A Parallelism Analyzer for Conservative Parallel Simulation

Yung-Chang Wong, Shu-Yuen Hwang, and Jason Yi-Bing Lin

Abstract—**Most small-scale simulation applications are implemented by sequential simulation techniques. As the problem size increases, however, sequential techniques may be unable to manage the time complexity of the simulation applications adequately. It is natural to consider re-implementing the corresponding large-scale simulations using parallel techniques, which have been reported to be successful in reducing the time complexity for several examples. However, parallel simulation may not be effective for every application. Since the implementation of parallel simulation for an application is usually very expensive, it is required to investigate the performance of parallel simulation for a particular application before re-implementing the simulation. The Chandy-Misra parallel, discrete-event simulation paradigm has been utilized in many large-scale simulation experiments, and several significant extensions have been based on it. Hence the Chandy-Misra protocol is adopted here as a basic model of parallel simulation to which our performance prediction techniques are applied. For an existing sequential simulation program based on the process interaction model, this paper proposes a technique for evaluating Chandy-Misra parallel simulation without actually implementing the parallel program. The idea is to insert parallelism analysis code into the sequential simulation program. When the modified sequential program is executed, the time complexity of the parallel simulation based on the Chandy-Misra protocol is computed.**

Our technique has been used to determine whether a giant Signaling System 7 simulation (sequential implementation) should be re-implemented using the parallel simulation approach.

Index Terms— **Chandy-Misra protocol, critical path analysis, Discrete event simulation, parallelism, parallel simulation**

I. Introduction

IN A PARALLEL SIMULATION, the simulated system is partitioned into several subsystems, each of which consists of a nonoverlay subset of state variables. These subsystems are concurrently simulated by a set of processes that communicate by exchanging timestamped messages. The events scheduled for a process can modify only the state variables of the corresponding subsystem. The processes execute concurrently to complete a simulation run. To produce the correct simulation results, the executions of the processes must follow a set of synchronization rules [1]. The performance of parallel simulation depends on two factors: the parallelism existing in the system to be simulated and the overhead of the parallel simulation protocol running on a particular computer architecture. The inherent parallelism of a simulation application was first studied by Berry and Jefferson [2] and Livny [3]. Algorithms have been proposed to study the inherent application parallelism when every process is executed by a separate processor. Lin [4] proposed an inherent-parallelism analysis algorithm for the case where more than one process may be mapped to a processor under different process scheduling policies. This paper extends previous results by considering both the inherent parallelism and the parallel simulation protocol overhead. The paper proposes a parallelism analysis algorithm for the Chandy-Misra protocol [5], in which more than one process may be mapped to a processor. The parallelism analysis algorithm is integrated with the sequential simulation program. When this modified sequential simulation is executed, the time complexity of the parallel simulation based on the Chandy-Misra protocol is also computed. Our technique is a powerful tool for determining the performance of the Chandy-Misra parallel simulation for an existing sequential simulation program.

This paper is organized as follows. Section II introduces the concept of event precedence graph. Section III describes the Chandy-Misra protocol. Section IV proposes a parallelism analysis algorithm for Chandy-Misra parallel simulation.

Manuscript received March 18, 1993; revised August 1994. This work was supported in part by the National Science Council, Republic of China, under Grants NSC 82-0408-E-009-115 and NSC 83-0408-E-009-002.

Y.-C. Wong and S.-Y. Hwang are with the Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C.

J. Y.-B. Lin is with Bellcore, Morristown, NJ 07962 USA (liny@mookie.bellcore.com).

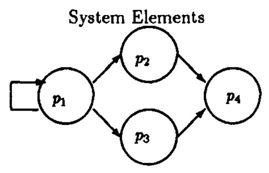
IEEE Log Number 9412400.

II. The Event Precedence Graph

The execution of a discrete event simulation follows causality constraints, and the relationships between the events can be described by an *event precedence graph* [2], [6], [7], [4]. The concept of event precedence graph is illustrated by the following example. Consider the simple network in Fig. 1(a). Fig. 1(b) shows the event precedence graph for a simulation scenario of the network.

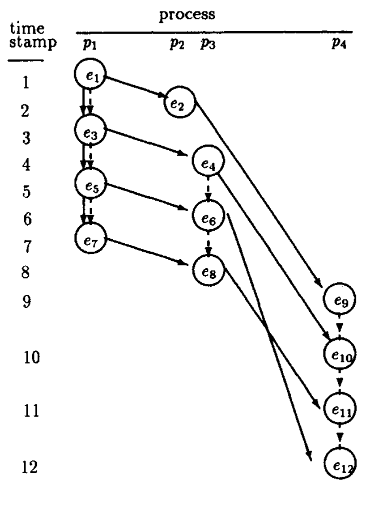
In this figure, the timestamp of event e_i is i.

In the event precedence graph, a vertex represents the occurrence of an event. A dashed arrow from event e_i to event e_j means that both e_i and e_j are scheduled for the same process, and e_i occurs earlier than e_j does (cf. events e_4 and e_6 in Fig. 1(b)). A solid arrow from e_i to e_j means that the scheduling of e_j is due to the occurrence of e_i (cf. events e_1 and e_2 in Fig. 1(b)). To correctly simulate the behavior of the network, event e_i must be processed before e_j if there is an arrow (either dashed or solid) from e_i to e_j in the event precedence graph. In a sequential simulation implementation, all events are processed in nondecreasing timestamp order. This sequential execution engine guarantees



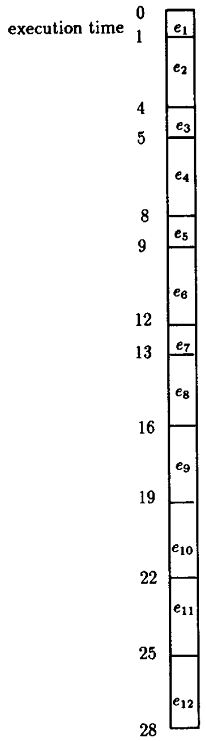
The simulated system

(a)



The event precedence graph

(b)



The sequential execution

(c)

Fig. 1. Topology, the event precedence graph, and the sequential execution time.

that the relationship in the event precedence graph is not violated. Suppose that the time to process event e_1, e_3, e_5 or e_7 is 1 unit, and the time to process any one of the other events is 3 units. The execution order and the elapsed time after an event is executed in a sequential simulation are given in Fig. 1(c).

In Fig. 1(b), an event execution time is associated with each vertex (i.e., event). A communication delay is associated with each solid arrow (the cost for the dashed arrow is 0). Since the graph is acyclic, a maximal weighted path can be found. This path is called the *critical path* and its cost is the minimal time required to finish the execution of the parallel simulation. The critical path does not consider the overhead for parallel simulation protocols. In other words, the cost for the critical path is a lower bound for the execution time of any parallel simulation approach. To evaluate the time complexity for a particular parallel simulation protocol, new techniques (such as the Chandy-Misra parallelism analyzer developed in this paper) are required.

III. The Chandy-Misra Protocol

From the definition of the event precedence graph, a parallel simulation protocol is correct if all events occurring at a logical process are executed in nondecreasing timestamp order. The Chandy-Misra protocol follows two waiting rules to satisfy the causality constraint. We first describe the assumptions of a Chandy-Misra simulation.

The FIFO message sending assumption: Communication between two processors preserves the first-in-first-out (FIFO) property (i.e., the messages are received in the order they are sent).

The static topology assumption: The network topology is static (i.e., the communication channels between processes do not change during simulation). If process p_i may send a message to process p_j, then a communication link is directed from p_i to p_j. The link is called an *input channel* (*output channel*) for p_j (p_i).

The original Chandy-Misra protocol [5] further assumes that the buffer capacity of a process to store the incoming messages is limited. The purpose of this restriction was to limit the memory usage of a Chandy-Misra simulation. However, Lin and Preiss [8] and Jefferson [9] showed that in general limiting the input buffer capacities of processes does not limit the total memory usage for a Chandy-Misra simulation. We assume that the input buffer capacity of a process is unlimited. For simplicity, we make three assumptions about the simulated network. It is easy to see that our results can be generalized.

- There are three types of processes in the simulation. A *source* process does not receive any messages from other processes. A *server* process may send and receive messages. A *sink* process does not send any messages to other processes. In Fig. 1(a), p_1 is a source process, p_2 and p_3 are server processes, and p_4 is a sink process.
- A source process may schedule events to itself. Neither server processes nor sink processes may schedule events to themselves.
- **[The FIFO event generation assumption]** A source process p_i schedules events in nondecreasing timestamp order. That is, if p_i schedules e_1 earlier than e_2, then $e_1.ts \leq e_2.ts$ (where $e.ts$ is the timestamp of e).

