

Efficient Algorithms for Conservative Parallel Simulation of Interconnection Networks

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

Abstract

This paper addresses the use of parallel simulation techniques to speedup the simulation of multistage interconnection networks. The conventional null-message approach to resolving deadlock problem in conservative simulation is based on a lookahead mechanism. For some application domains, unfortunately, the lookahead information is not available. Consequently, the simulation using null messages will be trapped in a livelock. 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 to address the combinatoric explosion of using null-message in conservative simulation is also discussed. Our analysis shows that the proposed flushing mechanism effectively reduces the number of null messages from exponential to linear.

1 Introduction

During the last two decades there are increasing interests in multiprocessor systems consisting of hundreds, or even thousands, of processors. In these systems a communication network is required to connect processors or to connect processors to memory modules. Multistage interconnection network (MIN) offers a tradeoff between cost and performance. As the performance of a multiprocessor system 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 [2]. As mathematical analysis does not provide adequate results, the significance of simulation studies on the MIN's performance becomes apparent.

A promising avenue to reduce the MIN's simulation runtime is to exploit parallelism in the switching components, and map the simulation program onto multiprocessor machines. 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 packet transmission in the context of modeling and parallel simulation. Section 3 presents the enhanced process-oriented model for large MIN simulation, and a method 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 proposed process-oriented parallel simulation model consists of the following five logical processes: two packet generators, one service station with parallel servers, and two sinks. Two buffers of finite size are embedded in the station to model a blocking switch. Each process has its own local clock to indicate its simulation progress. A blocked message due to the unavailability of free space in the receiving buffer will have to wait in the generator until it is allowed to proceed. 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 message arrives. To ensure that messages are not lost at a service station when the buffer is full, the simulation processes must adhere to some *hand-shaking protocols* during transmission [4]. That is, before a message 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.
- 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".
- 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.
- 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 given in figure 1.

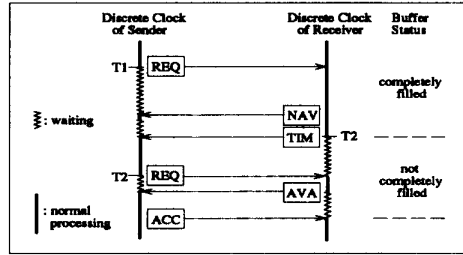


Figure 1: Resolving Traffic Contention Using Delayed-Transmission

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

Iteration Number	0	0	1	2	3	4
Discrete Clock	[0]	[0]	[4]	[4]	[7]	[10]
BOT	[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] event is selected for execution based on priority scheme. schedule next ARRIVAL[0] & DEPARTURE[0] events at clocktimes 12 & 7 respectively.	ARRIVAL[1] event is selected for execution based on minimum event occurrence time. schedule next ARRIVAL[1] event at clocktime 11.	DEPARTURE[0] event is selected for execution. send message on out-going link[0]. schedule next DEPARTURE[0] event at clocktime 10.	DEPARTURE[0] event is selected for execution. send message on out-going link[0].

BOT : [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 2: Serializing Routing Conflict

forwarding of messages to the *j*th 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. Figure 2 depicts how routing conflict problem is resolved. For illustration purpose, let the first arrival times on both links be 4 (μ sec, say) and both arrivals are to

be forwarded to the upper out-going link. Let the second arrival times on both links be 12 μ sec and 11 μ sec respectively and the switch delay be 3 μ sec. The discrete-event simulation algorithm serializes the departures of two concurrent arrivals at *clocktimes* 7 μ sec and 10 μ sec 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 3. We assume that the arrival times are the same

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

Figure 3: Parallel Transmissions on Different Out-going Links

as those in figure 2, but the concurrent arrivals at *clocktime* 4 are sent to different out-going links. Figure 3 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. A detailed discussion containing the modeling of preemptive packet transmissions, message broadcast, and resolution of forward-cascading and backward-cascading deadlock problems can be found in [4].

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 4a where SW_{ij} represents the j th switching element in the i th 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 4b).

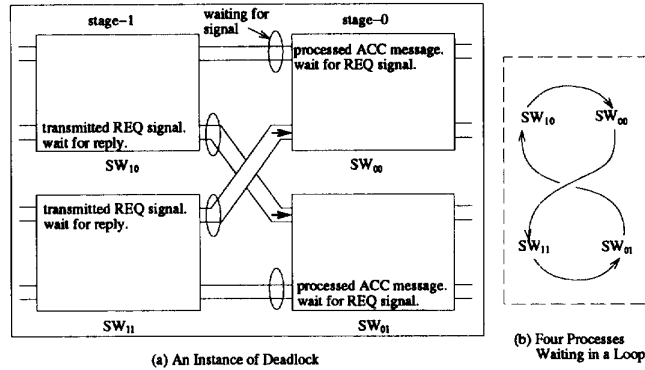


Figure 4: An Instance of Circular Wait in a 4×4 MIN Simulation

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 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 [3]. 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

A detailed description of *relaying protocols* to resolve the deadlock problem is in [4]. 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, the process is safe to proceed with the simulation, thereby resolving a deadlock.

