Modeling and Efficient Distributed Simulation of Multistage Interconnection Networks

Yong Meng TEO and Seng Chuan TAY
Department of Information Systems & Computer Science
National University of Singapore
Lower Kent Ridge Road
Singapore 0511
email: {teoym,taysengc}@iscs.nus.sg

Abstract

Multistage interconnection networks are used in a number of application areas such as parallel computers and high-speed communication systems. As the performance of these systems lies on an efficient design of the interconnection network, a thorough analysis of the network's performance is important. Mathematical analysis so far provides inadequate results and simulation analysis using a uniprocessor usually requires extremely long run time to evaluate large networks. This paper addresses the use of parallel simulation techniques to speedup the simulation of multistage interconnection networks. The conventional null-message approach for resolving deadlock problem in conservative simulation may cause livelock if lookahead is not guaranteed. We propose a deadlock/livelock free scheme using null messages, but without the guaranteed lookahead, to coordinate the simulation, and different partitioning techniques for mapping of the simulation program onto multicomputers. A flushing mechanism is also used to resolve the null-message explosion problem. Our analysis shows that the proposed flushing mechanism effectively reduces the number of null messages from exponential to linear.

Keywords: simulation modeling, conservative approach, multistage interconnection network, deadlock, livelock

1 Introduction

Switching architectures have been proposed for use in the communication networks of parallel/distribution systems. Among them multistage interconnection network (MIN) offers a tradeoff between cost and performance. More recently, there are growing interests in using MIN as interconnection structure for the switch nodes of high-speed communication networks. MIN is also a potential candidate for high speed switching systems such as broadband integrated services digital network (B-ISDN), and transport systems based on the asynchronous transfer mode (ATM) [8, 9]. As the performance of these systems depends on the efficient design of the communication subsystem, a thorough analysis of a MIN characteristics such as bandwidth, latency, etc. is important. Most of the mathematical studies assume that each input component generates random and independent packet transmissions distributed uniformly over all output components. So far, no closed-form result is available for non-uniform distribution. Researchers also make unrealistic assumptions such as ignoring blocked transmissions [6]. In the real system, this will mean that a processor may proceed with its computation before the required data is available in the processor. In addition, with increased system complexity and when a more detailed level of modeling is required such as more parameters, the mathematical analysis becomes analytically intractable and numerically probabilistic to evaluate. As mathematical analysis does not provide adequate results, the significance of simulation studies on the MIN's performance becomes apparent. Since classical sequential simulation is time-consuming especially for large interconnection networks, using multiple processors for simulation appears to be a promising approach to improve their speed. The two main synchronization approaches of parallel discrete-event simulation known as *conservative* and *optimistic* are discussed in [2, 3, 5].

This paper focuses on a *conservative approach* in the parallel simulation of MINs, efficient process modeling and partitioning schemes, and an analysis of its performance. Section 2 introduces a proposed *process-oriented simulation model* for a switching element; the smallest component in a multi-stage interconnection network. We explain the methods for handling traffic contention, routing conflict resolution, parallel and preemptive packet transmissions, and resolving forward and backward cascading deadlocks in the context of modeling and parallel simulation. Section 3 presents the enhanced process-

0-7803-2018-2/95/$4.00 © 1995 IEEE

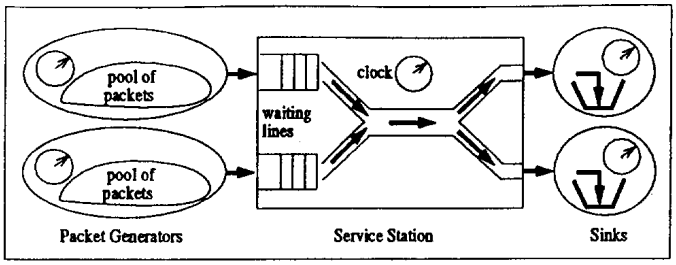


Figure 1: A Queuing Model for a 2 × 2 Switching Element

oriented model for large MIN simulation, and a set of protocols to resolve the deadlock and livelock problems. We also proposed a simple and yet effective mechanism to control the flooding of null messages during simulation. Section 4 introduces some partitioning schemes for mapping the simulation application onto a limited number of processors, and an analysis of the performance results. Section 5 contains our concluding remarks.

