Parallelism Analyzers for Parallel Discrete Event Simulation

YI-BING LIN
Bellcore

Discrete event simulation is usually time consuming. Recently, there has been a great deal of interest in using parallel computers to speed up the simulation process. Before the parallel simulation approach is applied, it is important to understand the inherent parallelism of simulation applications. A simple technique called *critical path analysis* was proposed to study parallelism of simulation applications. This paper describes three critical path analysis algorithms based on different event-scheduling (process scheduling) policies. These algorithms are much simpler than a previous approach, where the events must be recorded in a trace and an extra pass is required to process the event trace. In our approach, the critical path analysis algorithms are integrated with the sequential simulation. At the end of the sequential simulation, the optimal parallel execution time is also computed. Livny proposed an algorithm similar to our approach (His, however, was designed for a specific language). Our algorithms can be integrated with sequential simulation programs written by users or be integrated with simulation languages. Another advantage of our algorithms over previous approaches is that ours can be used to study load balancing under different event-scheduling policies. Since our algorithms can be easily inserted in sequential simulation programs, critical path analysis can be applied to existing sequential programs without difficulty. The results can then be used to predict the performance of parallel simulation on similar applications. An example is given to show how useful information can be obtained from our algorithms.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**:] Process Management—*synchronization*; D.4.7 [**Operating Systems**]: Organization and Design—*distributed systems*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computations—*parallelism*

General Terms: Performance

Additional Key Words and Phrases: Discrete event simulation, event precedence graph, inherent parallelism, parallel simulation

1. INTRODUCTION

A discrete event simulation is represented by some set of data, called the *system state*, which contains all the information required to characterize the system at one point in time. The state remains unchanged until some *event* occurs that causes a discrete change in the state. In other words, state transitions are done via executions of series of events, with times when they occur. Execution of an event modifies a subset of a system state, which moves

Author's address: Bellcore, 445 South St., Rm. 2D297, P.O.B. 1961, Morristown, NJ 07962-1961.

```
  while not complete do
      /* Let $e$ be the next event to be executed in the sequential simulation */
1     execute $e$;
      /* Let $e.E$ be the set of events scheduled due to the execution of $e$ */
2     for all $e' \in e.E$ do
3         insert $e'$ into the event list according to the timestamp order;
  end while
```

Fig. 1. A simple discrete event simulator.

the system to a new state. Execution of any event can give rise to any number of events with later timestamps. A simple discrete event simulation algorithm is given in Figure 1.

Since most discrete event simulations are time consuming, several approaches [7, 8, 12] have been proposed to speed up the simulation process. A popular approach is for multiple computers to cooperate to execute a simulation run [5, 9]. It is important to analyze the inherent parallelism of the simulation application before we apply this approach.

Berry and Jefferson [4] and Livny [16] proposed a simple technique called *critical path analysis* to study the inherent parallelism of simulation applications. This technique can also be used to evaluate the performance of existing parallel simulation protocols. For example, Fujimoto [6] and Wagner and Lazowska [22] reported speed ups of up to 3 for a 5-node central server model. From the speed-up figures, it is difficult to see whether the parallel simulation protocols are efficient. Critical path analysis indicates that the maximum speed up that can be achieved in this benchmark is 3.67. Thus, the speedups obtained by Fujimoto et al. are actually quite respectable.

This paper proposes three critical path analysis algorithms based on different event-scheduling (process-scheduling) policies. These algorithms are much simpler than the previous approach [4], where the events must be recorded in a trace and an extra pass is required to process the event trace. In our approach, the critical path analysis algorithms are integrated with the sequential simulation. At the end of the sequential simulation, the optimal parallel execution time is also computed. Berry and Jefferson [3] implemented a similar algorithm for a special case where every process is executed by a dedicated processor. Livny [16] proposed an algorithm similar to ours. His, however, was designed for a specific simulation language, DISS. If the algorithm is integrated with a general sequential simulator as shown in Figure 1, the result is not correct. Our algorithms are easy to implement (although the correctness proofs are not trivial), and can be integrated with sequential simulation programs written by users or integrated with simulation languages. Another advantage of our algorithms over previous attempts [4, 16] is that ours can be used to study load balancing under different event-scheduling policies.

This paper is organized as follows. Section 2 gives an introduction to critical path analysis. Sections 3, 4, and 5 propose and prove the correctness of critical path analysis algorithms under three event-scheduling policies. Section 6 demonstrates how we can learn from critical path analysis by an example. Finally, Section 7 gives the conclusions. The notation used in this paper is listed in Appendix A.

2. CRITICAL PATH ANALYSIS

The idea of parallel simulation is based on the following observation: If two events are independent of each other, they can be executed in parallel. Two events, e and e', are independent if execution of e modifies the same subset of state variables (and the modified variables have the same values), no matter whether e' is executed before or after e, and vice versa. In a parallel simulation, the simulated system is partitioned into N subsystems. Subsystem i consists of a subset of state variables S_i such that

$$S_i \cap S_j = \emptyset, \quad \text{for} \quad 1 \le i \ne j \le N, \quad \text{and} \quad \bigcup_{1 \le i \le N} S_i = S$$

where S is the set of state variables. These subsystems are concurrently simulated by a set of processes that communicate by exchanging timestamped messages. The events scheduled for process i can only modify state variables in S_i.

After the simulated system is partitioned, execution of events follows two sequential constraints [4, 10, 16]:

Constraint 1. If two events are schedules for the same process, the event with smaller timestamp must be executed before the one with larger timestamp.

Constraint 2. If an event executed at a process results in the scheduling of another event at a different process, then the former must be executed before the latter.

Partitioning the simulated system into subsystems is not a trivial task. If there are too many subsystems, the communication overhead due to Constraint 2 may outweigh the benefit provided by parallelism. On the other hand, if there are too few subsystems, independent events may be executed sequentially due to Constraint 1. Several studies [18, 21] are devoted to the partitioning problem. This paper will propose algorithms that can be used to study the granularity issue.

Based on Constraints 1 and 2, an *event precedence graph* is built for each parallel simulation. Each vertex of the graph represents an event, and each edge represents a communication. An *event execution time* is associated with each vertex. A *communication delay* is associated with each edge. 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. We refer to a parallel simulation with this minimum execution time as an *optimal parallel simulation*.[1] Examples

[1] Note that some parallel simulation protocols (such as lazy cancellation for Time Warp) may beat the critical path in theory [2, 20]. Such protocols seldom outperform the critical path in practice.