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 [4]. 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 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].

4 Performance Analysis

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

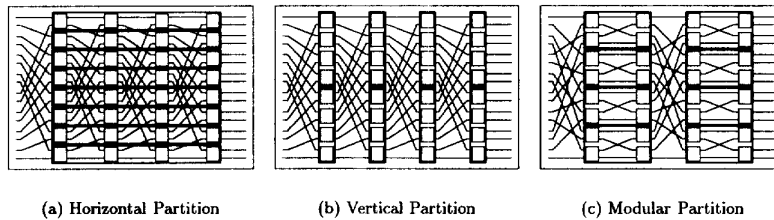


Figure 5: Partitioning Schemes for a Finite Number of Workstations

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 5. In the figure a square represents a simulation process while a rectangle represents the mapping of a number of simulation processes onto a workstation. The balance partitioning scheme (BPS) is performed by the PVM system, based on the machine performance and workload when processes are spawned.

4.1 Elapsed Time and Speedup

Figure 6 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 workstations. Our investigation also shows that the diameter of processor-networking produced by the PVM system when BPM 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 7 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.

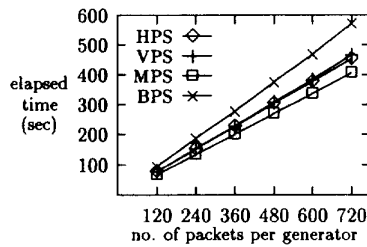


Figure 6: Elapsed Time for Different Partitioning Schemes

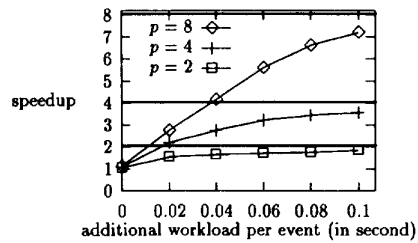


Figure 7 : Speedup versus Additional Workload per Event

4.2 NULL Message Overhead

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 remains 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 6, 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.

partition	percentage (%)	no. of messages generator			average
		240	480	720	
horizontal	$\frac{\text{Null}}{\text{Total}} \times 100$	79.80	79.90	79.83	79.84
	$\frac{\text{Protocols} - \text{ACC}}{\text{Total}} \times 100$	13.41	13.38	13.44	13.41
	$\frac{\text{ACC}}{\text{Total}} \times 100$	6.79	6.72	6.73	6.75
vertical	$\frac{\text{Null}}{\text{Total}} \times 100$	79.88	79.96	79.96	79.93
	$\frac{\text{Protocols} - \text{ACC}}{\text{Total}} \times 100$	13.38	13.35	13.35	13.36
	$\frac{\text{ACC}}{\text{Total}} \times 100$	6.74	6.69	6.69	6.71
modular	$\frac{\text{Null}}{\text{Total}} \times 100$	78.71	79.12	78.96	78.93
	$\frac{\text{Protocols} - \text{ACC}}{\text{Total}} \times 100$	14.02	13.92	14.04	13.99
	$\frac{\text{ACC}}{\text{Total}} \times 100$	7.27	6.96	7.00	7.08
balance	$\frac{\text{Null}}{\text{Total}} \times 100$	81.38	81.91	81.64	81.64
	$\frac{\text{Protocols} - \text{ACC}}{\text{Total}} \times 100$	12.38	12.05	12.24	12.22
	$\frac{\text{ACC}}{\text{Total}} \times 100$	6.24	6.04	6.12	6.14

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

5 Conclusions

We have developed parallel simulation algorithms to allow more realistic modeling and simulation of MIN as compared to mathematical analysis. As hand-shaking protocols and relaying protocols are required to produce correct simulation results and to prevent deadlock, the conservative simulation may incur large number of null-messages, aborting the simulation during execution time in the worst case. We have shown that by flushing the in-coming null messages, the overhead can be effectively reduced from *exponential* to *linear*. Our investigation also shows that the placement of simulation processes on the workstations affects the elapsed time of the program. In this aspect, it is necessary to reduce inter-processor communication overhead by using modular partitioning scheme. The speedup of the conservative simulation can be close-to-linear, even with the imposed transmission protocols, if the system to be simulated can be partitioned into a sufficient number of coarse-grain logical processes.

References

- [1] W. Cai and S. J. Turner. An Algorithm for Distributed Discrete-Event Simulation - The Carrier Null Message Approach. In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 3-8, January 1990.
- [2] M. Lee and C. L. Wu. Performance Analysis of Circuit Switching Baseline Interconnection Networks. In *Proc. of the 11th Annual International Symposium on Computer Architecture*, pages 82-90. IEEE Computer Society Press, 1984.
- [3] J. Misra and K. M. Chandy. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Comm. of the ACM*, Vol. 24, No. 4, pages 198-206, April 1981.
- [4] S. C. Tay and Y. M. Teo. *Conservative Parallel Simulation of Finite Buffered Multistage Interconnection Networks*. Technical Report TREG/94, Department of Information Systems and Computer Science, National University of Singapore, June 1994.