Mapping Parallel Simulation onto Distributed-Memory Systems: Model and Performance

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

Abstract

This paper presents an analytical model to evaluate the performance of conservative parallel simulation using a finite-buffered multistage interconnection network as an example. The model is formalized based on two important time components in parallel and distributed processing: *computation time* and *communication time.* Performance metrics such as *elapsed time*, *speedup* and *simulation bandwidth* associated with different schemes for partitioning/mapping parallel simulation onto physical processors are evaluated. Our mathematical analysis identifies the major constituents of simulation overheads in these mapping strategies necessary for improving parallel simulation efficiency. We also show that a perfectly balanced workload distribution may not necessarily translates into better performance. On the contrary, we have shown that a balanced mapping of workload may increase communication overheads resulting in a longer simulation elapsed time. The analytical performance model has been validated against implementation results from a parallel simulation model. The analytical framework can be easily extended to analyze other conservative parallel simulation schemes.

Keywords: simulation modeling, performance modeling, parallel simulation, multistage interconnection network, workload balancing

1 Introduction

Parallel discrete-event simulation (PDES) technique provides a promising avenue to reducing the runtime of large applications. Two related paradigms, *conservative approach* and *optimistic approach*, have been widely discussed [3, 4, 6, 7]. In brief, both approaches decompose the model to be simulated into loosely coupled components and simulate each component by a process. If the number of processors is sufficient, each process may be executed on one processor in parallel with the others, thereby speeding up the simulation runtime. Interaction among the processes in PDES is performed by passing timestamped messages, which usually represent events scheduled, or to be scheduled if feasible, by one process at another. To ensure causality correctness, the conservative approach executes safe events only, meaning that an event, say of occurrence time τ, is selected for execution only when all the events before τ are already executed. The optimistic approach, on the other hand, executes the events greedily. When causality error occurs, a rollback mechanism annuls those events simulated ahead of time.

Many factors contribute to the performance of a PDES program. Active research areas within this arena include new protocols, mathematical performance analysis, time parallelism, hardware support, load balancing and dynamic memory management [8]. Among them performance analysis provides an insight to the simulation modeling and identifies the cause of unsatisfactory implementations. The work done in this area is useful to simulationists as it provides the guidance necessary to improve, or even redesign, their programs. This paper focuses on the performance modeling of the conservative parallel simulation. Instead of using self-contrived queuing network models, our mathematical analysis is based on realistic multistage interconnection networks (MIN). The proposed simulation and performance models can easily be applied to study other simulation applications. The remaining of this paper is organized as follows. Section 2 gives an overview of conservative parallel simulation mechanism for modeling and simulating interconnection networks. We discuss the hand-shaking protocols used to ensure the correctness of finite-buffered simulation, the null-message relaying protocols to resolve deadlock problem, and a flushing mechanism to alleviate the combinatoric explosion of memory consumption. Section 3 abstracts the performance model for each component of MIN simulation. We derive the individual process elapsed time, and analyze the speedup and simulation bandwidth under light traffic and heavy traffic conditions. Section 4 analyzes the performance of different partitioning schemes for mapping a simulation application onto a finite number of processors. Section 5 compares the derived metrics from the proposed analytical performance model with results collected from the simulation model. We comment in particular the effect of load balancing on simulation performance. Based on the analytical results, section 6 discusses possible improvements in simulation modeling and process-to-processor mapping for achieving better performance. Lastly, section 7 contains our concluding remarks.

2 Conservative Parallel Simulation Model Overview

A detailed description of the MIN simulation is available in [9, 10]. In brief, the process-oriented parallel simulation model consists of three types of logical processes: *packet generator* (G), *switching element* (SW) and *sink* (S) (see figure 1). Based on the statistical distributions

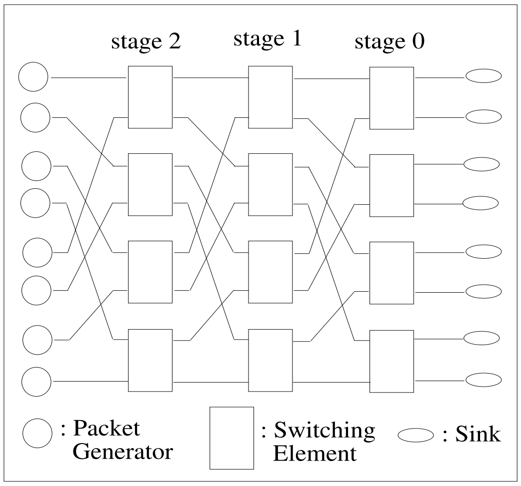


Figure 1: Components of MIN Simulation

for inter-arrival time and packet destination, each G process generates access request packets and sends them to a SW process on the $(\log_2 n - 1)$th stage. Each SW process serves as a relaying agent to forward the packets to their destinations. Two buffers of finite size are embedded in each SW process to model a blocking switch. Two types of simulation events are used: *arrival* and *departure*. A constant switch delay is used in each SW process. Each S process serves as a destination for access packets and its service time is assumed to be instantaneous. Each process has its own local clock to indicate its simulation progress. A blocked message due to the unavailability of free space in its receiving buffer will have to wait until the buffer space become available.

2.1 Ensuring the Correctness of Simulation Results

The simulation of a finite-buffered interconnection network is more sophisticated as compared to that of infinite buffered network. For example, the execution of each departure event in G and SW processes should not send the departing packet to their successor unless the receiving buffer contains at less one unoccupied buffer. This causes some complications in the simulation because the buffer status of the receiver is not known to the sender. To ensure that packets are not lost at a SW process when the buffer is completely occupied, the simulation processes therefore must adhere to some *hand-shaking protocols* during transmission [9]. That is, before a packet transmission is initiated, the sender must find out if the receiving buffer contains at least one unfilled slot to accommodate the packet. The following signals and messages are used in the hand-shaking protocols. The underlined indicates that a signal/message needs to be time-stamped.

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

Illustrations of hand-shaking protocols used in the packet transmission for light traffic and heavy traffic conditions are given in figures 2a and 2b respectively.

2.2 Resolving Deadlock Problem

In the conservative parallel MIN simulation, the execution of an arrival event in a SW process must be followed by the scheduling of a new arrival event so that causality can be ensured in each iteration. After an arrival event execution, the SW process will therefore have to enter a waiting state until the new packet arrives. If the waiting processes happen in a loop, *deadlock* will occur. In our MIN simulation null messages are used to resolve the deadlock problem.

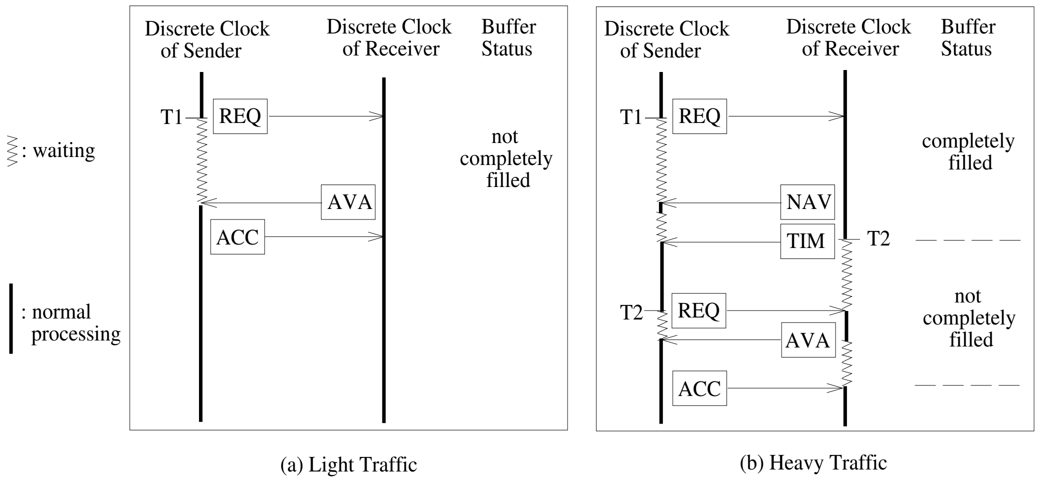


Figure 2: Hand-Shaking Protocols for Packet Transmission

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 predecessor(s) and successor(s). In this way the null messages are circulated among the network. 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.