2 Simulation Modeling of a Switching Element

The smallest component in our MIN simulation consists of a 2 × 2 crossbar switching element. The functionalities of each switching element include (i) receive messages from in-coming links, (ii) execute routing algorithm and (iii) forward messages to the desired out-going links. Therefore, a switching element can be treated as a queuing system. The proposed process-oriented parallel simulation model (see figure 1) consists of the following five logical processes: two packet generators, one service station with parallel servers, and two sinks. Two waiting lines (or buffers) of *finite queue length* are embedded in the station to model a blocking switch. Each process has its own local clock to indicate its simulation progress. All processes used for simulation can be executed in parallel. The generator and server processes are governed by inter-message arrival time and switching time respectively, and modeled using statistical distributions. However, to model closely the operations of the switching element the arrival of a message will have to be delayed if the waiting line at the receiving link is completely filled. Therefore, a blocked message will have to wait in the generator until it is allowed to proceed, instead of being discarded as commonly assumed in mathematical analysis [6]. For simplicity, we assume that the processing time required in a sink process is instantaneous as compared to the switching time. It follows that messages departing from a switching element are not blocked. We assume a constant switch delay in our simulation.

2.1 Modeling Traffic Contention

Traffic contention problem can occur in MIN's operation if the buffer associated with an in-coming link is full when a packet arrives. To ensure that packets are not lost at a service station when the buffer is full, the simulation processes must adhere to some *hand-shaking protocols* during transmission [11]. That is, before a packet transmission is initiated, the sender must find out if the receiving buffer contains at least one unfilled slot by sending some request signals. In the following discussions, when a transmission contains no information other than serving as a request or reply of link status, we refer to it as a *signal*. Otherwise we refer to it as a *message*. The underlined indicates that a signal/message needs to be time-stamped.

- *Request Progress* (REQ) - This signal is used by a generator to find out if the message buffer of the service station can accommodate additional message.
- *Slot Available* (AVA) - This signal is replied by a service station to notify that the message buffer of the desired in-coming link is not full.
- *Slot Not Available* (NAV) - This signal is replied by a service station to notify that the message buffer of the desired in-coming link is full.
- *Access Request* (ACC) - This message contains the source and destination addresses of an access request. It is sent to the service station upon receiving the reply signal "Slot Available".
- *Transmit Time* (TIM) - This message contains the time for a generator to re-transmit a blocked message. It is sent by a service station when a message is removed from its message buffer where a NAV signal was previously replied to the generator on that link.
- *Stop* (STO) - This signal is used by a service station/sink to inform the generator/service station that its process is going to terminate.

An illustration of how the hand-shaking protocols resolve the traffic contention problem is shown in figure 2.

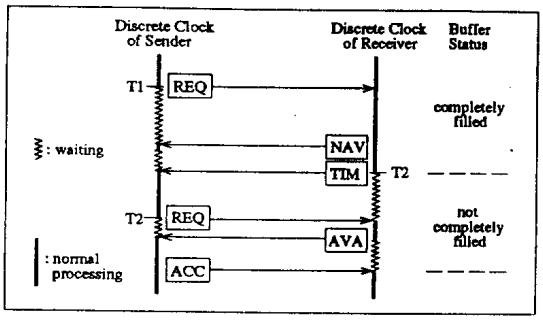


Figure 2: Resolving Traffic Contention Using Delayed-Transmission

2.2 Modeling Routing Conflict

Routing conflict occurs in a switching element when two messages coming from different in-coming links are sent to the same out-going link at the same or overlapped time interval. When this occurs, an arbitration scheme is required to serialize the accesses. In our simulation, the upper link is given the priority to proceed first.

Four types of event are modeled in the simulation to resolve routing conflict. The first two event types, denoted by $ARRIVAL[i]$, where i is 0/1 for upper/lower in-coming link, are the arrival events representing the processing of in-coming messages from the upper and lower links respectively. The last two event type, denoted by $DEPARTURE[j]$, where j is 0/1 for the upper/lower out-going link, are the departure events representing the forwarding of messages to the jth out-going link. All events are stamped with an occurrence time.

It is important to note that event execution must be performed in ascending order of occurrence time to ensure causality. Figure 3 depicts how routing conflict problem is resolved. For illustration purpose, let the first arrival times on both links be 4 (μsec, say) and both arrivals are to be forwarded to the upper out-going link. Let the second arrival times on both links be 12 μsec and 11 μsec respectively and the switch delay be 3 μsec. As shown in figure 3, the discrete-event simulation algorithm serializes the departures of two concurrent arrivals at *clocktimes* 7 μsec and 10 μsec respectively on the upper out-going link.

2.3 Modeling Parallel Packet Transmissions

Since the simulation model consists of two parallel servers and the assignments of departure times on the two out-going links are independent, parallel transmission in the real switch can be modeled and simulated. An illustration of how parallel transmissions are simulated is depicted in figure 4.

