channel and single output channel such as a server or match.
Fork	Receives messages from a single input, sends to multiple outputs according to routing information in the message.
Join	Multiple inputs, single output. Serves to bring message streams together.
Transfer	Serves as both a fork and join in one node.
Transfer Pair	Refers to a Join - Fork pair with only linear nodes on the path from the join to the fork.

We also need the concept of a *Closed Loop* for a transfer or a transfer pair. A closed loop consists of an input link and an output link on a transfer such that every message which leaves from the output returns to that input, and vice-versa (with the possible exception of messages in the system at startup time). A *Closed Loop* system is one in which all multiple routing links occur in transfers or transfer pairs and all inputs and outputs to transfers or transfer pairs are paired into closed loops. An *Almost Closed Loop* system is one in which each transfer can have at most one input and one output which are not in closed loops, and these particular links cannot be part of any cycle in the system.

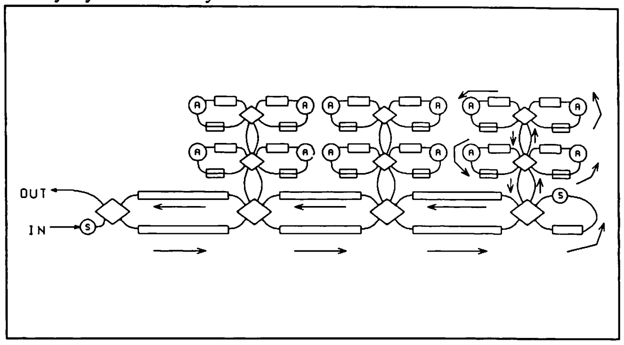

Figure 1. A Flexible Manufacturing System

The flexible manufacturing system shown in Figure 1 is an example of a closed loop system which occurs directly from the representation of the physical system (except that transfers in the system act as delays as well as routing nodes). Each item, either a work-in-progress tray or a parts tray, moves to the right along the horizontal spine and on its return trip is sent up one of the vertical branches to a manufacturing station. WIP routes through all four stages on one vertical branch while parts go to only the one where they are used. Completed items and empty parts trays are routed to the right where they are replaced in the system by fresh parts. Thus although different items follow different routes, at each transfer the closed loop condition is satisfied. This example is taken from (Winters, 1988).

Nevison (1990) proves that a closed loop system, or an almost closed loop system with guaranteed replenishment on the non-closed input, which has only one transfer can be made to run deadlock free with an appropriate *guard* on the closed loops. The simplest guard, which is sufficient, is the *count guard*. A count guard is a counter for each closed loop on the transfer which is incremented whenever a message goes out that loop and decremented when one comes in. The transfer must check every input before sending a message out to guarantee correct time order of messages. But when the count on a loop is zero, the transfer need not wait on that loop. If the transfer is not zero, then the message will always return eventually, so that loop cannot cause deadlock. In the case of match nodes, the count must include checking for all the types of message which are needed to release a message to return, but the idea is the same.

In a logical network with more than one transfer and no match nodes, deadlock can occur only if there are messages on two or more transfers which cause a mutual blocking condition. This is shown in Figure 2. Deadlock cannot occur because of other nodes on the system. Again, with match nodes, things are a bit more complicated, but deadlock can again be reduced a situation where the essential interaction is between two or more transfer nodes.

The consequence of these results is that if we can restructure the logical network for a simulation to one which gives an equivalent simulation but which forms a single transfer closed loop system, then that logical network is deadlock free. Consequently **no time synchronization protocol whatsoever is needed** other than the logic for the guards on the transfer. Empirical work shows that this can result in very efficient parallel simulations.