The concept *lookahead* is used in the Chandy-Misra protocol. Let $e.E$ be the set of events scheduled due to the execution of event e. The lookahead of e is a quantity $\varepsilon(e)$ such that for every event $e' \in e.E$, $e'.ts \geq e.ts + \varepsilon(e)$. In a queueing network simulation, lookahead can be regarded as the minimum service time of a server. Studies [10], [11] have indicated that the larger the lookahead values, the better the performance of a Chandy-Misra simulation becomes. Several techniques [12], [13] have been proposed to explore lookahead.

Two waiting rules ensure the correctness of the Chandy-Misra protocol.

The input waiting rule: Before process p_i executes an event e, p_i must receive from each of its input channels an event (including e), and e must have the smallest timestamp among the events in the input channels.

The output waiting rule: Consider an event e' created at process p_i, which is scheduled for process p_j. Event e' is sent to p_j after p_i has started executing event e, where $e'.ts \leq e.ts + \varepsilon(e)$. If several events satisfy this inequality, then they are sent to p_j in nondecreasing timestamp order.

The output waiting rule and the FIFO message sending assumption ensure that a process always receives messages from an input channel in nondecreasing timestamp order. This property, together with the input waiting rule, guarantees that all events occurring at a process are executed in nondecreasing timestamp order. Note that the output waiting rule is not required for a source process because of the FIFO event generation assumption. We define the lookahead of every event

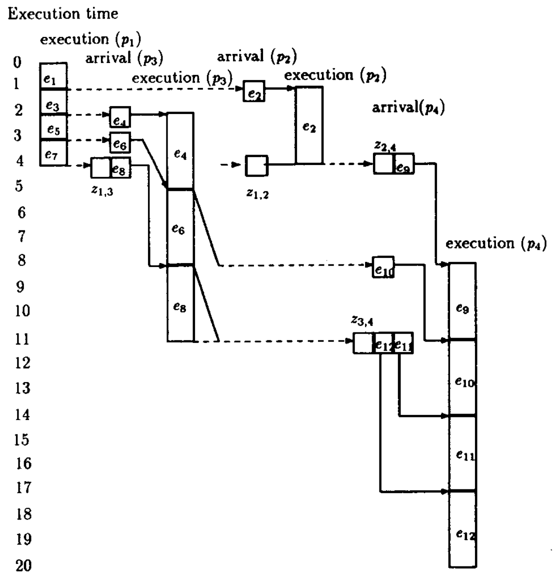


Fig. 2. Execution of the Chandy-Misra simulation for the 4-process network.

e executed at a source process as $\varepsilon(e) = \infty$. (Note that $\varepsilon(e) = \infty$ does not imply that the execution of e at a source process will schedule an event with timestamp ∞. The infinite lookahead value is used to bypass the output waiting rule for the source process.)

Two types of control messages are introduced in the Chandy-Misra protocol: *end-of-simulation* (eos) messages and *null* messages. The eos messages are used to terminate the parallel simulation. After a source process has generated the last event, it sends an eos message to each of its output channels and enters the termination state. An eos message has timestamp ∞ and lookahead value ∞. When a server process p_i executes an eos message, it generates and sends an eos message to each of its output channels. Then p_i enters the termination state. When a sink process executes an eos message, it simply enters the termination state. All processes eventually enter the termination state and the parallel simulation terminates. Note that after a process enters the termination state, it never become active again.

Fig. 2 illustrates the execution of the Chandy-Misra simulation for the event precedence graph in Fig. 1(b).

We assume that the message sending delays are 0 in the Chandy-Misra simulation. We further assume that the lookahead values for $e_2, e_4, e_6, e_8, e_9, e_{10}, e_{11}$, and e_{12} are 3. Note that the lookahead values for e_1, e_3, e_5, and e_7 are ∞ because they are executed by the source process p_1. After p_1 has executed an event, the newly scheduled events are sent to the destinations immediately (cf. e_2 and e_3). Process p_3 (p_2) has only one input channel. According to the input waiting rule, an arrival event is executed immediately if p_3 is idle (cf. e_4) or is executed after p_3 has executed the previous event (cf. e_6). According to the output waiting rule, e_{10} is not sent to p_4 until time 8; i.e., when p_3 starts executing e_8. Note that $e_6.ts + \varepsilon(e_6) = 6 + 3 < e_{10}.ts = 10 < e_8.ts + \varepsilon(e_8) = 11$.

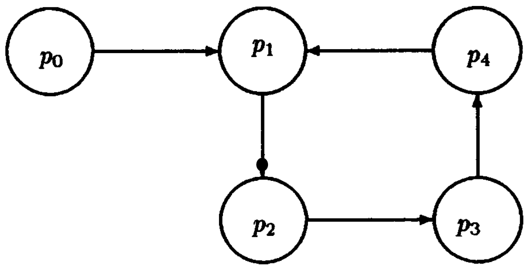


Fig. 3. Deadlock situation. Processes p_1, p_2, p_3, and p_4 are waiting for input messages without any progress.

Event e_9 arrives at p_4 at time 4. However, its execution is delayed until e_{10} arrives (due to the input waiting rule). The eos messages $z_{1,2}$ and $z_{1,3}$ are sent from p_1 to p_2 and p_3, respectively. These messages arrive at their destinations at time 4 (the time when the execution of e_7 is completed) because there is no message sending delay. After p_3 has processed $z_{1,3}$ (the execution time for an eos message is 0), a new eos message $z_{3,4}$ is sent to p_4.

A null message provides only timing information. For example, after p_i has executed an event e, it may send a null message e', where $e'.ts = e.ts + \varepsilon(e)$, to the output channel connected to p_j. When p_j receives e', it knows that it will never receive any message with timestamp less than $e'.ts$ from p_i. This information is used to reduce the overhead of the input waiting rule as well as to avoid deadlock [5]. In a Chandy-Misra simulation, deadlock may occur in a feedback loop. Consider the feedback network in Fig. 3. The initial events are generated by the source process p_0. At the beginning, p_0 sends an event message e to process p_1. According to the input waiting rule, p_1 cannot handle e before it receives a message from process p_4. Unfortunately, p_4 will not produce any output message before p_1 produces the first output message. Thus, processes p_1, p_2, p_3, and p_4 fall into a deadlock situation.

Two approaches have been proposed to resolve the deadlock situation. *Deadlock avoidance* [14] uses null messages to avoid deadlock. In Fig. 3, suppose that process p_i (for $1 \le i \le 4$) has a constant lookahead value ε_i and its local clock $ck_i = 0$ initially. At the beginning of execution, p_i sends a null message with timestamp $ck_i + \varepsilon$ to the output channel. When the destination p_j receives the null message, p_j is essentially promised by p_i that it will not send a message to p_j carrying a timestamp smaller than ε_i, and ck_j is incremented from 0 to ε_i. Then p_j sends a null message with timestamp $ck_j + \varepsilon_j = \varepsilon_i + \varepsilon_j$ to its output channel. After the null messages have circulated in the loop several times, p_1 eventually receives a null message with timestamp larger than $e.ts$. According to the input waiting rule, p_1 executes e and the deadlock is avoided.

In *deadlock recovery*, no null messages are sent. A separate mechanism is used to detect when the simulation is deadlocked, and another mechanism is used to break the deadlock. Deadlock detection mechanisms are described in [15], [16], [17]. In the deadlock recovery mechanism, all processes cooperate to find the events with the smallest timestamp in the system. These events can be safely executed, and the deadlock situation is thus recovered.

IV. A Parallelism Analyzer

We first consider the parallelism analyzer for deadlock avoidance simulation. Then we extend the algorithm for deadlock recovery simulation.

Consider an existing sequential simulation program. We investigate the performance of the corresponding Chandy-Misra parallel simulation without actually implementing the parallel program. The idea is to insert some instructions (to be described) into the sequential simulation program. The inserted code computes the elapsed time of the corresponding Chandy-Misra parallel simulation along with the execution of the sequential simulation.

We assume that the sequential program follows the process interaction model [18], in which the simulated system is modeled by a set of objects. These objects can be directly mapped to the logical processes in parallel simulation. We assume that every process is executed by a dedicated processor (we thus use the terms "process" and "processor" interchangeably). This restriction is relaxed later in this section. The modified sequential simulation that performs parallelism analysis for the corresponding Chandy-Misra simulation is referred to as the *parallelism analyzer.* The process-to-processor mapping affects the performance of the parallel simulation. To study the process assignment problem, one may execute the parallelism analyzer with different mappings.

In the parallelism analyzer the eos messages and the null messages are also included to simulate the Chandy-Misra protocol. The parallelism analyzer generates an eos event for every downstream process of a source process p_i after it processes the last event scheduled for p_i. When the parallelism analyzer processes an eos event for a server process p_j, it generates new eos events for the downstream processes of p_j. Suppose that an event e occurs at process p_i, and its occurrence results in the scheduling of another event e' for process p_j. When the parallelism analyzer processes e, it generally schedules a null message with timestamp $e.ts + \varepsilon(e)$ for every downstream process of p_i. (In some implementations, no null message is sent to p_j. In other implementations, null messages may be sent by demand [17]. Our parallelism analyzer can easily be tailored to study implementations with different null message sending policies.)