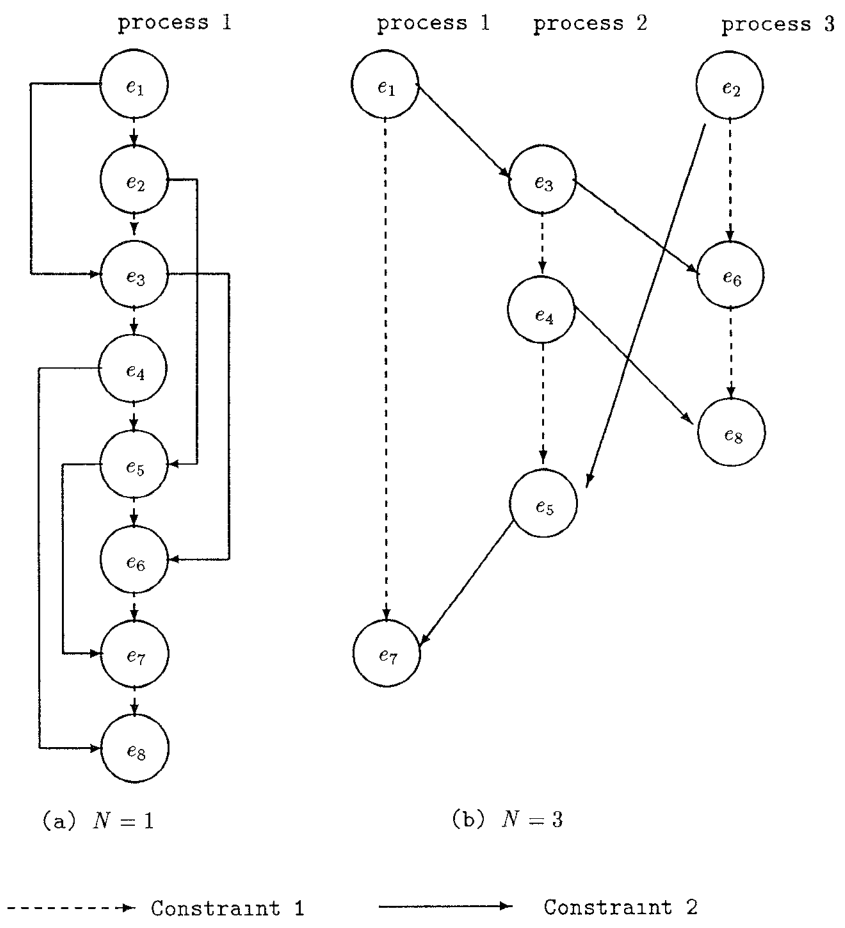

Fig. 2. Event precedence graphs.

of an event precedence graph are given in Figure 2. A dashed arrow represents the scheduling constraint of Constraint 1, and a continuous arrow represents the process communication due to Constraint 2. The event precedence graph for sequential simulation is given in Figure 2a, and the graph for 3 processes is in Figure 2b, where events e_1 and e_7 are scheduled for process 1, e_3, e_4, and e_5 are scheduled for process 2, and e_2, e_6, and e_8 are scheduled for process 3. The cost for the critical path can be derived quantitatively [1, 11, 13, 16] as follows. Let g_e be an event such that event e is scheduled due to the execution of g_e. In Figure 2b, $g_{e_7} = e_5$. An event e is prescheduled if g_e does not exist. In Figure 2, events e_1 and e_2 are prescheduled. Let p_e be an event such that both events e and p_e are scheduled for the same process, and the execution of p_e is followed by the execution of e. In Figure 2b, $p_{e_7} = e_1$. Let $\tau(e)$ be the earliest time when e's execution starts. Let $\eta(e)$ be the execution time for the event e. Let $\bar{\tau}(e)$ be the earliest time when e's execution completes. If every process is executed by a dedicated processor, then

$$\bar{\tau}(e) = \tau(e) + \eta(e). \tag{1}$$

Let $\delta(e)$ be the time to schedule event e. If g_e and e are scheduled at different processes, then $\delta(e)$ represents the message-sending delay. Otherwise, $\delta(e)$ is assumed to be 0 in our study. From Constraints 1 and 2,

$$\tau(e) = \begin{cases} 0 & \text{if neither } g_e \text{ nor } p_e \text{ exist,} \\ \bar{\tau}(p_e) & \text{if } g_e \text{ does not exist,} \\ \bar{\tau}(g_e) + \delta(e) & \text{if } p_e \text{ does not exist,} \\ \max[\bar{\tau}(p_e), \bar{\tau}(g_e) + \delta(e)] & \text{otherwise.} \end{cases} \tag{2}$$

Let T_p be the cost for the critical path (i.e., the execution time of the optimal parallel simulation). Let T_s be the sequential execution time. Then T_p and T_s are expressed as

$$T_p = \max_{\forall e} \bar{\tau}(e), \quad \text{and} \quad T_s = \sum_{\forall e} \eta(e). \tag{3}$$

The optimal parallel simulation time is computed based on (2). However, (2) is only adequate for the case when every process is executed by a dedicated processor. If the number of processors P is less than the number of processes N, then T_p is also affected by process assignment (this paper only considers static process assignment) and event scheduling (or process scheduling). If more than two processes are assigned to a processor, (2) only represents the time when an event is available for execution. More than two events from different processes may be available for execution at a processor at the same time. An event-scheduling policy is required to determine the next event to be executed. Let P_k be the set of indexes of processes mapped into processor k (i.e., $i \in P_k$ if process i is mapped to processor k). Let $\bar{e}_j(t)$ be an event scheduled for process j such that for all events scheduled for process j, $\bar{e}_j(t)$ is the next event to be executed after time t (and at time t, processor k is available to execute the next event). Let $e.ts$ be the timestamp of e. We consider three event-scheduling policies.

Definition 1 (*Event-scheduling policy* I). Suppose that processor k is available to execute the next event after time t. The event $\bar{e}_i(t)$ is the next event to be executed at processor k if and only if for all $j \neq i$, $i, j \in P_k$, $\bar{e}_j(t).ts > \bar{e}_i(t).ts$.