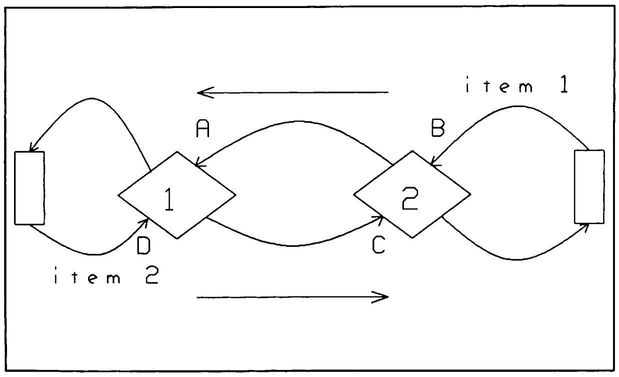

Figure 2. Deadlock Between Two Transfers

In some situations it is profitable to replace the count guards on the closed loops by more the complicated *time guards*. In addition to keeping track of how many messages are out on a closed loop, a time guard keeps a record of either a definite or an estimated next arrival time from that loop. The time is derived from whatever look-ahead information is available from the linear nodes on the loop. For example in a non-preemptive queuing system, the service times can be calculated one step ahead and piggy-back around to the transfer on the previous message, giving the transfer the information needed for an exact next arrival time. With this information, checking an input loop can be delayed until that projected time. In the case where exact times are not available, estimated times which are lower bounds can be used, as when there is a known minimum delay around a loop. The tradeoff on whether the more complicated logic of a time guard is worth it depends on the traffic in the system and the amount of real time delay (calculation time) which messages encounter on the loops.

3. RESTRUCTURING NETWORKS

The key new result in this paper which enables us to take advantage of the results discussed in Section 2, is that any logical network with no source or sink nodes which is built from the nodes described in Section 2 can be restructured into one which does an equivalent simulation but is deadlock free. This is quite simple: we only need to pull all the fork, join, and transfer nodes together into one transfer node. Every segent in the original network which consists of a sequence of linear nodes starting and ending at a fork, join or transfer node becomes a closed loop off the single transfer node which remains in the modified network. Consequently we have a closed loop system and the results cited above apply.
We discuss how some of these might be handled below. However, many networks which arise in applications in fact are highly structured and have few distinct linear segments, so the few loops are reproduced by this restructuring.

Two examples from the literature which yield good results are the central server network shown in Figure 3, with the restructured version shown in Figure 4, and the eight cluster network shown in Figure 5, which reduces to the single transfer network in Figure 6.

This aggregation of routing nodes may seem to create a "hot spot" in the network, and for large and highly interconnected networks it does. In each of the restructured networks