We assume that the arrival times are the same as those in figure 3, but the concurrent arrivals at *clocktime* 4 are sent to different out-going links. Figure 4 shows that the concurrent messages are forwarded simultaneously, in terms of the discrete clock times, to their desired out-going links by the departure-event executed on different out-going links.

2.4 Modeling Preemptive Packet Transmissions

Preemptive packet transmission refers to the instance where packet(s) is forwarded to its desired out-going link before the departure(s) of packet(s) arriving earlier on the same or different in-coming link. Preemptive transmission is allowed in MIN when the desired out-going link of a packet becomes available while it is waiting in a buffer. Consider five arrivals A, B, C, D, E of timestamps 1, 5, 6, 7 and 8 respectively on in-coming link[0], where the first four arrivals are sent to out-going link[0], and the fifth arrival to out-going link[1] (see figure 5).

Let the timestamps of the three arrivals on in-coming link[1], denoted by F, G, H be 2, 3 and 4 respectively, and the first arrival is sent to out-going link[1] and the second and third arrivals to link[0]. Let the switch delay be 10. If the in-coming messages on link[0] are processed in sequential order, E will not be processed before D is removed from the message buffer at *clocktime* 51 ($1 + 5 \times 10$). In fact, E can be processed earlier because its desired out-going link is available after F departs the switching element at *clocktime* 12 (2 + 10). Therefore, the $DEPARTURE[1]$ event executed at *clocktime* 12 (for F) may schedule the next $DEPARTURE[1]$ event at *clocktime* 22 (12 + 10) (for E). To model the preemptive packet transmission, the in-coming messages are partitioned into two groups for out-going link[0] and link[1] respectively on their arrivals. Each group of messages is then sorted in ascending arrival-timestamp order. Pointers (figure 5a) or circular arrays (figure 5b) may be used for the data structure. In our implementation circular arrays are used to model the preemptive packet transmission.

Iteration Number	0	0	1	2	3	4
Discrete Clock	[0]	[0]	[4]	[4]	[7]	[10]
EOT	[0,0,+∞,+∞]	[4,4,+∞,+∞]	[12,4,7,+∞]	[12,11,7,+∞]	[12,11,10,+∞]	[12,11,+∞,+∞]
Remarks	initial values	receive ACCs of the same timestamp and same destination from both in-coming links.	ARRIVAL[0] is selected for execution based on priority scheme. schedule next ARRIVAL[0] & DEPARTURE[0] at clocktimes 12 & 7 respectively.	ARRIVAL[1] is selected for execution based on minimum event occurrence time. schedule next ARRIVAL[1] at clocktime 11.	DEPARTURE[0] is selected for execution. send message on out-going link[0]. schedule next DEPARTURE[0] at clocktime 10.	DEPARTURE[0] is selected for execution. send message on out-going link1[0].

EOT : [a, b, c, d,], where a = Event Occurrence Time of ARRIVAL[0],
b = Event Occurrence Time of ARRIVAL[1],
c = Event Occurrence Time of DEPARTURE[0],
d = Event Occurrence Time of DEPARTURE[1].

Figure 3: Serializing Routing Conflict

Iteration Number	0	0	1	2	3	4
Discrete Clock	[0]	[0]	[4]	[4]	[7]	[7]
EOT	[0,0,+∞,+∞]	[4,4,+∞,+∞]	[12,4,7,+∞]	[12,11,7,7]	[12,11,+∞,7]	[12,11,+∞,+∞]
Remarks	initial values	receive ACCs of the same timestamp & different destinations from both in-coming links.	ARRIVAL[0] is selected for execution based on priority scheme. schedule next ARRIVAL[0] & DEPARTURE[0] at clocktimes 12 & 7 respectively.	ARRIVAL[1] is selected for execution based on minimum event occurrence time. schedule next ARRIVAL[1] & DEPARTURE[1] at clocktimes 11 & 7 respectively.	DEPARTURE[0] is selected for execution. send message to out-going link[0]. set out-going link[0] to idle.	DEPARTURE[1] is selected for execution. send message to out-going link[1]. set out-going link[1] to idle.