In Policy I, all events arrive at a processor are executed in nondecreasing timestamp order.

Definition 2 (*Event-scheduling policy* II). Suppose that processor k is available to execute the next event after time t. The event $\bar{e}_i(t)$ is the next event to be executed at processor k if and only if $\bar{e}_i(t)$ arrives at processor k earlier than $\bar{e}_j(t)$, for all $j \neq i$, $i, j \in P_k$.

In Policy II, among the events (of different processes) available for execution at a processor, the event with the earliest arrival time is selected.

Definition 3 (*Event-scheduling policy* III). Suppose that processor k is available to execute the next event after time t. The event $\bar{e}_i(t)$ is the next event to be executed at processor k if and only if $\bar{e}_i(t)$ is the event with the smallest timestamp among the events $\bar{e}_j(t)$, for all $j \in P_k$, that arrive at processor k no later than t.

In Policy III, among the events available for execution at a processor, the event with the smallest timestamp is selected.

The speed that can be achieved by Policy I is less than that of Policies II and III. Consider the following example of 4-process-3-processor simulation. The event-precedence graph is illustrated in Figure 3. Suppose that process 1 is mapped into processor 1, processes 2 and 3 are mapped into processor 2, and process 4 is mapped into processor 3. Let the timestamp of event i be i. Assume that there is no communication cost, and the execution time for event 1 is 5 units, the execution time for event 5 is 4 units, and the execution times for other events are 1 unit. If we perform the critical path analysis under Policy I, the parallel execution time is 12 units, as shown in Figure 4. The minimum parallel execution time based on Policy II (or Policy III; in this example, both policies are identical) is 11 units, as shown in Figure 5. Note that events 4 and 8 are independent of events 3 and 7. In Figure 4, event 4 cannot be executed before event 3 arrives, even if processor 2 is idle. In Figure 5, both events 4 and 8 are executed before event 3 arrives at processor 2, which gives the maximum parallelism that can be exploited.

We are interested in Policy I for two reasons. First, the critical path analyzer based on Policy I is easy to implement. If we are only interested in the maximum parallelism of a simulation application (i.e., we only consider the case when $N = P$), then the algorithm we propose for Policy I is much simpler than the algorithms for Policies II and III. Also, this policy is adequate in the environment where the context-switching overhead is large. In such a case, it is more appropriate to implement a set of processes as a super process. This policy is also adequate for Chandy–Misra simulation of systems without lookahead, where processes of a strongly connected component are mapped into a processor [15].

Both policies II and III may beat each other under different conditions. The progress of global virtual time [9] for Policy III is better than that for Policy II. On the other hand, the critical path analyzer for Policy II proposed in this paper is easier to implement than that for Policy III. In the following three sections we describe critical path analysis algorithms under Policies I, II, and III, respectively.

3. A CRITICAL PATH ANALYZER BASED ON POLICY I

In [4], the critical path analysis is performed as follows. We first instrument a sequential simulation and take a trace of the events executed. The trace is then transformed into a event-precedence graph. Finally, the cost of the

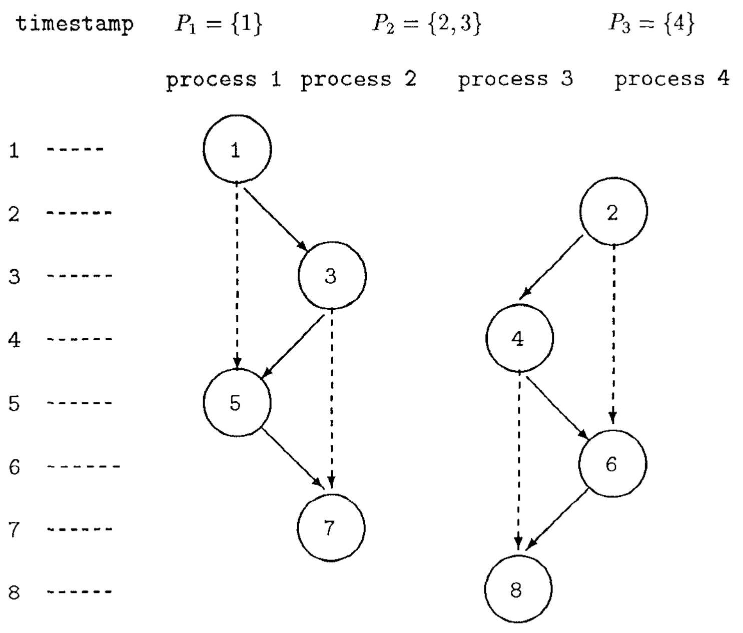

Fig. 3. The event precedence graph for a 4-process-3-processor simulation.

critical path in the graph is computed. The critical path analyzer proposed by Livny [16] is integrated with a specific simulation, DISS, and the cost for the critical path is computed along with the execution of the simulation. Thus, no event trace is required, and no explicit construction of a precedence graph is necessary. It is not clear how the algorithm works in DISS. However, if we integrate Livny's algorithm with a general sequential simulator (as shown in Figure 1), the result is not correct. Consider the example in Figure 2b. If the event execution time is 1 unit and the communication cost is 0, then the execution time for the optimal parallel simulation for the first three events (i.e., e_1, e_2, and e_3) is 2 units. However, the value computed from Livny's algorithm is 1 unit.

The above two approaches are primarily designed for the case when $P = N$. We now describe a correct critical path analyzer which can be integrated with the sequential simulation program. This algorithm, referred to as Algorithm 1 (cf., Figure 6), is designed under the assumption that every process is executed by a dedicated processor. (In this section the terms "process" and "processor" are used interchangeably.) The extension of this algorithm for Policy I is trivial. The processes mapped into a processor are considered a "super" process, and all events are executed in the timestamp order at the processor (this approach is also used in Livny's algorithm).