For completeness, we also include the *null-message relaying protocols* in this paper. In the following descriptions, $toa[i]$ and $tosc[j]$ refer to the time of arrival on incoming link[i], $i \in \{0,\ 1\}$ and time of service completion on out-going link[j], $j \in \{0,\ 1\}$, respectively where *0* refers to the event occurred on upper link and *1* lower link. 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. Let $i' = i \oplus_2 1$, and $j' = j \oplus_2 1$, where $\oplus_2$ is an additive operator of modulo two.

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

$$\alpha = \begin{cases} toa[0] & \text{if in-coming link[0] buffer is not full} \\ \Theta & \textit{otherwise} \end{cases}$$

$$\beta = \begin{cases} toa[1] & \text{if in-coming link[1] buffer is not full} \\ \Theta & \textit{otherwise} \end{cases}$$

$$\gamma = \begin{cases} tosc[j'] & \text{if out-going link}[j'] \text{ has a message} \\ & \text{scheduled for transmission} \\ \Theta + switch\ delay & \textit{otherwise} \end{cases}$$

   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

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

$$\beta = \begin{cases} tosc[0] & \text{if out-going link}[0] \text{ has a message} \\ & \text{scheduled for transmission} \\ \Theta + switch\ delay & otherwise \end{cases}$$

$$\gamma = \begin{cases} tosc[1] & \text{if out-going link}[1] \text{ has a message} \\ & \text{scheduled for transmission} \\ \Theta + switch\ delay & otherwise \end{cases}$$

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

2.3 Reducing Synchronization Messages

A major drawback of using null messages in parallel simulation is that the messages may cause combinatoric explosion in memory utilization for some sophisticated processes networking. As in our MIN simulation each *SW* process is connected to the other three processes and each inter-process link is a feedback loop, the null messages grow at an exponential rate during simulation run. In the worst case, the program is aborted due to insufficient memory space to accommodate the null messages. A flushing mechanism proposed in [10] is used in our implementation to handle this problem. 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. Such a flushing mechanism is able to maintain the liveness of simulation as at least one null message is circulating in the network throughout the simulation duration. Resulting from message absorption of the flushing mechanism, the overheads are reduced from *exponential* to *linear*.

2.4 Parallel Simulation Model Summary

The proposed parallel simulation model employs a number of modeling and synchronization techniques. Space precludes us for elaborating the details. An overview is given above, details can be found in [9, 11] and a summary is given here. The proposed parallel simulation model addresses the following issues:

1. *traffic contention* - resolved by transmission protocols;

2. *routing conflicts* - resolved by discrete event execution;

3. *parallel packet transmissions* - resolved by emulating two servers per switching element;

4. *preemptive packet transmissions* - resolved by sorting incoming messages as they arrive;
5. *backward- and forward-cascading deadlocks* - resolved by using pseudo-arrival messages and STO signals;
6. *circular-wait problem* - resolved by relaying null messages;
7. *livelock problem* - resolved by giving a higher priority to departure event execution;
8. *exponential growth in null (synchronization) messages* - resolved by a null message flushing scheme;
9. *application partitioning and mapping onto distributed systems* - loading balancing using transformation techniques to obtain a modular equivalent interconnection.

3 Analytical Performance Model

Table 1 contains a list of performance parameters used in our analysis. Based on problem size, MIN's characteristics, data transmission protocols and simulation mechanism, the model formalizes the workload of each simulation component in terms of their elapsed time.

We assume that the destination of each packet transmission is uniformly distributed. Each data transmission among the simulation processes is modeled by two components: *buffer access time* and *transit time*. The reception of data is modeled by buffer access time only and transit time is excluded to prevent double accounting. We divide the analysis of simulation performance into two parts: *light traffic* ($\lambda \leq \mu$) and *heavy traffic* ($\lambda > \mu$) conditions. In the speedup analysis and simulation bandwidth analysis, we assume that $p = 2 \times n + \frac{n \times \log_2 n}{2}$, *i.e.*, the number of processors is sufficient to achieve the maximum degree of parallelism for the MIN simulation. While modeling T_p, we assume that the elapsed time of each SW process outweighs those of G and S processes.

3.1 Light Traffic

In this case we let $\lambda_{eff} = \lambda$. It is highlighted that the actual number of packets sent by each generator during simulation may be smaller due to unavailability of buffer space. To simplify the analytical model we assume that the probability of such occurrences is negligible, and we further assume that the hand-shaking protocols used in each packet transmission include only REQ and AVA signals. In other words, when a process sends a REQ signal to its receiver to find out the buffer status, the reply is always "available" (see figure 2a). It follows that the time incurred in each set of hand-shaking protocols is $2 \times T_{buffer} + T_{transit}$, and the time incurred in each packet transmission is $T_{buffer} + T_{transit}$.

3.1.1 Packet Generator

The elapsed time of each G process is modeled by three components: *packet generation* (T_1^G), *hand-shaking protocols* (T_2^G) and *packet transmission* (T_3^G). Thus,

$$T_1^G = g \times T_{generate} \quad (1)$$
$$T_2^G = g \times (2T_{buffer} + T_{transit}) \quad (2)$$
$$T_3^G = g \times (T_{buffer} + T_{transit}) \quad (3)$$

parameter	description
t	duration of simulation
λ	arrival rate used in generator
λ_{eff}	effective arrival rate used in each generator
μ	service rate used in switching element
g	total number of packets sent per generator ($\lambda_{eff} \times t$)
n	number of inputs/outputs of MIN, *i.e.* network size
p	number of processors
T_{event}	event (arrival or departure) execution time
T_{buffer}	buffer (receive or transmit) access time
$T_{transit}$	packet/message/signal transmission time
$T_{generate}$	packet generation time
$T_{account}$	packet accounting time
T^G_{total}	total elapsed time of packet generator process
T^{SW}_{total}	total elapsed time of switching element process
T^S_{total}	total elapsed time of sink process
T_1	total elapsed time when one processor is used to execute simulation processes
T_p	total elapsed time when p processors are used to execute simulation processes
T_{first_p}	total elapsed time incurred by the leftmost processor
T_{travel}	traveling time for the last transmission to reach its destination
$S_p(scheme)$	speedup metric when p processors are used for a particular mapping scheme
$BW_p(scheme)$	bandwidth (number of events simulated per unit time) for a particular mapping scheme

Table 1: Performance Model Parameters

By equations (1), (2) and (3), the total elapsed time of each G process becomes

$$T^G_{total} = g \times (T_{generate} + 3T_{buffer} + 2T_{transit}) \tag{4}$$

3.1.2 Switching Element

We model the total elapsed time of each SW process by four components: *event execution* (T^{SW}_1), *hand-shaking protocols* (T^{SW}_2), *packet transmission and reception* (T^{SW}_3) and *null-message transmissions* (T^{SW}_4).

As there are two input links to each SW process, the number of received packets is $2 \times g$. Each packet, corresponding to one arrival event, will generate one departure event and therefore the total number of events to be executed in a SW process is $2 \times 2g$. It follows that

$$\begin{aligned} T^{SW}_1 &= 4g \times T_{event} && (5) \\ T^{SW}_2 &= 2g \times (2T_{buffer} + T_{transit}) + 2g \times (2T_{buffer} + T_{transit}) \\ &= 4g \times (2T_{buffer} + T_{transit}) && (6) \end{aligned}$$

For each SW process, $2g$ packets are received and $2g$ packets are transmitted. Therefore

$$\begin{aligned} T_3^{SW} &= 2g \times T_{buffer} + 2g \times (T_{buffer} + T_{transit}) \\ &= 4g \times T_{buffer} + 2g \times T_{transit} \end{aligned} \tag{7}$$

For each transmission of AVA signal and, REQ signal and ACC message on an in-coming link and out-going link respectively, null messages will be sent on the other three links of the SW process to prevent deadlock. We assume that the null messages are then discarded by their receivers. Since there are $2g$ packets to be received and $2g$ packets to be transmitted in each SW process, we have

$$\begin{aligned} T_4^{SW} &= 2g \times 1 \times 3 \times (T_{buffer} + T_{transit}) + 2g \times 2 \times 3 \times (T_{buffer} + T_{transit}) \\ &= 18g \times (T_{buffer} + T_{transit}) \end{aligned} \tag{8}$$

It is noteworthy that the actual elapsed time in processing the null messages is larger as the receiving SW process will also select the largest time indicator among all the arrived null message and relay it to the other three SW processes. To simplify the analytical model, we assume that the workload in relaying the null messages is negligible.