Figure 4: Parallel Transmissions on Different Out-going Links

2.5 Resolving Forward-Cascading Deadlock

Although the discrete-event simulation model using the proposed hand-shaking protocols is able to produce correct simulation results, deadlock may occur in the service-station when the simulation approaches the completion time. The reason is that the execution of an arrival event will have to schedule the next arrival. However, if the transmit time of the desired ACC message is beyond the completion time the generator will not activate the transmission, resulting in deadlock in the process on its receiving end (see figure 6a). In consequence, deadlock is cascaded forward to the sink process. An approach to resolve this problem is proposed in [12]. The generator and service-station processes intentionally send out a pseudo-arrival consisting of REQ signal and an ACC message both with the timestamp of $\tau + \delta$, where τ is the duration of simulation and $\delta > 0$, on all their out-going links when their clock reaches the completion time (see figure 6b). Note that in this instance no reply is required after the REQ signal is sent because the receiving process may have been terminated, thereby causing deadlock in the sending process if it waits for a reply signal.

2.6 Resolving Backward-Cascading Deadlock

Even with the inclusion of the pseudo-arrivals, deadlock may still occur in the parallel simulation. Consider a service station where the generator on link[0] is at the wait state due to unavailability of buffer space. Therefore the $ARRIVAL[0]$ event in the receiving process will not be selected for execution until a departure event that creates space on link[0] is executed. Suppose the occurrence time of $DEPARTURE[0]$ and $DEPARTURE[1]$ are also beyond the duration of simulation. If the receiving process executes an $ARRIVAL[1]$ event and schedules the next $ARRIVAL[1]$ such that the occurrence time is beyond the completion time, the receiving process will be terminated in the next iteration because all the four events are not eligible for execution. Consequently, the generator process on link[0] will be locked indefinitely (see figure 7a). It is clear that deadlock will be cascaded backward if the generator

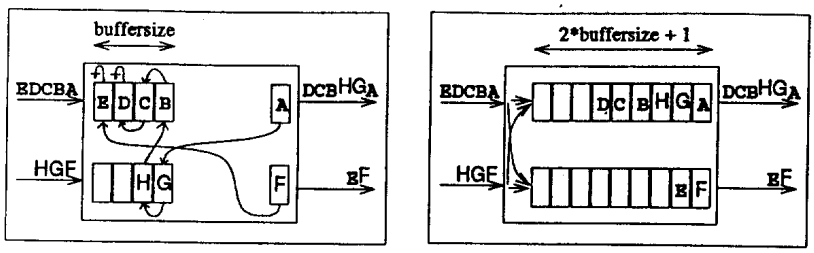


Figure 5: Data Structures for Preemptive Packet Transmissions

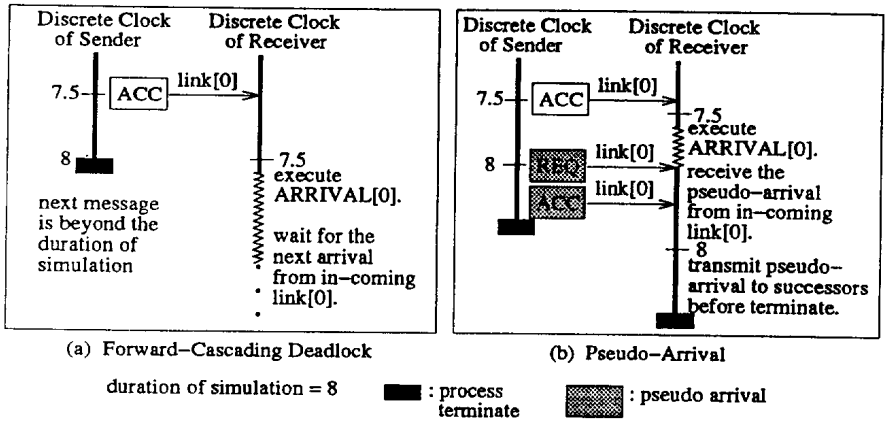


Figure 6: Solving Forward-Cascading Deadlock using Pseudo-Arrival

is also connected to the out-going links of some processes. This problem can be resolved by the service-station sending a STO signal to the generator before it terminates. On the other hand, after a generator sends a REQ signal it should also accept the STO signal as the reply, not just the AVA and NAV signals only. If the reply is a STO signal, meaning that its receiving process has already terminated, the generator should also terminate (see figure 7b).