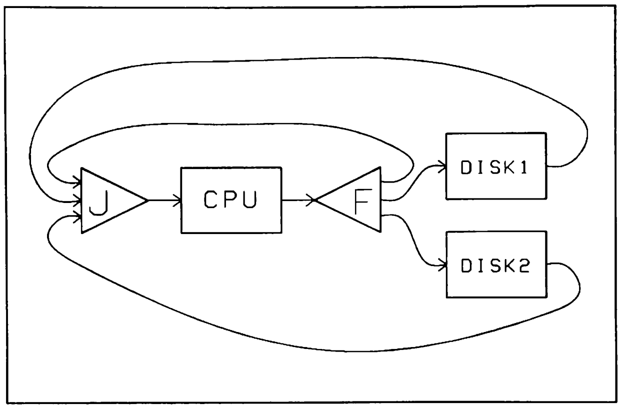

Figure 3. Central Server Network

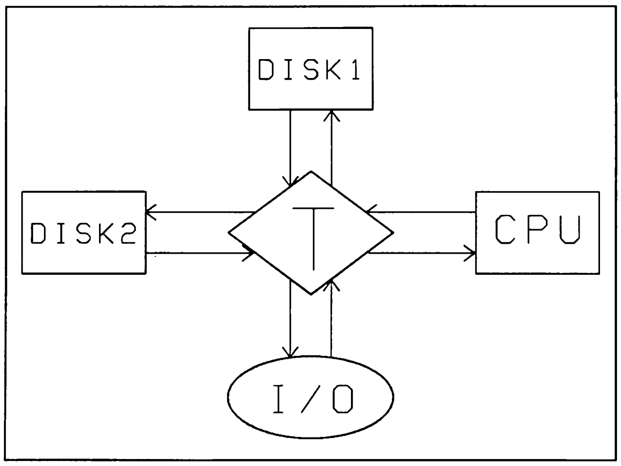

Figure 4. Restructured Central Server Network with I/O

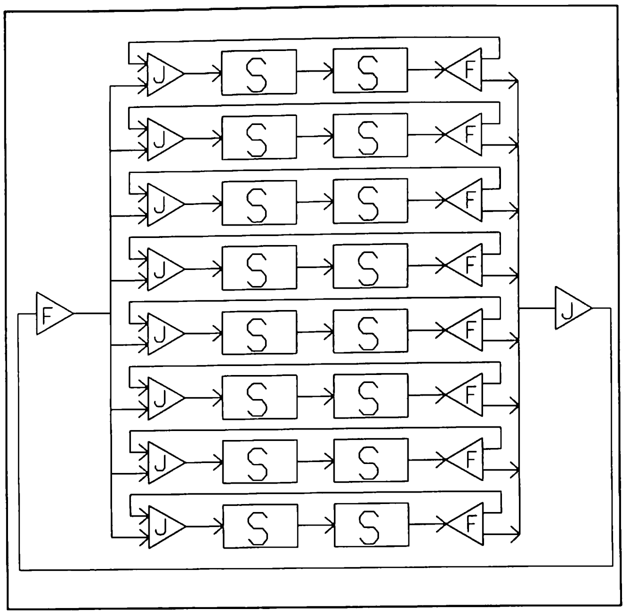

Figure 5. Eight Cluster Network

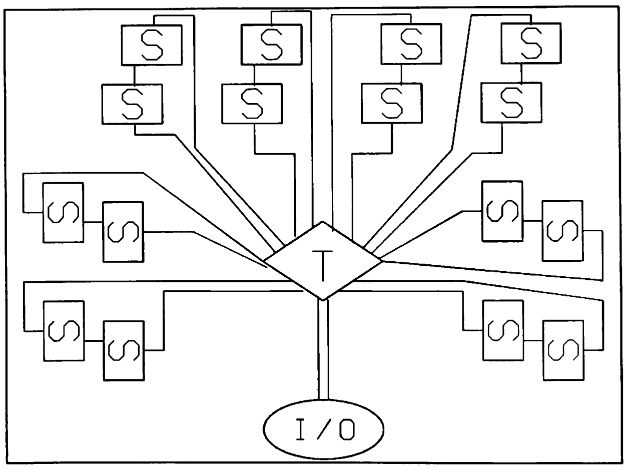

Figure 6. Eight Cluster Network Restructured with I/O

we have added a node labeled IO to indicate how the input and output are added to the system. In Section 5 we discuss our simulation results for these two models.

The eight cluster network begins to be large for this technique to work well, as all the traffic comes through the one transfer point. In cases such as this and in larger examples, we can use our technique to simplify the network without completely eliminating the possibility of deadlock. For example, the eight cluster network might be restructured as shown in Figure 7. Here, there can be a deadlock of the type shown in Figure 2 between two of the three transfers. However, this type of deadlock can be avoided by using null messages between the transfers only. In order to advance time efficiently, these null messages should use as much look-ahead information as can be made available at the transfer. In this way, we have a network where the synchronization messages are limited in scope and relatively simple to handle, in contrast to using null messages which would circulate throughout the original network. Consequently, a parallel implementation of this system should yield good speedups.

DeVries (1990) recommends a similar approach to reducing the number of null messages needed in a network, although his paper does not include any empirical results.

4. PREVIOUS WORK

Chandy and Misra showed that a system of null messages could be used to implement deadlock free parallel simulations of networks under very broad conditions. They also discussed the alternative of deadlock detection and recovery. (Chandy and Misra, 1981) More recently, several researchers have tested the Chandy and Misra methods on parallel computers for a variety of different networks. For the most part these tests have been disappointing. In both (Reed and Maloney, 1988) and (Fujimoto, 1988) the authors conclude that for the types of models which they tested, the conservative methods do not yield good speedups, largely because the cost of either null messages for deadlock avoidance or of deadlock detection and recovery schemes outweighs the potential parallelism which can be realized. Reed and Maloney included the central server network and the cluster networks which we discussed above and for