By equations (5), (6), (7) and (8), the total elapsed time of each SW process becomes

$$T_{total}^{SW} = 2g \times (2T_{event} + 15T_{buffer} + 12T_{transit}) \tag{9}$$

3.1.3 Sink

The elapsed time of each S process is modeled by two components: *packet accounting* (T_1^S) and *hand-shaking protocols* (T_2^S). Since a S process is connected to each output link of the MIN, the number of received packets is g. It follows that

$$T_1^S = g \times T_{account} \tag{10}$$

$$T_2^S = g \times (2T_{buffer} + T_{transit}) \tag{11}$$

By equations (10) and (11), the total elapsed time of each S process becomes

$$T_{total}^{S} = g \times (T_{account} + 2T_{buffer} + T_{transit}) \tag{12}$$

3.1.4 Speedup

Speedup for p processors is defined as the execution time for the best serial algorithm in a single processor (T_1) divided by the execution time for the parallel algorithm using p processors (T_p) [2]. In our analysis, T_1 is modeled by the elapsed time of MIN simulation incurred by a sequential program. As causality correctness is easily ensured in sequential simulation, the use of transmission protocols is not necessary in the simulation program and is discarded in our derivation of T_1. This presents a fairer speedup measure than sometime reported in the literature whereby the value of T_1 is measured by running the parallel algorithm on one processor. This later instance certainly gives better speedup. Given that the number of simulated packets transmitted across the MIN is $n \times g$, the elapsed time of the simulation program becomes

$$\begin{aligned} T_1 &= n \times g \times T_{generate} + \frac{n \log_2 n}{2} \times 2g \times 2T_{event} + n \times g \times T_{account} \\ &= ng \times (T_{generate} + 2\log_2 n \times T_{event} + T_{account}) \end{aligned} \tag{13}$$

Assuming that the elapsed time of each SW process outweighs those of G and S processes, we have

$$T_p = T^{SW}_{total} = 2g \times (2T_{event} + 15T_{buffer} + 12T_{transit}) \tag{14}$$

By equations (13) and (14), under light traffic condition the speedup when p processors (where $p = 2 \times n + \frac{n \times \log_2 n}{2}$) are used becomes

$$\begin{aligned} S_p(nil) &= \frac{ng \times (T_{generate} + 2\log_2 n \times T_{event} + T_{account})}{2g \times (2T_{event} + 15T_{buffer} + 12T_{transit})} \\ &\approx \frac{p}{1 + \frac{15T_{buffer} + 12T_{transit}}{2T_{event}}} \end{aligned} \tag{15}$$

3.1.5 Simulation Bandwidth

Simulation bandwidth, a measure of simulation implementation efficiency, is defined as the number of events executed per second. The total number of events to be executed is

$$2 \times 2g \times \frac{n \times \log_2 n}{2} \tag{16}$$

By equations (14) and (16), the simulation bandwidth under light traffic condition becomes

$$BW_p(nil) \approx \frac{p}{T_{event} + \frac{15}{2}T_{buffer} + 6T_{transit}} \tag{17}$$

3.2 Heavy Traffic

When the arrival rate is greater than the service rate of the SW processes connected to the G processes, the buffer spaces of SW processes will be filled up progressively until all slots are occupied. Subsequently, the packet arrival rate is moderated by the SW processes to prevent packet loss. When the steady stage is reached $\lambda_{eff} = \mu$. Our worst case analysis assumes that the time interval before the steady stage is negligible. The hand-shaking protocols used by generators for packet transmission include REQ, NAV and AVA signals and TIM message to delay the packet arrival times (see figure 2b). It follows that the elapsed times incurred in each set of hand-shaking protocols used by G processes for packet transmission and in SW process on the $(n-1)th$ stage for packet reception are $5T_{buffer} + 2T_{transit}$ and $5T_{buffer} + 3T_{transit}$ respectively. The hand-shaking protocols used in the remaining SW processes, on the other hand, remain unchanged as the arrival rate to these processes has been moderated by their service rate. In this aspect, we assume that the buffer space from the *0th* to $(\log_2 n - 2)th$ stages are available throughout the simulation duration. The following analysis include only the elapsed times of each generator process and each SW process in the $(\log_2 n - 1)th$ stage. The elapsed times of the remaining processes remain unchanged.

3.2.1 Packet Generator

Packet generation time (T^G_1) and packet transmission time (T^G_3) remain unchanged. The elapsed time due to hand-shaking protocols is as follows:

$$T^G_2 = g \times (5T_{buffer} + 2T_{transit}) \tag{18}$$

By equations (1), (3) and (18), the total elapsed time of each G process becomes

$$T^{G}_{total} = g \times (T_{generate} + 6T_{buffer} + 3T_{transit}) \tag{19}$$

3.2.2 Switching Element on the $(\log_2 n - 1)$th Stage

Event execution time (T^{SW}_1) and packet transmission and reception time (T^{SW}_3) remain unchanged. The elapsed time due to hand-shaking protocols is as follows:

$$\begin{aligned} T^{SW}_2 &= 2g \times (5T_{buffer} + 3T_{transit}) + 2g \times (2T_{buffer} + T_{transit}) \\ &= 2g \times (7T_{buffer} + 4T_{transit}) \end{aligned} \tag{20}$$

For each transmission of AVA and NAV signals and, REQ signal and ACC message on an in-coming link and out-going link respectively, null messages will also be sent on the other *three* links to prevent deadlock (see section 2.2). Again, we assume that the null messages are then discarded by their receivers. Therefore

$$\begin{aligned} T^{SW}_4 &= 2g \times 2 \times 3 \times (T_{buffer} + T_{transit}) + 2g \times 2 \times 3 \times (T_{buffer} + T_{transit}) \\ &= 24g \times (T_{buffer} + T_{transit}) \end{aligned} \tag{21}$$

By equations (5), (7), (20) and (21), the total elapsed time of each SW process on the $(\log_2 n - 1)th$ stage becomes

$$T^{SW}_{total} = 2g \times (2T_{event} + 21T_{buffer} + 17T_{transit}) \tag{22}$$

3.2.3 Speedup

With heavy traffic, T_1 can be modeled by equation (13) with $g = \lambda_{eff} \times t = \mu \times t$. By equation (22), we have

$$T_p = T^{SW}_{total} = 2g \times (2T_{event} + 21T_{buffer} + 17T_{transit}) \tag{23}$$

By equations (13) and (23), the speedup for p processors (where $p = 2 \times n + \frac{n \times \log_2 n}{2}$) under heavy traffic condition becomes

$$\begin{aligned} S_p(nil) &= \frac{ng \times (T_{generate} + 2\log_2 n \times T_{event} + T_{account})}{2g \times (2T_{event} + 21T_{buffer} + 17T_{transit})} \\ &\approx \frac{p}{1 + \frac{21T_{buffer} + 17T_{transit}}{2T_{event}}} \end{aligned} \tag{24}$$

3.2.4 Simulation Bandwidth

By equations (16) and (23), the simulation bandwidth for heavy traffic condition becomes

$$BW_p(nil) \approx \frac{p}{T_{event} + \frac{21}{2}T_{buffer} + \frac{17}{2}T_{transit}} \tag{25}$$

A summary of the analytical results is given in tables 2a and 2b.

traffic	elapsed time
Light	$2g \times (2T_{event} + 15T_{buffer} + 12T_{transit})$
Heavy	$2g \times (2T_{event} + 21T_{buffer} + 17T_{transit})$