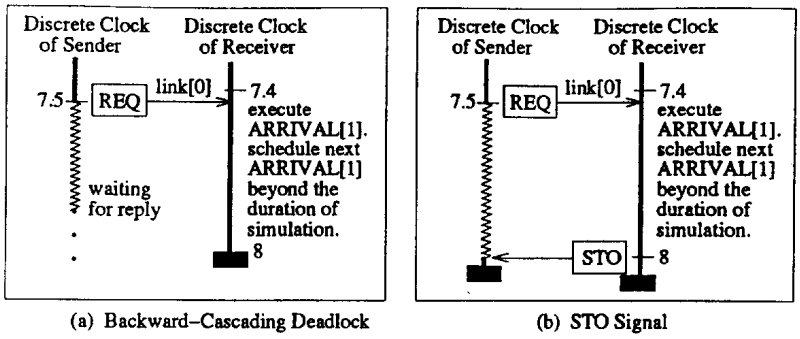
3 Parallel Simulation of MIN

The process-oriented nature for modeling and simulating a switching element can be extended to MIN. In the operation of a MIN all switching components operate autonomously and so does its counterparts in simulation. Each switching component in the MIN is simulated by a service-station process executing discrete events. The interactions among the processes is by passing signals and messages, and all processes are executed in parallel. Enhancement to the simulation model for the switching element is necessary, i.e. the out-going messages from a service station should adhere to the hand-shaking protocols due to finite buffer space.

3.1 Deadlock and Livelock

Consider a simple 4×4 Omega network as shown in figure 8a where SW_{ij} represents the jth switching element in the ith stage. This figure corresponds to the instance where SW_{10} and SW_{11} are executing departure events, and SW_{00} and SW_{01} are executing arrival events. Both SW_{10} and SW_{11} are waiting for the status of the receiving buffer on their lower and upper out-going links respectively, while SW_{00} and SW_{01} are waiting for new arrival from their upper and lower in-coming links respectively. This results in a *deadlock* situation among the four switching elements (see figure 8b).

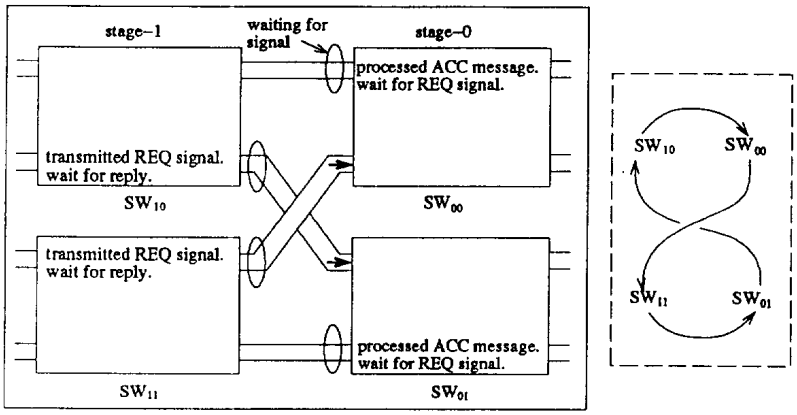
A modified null-message approach is used to resolve the deadlock problem in our MIN simulation. It is vital to note that in the conventional null-message approach, whenever a message of timestamp σ is transmitted on an out-going link, a null message containing a *time indicator* of $\sigma + \epsilon$, $\epsilon > 0$ (ϵ may be treated as the switch delay in MIN simulation), is



(a) Backward–Cascading Deadlock (b) STO Signal

Event occurrence times of DEPARTURE[0] and DEPARTURE[1] are both beyond the duration of simulation.

Figure 7: Resolving Backward-Cascading Deadlock using STO Signal



(a) An Instance of Deadlock (b) Four Processes Waiting in a Loop

Figure 8: An Instance of Circular Wait in a 4 × 4 MIN Simulation

also transmitted on all other out-going links to inform the receivers to proceed, instead of waiting, with their simulation up to the time indicated [7]. These null messages are then circulated with their indicators incremented by ϵ whenever they are forwarded from one process to the others. Unfortunately, such *lookahead* indicators are not available in MIN simulation when message buffer(s) is completely filled. For these instances null messages are forwarded without having the time indicators incremented. As a result, all processes will not schedule any event for execution after receiving the forwarded null messages, causing *livelock* in the parallel simulation.

3.2 Resolving Deadlock and Livelock Problems

The following *relaying protocols* protocols is proposed to resolve the deadlock problem in the MIN simulation. In general, a process on receiving a null message or requesting a buffer status should also send null messages, with lookahead timestamps if available, to all the other three links. In this way the null messages are circulated among the processes for MIN simulation. If a received null message indicator is greater than all other outstanding event times, a process is safe to proceed on with the simulation, thereby resolving a deadlock. In the following descriptions, α, β, and γ are the time indicators to inform the sender/receiver on the various links that the process will not send them any useful (hand-shaking) signals and messages before the specified time. We let $i' = i \oplus_2 1$, and $j' = j \oplus_2 1$, where $\oplus_2$ is an addi-

tive operator of modulo two. The underlined refers to in-coming link, and overlined out-going link.