Several data structures are used in the parallelism analyzer.

- Every event e is associated with a real number $e.\alpha$ which represents the (real) time when e arrives at its destination (i.e., the process that executes e) in the corresponding Chandy-Misra simulation. Initially, $e.\alpha = 0$ (if e is an event pre-scheduled at the beginning of the simulation) or $e.\alpha = \infty$ (otherwise; in this case, the arrival time of e will be computed and assigned to $e.\alpha$ later).
- For a channel directed from p_i to p_j, a set $Q_{i,j}$ is used in the parallelism analyzer to hold all "floating" events sent from p_i to p_j. The time when a "floating" event is to be executed in the Chandy-Misra simulation has not yet been determined. When an event is processed in the sequential simulation, the parallelism analyzer inserts the event into the corresponding $Q_{i,j}$. This event is removed from $Q_{i,j}$ after its execution time in the Chandy-Misra simulation

```
     /* Initialization */
0a   for all $i, j$ do $t_i \leftarrow 0, Q_{i,j} \leftarrow \emptyset$, and $O_i \leftarrow \emptyset$;
0b   for all $e \in L$ do $e.\alpha \leftarrow 0$; /* $e$ is a pre-scheduled event */
     /* The main loop */
0     while $L \neq \emptyset$ do
1       execute the next event $e$ in $L$, and $L \leftarrow L - \{e\}$;
2       for all $e' \in e.E$ do $e'.\alpha \leftarrow \infty$ and $L \leftarrow L \cup \{e'\}$;
3       $Q_{j,i} \leftarrow Q_{j,i} \cup \{e\}$; /* $e$ is created at $p_j$ and is scheduled for $p_i$ */
4       while there exists $n$ such that $\theta_n \neq \phi$ do /* suppose that $\theta_n \in Q_{m,n}$ */
5         $t_n \leftarrow \max[t_n, \rho(\theta_n)]$;
6         for all $e' \in O_n$ such that $e'.ts \leq \theta_n.ts + \varepsilon(\theta_n)$ do $e'.\alpha \leftarrow t_n + \delta(e')$ and $O_n \leftarrow O_n - \{e'\}$;
7         $t_n \leftarrow t_n + \eta(\theta_n)$;
8         if $\varepsilon(\theta_n) = \infty$ then for all $e' \in \theta_n.E$ do $e'.\alpha \leftarrow t_n + \delta(e')$;
9         else for all $e' \in \theta_n.E$ do $O_n \leftarrow O_n \cup \{e'\}$;
10        $Q_{m,n} \leftarrow Q_{m,n} - \{\theta_n\}$;
        end while
      end while
11    $T = \max_{1 \leq i \leq N} t_i$;
```

Fig. 4. The parallelism analyzer (assumes every process is executed by a dedicated processor).

is determined. Note that $Q_{i,i}$ exists if and only if p_i is a source process.

- For every server process p_i, the parallelism analyzer maintains a set O_i which holds the events generated at p_i for which the departure time of the events (from p_i) in the Chandy-Misra simulation has not yet been decided. An event e is removed from O_i after the parallelism analyzer has determined its departure time (and thus its arrival time; in other words, $e.\alpha$ is assigned a finite value) It is clear that a sink process does not need this data structure. According to the FIFO event generation assumption, $O_i = \emptyset$ for a source process p_i.
- For every process p_i, the parallelism analyzer maintains a variable t_i. When an event (occurring in p_i) is processed by the parallelism analyzer, t_i represents the elapsed time of the Chandy-Misra simulation after the event is executed. At the end of the modified sequential simulation, t_i represents the time after p_i finishes the last event in the Chandy-Misra simulation, and the execution time of the Chandy-Misra simulation is $T = \max_{1 \leq i \leq N} t_i$, where N is the process number.

The parallelism analyzer is described formally in Fig. 4.

In this figure, L represents the event list in the sequential simulation. The initialization phase resets the data structures $t_i, Q_{i,j}$, and O_i (Line 0a). At the beginning of the sequential simulation, several pre-scheduled events are in the event list L. These events are also pre-scheduled in the corresponding Chandy-Misra simulation, and their arrival times in the Chandy-Misra simulation are 0 (Lin 0b). The main loop (Lines 0-10) performs the sequential simulation (Lines 1 and 2) and determines the progress of a process in the corresponding Chandy-Misra simulation by modifying $Q_{i,j}$, t_i, and O_i (Lines 3-10). It is clear that the sequential simulation terminates when $L = \emptyset$ at Line 0. Line 1 removes the event e with the smallest timestamp from the event list L and executes the event. At Line 2, $e.E$ represents the set of events scheduled due to the execution of e. Every event $e' \in e.E$ is inserted in L in the timestamp order. The time when e' arrives at its destination in the Chandy-Misra simulation is not determined, and $e'.\alpha$ is assigned the value ∞. At Line 3, e is inserted in $Q_{j,i}$. In other words, e is sent from p_j to p_i in the Chandy-Misra simulation (however, its arrival time may not be determined at this moment). In the loop Lines 4-10, the parallelism analyzer tests whether the time when an event $e \in Q_{m,n}$ (for some m, n) available for execution in the Chandy-Misra simulation (i.e., the time when e satisfies the input waiting rule) is known. If so, the time when the execution of e is completed in the Chandy-Misra simulation is computed and assigned to t_n. The parallelism analyzer also tests whether any event $e' \in O_n$ satisfies the output waiting rule . If so, the time when e' arrives at its destination is computed and assigned to $e'.\alpha$.

At Line 4, the event θ_n is defined as follows: Let I_n be the set of processes that may schedule events (i.e., send messages) to p_n. Suppose that the following two conditions are satisfied in the parallelism analyzer:

- For every $p_m \in I_n$, $Q_{m,n} \neq \emptyset$.
- For every $p_m \in I_n$, let $e_{m,n}$ be the event with the smallest timestamp in $Q_{m,n}$. Then $e_{m,n}.\alpha < \infty$ (i.e., the arrival time of $e_{m,n}$ in the Chandy-Misra simulation is determined).

Then θ_n is the event with the smallest timestamp among $e_{m,n}$ for all $m, p_m \in I_n$. The time when θ_n is available for execution in the Chandy-Misra simulation is defined as

$$\rho(\theta_n) = \max_{\forall p_m \in I_n} e_{m,n}.\alpha.$$

In Appendix A (Lemma 1), we prove that at Line 4 of the parallelism analyzer, all events sent from p_m to p_n with timestamps less than $e_{m,n}.ts$ are processed before t_n in the Chandy-Misra simulation. Lemma 1 together with the definition of θ_n implies that after t_n, θ_n is the next event to be executed at p_n in the Chandy-Misra simulation. Lemma 2 (a) in Appendix A ensures that if $e.\alpha < \infty$, then this finite value is the time when the event arrives at its destination in the Chandy-Misra simulation. Thus, by the input waiting rule, $\rho(\theta_n)$ is the time when θ_n can be executed. This event is executed at time $\max[t_n, \rho(\theta_n)]$. Thus, after Line 5, t_n represents the time when θ_n's execution starts. At Line 6, O_n represents the set of events scheduled by p_n that have not been sent to the destinations before time t_n, i.e., before θ_n is selected for execution (cf. Lemma 6 (c) in Appendix 6). By the output waiting rule, these events e' are sent at time t_n if