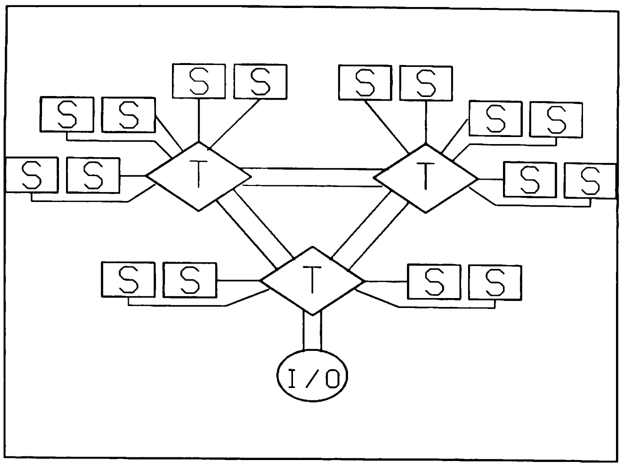

Figure 7. Eight Cluster Network Restructured with Three Transfer Nodes

which we have empirical results. They concluded that even with artificial computation loads inserted, by adding busy waits to the servers in the simulation, in order to emulate a higher computation to communication ratio than a simple queuing model would have, the Chandy-Misra methods did not produce good speedups. Since they measured speedup against a single processor version of the best parallel algorithm, rather than the standard event scheduling algorithm, these results are especially negative.

Fujimoto had more mixed results. He concluded that in cases where there was a high message load, reasonable speedups could be obtained, but in other cases the parallel techniques did not work well. Wagner, Lazowska, and Bershad had better results in their experiments (Wagner, Lazowska, Bershad, 1989). By using look-ahead techniques that were particularly appropriate to the shared memory architecture which they used, they obtained very good speedups for a variety of cases. However, their results show an anomaly which also occurred for Reed and Maloney: as the spin delay is increased to emulate higher computation times on each server, the speedup for the simulation peaks and then decreases significantly. This occurs for the central server model in (Wagner, Lazowska, Bershad, 1989) and is partially explained by the structure of the model, as they point out. But the decrease, especially with high message loads, goes beyond these structural effects. In (Reed and Maloney, 1988), the decrease in speedup for high spin times is not adequately explained. We shall show that for our model the effect of spin times is precisely what one would expect in terms of the model structure and the computation to communication ratio: in both the models which have been tested, the speedup improves with higher spin delays.

5. TEST CASES

We use both the central server model and the cluster model to test our methods. Both of these were tested by (Reed and Maloney, 1988) with poor speedups reported and the central server model was tested by (Wagner, Lazowska, Bershad, 1989) with good results, but with something of an anomaly for increased spin delays. In both of these studies we do not have true speedup values, since they compared parallel implementations to a parallel algorithm (deadlock detection and recovery) running on a single processor. Our base for speedup is an event scheduling implementation using a doubly linked list with middle pointer, searched from the top or bottom according to which half the event should go into. McCormack and Sargent (1981) have shown that this algorithm performs very well for short event lists such as we have for these examples. In addition to giving speedup values for parallel implementations, we also give the results for the parallel version running on a single processor. Since these values are always less than one, the speedup values would be significantly better if the same methodology as Reed and Maloney or Wagner et. al. were used.

Both models use servers with exponential service times, mean time 1 unit with simulation runs for 10,000 time units. In the central server model we also used simulated delays of 1, 3, 3 on the server and two disks respectively, in order to balance the load across processors (since the central server sees three times as many messages as each of the disks). The branching at the one fork in the central server model is done with equal probabilities, just as it is in both the studies referred to above. (See Figure 3.) Two cluster models were tested: one with four clusters of two servers each, as shown in Figure 8 and restructured in Figure 9; the other with eight clusters as shown in Figures 5 and 6. In the cluster model the branching