Let event e be scheduled for process i. In this algorithm, $e.\alpha$ represents the time when event e arrives at process i. The value of T_i after the execution of Line 1b is the completion time of e's execution in the optimal

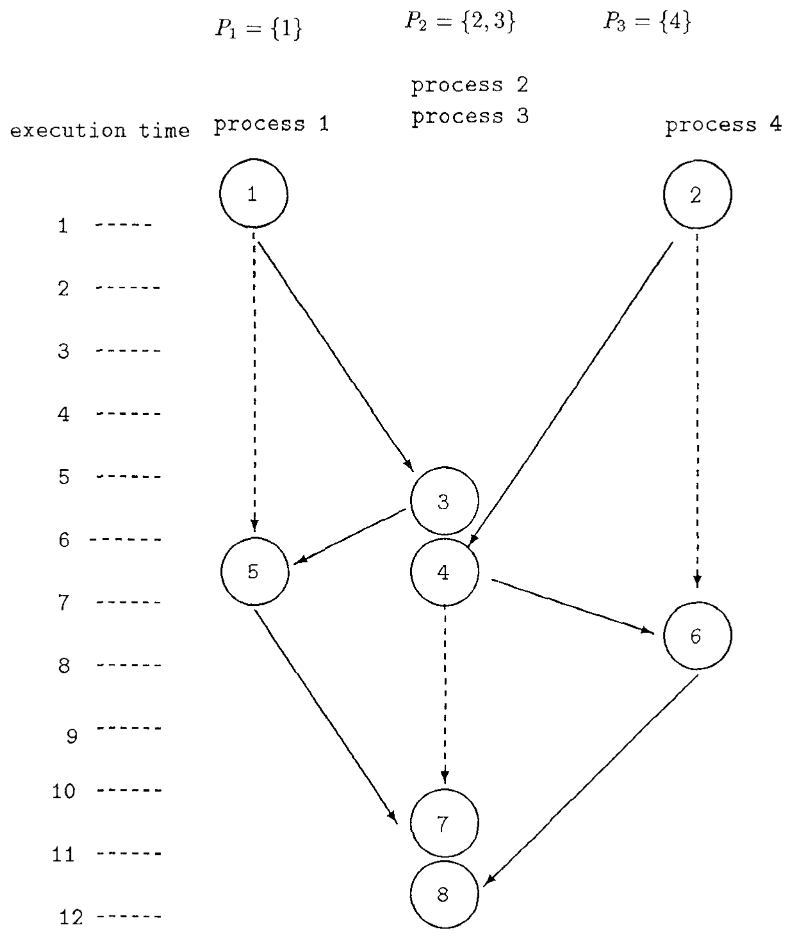

Fig. 4. The parallel execution time of the 4-process-3-processor simulation computed from Policy I. Assume that there is no communication cost, that the execution time for event 1 is 5 units, that the execution time for event 5 is 4 units, and that the execution time for other events is 1 unit.

parallel simulation. The correctness of the algorithm is proved as follows.

LEMMA 1. *In Algorithm* 1, *we have*

(*a*) $e.\alpha = 0$, *if* g_e *does not exist.*
(*b*) $T_i = 0$, *before e is executed if e is scheduled for process i and* p_e *does not exist.*

PROOF. (a) An event e is prescheduled in the event list if and only if g_e does not exist. For a prescheduled event e, we have $e.\alpha = 0$ (cf., line 0c in Figure 6). (b) If p_e does not exist, then e is scheduled for process i and is the first such event (i.e., the first event that may modify S_i) executed in the

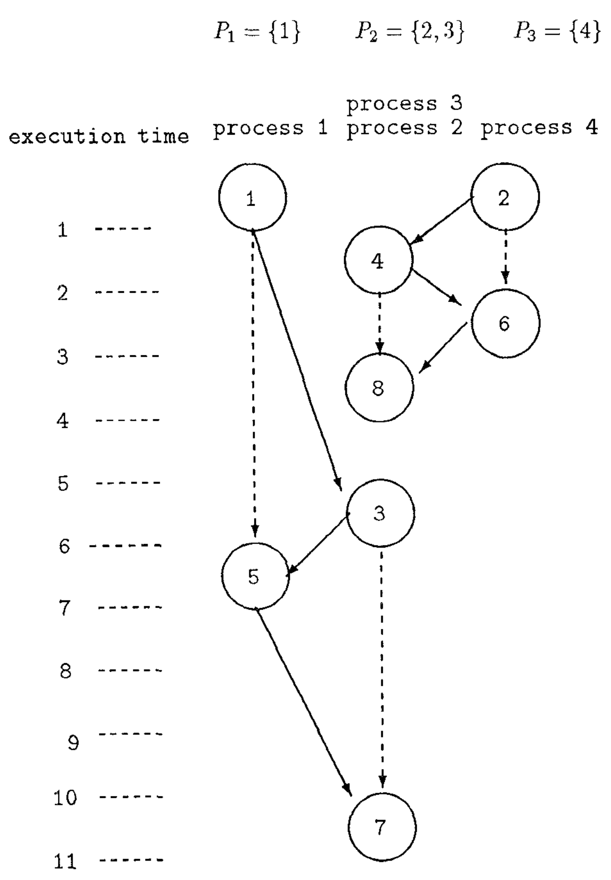

Fig. 5. The parallel execution time of the 4-process-3-processor simulation computed based on Policy II (III). Assume that there is no communication cost, that the execution time for event 1 is 5 units, that the execution time for event 5 is 5 units, and that the execution time for other events is 1 unit.

sequential simulation. In other words, T_i has not been modified before e is executed (cf., Figure 6). Thus $T_i = 0$ (cf., line 0a). □

LEMMA 2. *In Algorithm* 1, $T_i = \bar{\tau}(e)$ *after line* 1*b is executed, where event e is scheduled for process i.*

PROOF. Let $\varphi(e)$ be the iteration when e is processed in Algorithm 1. It is clear that $\varphi(e) > \varphi(p_e)$ and $\varphi(e) > \varphi(g_e)$. We prove by induction on φ that $T_i = \bar{\tau}(e)$ after line 1b is executed for every event e.