$$e'.ts \leq \theta_n.ts + \varepsilon(\theta_n).$$

At Line 6, for every $e' \in O_n$ that satisfies the above inequality, e' arrives at its destination at time $e'.\alpha = t_n + \delta(e')$, where $\delta(e')$ is the message sending delay for e'. Also, after t_n, e' is removed from O_n. At Line 7, the execution of θ_n is completed at time $t_n + \eta(\theta_n)$, where $\eta(\theta_n)$ is the execution time of θ_n. Lines 8 and 9 handle the events scheduled due to the execution of θ_n. If $\varepsilon(\theta_n) = \infty$ then either p_n is a source process or θ_n is an *eos* event. In either case, all events e' scheduled due to the execution of θ_n are sent to the destinations at θ_n's completion time. Thus, $e'.\alpha = t_n + \delta(e')$ at Line 8. If $\varepsilon(e) < \infty$, then by the output waiting rule, all events $e' \in e.E$ are stored in O_n at Line 9. Since θ_n's execution is completed at t_n, θ_n is removed from $Q_{m,n}$ at Line 10. After the modified sequential

```
    /* Initialization */
0a  for all $i,j$ do $Q_{i,j} \leftarrow \emptyset$, and $O_i \leftarrow \emptyset$;
0b  for all $e \in L$ do $e.\alpha \leftarrow 0$; /* $e$ is a pre-scheduled event */
0c  for $k = 1$ to $K$ do $t_k = 0$;
    /* The main loop */
0   while $L \neq \emptyset$ do
1     execute the next event $e$ in $L$, and $L \leftarrow L - \{e\}$;
2     for all $e' \in e.E$ do $e'.\alpha \leftarrow \infty$ and $L \leftarrow L \cup \{e'\}$;
3     $Q_{j,i} \leftarrow Q_{j,i} \cup \{e\}$; /* $e$ is created at $p_j$ and is scheduled for $p_i$ */
4     while there exists $n$ such that $\theta_n \neq \phi$ do /* suppose that $\theta_n \in Q_{m,n}$, and $p_n \in P_k$ */
5       $t_k \leftarrow \max[t_k, \rho(\theta_n)]$;
6       for all $e' \in O_n$ such that $e'.ts \leq \theta_n.ts + \varepsilon(\theta_n)$ do $e'.\alpha \leftarrow t_k + \delta(e')$ and $O_n \leftarrow O_n - \{e'\}$;
7       $t_k \leftarrow t_k + \eta(\theta_n)$;
8       if $\varepsilon(\theta_n) = \infty$ then for all $e' \in \theta_n.E$ do $e'.\alpha \leftarrow t_k + \delta(e')$;
9       else for all $e' \in \theta_n.E$ do $O_n \leftarrow O_n \cup \{e'\}$;
10      $Q_{m,n} \leftarrow Q_{m,n} - \{\theta_n\}$;
      end while
    end while
11  $T = \max_{1 \leq k \leq K} t_k$;
```

Fig. 5. The parallelism analyzer (more than one process may be mapped to a processor).

simulation has been completed (i.e., $L = \emptyset$), t_i represents the completion time of the last event executed at process p_i, and the execution time T of the Chandy-Misra simulation is computed as

$$T = \max_{1 \leq i \leq N} t_i$$

at Line 11 (cf. Theorem 1 in Appendix A). The execution of the parallelism analyzer in Fig. 4 for the event precedence graph in Fig. 1(b) is described in Appendix B.

The parallelism analyzer in Fig. 4 can easily be generalized to the case when more than one process is executed by a processor. Suppose there are K processors available for the parallel simulation, where $K \leq N$. Let P_k be the set of processes mapped to processor k. The modified parallelism analyzer is shown in Fig. 5. In this algorithm, variable t_i represents the progress of the processor i (where $1 \leq i \leq K$) and in Lines 5-8, t_n (in Fig. 4) is replaced by t_k, where $p_n \in P_k$. Fig. 5 is exactly the same as Fig. 4 except that the elapsed times considered in the Chandy-Misra simulation are for processors.

Our algorithm can also be extended to study the Chandy-Misra deadlock recovery protocol. To compute the execution time of a deadlock recovery simulation, the algorithm in Fig. 5 is modified as follows:

- No null messages are created in the parallelism analyzer.
- In an iteration of the loop in Lines 0-10, if $\theta_n = \emptyset$ (for all n) the first time Line 4 is executed, then a deadlock occurs. Basically, the condition $\theta_n = 0$ for all n implies that no process satisfies the input waiting rule in the Chandy-Misra simulation. The condition $L \neq \emptyset$ implies that the Chandy-Misra simulation has not yet been completed (cf., Lemma 3 in Appendix A). When the above two conditions are satisfied, a deadlock occurs. Let e be the event with the smallest timestamp in $\cup_{\forall i,j} Q_{i,j}$. Then for $1 \leq i \leq N, t_i = \Delta + \max_{1 \leq j \leq N} t_j$, where Δ is the elapsed time to detect and break the deadlock (i.e., to find the event e). If e is sent from process i to process j, and process j is mapped to processor k, then $Q_{i,j} \leftarrow Q_{i,j} - \{e\}$, and $t_k = t_k + \eta(e)$.

From the above discussion, the number of deadlocks occurring in the simulation can also be computed.

V. Summary

This paper proposed a technique to evaluate the time complexity of the Chandy-Misra parallel simulation protocol for an existing sequential simulation program. The idea is to insert parallelism analysis code into the sequential simulation program. When the modified sequential program is executed, the time complexity of the corresponding Chandy-Misra parallel simulation is also computed.

We described the parallelism analysis algorithm, and proved that the algorithm is correct. The algorithm assumes that the sequential simulation follows the process interaction model. This assumption does not restrict the applicability of our technique, because most modern simulation programs are implemented by object-oriented languages or environments that follow the process interaction model.

Our technique has been proven to be useful in several large-scale industrial applications. For example, a Signaling System 7 (SS7) network simulation was implemented by a sequential simulation technique that is adequate for small-scale networks. Because the number of new customers and services grows rapidly in a telephone network, it is important to study the performance of a scaled-up SS7 network. Since the SS7 network is very complex, it is difficult to execute a sequential simulation within a reasonable elapsed time when the network size is increased. A natural alternative for speeding up the simulation process is to use parallel simulation techniques. However, re-implementation of an SS7 simulation on a parallel platform is very expensive. It is necessary to investigate the performance of parallel simulation to determine whether it is cost effective. With our technique, the performance of Chandy-Misra parallel simulation can be determined without actually implementing the parallel program.

Acknowledgment

The authors would like to thank D. DeGroot and the three reviewers for their comments.

Appendix

A. Proof of the Parallelism Analysis Algorithm

The correctness of the parallelism analyzer is proved by means of three lemmas and one theorem.

For the ith event $e_{m,n}(i)$ sent from process p_m to p_n, the parallelism analyzer computes a value $t_{m,n}(i)$. Lemma 1 shows that $t_{m,n}(i) < t_{m,n}(j)$ if $e_{m,n}(i).ts < e_{m,n}(j).ts$. Lemma 1 is used in Lemma 2. Lemma 2 consists of three parts. Part (a) proves that the value $e.\alpha$ computed in the parallelism analyzer is the time when event e arrives at its destination. Part (b) shows that $t_{m,n}(i)$ in Lemma 1 is the completion time of $e_{m,n}(i-1)$ in the Chandy-Misra simulation. Part (c) is a hypothesis used to prove parts (a) and (b) by induction. Lemma 2 proves that the analyzer correctly computes the completion time of every event in the Chandy-Misra simulation. Lemma 3 shows that the parallelism analyzer appropriately terminates. Drawing on Lemmas 2 and 3, Theorem 1 proves the correctness of the parallelism analyzer.

Lemma 1: Let $e_{m,n}(i)$ be the ith message sent from p_m to p_n. Suppose that $\theta_n = e_{m,n}(i)$ at Line 4 in Fig. 4. Then the parallelism analyzer assumes that all events sent from p_m to p_n with timestamps less than $e_{m,n}(i).ts$ are processed before $t_{m,n}(i)$ in the Chandy-Misra simulation, where $t_{m,n}(i)$ is the value of t_n at Line 4.

Proof: We prove the following hypothesis by induction on i: for all $i' < i$, $e_{m,n}(i')$ are processed before $t_{m,n}(i)$ in the Chandy-Misra simulation.

Basis ($i = 1$): Trivial.

Inductive step ($i > 1$): Assume that the hypothesis holds for $e_{m,n}(i-1)$. We show that the hypothesis also holds for $e_{m,n}(i)$.

From the induction hypothesis, the parallelism analyzer assumes that $e_{m,n}(i')$ is processed before $t_{m,n}(i-1)$ in the Chandy-Misra simulation, for all $i' < i-1$. Suppose that $\theta_n = e_{m,n}(i)$ at Line 4 in Fig. 4. Then this implies that $e_{m,n}(i-1)$ has been deleted from $Q_{m,n}$ (cf. definition of θ_n). Since an event can be removed from $Q_{m,n}$ only at Line 10 in Fig. 4, the parallelism analyzer assumes that $e_{m,n}(i-1)$ is processed before $t_{m,n}(i)$ in the Chandy-Misra simulation, and the hypothesis holds. ■

Lemma 2: Consider the ith iteration when Loop Lines 4-10 in Fig. 4 are executed. Let $t_n^-(i)$ be the value of t_n at Line 4; i.e., the value before Line 5 is executed but after Line 4 has been executed (note that $t_n^-(i)$ in this lemma is $t_{m,n}(i)$ in Lemma 1). Let $t_n(i)$ be the value of t_n after Line 5 but before Line 7 is executed. Let $t_n^+(i)$ be the value of t_n after Line 7 is executed. Let $\theta_n(i) \neq \phi$ be the event selected at Line 4 and $O_n(i)$ be the value of O_n at Line 10.