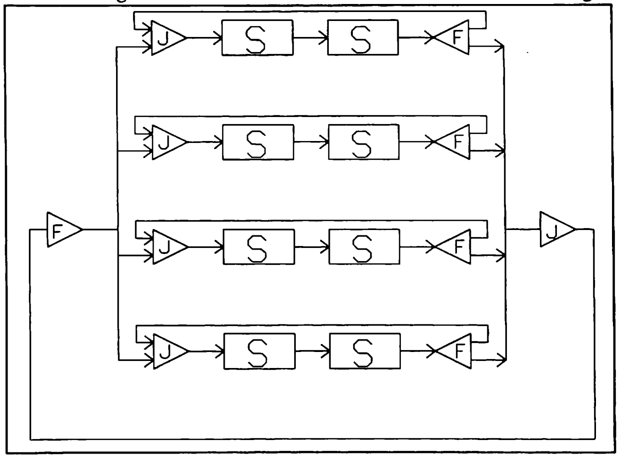

Figure 8. Four Cluster Network

follows the Reed and Maloney model, a probability of 1/2 of recycling on the same cluster and equal probabilities for each of the other clusters. In both models, we compute the next destination on the server and the transfer is purely a routing mechanism, not an addressing node. If the fork nodes in either of these models were to have simulated delays, these could easily be built into our model as well. Since they do not, the single transfer network which we derive is fully equivalent to the original model in each case.

Simulation runs for various loads, the number of jobs in the system, were done. In addition, we added artificial spin delays to the servers to emulate higher computation loads which would be found in some simulation applications. In both our models, the typical simulation statistics for each queue are kept, namely: total time spent in the queues, busy time for the servers, and number of customers or jobs served. Thus we are able to calculate statistics such as average wait, average queue length, and utilization for each server.

Our study was carried out on a network of T414 transputers. Each transputer is a high speed RISC microprocessor, with integer speeds slightly faster than an Intel 80386 (both running at 20 MHz)(Stiles, 1989). In addition, the transputer has been optimized for the parallel language Occam 2, based on Hoare's CSP (Hoare, 1979). Each transputer has four high

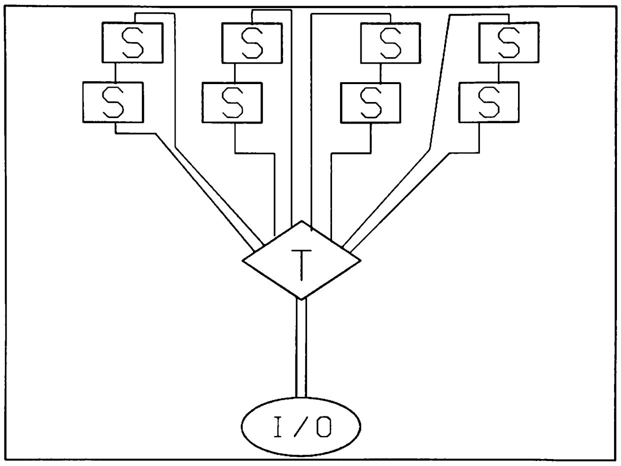

Figure 9. Restructured Four Cluster Network

speed (20 Megabits per second) bidirectional IO links which are connected to other transputers to form the network. In our study both the event scheduling version and the parallel versions were coded in Occam 2. Because the T414 transputers do not have numeric coprocessors the floating point calculations are done in software, adding to the computation load. The transputer network is hosted by an MS-DOS microcomputer.

For each model the network was connected to match the logical network as closely as possible. The empirical results for the central server include parallel versions running on one, two, three, and four processors, as shown in Figures 10, 11 and 12. The four cluster model is run on nine processors, one for each server and one for the transfer and I/O, and the eight cluster model is run on seventeen processors.