(a) Elapsed Time

traffic	speedup	simulation bandwidth
Light	$\frac{p}{1+\frac{15T_{buffer}+12T_{transit}}{2T_{event}}}$	$\frac{p}{T_{event}+\frac{15}{2}T_{buffer}+6T_{transit}}$
Heavy	$\frac{p}{1+\frac{21T_{buffer}+17T_{transit}}{2T_{event}}}$	$\frac{p}{T_{event}+\frac{21}{2}T_{buffer}+\frac{17}{2}T_{transit}}$

(b) Asymptotic Speedup and Simulation Bandwidth

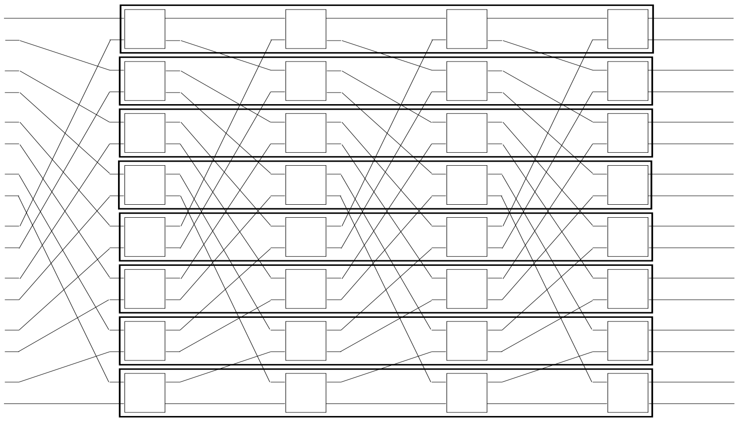
Table 2: Analytical Results for $p = 2 \times n + \frac{n \times \log_2 n}{2}$

4 Analysis for Different Mapping Schemes

Our conservative simulation model requires n G processes, $\frac{n \log_2 n}{2}$ SW processes and n S processes to simulate a $n \times n$ MIN. Mapping of such processes onto a particular processors configuration for efficient implementation is an NP-complete problem [1]. In this paper, we introduce and analyze the performance of *three mapping schemes*, namely, *horizontal partitioning scheme (HPS)*, *vertical partitioning scheme (VPS)* and *modular partitioning scheme (MPS)*. Details of the formulation and mathematical transformation used can be found in [9, 11]. Figures 3, 4 and 5 show the three schemes. A rectangular block denotes the mapping of a number of simulated processes onto *one* processor. As the G and S processes do not have to simulate the switch operations, their workloads are comparatively smaller as compared to that of a SW process. Therefore, in all mapping schemes each G process is placed together with the SW process on its out-going link in a same partition, and each S process is placed together with the SW process on its in-coming link. It is noteworthy that in MPS each module containing *four* SW processes where each process corresponding to a 2×2 switch, is executed/mapped on the same processor. However, in VPS each partition containing $\log_2 n$ SW *processes* is executed on a processor. Therefore, in the following speedup and simulation bandwidth comparison analysis we let $n = 2^{2^i}$ for some integer i. For ease of discussion, we further assume that $p = \frac{n}{2}$ and the transit time for intra-processor communication is negligible. Due to space constraint, we assume light traffic condition in the following elapsed time analysis, speedup analysis and simulation bandwidth analysis. The performance metrices for heavy traffic condition can be derived by the same steps.

4.1 Horizontal Partitioning Scheme

In HPS, the G, SW and S simulated processes on one horizontal rectangular box (see figure 3) are mapped onto one processor. Parallel simulation is terminated only after the execution of S processes in all processors are completed. Therefore, T_p is modeled by the maximum workload among processors. Before proceeding further, we re-compute the elapsed times of the affected processes due to this scheme.



Packet Generators and Sink are included in the same partition containing the Switching Element on their out–going link and incoming link respectively.

Figure 3: A Horizontal Partitioning Scheme for 16×16 Omega MIN

4.1.1 Packet Generator

As the intra-processor transit time between the generator process and the SW process on the $(\log_2 n - 1)$th stage is negligible, the total elapsed time for each G process (see equation (4)) becomes

$$\begin{aligned} T^{G}_{total} &= g \times (T_{generate} + 3T_{buffer} + 2 \times 0) \\ &= g \times (T_{generate} + 3T_{buffer}) \end{aligned} \tag{26}$$

4.1.2 SW Process on the $(\log_2 n - 1)$th Stage

While modeling the maximum workload among processors, we do not consider the top and the bottom partitions due to the diminished inter-processor links which reduce the total elapsed time. The elapsed times due to event execution (equation (5)) and packet transmission and reception (equation (7)) remain unchanged. The elapsed time due to hand-shaking protocols (equation (6)) for the SW processes on the $(\log_2 n - 1)$th stage is reduced to

$$\begin{aligned} T^{SW}_{2} &= 2g \times (2T_{buffer} + 0) + 2g \times (2T_{buffer} + T_{transit}) \\ &= 2g \times (4T_{buffer} + T_{transit}) \end{aligned} \tag{27}$$

For the same reason, the elapsed time due to null-message transmission (equation (8)) is reduced to

$$\begin{aligned} T^{SW}_{4} &= 2g \times 1 \times 3 \times (T_{buffer} + 0) + 2g \times 2 \times 3 \times (T_{buffer} + T_{transit}) \\ &= 6g \times (3T_{buffer} + 2T_{transit}) \end{aligned} \tag{28}$$

By equations (5), (7), (27) and (28), the total elapsed time of each SW process on the $(\log_2 n - 1)th$ stage becomes

$$T^{SW}_{total} = 2g \times (2T_{event} + 15T_{buffer} + 8T_{transit}) \tag{29}$$

4.1.3 SW Process on the 0th Stage:

The elapsed time due to event execution (equation (5)) remain unchanged. The elapsed time due to hand-shaking protocols (equation (6)) is reduced to

$$\begin{aligned} T_2^{SW} &= 2g \times (2T_{buffer} + T_{transit}) + 2g \times (2T_{buffer} + 0) \\ &= 2g \times (4T_{buffer} + T_{transit}) \end{aligned} \tag{30}$$

The elapsed time due to packet transmission and reception (equation (7)) becomes

$$\begin{aligned} T_3^{SW} &= 2g \times T_{buffer} + 2g \times (T_{buffer} + 0) \\ &= 4g \times T_{buffer} \end{aligned} \tag{31}$$

The elapsed time due to null-message transmission for the SW processes (equation (8)) is reduced to

$$\begin{aligned} T_4^{SW} &= 2g \times 1 \times (3T_{buffer} + T_{transit}) + 2g \times 2 \times (3T_{buffer} + 2T_{transit}) \\ &= 2g \times (9T_{buffer} + 5T_{transit}) \end{aligned} \tag{32}$$

By equations (5), (30), (31) and (32), the total elapsed time of each SW process on the *0th* stage becomes

$$T_{total}^{SW} = 2g \times (2T_{event} + 15T_{buffer} + 6T_{transit}) \tag{33}$$

4.1.4 Sink Process

The elapsed time of each S process (see equation (12)) is reduced to

$$T_{total}^{S} = g \times (T_{account} + 2T_{buffer}) \tag{34}$$

By multiplying the respective numbers of processes to equations (9), (26), (29), (33) and (34), we have

$$\begin{aligned} T_p &= 2 \times g \times (T_{generate} + 3T_{buffer}) + \\ &\quad 1 \times 2g \times (2T_{event} + 15T_{buffer} + 8T_{transit}) + \\ &\quad (\log_2 n - 2) \times 2g \times (2T_{event} + 15T_{buffer} + 12T_{transit}) + \\ &\quad 1 \times 2g \times (2T_{event} + 15T_{buffer} + 6T_{transit}) + \\ &\quad 2 \times g \times (T_{account} + 2T_{buffer}) \\ &= 2g \times (T_{generate} + 2\log_2 n \times T_{event} + T_{account} + \\ &\quad (5 + 15\log_2 n) \times T_{buffer} + 2(6\log_2 n - 5) \times T_{transit}) \end{aligned} \tag{35}$$