(a) If $e.\alpha < \infty$, then $e.\alpha$ is the time when e arrives at its destination in the Chandy-Misra simulation.

(b) After Line 7 is executed, $t_n = t_n^+(i)$ is the time when $\theta_n(i)$'s execution is completed in the Chandy-Misra simulation.

(c) At Line 10, $O_n(i)$ represents the set of events that have not been sent to their destinations at time $t_n^+(i)$ in the Chandy-Misra simulation.

Proof: We prove by induction on i that hypotheses (a), (b), and (c) hold.

Basis ($i = 1$): Since the value $e.\alpha$ for each event e can be modified only in Line 6 or Line 8 in Fig. 4, we have $\theta_n(1).\alpha = 0$ and $\rho(\theta_n(1)) = 0$ (cf. definition of θ_n and ρ). That is, $\theta_n(1)$ is a pre-scheduled event for a source process p_n, and $\theta_n(1).\alpha = 0$ is the time when $\theta_n(1)$ arrives at p_n in the Chandy-Misra simulation. Therefore, after Line 5 is executed, $t_n = t_n(i)$ is the time when the execution of $\theta_n(1)$ starts. After Line 7 is executed, t_n is the time when $\theta_n(1)$'s execution is completed (cf. definition of η). Hypothesis (b) holds. It is clear that $O_n(1) = \emptyset$ at $t_n^-(i)$ in the Chandy-Misra simulation (cf. Line 0a). Since $\varepsilon(\theta_n(1)) = \infty$ (because p_n is a source process), every event e in $\theta_n(1).E$ can be sent from p_n as soon as the execution of $\theta_n(1)$ is completed, and the value $e.\alpha$ assigned at Line 8 is the time when e arrives at its destination in the Chandy-Misra simulation (cf. definition of δ). Thus hypothesis (a) holds. Also, at Line 10, $O_n(i) = \emptyset$. Since O_n is always empty for a source process p_n, hypothesis (c) holds.

Inductive step ($i > 1$): Assume that hypotheses (a), (b), and (c) hold for the first $i-1$ iterations of the loop Lines 4-10. We show that these hypotheses also hold for the ith iteration. From Lemma 1 and the definition of the input waiting rule, $\theta_n(i)$ is the next event to be executed at p_n after time $t_n^-(i)$. At that time, we have $e_{m,n}.\alpha < \infty$ (cf. definition of θ_n), for all p_m in I_n. According to hypothesis (a), $e_{m,n}.\alpha$ is the arrival time of $e_{m,n}$ in the Chandy-Misra simulation, and from hypothesis (b) $t_n^-(i)$ is the time when the execution of the last event processed at p_n is completed. Therefore, $\theta_n(i)$ is available for execution at time $\rho(\theta_n(i))$ in the Chandy-Misra simulation, and the value of t_n after Line 5 is the time when $\theta_n(i)$'s execution starts. Since it takes time $\eta(\theta_n(i))$ to execute event $\theta_n(i)$, $t_n^+(i)$ is the time when the execution of $\theta_n(i)$ is completed in the Chandy-Misra simulation. Hypothesis (b) holds. From hypothesis (c), $O_n(i-1)$ is the set of events that have not been sent to their destinations at time $t_n^+(i-1) = t_n^-(i)$ in the Chandy-Misra simulation. By the output waiting rule, we have

Fact 1. An event $e' \in O_n(i-1)$ is sent from p_n at time $t_n(i)$ (i.e., before event $\theta_n(i)$ is executed) if and only if $e'.ts \leq \theta_n(i).ts + \varepsilon(\theta_n(i))$.

Fact 2. $O_n(i) = \{e' \in O_n(i-1) \cup \theta_n(i).E | e'.ts > \theta_n(i).ts + \varepsilon(\theta_n(i))\}$

From Fact 2 and the execution of Lines 6-9, $O_n(i)$ is the set of events that have not been sent to their destinations at time $t_n^+(i)$, and hypothesis (c) holds.

From Fact 1 and the execution of Line 6 and Line 8 (if p_n is a source process), the value $e'.\alpha$ assigned either at Line 6 or at Line 8 is the time when e' arrives at its destination. Thus hypothesis (a) holds. ■

Lemma 3: If $L = \emptyset$ at Line 0 in Fig. 4, then $Q_{m,n}$ either is an empty set or contains only the *eos* message $z_{m,n}$. In other words, all non-*eos* events in the Chandy-Misra simulation have been processed when $L = \emptyset$ at Line 0.

Proof: We prove the lemma by contradiction. Suppose that there exists $Q_{m,n}$ which contains non-*eos* events when $L = \emptyset$ at Line 0. This implies that θ_n does not exist after the ith time when Loop Lines 4-10 are executed, for some integer i. Therefore, one of the following two cases must be true according to the definition of θ_n:

Case 1: There exists $p_{m_1} \in I_n$, $m_1 \neq m$, such that $Q_{m_1,n} = \emptyset$. Note that for all k, l, the *eos* message $z_{k,l}$ should have been inserted into $Q_{k,l}$ before $L = \emptyset$ at Line 0. (Otherwise, there is an un-executed event in the sequential simulation and L cannot be empty.) Since $Q_{m_1,n} = \emptyset$, event $z_{m_1,n}$ must have been processed at p_n and then been deleted from $Q_{m_1,n}$. Note that $\theta_n = z_{m_1,n}$ if and only if $e_{m_2,n}$ is an *eos* message for all $m_2 \in I_n$ (from the definition of θ_n and the fact that $e.ts < \infty$ for every noneos event). In other words, there are no noneos events in $Q_{m,n}$, a contradiction.