1. When a REQ signal of timestamp Θ is sent to an out-going link[j], three NULL messages containing the time indicators α, β and γ respectively, where

$$\alpha = \begin{cases} toa[0] & \text{if } \underline{link[0]} \text{ full} \\ \Theta & \text{otherwise} \end{cases}$$

$$\beta = \begin{cases} toa[1] & \text{if } \underline{link[1]} \text{ not full} \\ \Theta & \text{otherwise} \end{cases}$$

$$\gamma = \begin{cases} tosc[j'] & \text{if } \overline{link[j']} \text{ not empty} \\ \Theta + switch\ delay & \text{otherwise} \end{cases}$$

   are also sent to in-coming link[0], in-coming link[1] and out-going link[j'] respectively.

2. When a NULL message is received while a process is waiting on an out-going link[j] for the reply to a REQ signal, the process should also forward the NULL message on the other three links with the time indicators specified in clause 1.

3. When scheduling a new arrival event on an in-coming link[i], a process should wait for a REQ signal and also process a NULL message if it arrives.

4. If a NULL message is received in clause 3, three NULL messages containing the time indicators α, β and γ respectively, where

$$\alpha = \begin{cases} toa[i'] & \text{if } \underline{link[i']} \text{ not full} \\ \min\,(tosc[0],\ tosc[1]) & \text{otherwise} \end{cases}$$

$$\beta = \begin{cases} tosc[0] & \text{if } \overline{link[0]} \text{ not empty} \\ \Theta + switch\ delay & \text{otherwise} \end{cases}$$

$$\gamma = \begin{cases} tosc[1] & \text{if } \overline{link[1]} \text{ not empty} \\ \Theta + switch\ delay & \text{otherwise} \end{cases}$$

   are also sent to in-coming link[i'], out-coming link[0] and out-going link[1] respectively.

The way we resolve the livelock problem is by giving a higher priority to the execution of departure event whenever a departure time is equal to a null-message indicator. By using this priority scheme in the event scheduling, a process will proceed with the simulation instead of keep relaying null messages to other processes.

3.3 Reducing Null-Message Overhead

The transmission protocols and the simulation mechanism are sufficient to ensure that deadlock and livelock will not occur during the conservative simulation. However, the number of null messages generated grows at exponential rate. Cai and Turner [1] used routing history to reduce the amount of null messages. As their method assumes that lookahead is guaranteed in null messages, it cannot be used in our simulation. Instead a flushing mechanism is used in our implementation. When a null message is received, the process will flush in all the null messages that have already arrived on that link and select the *largest* time indicator. The forwarding of null messages is only on the largest time indicator and the rest are discarded. Our implementation results show that the amount of null message overhead is effectively reduced from *exponential* to *linear* when this mechanism is used.

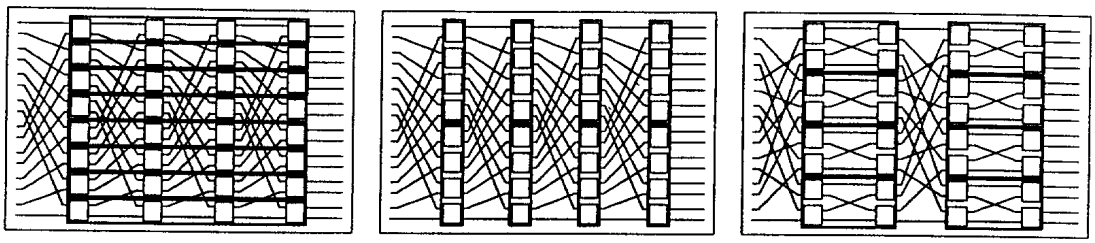
4 Performance Evaluation

We have implemented the parallel simulation algorithms in the C language. The PVM (Parallel Virtual Machine) software [4] is used to distribute the simulation processes for execution on a network of workstations.

For comparison purpose, four mapping schemes are used to map the simulation processes to a finite number of workstations. The horizontal partitioning scheme (HPS), vertical partitioning scheme (VPS) and modular partitioning scheme (MPS) for simulating a 16 × 16 MIN using 8 workstations are shown in figure 9. In the figure a rectangle represents a workstation while a square represents a simulation process. The balance partitioning scheme (BPS) is performed by the PVM system, based on the machine performance and workload when processes are spawned, while the MPS is obtained by a series of transformations on the omega networks [11].