Basis. $\varphi(e) = 1$. Event e is prescheduled in the event list, and is the first event executed in the sequential simulation. Before line 1b is executed, $T_i = 0$ (cf., Lemma 1(b)) and $e.\alpha = 0$ (cf., Lemma 1 (a)). Thus, after line 1b is executed, $T_i = \max(0,0) + \eta(e) = \eta(e) = \bar{\tau}(e)$ (cf., (1) and the first case in (2)).

```
    /* initialization */
0a  for all $i$ do $T_i = 0$;
0b  for all $e$ pre-scheduled in the event list do
0c      $e.\alpha = 0$;
    end for;
    /* the main loop */
    while not complete do
        /* Let $e$ be the next event to be executed in the sequential simulation */
1a      execute $e$;
        /* Let $e$ be scheduled for process $i$ */
1b      $T_i = \max(T_i, e.\alpha) + \eta(e)$;
        /* Let $e.E$ be the set of events scheduled due to the execution of $e$ */
2       for all $e' \in e.E$ do
3a          $e'.\alpha = T_i + \delta(e')$;
3b          insert $e'$ into the event list according to the timestamp order;
        end for
4           $T_s = T_s + \eta(e)$;
    end while
```

Fig. 6. Algorithm 1: A critical path analyzer for Policy I.

Inductive step. Assume that the hypothesis holds for all $\varphi(e') \le n$. We show that it also holds for $\varphi(e) = n + 1$. If g_e does not exist, then

$$e.\alpha = 0 \tag{4}$$

(cf., Lemma 1 (a)). If g_e exists, then consider $\varphi(g_e)$, the iteration when g_e is executed. Event e is created after line 2 at $\varphi(g_e)$. Note that g_e is executed at process i and e is scheduled for process j. After line 3a is executed,

$$e.\alpha = \bar{\tau}(g_e) + \delta(e) \tag{5}$$

(from the hypothesis). Based on (4) and (5), we have

$$e.\alpha = \begin{cases} 0 & g_e \text{ does not exist} \\ \bar{\tau}(g_e) + \delta(e) & \text{otherwise.} \end{cases} \tag{6}$$

Now consider $\varphi(e)$, the iteration when e is executed. Note that at this iteration, e is scheduled for process i (i.e., process i at e's iteration is process j at g_e's iteration, if g_e exists). If p_e does not exist, then $T_i = 0$ (cf., Lemma 1 (a)). If p_e exists, then $\varphi(p_e) < \varphi(e)$ (cf., definition of φ). Before line 1b is executed, the last time when T_i is modified is at p_e's iteration, and the value is $T_i = \bar{\tau}(p_e)$ (from the hypothesis), and

$$T_i = \begin{cases} 0 & \text{if } p_e \text{ does not exist} \\ \bar{\tau}(p_e) & \text{otherwise.} \end{cases} \tag{7}$$

Based on (2), (4), (6), and (7), $T_i = \bar{\tau}(e)$ after line 1b is executed. □

THEOREM 1. *At the end of Algorithm* 1, $T_p = max_{\forall i} T_i$.

PROOF. Let Ψ_i be the set of events scheduled for process i. Without loss of generality, assume that $e.ts \ne e'.ts$, where $e, e' \in \Psi_i$. At the end of the algorithm, $T_i = \bar{\tau}(e) > \bar{\tau}(e')$, where $e, e' \in \Psi_i$ and $e.ts > e'.ts$ (From Lemma

2 and the fact that events are executed in nondecreasing timestamp order in the sequential simulation). From (3), at the end of Algorithm 1, we have

$$T_p = \max_{\forall i} \bar{\tau}(e) = \max_{\forall i}\left[\max_{\forall e \in \Psi_i} \bar{\tau}(e)\right] = \max_{\forall i} T_i \quad \square$$

It is clear that at the end of the execution of Algorithm 1, the value of T_s is the sequential execution time defined at (3).

4. A CRITICAL PATH ANALYZER BASED ON POLICY II

This section proposes a critical path analyzer (referred to as Algorithm 2; cf., Figure 7) for Policy II. Similar to Algorithm 1, this algorithm is integrated with the sequential simulation, and no explicit construction of an event precedence graph is needed.

Let t be the time when processor k completes the execution of an event in the optimal parallel simulation under Policy II. Let $\Phi_k(t) = \{\bar{e}_j(t),\ j \in P_k\}$. Let $\sigma_k(t) = e$ if $e \in \Phi_k(t)$ and for all $e' \neq e$, $e' \in \Phi_k(t)$, e arrives at processor k earlier than e'. According to Policy II, $\sigma_k(t)$ is the next event to be executed at processor k after time t.

The idea behind Algorithm 2 is the following. Let event e be scheduled for process i, where $i \in P_k$. When e is processed at lines 4–9 in Algorithm 2, $\tau(e)$ cannot be determined as in Algorithm 1. It is possible that there exists another event e', $e'.ts > e.ts$, and e' is scheduled for process j, $j \in P_k$ such that e' arrives at processor k earlier than e does. If e' is the next event to be executed at process j, then event e' must be executed before e under Policy II. In the main loop (lines 4–13) of Algorithm 2, e is processed earlier than e', and $\tau(e)$ cannot be determined before e' is processed in Algorithm 2. Thus, for every process, Algorithm 2 maintains a queue Q_i which contains every event e scheduled for process i such that $\tau(e)$ has not been determined. The event e will not be deleted from Q_i until $\bar{\tau}(e)$ is computed.

Suppose that Algorithm 2 is executed up to a timestamp T (after lines 10–13 are executed). Let $e_{(i)}$ be the event with the smallest timestamp in Q_i; i.e.,. $e_{(i)} \in Q_i$ and $e_{(i)}.ts = \min_{\forall e \in Q_i} e.ts$. Let $e_{(i)} = \phi$ if $Q_i = \emptyset$. Let $\Phi_k^* = \{e_{(i)} \mid i \in P_k\}$. Let $\Phi^* = \{e_{(i)}, \forall i\}$. Let $\theta(\Phi^*, T)$ be a predicate that returns *true* if and only if (i) for all i, $e_{(i)} \neq \phi$, and (ii) for all i, $e_{(i)}.ts < T$. In other words, θ is a predicate to test whether the timestamp of $e_{(i)}$, for all i, is less than T. If the **if** statement at line 10 is true, then the last event has been processed in the sequential simulation, and for every i, $1 \leq i \leq N$, an event z_i with timestamp $z_i.ts = \infty$ is artificially created to terminate Algorithm 2. We will elaborate later. At line 15, let

$$\sigma^* = \begin{cases} e & \text{if } e.\alpha = \min_{e' \in \Phi^*} e'.\alpha \\ \phi & \text{otherwise.} \end{cases} \tag{8}$$