4.1.5 Speedup

By equations (13) and (35), the speedup when p processors (where $p = \frac{n}{2}$) are used becomes

$$\begin{aligned} S_p(HPS) &\approx \frac{ng \times 2\log_2 n \times T_{event}}{2g \times (2\log_2 n \times T_{event} + 15\log_2 n \times T_{buffer} + 12\log_2 n \times T_{transit})} \\ &\approx \frac{p}{1 + \frac{15T_{buffer} + 12T_{transit}}{2T_{event}}} \end{aligned} \tag{36}$$

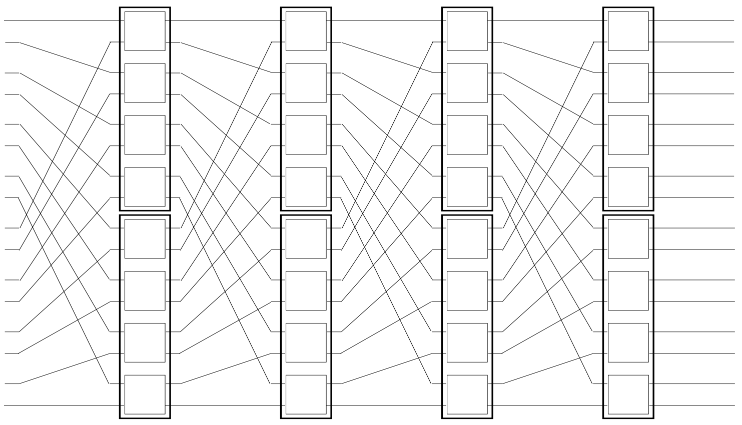
4.1.6 Simulation Bandwidth

By equations (16) and (35), the simulation bandwidth of HPS becomes

$$\begin{aligned} BW_p(HPS) &\approx \frac{2gn\ \log_2 n}{2g \times (2\log_2 n \times T_{event} + 15\log_2 n \times T_{buffer} + 12\log_2 n \times T_{transit})} \\ &\approx \frac{2p}{2T_{event} + 15T_{buffer} + 12T_{transit}} \end{aligned} \tag{37}$$

4.2 Vertical Partitioning Scheme

In VPS, each processor executes $\log_2 n$ SW processes belonging to the same MIN stage (see figure 4). During simulation, ACC packets are created by the generators, relayed by SW processes in each stage, and absorbed by the Sinks. As the mapping is performed on a stage basis, parallel simulation is completed only when the last (rightmost) processor terminates. T_p is therefore modeled by two components: *elapsed time of the first (leftmost) processor*, and the *traveling time for the last transmission to reach the sink process.*



Packet Generators and Sink are included in the same partition containing the Switching Element on their out–going link and incoming link respectively.

Figure 4: A Vertical Partitioning Scheme for 16×16 Omega MIN

By multiplying the number of processes to equations (26) and (29), we have

$$\begin{aligned} T_{first_p} &= 2\log_2 n \times g \times (T_{generate} + 3T_{buffer}) + \\ &\quad \log_2 n \times 2g \times (2T_{event} + 15T_{buffer} + 8T_{transit}) \\ &= 2g \times \log_2 n \times (T_{generate} + 2T_{event} + 18T_{buffer} + 8T_{transit}) \end{aligned} \tag{38}$$

By multiplying the required numbers of processes to equations (9), (33) and (34), and assuming one transit packet, we have

$$\begin{aligned} T_{travel} &= (\log_2 n - 2) \times 1 \times (2T_{event} + 15T_{buffer} + 12T_{transit}) + \\ &\quad 1 \times (2T_{event} + 15T_{buffer} + 6T_{transit}) + \\ &\quad 1 \times (T_{account} + 2T_{buffer}) \\ &= 2(\log_2 n - 1)T_{event} + T_{account} + \\ &\quad (15\log_2 n - 13)T_{buffer} + 6(2\log_2 n - 3)T_{transit} \end{aligned} \tag{39}$$

By equations (38) and (39), we have

$$\begin{aligned} T_p &= T_{first_p} + T_{travel} \\ &= 2g\log_2 n \times T_{generate} + 2(\log_2 n \times (2g+1) - 1)T_{event} + T_{account} + \\ &\quad (3\log_2 n \times (12g+5) - 13)T_{buffer} + 2(2\log_2 n \times (4g+3) - 9)T_{transit} \end{aligned} \tag{40}$$

4.2.1 Speedup

From equations (13) and (40), the speedup for VPS becomes

$$\begin{aligned} S_p(VPS) &\approx \frac{ng \times 2T_{event}}{2gT_{generate} + 4gT_{event} + 36gT_{buffer} + 16gT_{transit}} \\ &\approx \frac{p}{1 + \frac{T_{generate} + 18T_{buffer} + 8T_{transit}}{2T_{event}}} \end{aligned} \tag{41}$$

4.2.2 Simulation Bandwidth

From equations (16) and (40), we have

$$BW_p(VPS) \approx \frac{2p}{T_{generate} + 2T_{event} + 18T_{buffer} + 8T_{transit}} \tag{42}$$

4.3 Modular Partitioning Scheme

The MPS is based on partitions of *four SW processes* spread over two MIN stages as shown in figure 5. A proof of the transformation from the Omega MIN to its equivalent modular topology can be found in [10]. Each processor executes $\log_2 n$ SW processes arranged in $\frac{\log_2 n}{4}$ blocks. For the same reason that sink processes are contained in the rightmost module only, we also model T_p by the same two components as in VPS.

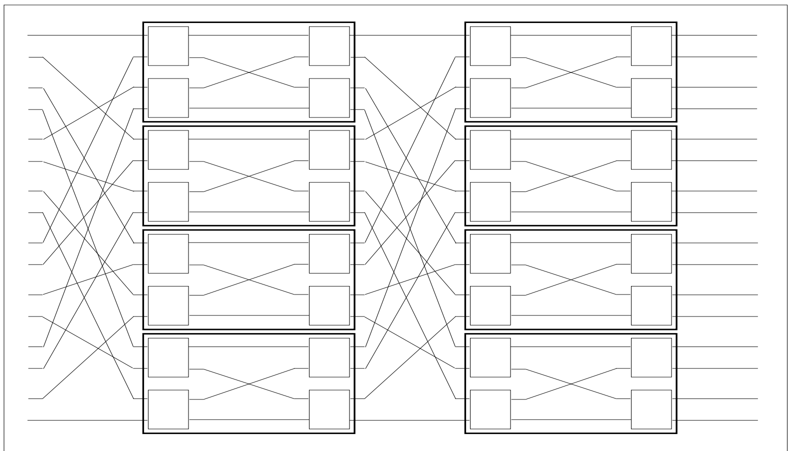


Figure 5: A Modular Partitioning Scheme for 16×16 Omega MIN

In the following derivation, we assume that $n \geq 16$. First, we compute the elapsed time of the leftmost block. By multiplying the number of processes to equations (4), (9) and (29), and assuming $T_{transit} = 0$ in the first two equations, we have

$$\begin{aligned}
&4 \times g \times (T_{generate} + 3T_{buffer} + 0) + \\
&2 \times 2g \times (2T_{event} + 15T_{buffer} + 0) + \\
&2 \times 2g \times (2T_{event} + 15T_{buffer} + 8T_{transit}) \\
&= 4g \times (T_{generate} + 4T_{event} + 33T_{buffer} + 8T_{transit})
\end{aligned} \tag{43}$$

By multiplying the number of modules in the leftmost processor to equation (43), we obtain

$$\begin{aligned}
T_{first_p} &= \frac{\log_2 n}{4} \times 4g \times (T_{generate} + 4T_{event} + 33T_{buffer} + 8T_{transit}) \\
&= g \times \log_2 n \times (T_{generate} + 4T_{event} + 33T_{buffer} + 8T_{transit})
\end{aligned} \tag{44}$$

Next, we compute the elapsed time of each transmission across an intermediate block. By multiplying the respective numbers of processes in equations (29) and (33), and assuming one transit packet, we have

$$\begin{aligned}
&(2T_{event} + 15T_{buffer} + 6T_{transit}) + \\
&(2T_{event} + 15T_{buffer} + 8T_{transit}) \\
&= 4T_{event} + 30T_{buffer} + 14T_{transit}
\end{aligned} \tag{45}$$