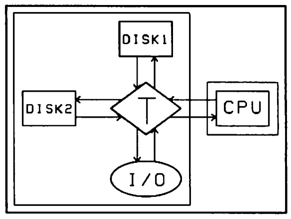

Figure 10. CPU Model on Two Processors

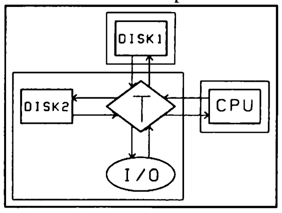

Figure 11. CPU Model on Three Processors

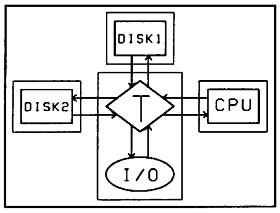

Figure 12. CPU Model on Four Processors

6. EMPIRICAL RESULTS

For the central server we have a series of results indicating speedup values for one to four processors with the parallel code as measured against the event scheduling algorithm. These are plotted for a range of load values from 1 to 40, which shows how the speedup increases with higher message densities. The load is the number of jobs in the simulated system. In all cases, of course, a single message means the whole simulation is serial, so the parallel versions show the same running time as the single processor parallel version. In all cases the speedup reaches a saturation point and levels off, showing the critical number of messages for keeping the nodes processing the servers constantly busy. The results for this model are shown in Figures 13 through 16.

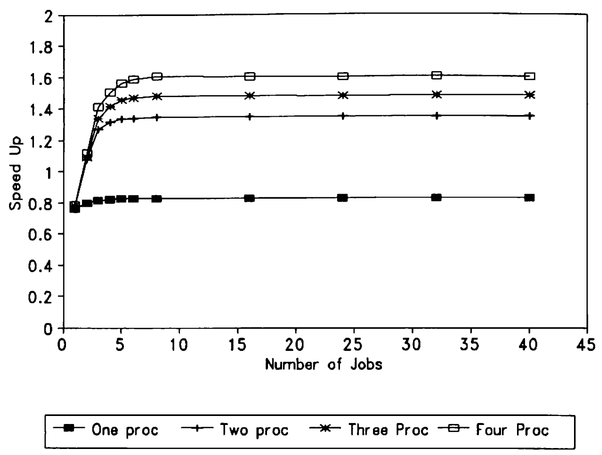

Figure 13. Results for Central Server with
Service Times: CPU 1 Disks 1
Real Delay: CPU 0 ms Disks 0 ms

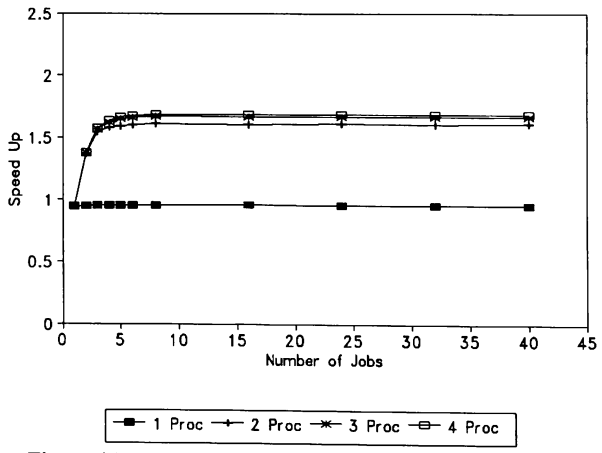

Figure 14. Results for Central Server with
Service Times: CPU 1 Disks 1
Real Delay: CPU 1.5 ms Disks 1.5 ms

In Figure 13 we see that for no spin delay, even two processors capture most of the parallelism, and that the maximum speedup, for four processors, levels off at about 1.6 with a load of 8. The second graph, Figure 14, adds a 1.5 msec. delay to the servers each time they execute. In this case, as Wagner et al. observed, the maximum speedup should be about 1.67, since the central server has three times the load of the other two servers. We see that even two processors achieve most of this potential speedup, and the fourth processor adds nothing significant.

In the third graph, in Figure 15, we use a spin delay three times as large for the peripheral servers, so the potential load for each logical server node is equal. For two processors, we cannot expect anything more than a 1.5 speedup, since with the

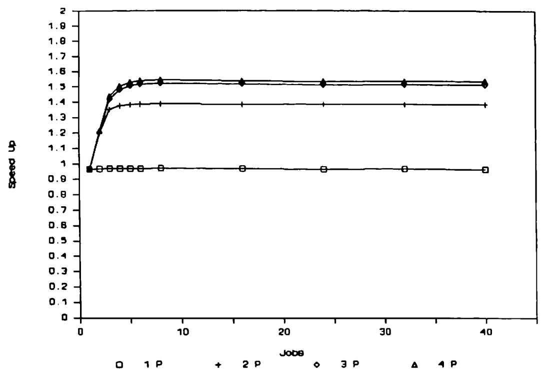

Figure 15. Results for Central Server with
Service Times: CPU 1 Disks 1
Real Delay: CPU 1.5 ms Disks 4.5 ms