When an event e (scheduled for process i) is processed in lines 4–13 in Algorithm 2, the event is inserted in Q_i. Then we test if $\theta(\Phi^*, e.ts)$ is true at line 14. If so, then (from Theorem 2) $e_{(j)} = \bar{e}_j(T_k)$ for all $k, j \in P_k$, and $\sigma^* \neq \phi$ (let σ^* be scheduled for process m and $m \in P_l$) is the next event to be executed at processor l after real time T_l in optimal parallel simulation.

```
   /* initialization */
1  for $k = 1$ to $K$ do $T_k = 0$;
2  for all $e$ pre-scheduled in the event list do $e.\alpha = 0$;
3  for $i = 1$ to $N$ do $Q_i = \emptyset$;
   /* the main loop */
   while not complete do
       /* Let $e$ be the next event to be executed in the sequential simulation */
4      execute $e$;
       /* Let $e.E$ be the set of events scheduled due to the execution of $e$ */
5      for all $\bar{e} \in e.E$ do
6          $\bar{e}.\alpha = \infty$;
7          insert $\bar{e}$ into the event list according to the timestamp order;
       end for
       /* Let $e$ be scheduled for process $i$ */
8      $Q_i = Q_i \cup \{e\}$;
9      $T_s = T_s + \eta(e)$;
10     if $e$ is the last event to execute then
11         for $j = 1$ to $N$ do $Q_j = Q_j \cup \{z_j\}$; /* where $z_j.ts = \infty$ */
12         $T = \infty$;
13     else $T = e.ts$;
14     while $\theta(\Phi^*, T) = true$ do
15         $e^* = \sigma^*$; /* cf. (8) */
           /* Let $e^*$ be scheduled for process $m$, and $m \in P_l$ */
16         $T_l = \max(T_l, e^*.\alpha) + \eta(e^*)$;
17         for all $e' \in e^*.E$ do $e'.\alpha = T_l + \delta(e')$;
18         $Q_m = Q_m - \{e^*\}$;
19     end while
   end while
```

Fig. 7. Algorithm 2: A critical path analyzer for Policy II.

(In other words, $\sigma^* = \sigma_l(T_l)$.) The execution of Algorithm 2 on the 4-process-3-processor simulation is given in Table I. The first row of the table gives the iteration when event e is processed in Algorithm 2. The column e^- gives the values of variables when event e is processed up to line 14. The column e represents the values after line 19 is executed. For example, the value of Q_1 at line 14 of event 5's iteration is $\{1, 5\}$ (at column 5^-), and the value is $\{5\}$ at the end of event 5's iteration (at column 5). At the end of Algorithm 2, special event z_i is inserted in Q_i for all i (cf., line 11 in Figure 7 and column 8^- in Table I). The timestamp of z_i is ∞. These artificial events are only used to compute the parallel execution time of last events. They are not processed in the sequential simulation. Now we prove that Algorithm 2 is correct.

THEOREM 2. *If* $\theta(\Phi^*, T) =$ true *at line* 14 *of Algorithm* 2, *then*

(*a*) *for all* $j \in P_k$, $e_{(j)} = \bar{e}_j(T_k)$ *after time* T_k *in the optimal parallel simulation under Policy* II; *in other words,* $\Phi_k^* = \Phi_k(T_k)$;

(*b*) $\sigma^* \neq \phi$, *and* $\sigma^*.\alpha < \infty$ *is the* (*real*) *time when* σ^* *arrives at processor* l *in the optimal parallel simulation* (*where* σ^* *is scheduled for process* m, *and* $m \in P_l$); *and*

Table I. Execution of Algorithm 2 for 4-Process-3-Processor Simulation

e	1	2	3	4	5^-	5	6^-	6
T	1	2	3	4	5	5	6	6
T_1	0	0	0	0	0	5	5	5
T_2	0	0	0	0	0	0	0	0
T_3	0	0	0	0	0	0	0	1
Q_1	$\{1\}$	$\{1\}$	$\{1\}$	$\{1\}$	$\{1,5\}$	$\{5\}$	$\{5\}$	$\{5\}$
Q_2	$\emptyset$	$\emptyset$	$\{3\}$	$\{3\}$	$\{3\}$	$\{3\}$	$\{3\}$	$\{3\}$
Q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\{4\}$	$\{4\}$	$\{4\}$	$\{4\}$	$\{4\}$
Q_4	$\emptyset$	$\{2\}$	$\{2\}$	$\{2\}$	$\{2\}$	$\{2\}$	$\{2,6\}$	$\{6\}$
Φ^*	$\{1\}$	$\{1,2\}$	$\{1,2,3\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$		$\{2,3,4,5\}$	
$\theta(\Phi^*,T)$	false	false	false	false	true		true	
e^*					1		2	
$1.\alpha$	0	0	0	0	0	0	0	0
$2.\alpha$	0	0	0	0	0	0	0	0
$3.\alpha$	∞	∞	∞	∞	∞	5	5	5
$4.\alpha$	-	∞	∞	∞	∞	∞	∞	1
$5.\alpha$	-	-	∞	∞	∞	∞	∞	∞
$6.\alpha$	-	-	-	∞	∞	∞	∞	∞
$7.\alpha$	-	-	-	-	-	∞	∞	∞
$8.\alpha$	-	-	-	-	-	-	∞	∞

(c) *after time T_l, $\sigma^* = \sigma_l(T_l)$, and $\tau(\sigma^*) = max(\sigma^*.\alpha, T_l)$ in the optimal parallel simulation.*

PROOF. Let $\psi = n$ represent the nth time that $\theta(\Phi^*, T)$ is true at line 14 of Algorithm 2. We prove by induction on ψ that Hypotheses (a), (b), and (c) are true.

Basis $\psi = 1$. (a) From the definition of θ, $Q_j \neq \emptyset$, for all j, and all events in Q_j with timestamps less than T are the events with timestamps less than T scheduled for process j. (Note that every event e processed in Algorithm 2 is inserted in Q_j for some j; cf., line 8. The event e can only be deleted from Q_j at line 17, and before $\psi = 1$, no event is deleted.) Thus, $e_{(j)}$ is the first event scheduled for process j, and since $T_k = 0$, where $j \in P_k$ (cf., line 1), we have $e_{(j)} = \bar{e}_j(T_k)$ after time T_k, and (a) holds.