The elapsed time across the rightmost block is computed by equation (33), setting $T_{transit} = 0$ in equations (9) and (12), and assuming one transit packet, giving

$$\begin{aligned}
&2T_{event} + 15T_{buffer} + 6T_{transit} + \\
&2T_{event} + 15T_{buffer} + 0 + \\
&T_{account} + 2T_{buffer} + 0 \\
&= 4T_{event} + T_{account} + 32T_{buffer} + 6T_{transit}
\end{aligned} \tag{46}$$

By multiplying the respective numbers of blocks to equations (45) and (46), we have

$$\begin{aligned}
T_{travel} &= (\frac{\log_2 n}{2} - 2) \times (4T_{event} + 30T_{buffer} + 14T_{transit}) + \\
&\quad 1 \times (4T_{event} + T_{account} + 32T_{buffer} + 6T_{transit}) \\
&= (2\log_2 n - 4)T_{event} + T_{account} + (15\log_2 n - 28)T_{buffer} + \\
&\quad (7\log_2 n - 22)T_{transit}
\end{aligned} \tag{47}$$

By equations (44) and (47), we have

$$\begin{aligned}
T_p &= T_{first_p} + T_{travel} \\
&= g\log_2 n \times T_{generate} + 2((2g+1)\log_2 n - 2) \times T_{event} + T_{account} + \\
&\quad (3(11g+5)\log_2 n - 28) \times T_{buffer} + ((8g+7)\log_2 n - 22) \times T_{transit}
\end{aligned} \tag{48}$$

4.3.1 Speedup

By equations (13) and (48), the speedup for MPS becomes

$$S_p(MPS) \approx \frac{ng \times 2T_{event}}{gT_{generate} + 2(2g+1)T_{event} + 3(11g+5)T_{buffer} + (8g+7)T_{transit}}$$

$$\approx \frac{p}{1 + \frac{\frac{T_{generate}}{2} + \frac{33}{2} T_{buffer} + 4T_{transit}}{2T_{event}}} \tag{49}$$

4.3.2 Simulation Bandwidth

By equations (16) and (35), we have

$$BW_p(MPS) \approx \frac{2p}{\frac{T_{generate}}{2} + 2T_{event} + \frac{33}{2} T_{buffer} + 4T_{transit}} \tag{50}$$

A summary of the analytical results is given in tables 3a and 3b.

scheme	elapsed time
HPS	$2g \times (T_{generate} + 2\log_2 n \times T_{event} + T_{account} +$ $(5 + 15\log_2 n) \times T_{buffer} + 2(6\log_2 n - 5) \times T_{transit})$
VPS	$2g\log_2 n \times T_{generate} + 2(\log_2 n \times (2g+1) - 1)T_{event} + T_{account} +$ $(3\log_2 n \times (12g+5) - 13)T_{buffer} + 2(2\log_2 n \times (4g+3) - 9)T_{transit}$
MPS	$g\log_2 n \times T_{generate} + 2((2g+1)\log_2 n - 2) \times T_{event} + T_{account} +$ $(3(11g+5)\log_2 n - 28) \times T_{buffer} + ((8g+7)\log_2 n - 22) \times T_{transit}$

(a) Elapsed Time

scheme	speedup	simulation bandwidth
HPS	$\frac{p}{1+\frac{15T_{buffer}+12T_{transit}}{2T_{event}}}$	$\frac{2p}{2T_{event}+15T_{buffer}+12T_{transit}}$
VPS	$\frac{p}{1+\frac{T_{generate}+18T_{buffer}+8T_{transit}}{2T_{event}}}$	$\frac{2p}{T_{generate}+2T_{event}+18T_{buffer}+8T_{transit}}$
MPS	$\frac{p}{1+\frac{\frac{T_{generate}}{2}+\frac{33}{2}T_{buffer}+4T_{transit}}{2T_{event}}}$	$\frac{2p}{\frac{T_{generate}}{2}+2T_{event}+\frac{33}{2}T_{buffer}+4T_{transit}}$

(b) Asymptotic Speedup and Simulation Bandwidth

Table 3: Analytical Results for $p = \frac{n}{2}$

5 Model Validation and Performance Analysis

We implemented the conservative parallel MIN simulation model in C language on a network of workstations which mimics a distributed-memory parallel computer. PVM software [5] was used for spawning the simulation processes and for handling message passing. The simulation model results presented are based on simulating a 16×16 Omega MIN on a network of *eight* homogeneous SUN workstations connected using a 10Mbps Ethernet network. Altogether 64 PVM processes are spawned on the processors.

Five parameter values used in the analytical model are obtained by taking measurements on the workstation network (see table 4). Additional statistical measures used in the analysis are

parameter	time (μsec)
$T_{generate}$	926
T_{event}	2990
$T_{account}$	479
T_{buffer}	3238
$T_{transit}$	1821

Table 4: Performance Parameter Measurement

defined. Let s_i be a simulation process and $|s_i|$ denotes its workload. In the following analysis, the workload for each simulation process is represented by the size of its object codes, and normalized to that of the S process. For simplicity, we let $|S| = 1$. Thus, the workloads for $|G|$ and $|SW|$ are 1.93 and 6.24 respectively. The normalized coefficient of $T_{transit}$ is denoted by $\mathcal{T}$. The percentage deviation between the measured and the predicted values δ is computed as $\frac{|measured - predicted|}{predicted} \times 100$. The average of δ over six different sets of simulation parameters is denoted by $\overline{\delta}$.

Let M be a set containing all s_i required in a MIN simulation. Let $\widehat{P}$ be the processor that incurs the largest workload among all processors. We define the *workload distribution coefficient* (ω) as $\omega = \frac{\sum_{s_i \in M} |s_i|}{\sum_{s_j \in \widehat{P}} |s_j|}$. The load balancing factor (η) is then define as $\eta = \frac{p}{\omega}$. Thus, the larger the value of η, the more unbalance is the workload distribution. The smallest (best) value of η equals to 1 corresponds to a perfectly balanced distribution of workload.

We first compare the elapsed times among the different schemes, and between the simulation model *(measured)* and the analytical model *(predicted)*. For VPS and MPS, the latter outperforms in terms of elapsed time as shown in table 5. This phenomenon is due to the poor

scheme		no. of messages per generator						$\overline{\delta}$ (%)	$\mathcal{T}$	η
		120	240	360	480	600	720			
HPS	measured	76.79	151.15	226.49	304.09	377.68	454.93	3.66	4	1.00
	predicted	*73.19*	*146.38*	*219.60*	*292.76*	*365.95*	*439.14*	-	-	-
VPS	measured	79.61	160.14	239.93	322.23	403.43	482.46	4.78	2	1.31
	predicted	*76.75*	*153.28*	*229.81*	*306.33*	*382.86*	*459.39*	-	-	-
MPS	measured	67.29	134.00	200.25	269.73	337.18	406.74	4.31	1	1.06
	predicted	*64.58*	*129.03*	*193.48*	*257.93*	*322.38*	*386.83*	-	-	-

Table 5: Comparison of Elapsed Times (in second)

workload distribution and larger inter-processors communication overheads of VPS ($\mathcal{T} = 2$) compared to those of MPS ($\mathcal{T} = 1$). The comparisons of VPS with HPS is more interesting. Although the interprocessor communication overheads of VPS is only half of that of HPS, it seems that the imbalance workload factor is more dominant, making the elapsed time of VPS ($\eta = 1.31$) longer than that of HPS ($\eta = 1$). In this aspect, our explanation is that for VPS the worst number of processes in the processors is larger than that of HPS. Therefore, the elapsed time of VPS consists of a larger number of context switching time incurred by the task scheduler of the UNIX operating system. The comparison of HPS ($\eta = 1$) and MPS ($\eta = 1.06$)

is rather counter-intuitive. We observe that the HPS, a perfect load balancing scheme, incurs a significantly larger elapsed time as compared to that of a less balance scheme. Such an observation contradicts to the common belief, that the workload distribution among processors must be balanced in order to improve elapsed time. We note here that other factors such as inter-processors communication overheads incurred by a mapping scheme should also be taken into account while balancing the workload distribution. As the inter-processors communication overheads incurred by HPS is larger than that of MPS, its elapsed time is therefore aggravated despite the balanced workload distribution.