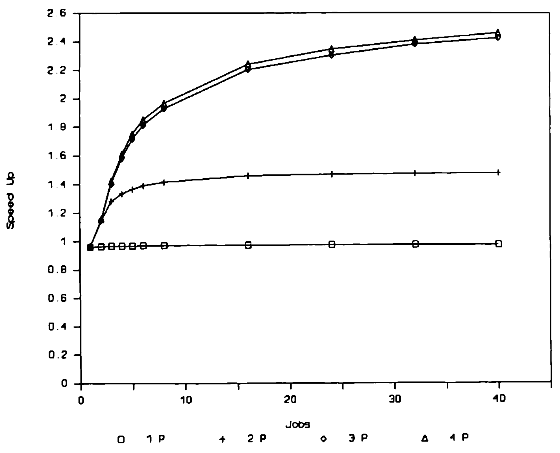

Figure 16. Results for Central Server with
Service Times: CPU 1 Disks 3
Real Delay: CPU 1.5 ms Disks 4.5 ms

static allocation of the network, one of the nodes has twice the load of the other. Three nodes achieve all the rest of the speedup available, which still only just over 1.5. This is due to the nature of the model and the fact that the logical nodes are statically assigned to processors. Although the computation load is balanced among the three logical nodes, the simulated traffic is not balanced. The simulation statistics show that the average queue length is 0.16 on the two peripheral servers. This means that when the transfer sends them a job, it is usually the only one there. Consequently, when using a count guard, as we are for this series of experiments, the transfer must wait for that job to return before it can process another message. This sequentializes the computations on the peripheral servers, resulting in lower parallelism and speedup. This example demonstrates how the structure of the model can limit the speedup available.

Figure 16 shows the graph for the model with both the spin delay and the simulated service times in balance. Since the simulated loads are balanced, as the number of messages increases, the likelihood of serialization observed before diminishes, and the model approaches the maximum possible speedup of three. The peak speedup is 2.4 for three processors, an efficiency of 80%.

These speedup results are much better than those given in (Reed and Maloney, 1988) and compare quite well to those in (Wagner, Lazowska, Bershad, 1989).