4.1 Elapsed Time and Speedup

Figure 10 shows the elapsed time of a 16 × 16 MIN simulation on eight workstations. As shown in the figure, the elapsed time of HPS is larger than that of MPS. The explanation to this phenomenon is that for synchronous transmission, the waiting for arrival in a processor causes starvation to all its succeeding processors. As the diameter of the processor-networking for HPS is longer than that of MPS, starvation effect is more severe, causing a larger elapsed time in HPS. The elapsed time of VPS is larger than those for HPS and MPS due to the uneven distribution of generator and sink processes on the worksta-



(a) Horizontal Partition (b) Vertical Partition (c) Modular Partition

Figure 9: Partitioning Schemes for a Finite Number of Workstations

scheme	percentage (%)	no. of messages sent per generator						average
		120	240	360	480	600	720	
horizontal	$\frac{Null}{Total} \times 100$	79.82	79.80	79.86	79.90	79.96	79.83	79.86
	$\frac{Protocols-ACC}{Total} \times 100$	13.46	13.41	13.40	13.38	13.31	13.44	13.40
	$\frac{ACC}{Total} \times 100$	6.72	6.79	6.74	6.72	6.73	6.73	6.74
vertical	$\frac{Null}{Total} \times 100$	80.05	79.88	79.93	79.96	80.05	79.96	79.97
	$\frac{Protocols-ACC}{Total} \times 100$	13.27	13.38	13.35	13.35	13.28	13.35	13.33
	$\frac{ACC}{Total} \times 100$	6.68	6.74	6.72	6.69	6.67	6.69	6.70
modular	$\frac{Null}{Total} \times 100$	78.60	78.71	78.97	79.12	79.05	78.96	78.90
	$\frac{Protocols-ACC}{Total} \times 100$	14.23	14.02	14.03	13.92	13.97	14.04	14.04
	$\frac{ACC}{Total} \times 100$	7.17	7.27	7.00	6.96	6.98	7.00	7.06
balance	$\frac{Null}{Total} \times 100$	81.59	81.38	82.00	81.91	81.47	81.64	81.67
	$\frac{Protocols-ACC}{Total} \times 100$	12.25	12.38	11.97	12.05	12.34	12.24	12.21
	$\frac{ACC}{Total} \times 100$	6.16	6.24	6.03	6.04	6.19	6.12	6.12

Table 1: Percentage of Signals and Messages ($\xi = 0$)

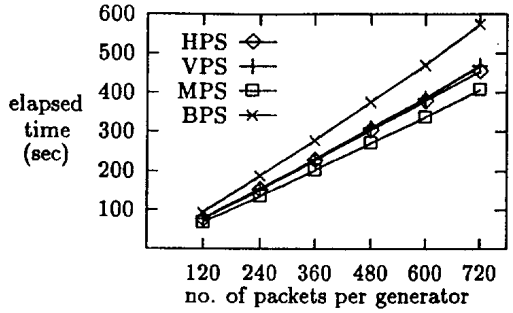


Figure 10: Elapsed Time of Different Partitioning Schemes

tions. Our investigation also shows that the diameter of processor-networking produced by the PVM system when BPS is used is always seven for eight workstations, meaning that the starvation effect of BPS is more critical as compared to VPS and MPS. The placement of processes using BPS is scattered on the workstations as compared to HPS. Therefore the inter-processor communication overhead for BPS is larger than that for HPS. In total, BPS incurs a larger elapsed time as compared to HPS.

In the speedup aspect, it is well-known that good speedup can be achieved only if the grain-size of each process is coarse. In a separate set of experiments using HPS to investigate the speedup performance of parallel MIN simulation, the workload in each process was gradually increased. The number of workstation varies from 1, 2, 4 to 8, and each generator sends 60 packets in the experiments. Let ξ be the addition workload per event. In the investigation ξ is simulated by a nested *for loop* with runtime ranging from 0.02 to 0.1 second. Figure 11 shows that as the workload to execute an event is increased so is the speedup of the parallel simulation, giving a close-to-linear speedup even with the imposed transmission protocols.