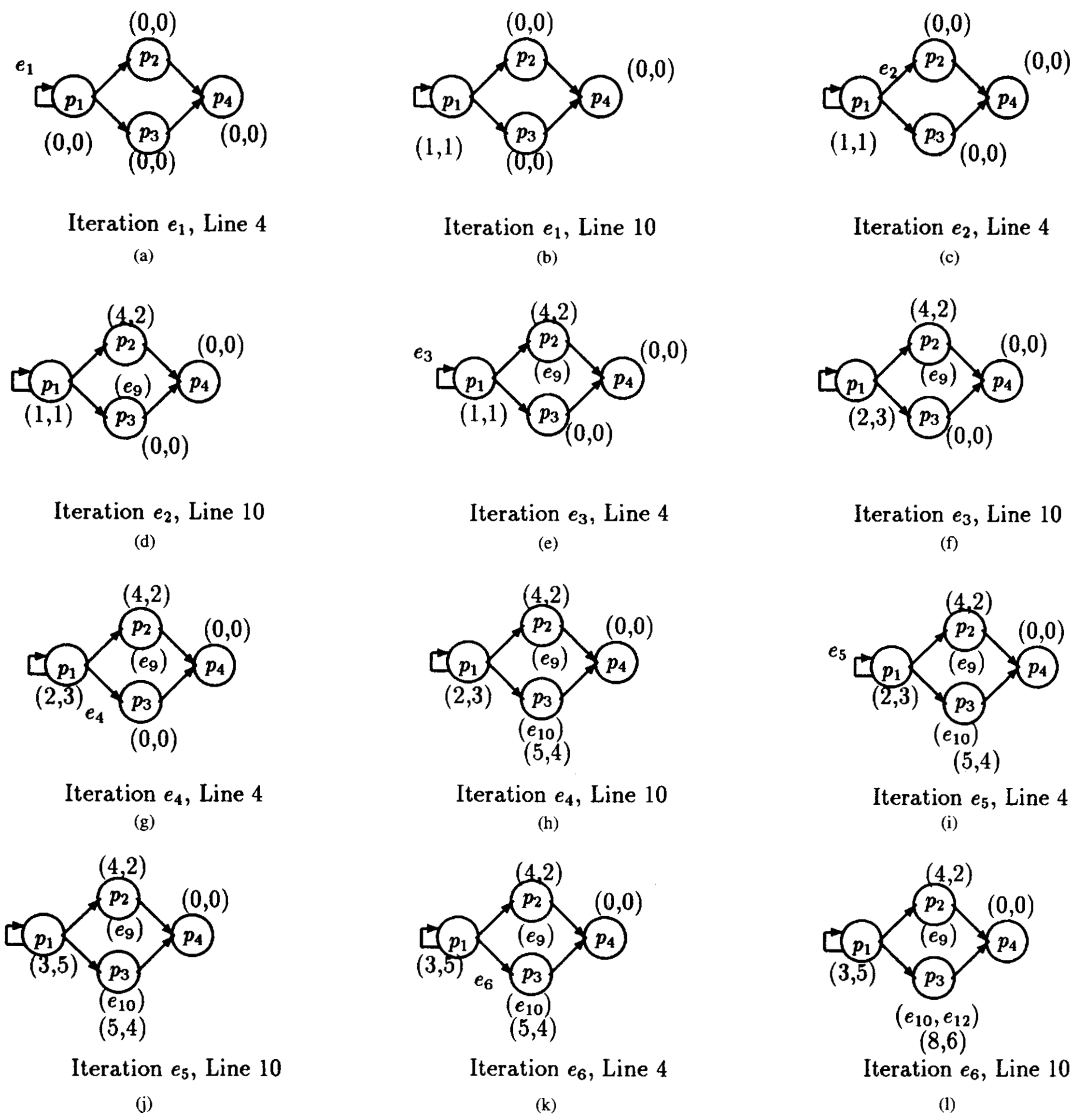


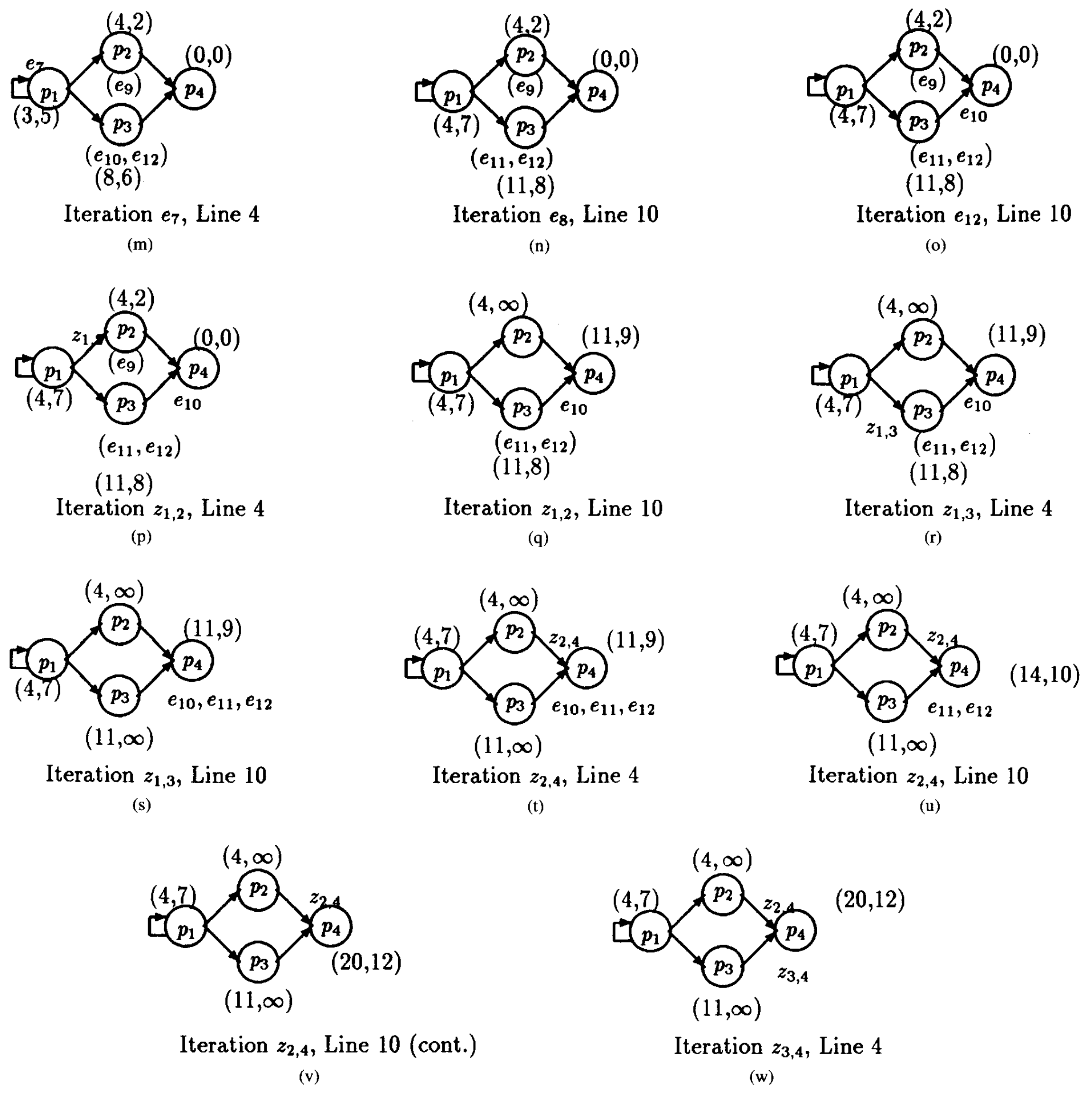
Fig. 6. The Chandy-Misra protocol simulated by the parallelism analyzer.

Case 2: There exists $p_{m_1} \in I_n$ such that $e_{m_1,n}.\alpha = \infty$. Let g_e be an event such that e is created by the execution of g_e. There are two possibilities:

Case 2a: p_{m_1} is a source process. Since $\varepsilon(g_{e_{m_1,n}}) = \infty$, $e_{m_1,n}.\alpha$ is assigned a finite value at Line 8 when $\theta_{m_1} = g_{e_{m_1},n}$, a contradiction.

Case 2b: p_{m_1} is not a source process. This implies that $g_{e_{m_1,n}}$ does not satisfy the output waiting rule at p_{m_1}. In other words, there exists $m_2 \in I_{m_1}$ such that Q_{m_2,m_1} contains non-*eos* events when $L = \emptyset$ at Line 0. Thus, we should backtrack the topology hierarchy of the simulation processes for one level and repeat the argument stated above. Since the number of processes in the network is limited, we will trace back to either a source process or p_{m_1} itself (if there is a feedback loop). The former is the same as **Case 2a**. For the latter, there is a cycle of blocked nonsource processes $p_n = p_{m_0} \leftarrow p_{m_1} \leftarrow p_{m_2} \leftarrow \ldots \leftarrow p_{m_k}$ where p_{m_l} waits to receive an event from $p_{m_{(l+1)(mod k+1)}}$, $1 \le l \le k$. In other words, a deadlock occurs. This contradicts the fact that the Chandy-Misra protocol (with deadlock avoidance) is deadlock-free. ■

Theorem 1: If $L = \emptyset$ at Line 0 in Fig. 4, then t_i represents the completion time of the last event processed at p_i in the Chandy-Misra simulation.



Iteration e_7, Line 4
(m)
Iteration e_8, Line 10
(n)
Iteration e_{12}, Line 10
(o)
Iteration $z_{1,2}$, Line 4
(p)
Iteration $z_{1,2}$, Line 10
(q)
Iteration $z_{1,3}$, Line 4
(r)
Iteration $z_{1,3}$, Line 10
(s)
Iteration $z_{2,4}$, Line 4
(t)
Iteration $z_{2,4}$, Line 10
(u)
Iteration $z_{2,4}$, Line 10 (cont.)
(v)
Iteration $z_{3,4}$, Line 4
(w)

Fig. 7. The Chandy-Misra protocol simulated by the parallelism analyzer.

Proof: Since every non-*eos* event in the Chandy-Misra simulation has been processed when $L = \emptyset$ at Line 0 (cf. Lemma 3) and every event is processed in nondecreasing timestamp order, t_i is the completion time of the last event processed at p_i (cf. Lemma 2) in the Chandy-Misra simulation. ■

B. An Example of Parallelism Analysis

This appendix illustrates the execution of the parallelism analyzer in Fig. 4 for the event precedence graph in Fig. 1(b). We note that $\eta(e_1) = \eta(e_3) = \eta(e_5) = \eta(e_7) = 1$, and $\eta(e_2) = \eta(e_4) = \eta(e_6) = \eta(e_8) = \eta(e_9) = \eta(e_{10}) = \eta(e_{11}) = \eta(e_{12}) = 3$, and the message sending delay times for all events are 0. and the message sending delay for all events are 0. Initially, $L = \{e_1\}$, $Q_{1,1} = Q_{1,2} = Q_{1,3} = Q_{2,4} = Q_{3,4} = \emptyset$, $O_2 = O_3 = \emptyset$, $e_1.\alpha = 0$ and $t_1 = t_2 = t_3 = t_4 = 0$.

Event e_1 is executed in the first iteration. Since e_1 is a prescheduled event, $e_1.\alpha = 0$. At the end of Line 3, $Q_{1,1} = \{e_1\}$ and $L = \{e_2, e_3\}$. At this point, we know the arrival time of e_1 in the Chandy-Misra simulation, as illustrated in Fig. 6(a). In this figure, a pair (t_i, ts_i) is associated with every process p_i, where t_i represents the (real) time of p_i in the Chandy-Misra simulation and ts_i represents the local clock of p_i (i.e., the timestamp of the event being executed by p_i) at time t_i. In the figures, a link directed from p_i to p_j is labeled by an event e if and only if $e.\alpha < \infty$ and $e \in Q_{i,j}$.