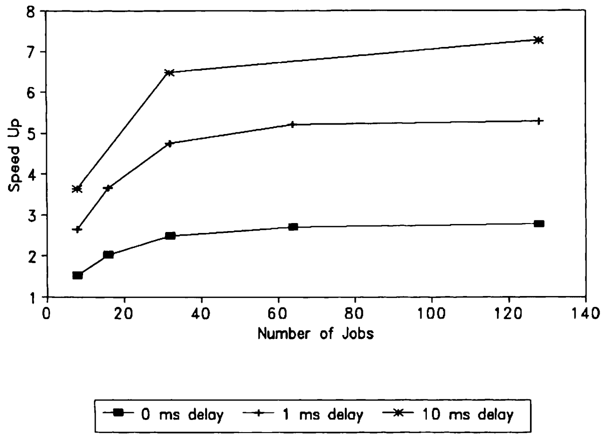

Figure 17. Results for Four Cluster Model

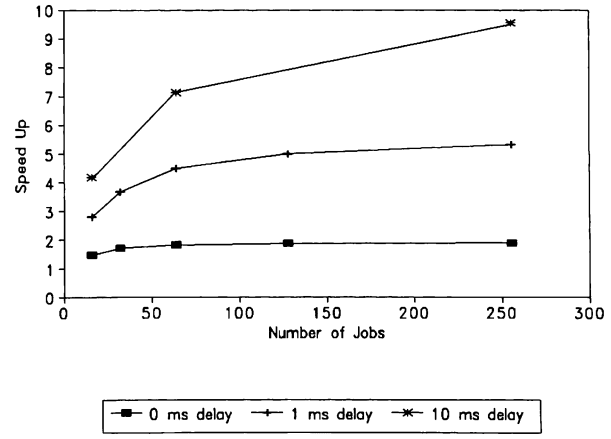

Figure 18. Results for Eight Cluster Model

The results for the four cluster model run on nine processors is shown in Figure 17. The eight cluster model run on seventeen processors is shown in Figure 18. The results show two chief characteristics: In both cases the speedup for the model with no real delay is very low, demonstrating the "hot spot" effect of pulling all routing into a single transfer node. Second, for both models the speedup increases as the real delay is increased and is quite good for a real delay of 10 ms per server. This indicates that this method would work well for models with a large computation to communication ratio on each node. The models yield efficiencies of 80% and 56% respectively for the highest loads tested.

In both the central server model and the cluster models, there is no anomalous decrease in speedup as the real delay is increased, as has occurred in other methods (Reed and Maloney, 1988; Wagner, Lazowska, Bershad, 1989).

7. CONCLUSIONS

We have shown that any logical network can, in principle, be restructured to a network which produces an equivalent simulation but is deadlock free when run in parallel in a conservative fashion. This enables us to simulate small to medium models in parallel with deadlock free logical networks, thus achieving very good speedups.

For larger systems, our method of restructuring should enable us to produce networks in which deadlock avoidance mechanisms can be used without the effects of flooding the system with null messages which has been observed in previous studies. We currently are developing simulations of larger models to test this hypothesis. Our restructuring technique can be used to advantage along with other conservative methods to help make large scale parallel simulations more effective.

ACKNOWLEDGEMENTS

This research was sponsored in part by the Consortium for Computer Applications and Software Engineering (CASE) at Syracuse University. Michael Freidin assisted with the programming and experimentation.

REFERENCES

Chandy, M. and J. Misra (1981), "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations," *Communications of the ACM 24*, 198-206.

De Vries, R.C. (1990), "Reducing Null Messages in Misra's Distributed Discrete Event Simulation Method," *IEEE Transactions on Software Engineering 16*, 82-91.

Fujimoto, R.M. (1988), "Performance Measurements of Distributed Simulation Strategies," In *Proceedings of the 1988 Distributed Simulation*, B. Unger and D. Jefferson, Eds. SCS, San Diego, CA, 14-20.

Hoare, C.A.R. (1978), "Communicating Sequential Processes," *Communications of the ACM 21*, 666-677.

Nevison, C.H. (1990), "Parallel Simulation of Manufacturing Systems: Structural Factors," In *Proceedings of the 1990 Distributed Simulation*, D. Nicol, Ed. SCS, San Diego, CA, 17-19.

Nicol, D. (1988), "Parallel Discrete Event Simulation of FCFS Stochastic Queueing Networks," In *Proceedings of the 1988 Winter Simulation Conference*, M.A. Abrams, P.L. Haigh, and J.C. Comfort, Eds. IEEE, Piscataway, NJ, 124-137.

Reed, D. and A. Maloney (1988), "Parallel Discrete Event Simulation: the Chandy-Misra Approach," In *Proceedings of the 1988 Distributed Simulation*, B. Unger and D. Jefferson, Eds. SCS, San Diego, CA, 8-13.

Stiles, G.S. (1989), "How the Transputer Stacks Up to Other Processors: A Comparison of Performance on Several Application Programs," In *NATUG 1: Proceedings of the First Conference of the North American Transputer Users Group*, G.S. Stiles, Ed. Salt Lake City, UT, 199-209.

Wagner, D., E. Lazowska, and B. Bershad (1989), "Techniques for Efficient Shared-Memory Parallel Simulation," In *Proceedings of the 1989 Distributed Simulation*, B. Unger and R.M. Fujimoto, Eds. SCS, San Diego, CA, 29-37.

Winters, I. (1988), "Modeling Product Mix Capacity/Capability Functions of Flexible Manufacturing Systems," Ph.D. Thesis, University of Massachusetts, Amherst, MA.