(b) When $\theta(\Phi^*, T)$ is true, we have (i) for all j, $e_{(j)} \neq \phi$ (definition of θ), (ii) for all j, $e_{(j)}$ is the first event scheduled for process j (because $T_k = 0$, for all k, and Hypothesis (a) holds at $\psi = 1$), and (iii) there exist i such that $e_{(i)}$ is a prescheduled event (because the first event executed in Algorithm 2 is a prescheduled event). If $e_{(j)}$ is a prescheduled event, then $e_{(j)}.\alpha = 0$ (cf., line 2), otherwise, $e_{(j)}.\alpha = \infty$ (from line 6, and because line 17 is never executed before $\psi = 1$). Thus, σ^* is a prescheduled event and $\sigma^*.\alpha = 0$ is the time when σ^* arrives at processor l (definition of σ^*). In other words, (b) holds.

(c) Since $T_l = 0$ and because σ^* is a prescheduled event (from line 1 and because (i) line 16 is never executed before $\psi = 1$, and (ii) Hypothesis (b) holds when $\psi = 1$), (c) also holds.

Table I. (*Continued*)

e	7^-	7	8^-	8^-	8^-	8^-	8^-	8^-
T	7	7	∞	∞	∞	∞	∞	∞
T_1	5	5	5	5	5	5	10	10
T_2	0	2	2	2	4	6	6	11
T_3	1	1	1	3	3	3	3	3
Q_1	$\{5\}$	$\{5\}$	$\{5, z_1\}$	$\{5, z_1\}$	$\{5, z_1\}$	$\{5, z_1\}$	$\{z_1\}$	$\{z_1\}$
Q_2	$\{3, 7\}$	$\{3, 7\}$	$\{3, 7, z_2\}$	$\{3, 7, z_2\}$	$\{3, 7, z_2\}$	$\{7, z_2\}$	$\{7, z_2\}$	$\{z_2\}$
Q_3	$\{4\}$	∞	$\{8, z_3\}$	$\{8, z_3\}$	$\{z_3\}$	$\{z_3\}$	$\{z_3\}$	$\{z_3\}$
Q_4	$\{6\}$	$\{6\}$	$\{6, z_4\}$	$\{z_4\}$	$\{z_4\}$	$\{z_4\}$	$\{z_4\}$	$\{z_4\}$
Φ^*	$\{3, 4, 5, 6\}$		$\{3, 5, 6, 8\}$	$\{3, 5, 8, z_4\}$	$\{3, 5, z_3, z_4\}$	$\{5, 7. z_3, z_4\}$	$\{7, z_1, z_3, z_4\}$	$\{z_1, z_2, z_3, z_4\}$
$\theta(\Phi^*, T)$	true		true	true	true	true	true	false
e^*	4		6	8	3	5	7	
$1.\alpha$	0	0	0	0	0	0	0	0
$2.\alpha$	0	0	0	0	0	0	0	0
$3.\alpha$	5	5	5	5	5	5	5	5
$4.\alpha$	1	1	1	1	1	1	1	1
$5.\alpha$	∞	∞	∞	∞	∞	6	6	6
$6.\alpha$	∞	2	2	2	2	2	2	2
$7.\alpha$	∞	∞	∞	∞	∞	∞	10	10
$8.\alpha$	∞	∞	∞	3	3	3	3	3

Inductive step. Assume that Hypotheses (a), (b), and (c) hold for $\psi \leq n$, where $n > 0$. Now consider $\psi = n + 1$.

(a) For all j, let e_j^- be the last event deleted from Q_j at $\psi = n' < n + 1$. By convention, $e_j^- = \phi$ and $e_j^-.ts = -1$[2] if and only if no event is deleted from Q_j before $\psi = n + 1$. Then we have the following lemma.

LEMMA 3. *At $\psi = n + 1$, the events in Q_j with timestamps less than T are the events scheduled for process j with timestamps in the range $(e_j^-.ts, T)$. At least one such event exists in Q_j.*

PROOF. If $e_j^- = \phi$, then follows an argument similar to the proof for (a) in Basis, the events in Q_j with timestamps less than T are the events scheduled for process j with timestamps in the range $[0, T)$. From the definition of θ, at least one such event exists in Q_j at $\psi = n + 1$.

Now consider the case $e_j^- \neq \phi$. When the event with timestamp T is processed in Algorithm 2, all events scheduled for process j with timestamps less than T have been processed and inserted in Q_j (cf., line 8). Among them, the events with timestamps no larger than e_j^- have been deleted from Q_j (cf., line 18). At least one such event exists at $\psi = n + 1$ (definition of θ). □

Since Hypothesis (c) holds for $\psi \leq n$, we have $T_k = \max_{\forall j \in P_k} \bar{\tau}(e_j^-)$, and from Lemma 3, $e_{(j)} = \bar{e}_j(T_k)$ after T_k. Thus, Hypothesis (a) holds.

(b) To prove that Hypothesis (b) holds, the following lemmas are introduced.

LEMMA 4. *At $\psi = n + 1$, suppose that $e_{(j)}$ is not a prescheduled event. Let $g_{e_{(j)}}$ be scheduled for process i.*

(i) $g_{e_{(j)}}$ is either in Q_i, or

(ii) $g_{e_{(j)}}$ is deleted from Q_i at line 18 at $\psi = n' < n + 1$, for some n'.