TABLE I
EXECUTION OF e_2, e_3, e_4, AND e_5

Iteration	e_2	e_2	e_3	e_3	e_4	e_4	e_5	e_5
	Line 4	Line 10	Line 4	Line 10	Line 4	Line 10	Line 4	Line 10
L	$\{e_3, e_9\}$	$\{e_3, e_9\}$	$\{e_4, e_5, e_9\}$	$\{e_4, e_5, e_9\}$	$\{e_5, e_9, e_{10}\}$	$\{e_5, e_9, e_{10}\}$	$\{e_6, e_7, e_9, e_{10}\}$	$\{e_6, e_7, e_9, e_{10}\}$
$e_2.\alpha$	1							
$e_3.\alpha$	1	1	1					
$e_4.\alpha$			∞	2				
$e_5.\alpha$			∞	2	2	2		
$e_6.\alpha$							∞	3
$e_7.\alpha$							∞	3
$e_9.\alpha$	∞	∞	∞	∞	∞	∞	∞	∞
$e_{10}.\alpha$					∞	∞	∞	∞
$Q_{1,1}$	$\emptyset$	$\emptyset$	$\{e_9\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_5\}$	$\emptyset$
$Q_{1,2}$	$\{e_2\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{1,3}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_4\}$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{2,4}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{3,4}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
O_2	$\emptyset$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$
O_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_{10}\}$	$\{e_{10}\}$	$\{e_{10}\}$
(t_1, ts_1)	(1,1)	(1,1)	(1,1)	(2,3)	(2,3)	(2,3)	(2,3)	(3,5)
(t_2, ts_2)	(0,0)	(4,2)	(4,2)	(4,2)	(4,2)	(4,2)	(4,2)	(4,2)
(t_3, ts_3)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(5,4)	(5,4)	(5,4)
(t_4, ts_4)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
θ_1	ϕ		e_3		ϕ		e_5	
θ_2	e_2		ϕ		ϕ		ϕ	
θ_3	ϕ		ϕ		e_4		ϕ	
θ_4	ϕ		ϕ		ϕ		ϕ	
Figure	6(c)	6(d)	6(e)	6(f)	6(g)	6(h)	6(i)	6(j)

TABLE II
THE EXECUTIONS OF e_6, e_7, e_8, AND e_{12}

Iteration	e_6	e_6	e_7	e_7	e_8	e_{12}
	Line 4	Line 10	Line 4	Line 10	Line 10	Line 10
L	$\{e_7, e_9, e_{10}, e_{12}\}$	$\{e_7, e_9\} e_{10}, e_{12}\}$	$\{e_8, e_9, e_{10}, e_{12}, z_{1,2}, z_{1,3}\}$	$\{e_8, e_9, e_{10}, e_{12}, z_{1,2}, z_{1,3}\}$	$\{e_9, e_{10}, e_{11}, e_{12}, z_{1,2}, z_{1,3}\}$	$\{z_{1,2}, z_{1,3}\}$
$e_7.\alpha$	3	3				
$e_8.\alpha$			∞	4		
$e_9.\alpha$	∞	∞	∞	∞	∞	∞
$e_{10}.\alpha$	∞	∞	∞	∞	8	8
$e_{11}.\alpha$					∞	∞
$e_{12}.\alpha$	∞	∞	∞	∞	∞	∞
$z_{1,2}.\alpha$			∞	4	4	4
$z_{1,3}.\alpha$			∞	4	4	4
$Q_{1,1}$	$\emptyset$	$\emptyset$	$\{e_7\}$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{1,2}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{1,3}$	$\{e_6\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{2,4}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_9\}$
$Q_{3,4}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_{10}, e_{11}, e_{12}\}$
O_2	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$	$\{e_9\}$
O_3	$\{e_{10}\}$	$\{e_{10}, e_{12}\}$	$\{e_{10}, e_{12}\}$	$\{e_{10}, e_{12}\}$	$\{e_{11}, e_{12}\}$	$\{e_{11}, e_{12}\}$
(t_1, ts_1)	(3,5)	(3,5)	(3,5)	(4,7)	(4,7)	(4,7)
(t_2, ts_2)	(4,2)	(4,2)	(4,2)	(4,2)	(4,2)	(4,2)
(t_3, ts_3)	(5,4)	(8,6)	(8,6)	(8,6)	(11,8)	(11,8)
(t_4, ts_4)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
θ_1	ϕ		e_7			
θ_2	ϕ		ϕ			
θ_3	e_6		ϕ			
θ_4	ϕ		ϕ			
Figure	6(k)	6(l)	7(m)		7(n)	7(o)

This implies that the parallelism analyzer "simulates" the arrival of e at p_j in the Chandy-Misra simulation. At Line 4, $\theta_1 = e_1 \neq \phi$. Since $\varepsilon(e_1) = \infty$, $\eta(e_1) = 1$, $e_1.E = \{e_2, e_3\}$, and $\delta(e_2) = \delta(e_3) = 0$, at the end of Line 10, we have $t_1 = 1$, $ts_1 = 1$, $e_2.\alpha = e_3.\alpha = 1$, and $Q_{1,1} = \emptyset$ (cf. Fig. 6(b)).

Table I illustrates the execution of events e_2, e_3, e_4, and e_5. The column for Line 4 of iteration e_2 illustrates the related variable values after Line 4 is executed at e_2's iteration. The last item "Figure" points to the figure illustrating the execution. Note that in Fig. 6(d), the notation (e_9) under p_2 means that e_9 is an event scheduled by p_2, which has not been sent to the destination in the Chandy-Misra simulation.

Table II illustrates the execution of e_6, e_7, e_8, and e_{12}. The presentations for iterations e_9, e_{10}, and e_{11} are similar and hence are omitted.

Table III illustrates the executions of $z_{1,2}, z_{1,3}, z_{2,4}$, and $z_{3,4}$.

TABLE III
THE EXECUTION OF $z_{1,2}, z_{1,3}, z_{2,4}$, AND $z_{3,4}$

Iteration	$z_{1,2}$	$z_{1,2}$	$z_{1,3}$	$z_{1,3}$	$z_{2,4}$	$z_{2,4}$	$z_{3,4}$	$z_{3,4}$
	Line 4	Line 10	Line 4	Line 10	Line 4	Line 10	Line 4	Line 10
L	$\{z_{1,3}, z_{2,4}\}$	$\{z_{1,3}, z_{2,4}\}$	$\{z_{2,4}, z_{3,4}\}$	$\{z_{2,4}, z_{3,4}\}$	$\{z_{3,4}\}$	$\{z_{3,4}\}$	$\emptyset$	$\emptyset$
$e_9.\alpha$	∞							
$e_{10}.\alpha$	8	8	8	8	8			
$e_{11}.\alpha$	∞	∞	∞	11	11			
$e_{12}.\alpha$	∞	∞	∞	11	11			
$z_{1,3}.\alpha$	4	4						
$z_{2,4}.\alpha$	∞	4	4	4	4	4	4	
$z_{3,4}.\alpha$			∞	11	11	11	11	11
$Q_{1,1}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{1,2}$	$\{z_{1,2}\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{1,3}$	$\emptyset$	$\emptyset$	$\{z_{1,3}\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$Q_{2,4}$	$\{e_9\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{z_{2,4}\}$	$\{z_{2,4}\}$	$\{z_{2,4}\}$	$\emptyset$
$Q_{3,4}$	$\{e_{10}, e_{11}, e_{12}\}$	$\{e_{10}, e_{11}, e_{12}\}$	$\{e_{10}, e_{11}, e_{12}\}$	$\{e_{10}, e_{11}, e_{12}\}$	$\{e_{10}, e_{11}, e_{12}\}$	$\emptyset$	$\{z_{3,4}\}$	$\{z_{3,4}\}$
O_2	$\{e_9\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
O_3	$\{e_{11}, e_{12}\}$	$\{e_{11}, e_{12}\}$	$\{e_{11}, e_{12}\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
(t_1, ts_1)	(4,7)	(4,7)	(4,7)	(4,7)	(4,7)	(4,7)	(4,7)	(4,7)
(t_2, ts_2)	(4,2)	$(4,\infty)$	$(4,\infty)$	$(4,\infty)$	$(4,\infty)$	$(4,\infty)$	$(4,\infty)$	$(4,\infty)$
(t_3, ts_3)	(11,8)	(11,8)	(11,8)	$(11,\infty)$	$(11,\infty)$	$(11,\infty)$	$(11,\infty)$	$(11,\infty)$
(t_4, ts_4)	(0,0)	(11,9)	(11,9)	(11,9)	(11,9)	(20,12)	(20,12)	$(20,\infty)$
θ_1	ϕ		ϕ		ϕ		ϕ	
θ_2	$z_{1,2}$		ϕ		ϕ		ϕ	
θ_3	ϕ		$z_{1,3}$		ϕ		ϕ	
θ_4	ϕ		ϕ		e_{10}		$z_{2,4}$	
Figure	7(p)	7(q)	7(r)	7(s)	7(t)	7(u),7(v)	7(w)	

In iteration $z_{1,2}$, the while loop in Lines 4-10 is executed twice. In the first iteration, $\theta_2 = z_{1,2}$, and in the second iteration, $\theta_4 = e_9$. Line 10 of iteration $z_{1,2}$ in Table III shows the variable values at the end of the second iteration.