The averaged percentage deviations $\overline{\delta}$ show that accuracy of the analytical model for predicting elapsed time is good for all the three mapping schemes. However, the predicted timings show slightly better performance than the measured simulation model timings because the context switching time incurred by the scheduler of the UNIX OS is not accounted in our analytical model.

Tables 6 and 7 show that our performance model closely predicts the parallel simulation speedups and simulation bandwidths. The predicted values for all the three mapping schemes

scheme		no. of messages per generator						$\overline{\delta}$ (%)
		120	240	360	480	600	720	
HPS	measured	0.61	0.62	0.62	0.61	0.62	0.63	6.31
	predicted	*0.66*	*0.66*	*0.66*	*0.66*	*0.66*	*0.66*	-
VPS	measured	0.59	0.58	0.58	0.59	0.60	0.59	6.61
	predicted	*0.63*	*0.63*	*0.63*	*0.63*	*0.63*	*0.63*	-
MPS	measured	0.72	0.71	0.72	0.71	0.71	0.71	4.89
	predicted	*0.75*	*0.75*	*0.75*	*0.75*	*0.75*	*0.75*	-

Table 6: Comparison of Speedup

scheme		no. of messages per generator						$\overline{\delta}$ (%)
		120	240	360	480	600	720	
HPS	measured	200.02	203.24	203.45	202.05	203.35	202.58	2.98
	predicted	*209.85*	*209.87*	*209.86*	*209.86*	*209.87*	*209.87*	-
VPS	measured	191.68	191.07	192.97	191.94	191.54	192.73	4.26
	predicted	*200.12*	*200.41*	*200.51*	*200.57*	*200.59*	*200.61*	-
MPS	measured	228.26	229.25	230.11	227.78	227.77	226.58	4.13
	predicted	*237.85*	*238.09*	*238.16*	*238.20*	*238.23*	*238.24*	-

Table 7: Comparison of Simulation Bandwidth (events/second)

are slightly better than the measured values for the same reason. The speedup efficiency is unacceptably low, i.e. around 10%. Our analysis reveals that this is due to the higher than expected cost of PVM communication. Better speedup can be obtained on parallel machine where communication is much more efficient and less costly. In the following section, we present a breakdown of various communication costs, and analyze the effect of reducing these overheads on simulation performance.

In summary, the measured and predicted values for elapsed time, speedup and simulation bandwidth show the following important findings:

- A perfectly balanced workload distribution, such as HPS, may not necessary translate into better performance.
- Inter-processors communication overheads may cause dominant aggravation to the program elapsed time.
- A good mapping scheme should take into account the load balancing as well as communication overheads. Ignoring either factor may result in poor implementation performance.

6 Improving Implementation Performance and Simulation Modeling

As shown in our performance model, the communication overheads comprise two major components: *buffer access time* and *message transmission time*. In order to improve the implementation performance, we therefore have to reduce these two constituents. Table 8 depicts the asymptotic elapsed times for MIN simulation. The comparisons show that by transforming the MIN into a more modular equivalent interconnection, the message transit time can be significantly reduced. Nonetheless, table 6 shows that even for MPS the speedup of parallel simulation using eight workstations is less than one, i.e. it is slower to run the simulation program on the parallel processors than on a uniprocessor. The poor performance is caused by the high cost of communication in the PVM platform (refer to table 4). Using the MPS as an example, we analyze how the overheads affect parallel simulation performance. We compared the implications of setting overheads to zero (*no overheads*), buffer access time to zero (*zero* T_{buffer}), message transmission time to zero (*zero* $T_{transit}$), reducing T_{buffer} to half ($T_{buffer}/2$) and to a quarter ($T_{buffer}/4$), and reducing $T_{transit}$ by half ($T_{transit}/2$).

scheme	elapsed time ($\times 2g \log_2 n$)
HPS	$2T_{event} + 15T_{buffer} + 12T_{transit}$
VPS	$T_{generate} + 2T_{event} + 18T_{buffer} + 8T_{transit}$
MPS	$\frac{T_{genarate}}{2} + 2T_{event} + \frac{33}{2}T_{buffer} + 4T_{transit}$

Table 8: Asymptotic Elapsed Time for Different Partitioning Schemes

Figures 6, 7 and 8 depict the elapsed time versus the number of messages, and the speedup and the simulation bandwidth against the number of workstations respectively. In particular, figure 7 shows that good parallel simulation speedup can be achieved even with the imposed transmission protocols, provided the communication overheads are negligible in comparison with the event grain size. We observed that the worst bottleneck is attributed to T_{buffer} followed by $T_{transit}$. Thus, more effort should be placed on reducing PVM-based memory allocation time, and improving access protocols to reduce overall message buffering time than on improving the network transmission speed. A significant reduction in elapsed time is observed when T_{buffer} is halved as compared to setting $T_{transit}$ to zero (figure 6).

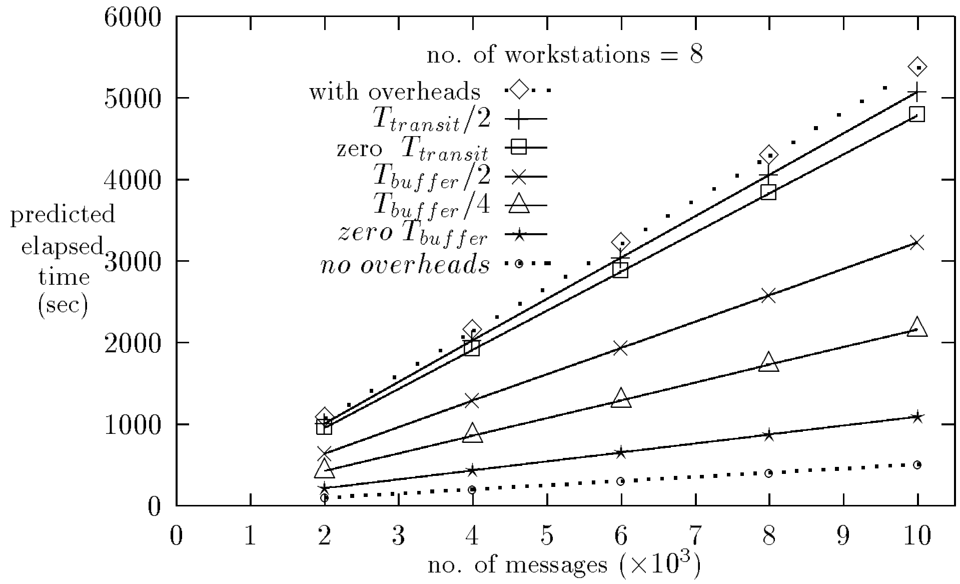


Figure 6: Elapsed Time Analysis (using MPS Mapping Scheme)

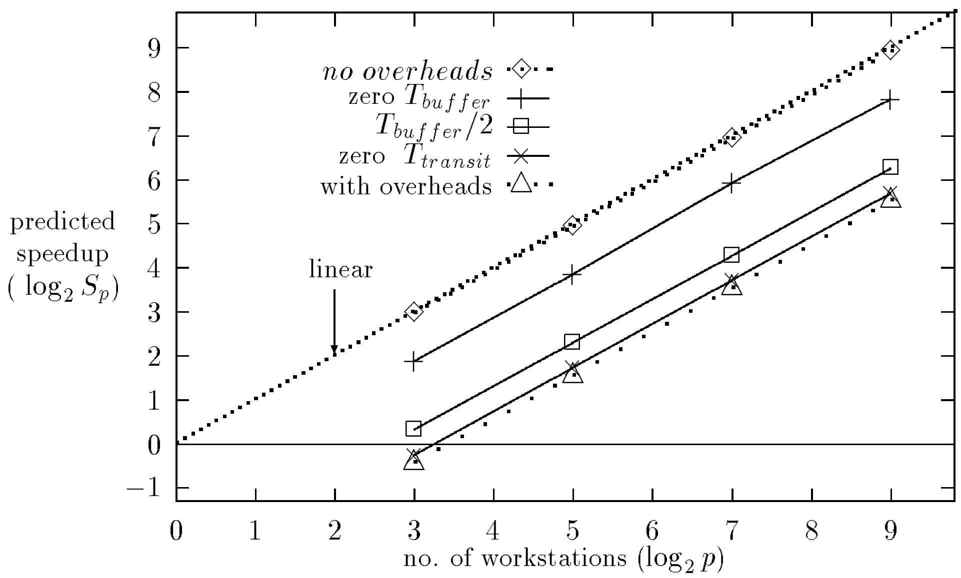


Figure 7: Speedup Analysis (using MPS Mapping Scheme)

Buffer access time and message transit time can be reduced by improving the simulation model. We proposed partitioning the MIN's configuration into p modules, where each module is simulated by one process that is mapped onto one processor (see figure 9). The new partitioning/mapping scheme will significantly reduce intra-processor communication overheads. This also results in coarser-grain process thereby making the inter-processors communication overheads less significant as compared to the previous model.