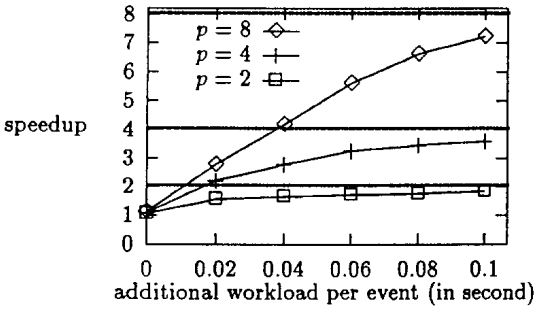


Figure 11: Speedup versus Additional Workload per Event

4.2 NULL Messages

The number of null messages, protocol signals and messages, and the number of packets (ACC messages) transmitted are also recorded. As shown in table 1, the null message ratio maintains a fairly constant for a given partitioning scheme regardless of the number of packets transmitted. This shows that the flushing mechanism is effective to reduce the exponential-growth of null messages to linear. Table 1 also sheds light on the variations in elapsed time for various partitioning schemes. As the increase in the null-message ratio correlates to the elapsed time in figure 10, we conclude that the increase in the elapsed time for various partitioning schemes is caused by the workload in processing the increasing null messages and the additional null-message communication overhead. The best partitioning scheme for the finite-buffer MIN simulation is MPS due to the close proximity of processes for intra-processor communication, and higher degree of parallelism for inter-processor communication. Although the null-message ratio for all the partitioning schemes are high there should be no alarm. Our justification to the statement is based on the complexity of the networking. In conservative simulation, transmission and relaying of null messages are required to prevent each process from waiting for itself. In the finite-buffer MIN simulation, each inter-process link is a feedback loop (bi-directional) and therefore every transmission of a useful packet is accompanied by the transmission of three null messages, resulting in *exponential* growth of null-message overhead. Nonetheless, the proposed flushing scheme reduces the overhead to *linear*. In a detailed study of several circuits, Soule [10] concluded that the number of deadlocks is high in parallel event driven simulation of logic circuits, and 50% to 80% of the execution time was being spent in the deadlock detection and recovery phase.

5 Conclusions

We have developed an asynchronous distributed simulation algorithm based on a conservative synchronization paradigm for analyzing the performance of MIN. Modeling and parallel simulation issues addressed include handling of traffic contention, routing conflicts, parallel packet transmissions, preemptive packet transmissions, backward- and forward-cascading deadlocks, and circular-wait and livelock

problems. In addition, we have shown that the *exponential* growth in null (synchronization) messages can be significantly reduced by our proposed null-message flushing scheme. Our investigation also shows that coarse-grained task partitioning and processor allocation are critical. In this aspect, an efficient transformation technique that transform an Omega MIN into a modular equivalent MIN of a larger grain to reduce inter-processor communication overhead is proposed an analyzed.

References

[1] W. Cai and S.J. Turner, *"An Algorithm for Distributed Discrete-Event Simulation - The Carrier Null Message Approach"*, Proc. of the SCS Multiconference on Distributed Simulation, pp. 3-8, January 1990.

[2] P.A. Fishwick, *"Simulation Model Design and Execution"*, Prentice-Hall, 1995.

[3] R.M. Fujimoto, *"Parallel Discrete Event Simulation"*, Comm. of ACM, Vol. 33, No. 10. pp. 31-53, October 1990.

[4] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *"PVM 3 User's Guide and Reference Manual"*, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 1993.

[5] D.R. Jefferson, *"Virtual Time"*, ACM Trans. on Programming Languages and Systems, Vol. 7, No. 3, pp. 404-425, July 1985.

[6] M. Lee and C.L. Wu, *"Performance Analysis of Circuit Switching Baseline Interconnection Networks"*, Proc. of the 11th Annual International Symposium on Computer Architecture, pp. 82-90, IEEE Computer Society Press, 1984.

[7] J. Misra and K. M. Chandy, *"Asynchronous Distributed Simulation via a Sequence of Parallel Computations"*, Comm. of the ACM, Vol. 24, No. 4, pp. 198–206, April 1981.

[8] A. Pattavina, *"Nonblocking Architectures for ATM Switching"*, IEEE Communications Magazine, pp. 38-48, February 1993.

[9] R. Rooholamini et. al, *"Finding the Right ATM Switch for the Market"*, IEEE Computer, pp. 16-28, April 1994.

[10] L.P. Soule, *"Parallel-Logic Simulation: An Evaluation of Centralized and Distributed-Time Algorithms"*, Technical Report: CSL-TR-92-527, Computer Systems Laboratory, Stanford University, 1992.

[11] S.C. Tay and Y.M. Teo, *"Conservative Parallel Simulation of Finite Buffered Multistage Interconnection Networks"*, Technical Report TRE6/94, Department of Information Systems and Computer Science, National University of Singapore, pp. 24, June 1994.

[12] Y.M. Teo and S.C. Tay, *"Conservative Parallel Simulation on a Network of Workstations"*, in Advances in Simulation Methodology and Practices, and Artificial Intelligence in Simulation, Proceedings of the European Simulation Symposium, pp. 158-163, Society for Computer Simulation International, 1994.