In iteration $z_{2,4}$, the while loop in Lines 4-10 is executed three times. The first iteration handles e_{10}, and at the end of Line 10, $(t_4, ts_4) = (14, 10)$, as shown in Fig. 7 (u). Line 10 of iteration $z_{2,4}$ in Table III shows the results at the end of the third iteration.

At the end of Line 10 at Iteration $z_{3,4}$, no θ_k exists, and the while loop in Lines 4-10 is skipped. Since $L = \emptyset$, the while loop in Lines 0-10 is skipped and the elapsed time for the Chandy-Misra simulation is executed at Line 11: $T = 20$, and the program exits.

C. Notation

This section summarizes the notation used in this paper.

$\delta(e)$	The message transmission delay of event e in the Chandy-Misra simulation.
$\varepsilon(e)$	The *lookahead* value of e.
e	An event.
$e_{j,i}$	The event with the smallest timestamp in $Q_{j,i}$.
$e_{m,n}(i)$	The ith message sent from p_m to p_n.
$e.\alpha$	The time when e arrives at its destination process in the Chandy-Misra simulation (if the value is finite).
$e.ts$	The timestamp of event e.
$e.E$	The set of events scheduled due to the execution of e.
$\eta(e)$	The execution time of event e.
θ_n	The event with the smallest timestamp among $e_{m,n}$, for all $p_m \in I_n$ such that the following two conditions are satisfied: a. For every $p_m \in I_n$, $Q_{m,n} \neq \emptyset$. b. For every $p_m \in I_n$, $e_{m,n}.\alpha < \infty$ (i.e., the arrival time of $e_{m,n}$ in the Chandy-Misra simulation is determined).
$\theta_n(i)$	The value of θ_n at the ith time when Loop Lines 4-10 in Fig. 4 is executed.

g_e	An event such that its execution schedules event e(i.e., $e \in g_e.E$).
I_i	The set of processes that may schedule events to p_i.
K	The number of processors available in the parallel simulation ($K \leq N$).
L	The event list for the sequential simulation.
	The number of processes.
O_i	(The set of) the events generated at p_i such that their departure times in the Chandy-Misra simulation have not been decided in the parallelism analyzer.
$O_n(i)$	The value of O_n for the ith time when Line 10 in Fig. 4 is executed.
p_i	A process.
P_k	The set of processes mapped to processor k.
$Q_{j,i}$	(The set of) the events sent from p_j to p_i such that the start execution times of these events in the Chandy-Misra simulation have not been determined by the parallelism analyzer.
$\rho(\theta_n)$	The time when event θ_n is available for execution in the Chandy-Misra simulation.
t_i	The progress (i.e., real time) of process p_i.
$t_{m,n}(i)$	The value of t_n when $\theta_n = e_{m,n}(i)$ at Line 4 in Fig. 4.
$t_n^-(i)$	The value of t_n at the ith time after Line 4 (in Fig. 4) but before Line 5 is executed.
$t_n(i)$	The value of t_n at the ith time when Line 6 in Fig. 4 is executed.
$t_n^+(i)$	The value of t_n at the ith time when Line 7 in Fig. 4 is executed.
ts_i	The local clock of p_i at (real) time t_i in the Chandy-Misra simulation.
T	The execution time of the Chandy-Misra simulation.
$z_{j,i}$	The *end-of-simulation* (*eos*) message sent from p_j to p_i in the Chandy-Misra simulation.

References

[1] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM*, vol. 33, pp. 31–53, Oct. 1990.

[2] O. Berry and D. Jefferson, "Critical path analysis of distributed simulation," in *Proc. 1985 SCS Multiconf. Distribut. Simulation*, Jan. 1985, pp. 57–60.

[3] M. Livny, "A study of parallelism in distributed simulation," in *Proc. 1985 SCS Multiconf. Distrib. Simulation*, Jan. 1985, pp. 94–98.

[4] Y.-B. Lin, "Parallelism analyzers for parallel discrete event simulation," *ACM Trans. Model. Comput. Simulation*, vol. 2, no. 3, 1993.

[5] K. M. Chandy and J. Misra, "Distributed simulation: A case study in design and verification of distributed programs," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 440–452, Sept. 1979.

[6] R. M. Fujimoto, "Optimistic approaches to parallel discrete event simulation," *Trans. Soc. Comput. Simulation*, vol. 7, pp. 153–191, June 1990.

[7] ——, "Performance of time warp under synthetic workloads," in *Proc. 1990 SCS Multiconf. Distrib. Simulation*, Jan. 1990, pp. 23–28.

[8] Y.-B. Lin and B. R. Preiss, "Optimal memory management for time warp parallel simulation," *ACM Trans. Model. Comput. Simulation*, vol. 1, pp. 283–307, Oct. 1991.

[9] D. Jefferson, "Virtual time II: The cancelback protocol for storage management in time warp," in *Proc. 9th Annu. ACM Symp. Princip. Distrib. Comput.*, Aug. 1990, pp. 75–90.

[10] R. M. Fujimoto, "Lookahead in parallel discrete event simulation," in *Proc. Int. Conf. Parallel Process.*, vol. III, 1988, pp. 34–41.

[11] B. R. Preiss and W. M. Loucks, "Impact of lookahead on the performance of conservative distributed simulation," in *Proc. SCS Europ. Multiconf. Simulation Method., Lang., Architect.*, 1990, pp. 204–209.

[12] D. M. Nicol, "Parallel discrete-event simulation of FCFS stochastic queueing networks," in *Proc. ACM SIGPLAN Symp. Parallel Programm.: Experience Applicat., Lang., Syst.*, 1988, pp. 124–137.

[13] Y.-B. Lin and E. D. Lazowska, "Exploiting lookahead in parallel simulation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, pp. 457–469, Oct. 1990.

[14] K. M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM*, vol. 24, pp. 198–206, Apr. 1981.

[15] E. W. Dijkstra and C. S. Scholten, "Termination detection for diffusing computations," *Inform. Process. Lett.*, vol. 11, no. 3, pp. 1–4, 1980.

[16] L. Liu and C. Tropper, "Local deadlock detection in distributed simulations," in *Proc. 1990 SCS Multiconf. Distrib. Simulation*, Jan. 1990, pp. 64–69.

[17] J. Misra, "Distributed discrete-event simulation," *Comput. Surveys*, vol. 18, pp. 39–65, Mar. 1986.

[18] J. A. Payne, *Introduction to Simulation*. New York: McGraw-Hill, 1982.

Yung-Chang Wong received the B.S. degree in applied mathematics from National Chiao Tung University, Taiwan, R.O.C. in 1989, and the M.S. degree in computer science and information engineering from National Chiao Tung University in 1991.

He has been a Graduate Student in the department of computer science and information engineering at National Chiao Tung University since 1991. His research interests include distributed simulation, parallel processing, and artificial intelligence.

Shu-Yuen Hwang received the B.S. and M.S. degrees in electrical engineering from National Taiwan University in 1981 and 1983, respectively, and the M.S. and Ph.D. degrees in computer science from the University of Washington in 1987 and 1989, respectively.

He is currently Associate Professor and Head of the Institute of Computer Science and Information Engineering, National Chiao-Tung University. His research interests include computer vision, artificial intelligence and computer simulation.

Dr. Hwang is a member of the ACM.

Jason Yi-Bing Lin received the BSEE degree from National Cheng Kung University in 1983, and the Ph.D. degree in computer science from the University of Washington in 1990.

Since then, he has been with the Applied Research Area at Bell Communications Research (Bellcore), Morristown, NJ. His current research interests include design and analysis of telecommunication networks, distributed simulation, and performance modeling. He is the Guest Editor of the Special Issue on Simulation of Communication Systems for the *International Journal of Computer Simulation*, the Guest Editor of the Special Issue on Parallel and Distributed Simulation for the *Journal of Parallel and Distributed Computing*, the editor of a book *Advanced Topics in Distributed Simulation*, an Associate Editor of the *International Journal of Computer Simulation*, and Program Chair for the 8th Workshop on Distributed and Parallel Simulation. He is an Adjunct Research Fellow at the Center for Telecommunications Research, National Chiao-Tung University, Taiwan, R.O.C.