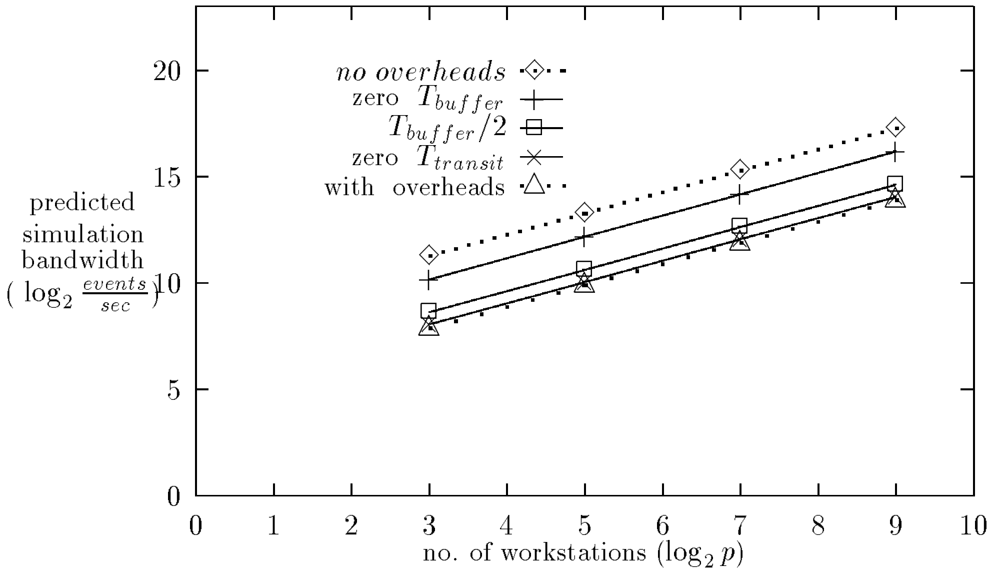


Figure 8: Simulation Bandwidth Analysis (using MPS Mapping Scheme)

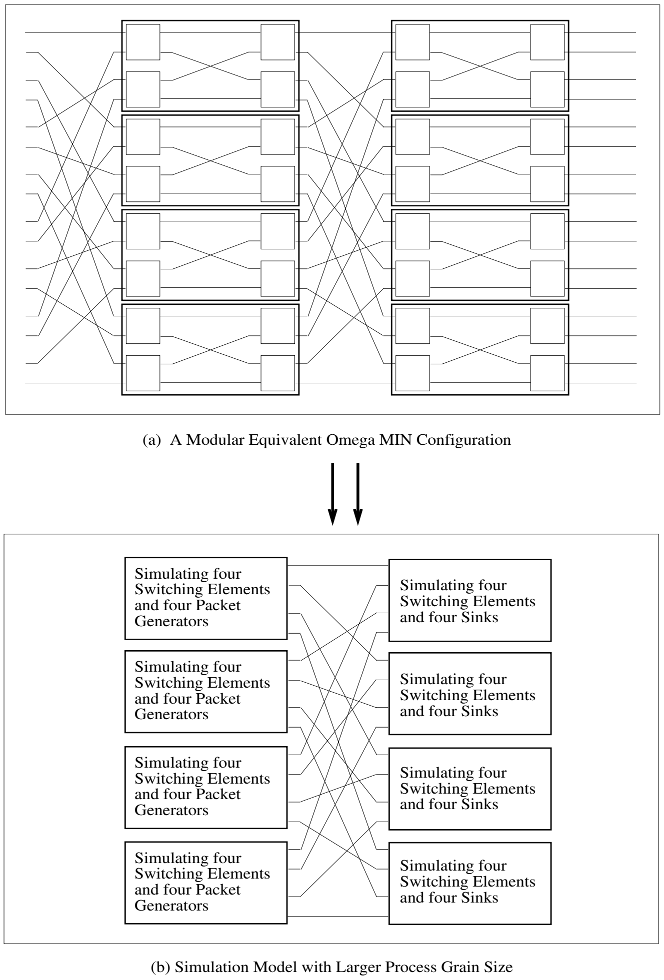


Figure 9: Improved Mapping Scheme for 16×16 Omega MIN using 8 Processors

7 Conclusions

We have developed an analytical model that characterized the performance of a conservative parallel simulation model using five timing parameters. Performance measures are derived for simulation elapsed time, speedup and bandwidth for both light and heavy traffic conditions. Validation experiments comparing the performance metrics from the analytical model and the simulation model show an acceptable 5% difference. We analyzed the performance for three process-to-processor mapping schemes to identify the main causes of poor performance. Using the performance model, we proposed improvements to simulation modeling and in implementation. The performance model can be easily extended to analyze other applications using the same parallel mechanism. Our work reveals that in the PVM-based implementation reducing buffer access time is more critical than network transmission time. In addition, we dispel the common belief that to reduce the elapsed time of a parallel program the workload distribution among the processors must be balanced. We observed that other factors such as inter-processors communication overheads may also cause dominant aggravation to the simulation performance despite the balanced workload distribution. Therefore, a balanced workload distribution may not necessary translate into better performance. Both the analytical and implementation results confirm such an exceptional phenomenon.

PDES is increasingly studied to exploit simulation parallelism, and to speedup simulation by utilizing idle processor cycles notably in distributed computing. However, the two main synchronization schemes used, conservative and optimistic, are intuitively promising but generally inefficient in practice. This is due to high simulation synchronization overheads, fine granularity of events, strong coupling of exploitable parallelism and the system to be simulated, inherent limited lookahead, etc. One important thread of investigation is to study the harmonization of synchronization needed in organizing PDES and in parallel processing. In this aspect, much remains to be done to realise the full potential of PDES.

References

[1] S. Bokhari, *"On Mapping Problem"*, IEEE Trans. of Computers, Vol. C-30, No. 3, pp. 207-214, March 1981.

[2] A.L. Decegama, *"The Technology of Parallel Processing - Parallel Architectures and VLSI Hardware Volume 1"*, Prentice-Hall, 1989.

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

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

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

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

[7] J. Misra, *"Distributed Discrete-Event Simulation"*, Computing Surveys, Vol. 18, No. 1, pp. 39-65, March 1986.

[8] D. Nicol and R. Fujimoto, *"Parallel Simulation Today"*, in Annals of Operations Research: Simulation and Modeling, edited by O. Balci, Vol. 53, 1994.

[9] 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.

[10] Y.M. Teo and S.C. Tay, *"Efficient Algorithms for Conservative Parallel Simulation of Interconnection Networks"*, Proc. of International Symposium on Parallel Architectures, Algorithms and Networks, Japan, pp. 286-293, IEEE Computer Society Press, December 1994.

[11] Y.M. Teo and S.C. Tay, *"Modeling and Efficient Distributed Simulation of Multistage Interconnection Network"*, Proc. of International Conference on Algorithms and Architectures for Parallel Processing, Australia, IEEE Computer Society Press, April 1995.