PROOF. Since $g_{e_{(j)}}.ts < e_{(j)}.ts < T$, $g_{e_{(j)}}$ is already processed in lines 4–9 in Algorithm 2, and is inserted in Q_i for some i (line 8). $g_{e_{(j)}}$ is deleted from Q_i if and only if $\sigma^* = g_{e_{(j)}}$ at $\psi = n' < n + 1$, for some n' (line 18). □

LEMMA 5. *Suppose that $e_{(j)}$ is not a prescheduled event. At $\psi = n + 1$, $e_{(j)}.\alpha$ is the time when $e_{(j)}$ arrives at processor k (where $j \in P_k$) if $g_{e_{(j)}}$ is deleted from Q_i at line 18 at $\psi = n' < n + 1$.*

PROOF. Since $e_{(j)}$ is not a prescheduled event, $g_{e_{(j)}}$ exists. $e_{(j)}.\alpha$ can only be modified at line 17. In other words, $e_{(j)}.\alpha < \infty$ if $\sigma^* = g_{e_{(j)}}$ at $\psi = n' < n + 1$. Since $n' < n + 1$, $T_l = \bar{\tau}(g_{e_{(j)}})$ (from Hypothesis (c)). Thus, after line 17 is executed at $\psi = n'$, $e_{(j)}.\alpha$ is the time when $e_{(j)}$ arrives at processor k. □

LEMMA 6. *At $\psi = n + 1$, there exist $e_{(j)} \in \Phi^*$ such that $e_{(j)}.\alpha < \infty$.*

If there exists $e_{(j)} \in \Phi^*$ such that $e_{(j)}$ is a prescheduled event, then $e_{(j)}.\alpha = 0 < \infty$. If no event in Φ^* is a prescheduled event, then we prove by contradiction, as follows. Assume that $e_{(j)} = \infty$ for all j. Let $g_{e_{(j)}}$ be scheduled

[2] We assume that all events occur in the simulation have timestamps larger than 0.

for process j_1. We have $j_1 \neq j$; otherwise since $g_{e_{(j)}}.ts < e_j^-.ts$, $g_{e_{(j)}}$ is deleted from Q_{j_1} before $\psi = n + 1$ (from Lemmas 3 and 4(ii)), and $e_{(j)}.\alpha < \infty$ (Lemma 5), a contradiction. Now consider $e_{(j_1)}$. Since $e_{(j_1)}.ts < g_{e_{(j)}}.ts < e_{(j)}.ts$, $g_{e_{(j_1)}}$ must be scheduled for process $j_2 \neq j_1, j$. Otherwise $g_{e_{(j_1)}}$ must be deleted before $\psi = n + 1$, a contradiction. If we repeat this process, we will find a sequence $j, j_1, j_2, \dots, j_{N-1}$ such that $g_{e_{(j_{N-1})}}$ cannot be found in any Q_ι. Thus, case (i) in Lemma 4 is not satisfied, which contradicts the assumption that $e_{(j)}.\alpha = \infty$ for all j. In other words, there exist at least one j such that $e_{(j)}.\alpha < \infty$. □

From Lemma 6 and the definition of σ^*, we have $\sigma^*.\alpha < \infty$. From Lemma 5, Hypothesis (b) holds.

(c) At $\psi = n + 1$, let σ^* be executed at processor l. Since Hypothesis (a) holds at $\psi = n + 1$, we have $\Phi_l^* = \Phi_l(T_l)$; that is, the next event to be executed at processor l after T_l in the optimal parallel simulation is among $e_{(j)}$, $j \in P_l$. We need to consider two cases.

—If $e_{(j)}.\alpha < \infty$ for all $j \in P_l$, then $e_{(j)}.\alpha$ is the time when $e_{(j)}$ arrives at processor l from Lemma 5), and $\sigma^* = \sigma_l(T_l)$ (cf., Definitions of σ^* and σ_l and because Hypothesis (a) is true).

—If there exists $j \in P_l$ such that $e_{(j)}.\alpha = \infty$, then we need to show that the arrival time of $e_{(j)}$ at processor l is later than $\sigma^*.\alpha$. Follow a argument similar to the proof for Lemma 6, we have:

LEMMA 7. *At $\psi = n + 1$, let σ^* be scheduled for processor l. If there exists $j \in P_l$ such that $e_{(j)}.\alpha = \infty$, then there exists i such that $e_{(i)}.\alpha < \infty$ and $e_{(j)}$ cannot be generated before $e_{(i)}$ is executed in the optimal parallel simulation. Thus, the arrival time of $e_{(j)}$ is later than $e_i.\alpha$.*

Since $\sigma^*.\alpha \le e_{(i)}.\alpha$ for all $e_{(i)}.\alpha < \infty$ (cf. definition of σ^*), the arrival time of $e_{(j)}$ is later than $\sigma^*.\alpha$ for all $e_{(j)}.\alpha = \infty, j \in \Phi_l^*$ (Lemma 7). From Hypothesis (a), $\Phi_l^* = \Phi_l(T_l)$, and we have $\sigma^* = \sigma_l(T_l)$ (cf. definitions of σ^* and σ_l). From Hypothesis (b) and Lemma 5, $\tau(\sigma^*) = \max(\sigma^*.\alpha, T_l)$.

Thus Hypothesis (c) holds. □

THEOREM 3. *At the end of Algorithm 2, $T_p = max_{1 \le k \le K} T_k$.*

PROOF. After the last event e is processed, an artificial envent z_ι is inserted in Q_ι (cf., line 11). Since $T = \infty$ at line 12, θ is true before all nonartificial events are processed in lines 14–18 (note that e is the last nonartificial event inserted into the queues, and all nonartificial events in the queues have timestamps no larger than e). After all nonartificial events are processed, $\Phi = \{z_k, 1 \le k \le K\}$, and θ is false. From Theorem 2, it is apparent that at the end of Algorithm 2, T_k is the minimum time to complete all events scheduled for all processes $j \in P_k$ based on Policy II. From (3), $T_p = \max_{1 \le k \le K} T_k$. □

5. A CRITICAL PATH ANALYZER BASED ON POLICY III

This section proposes a critical path analyzer (referred to as Algorithm 3; cf., Figure 8) for Policy III. Algorithm 3 is almost identical to Algorithm 2, except that line 15 in Algorithm 2 is replaced by lines 15a and 15b in Algorithm 3. In the remainder of this section we show how to select the processor l and the event e^* at lines 15a and 15b.

Let t be the time when processor k completes the execution of an event in the optimal parallel simulation under Policy III. Consider $\Phi_k(t)$. Let

$$\Upsilon_k(t) = \{e \mid e \text{ arrives at processor } k \text{ earlier than } t,\ e \in \Phi_k(t)\}.$$

Under Policy III, after time t, the next event to be executed at processor k is

$$\epsilon_k(t) = \begin{cases} \sigma_k(t) & \text{if } \Upsilon_k(t) = \emptyset \\ e & \text{if } \Upsilon_k(t) \neq \emptyset \text{ and } e.ts = \min\limits_{\forall e_{(j)} \in \Upsilon_k(t)} e_{(j)}.ts. \end{cases}$$

Note that if $\Upsilon_k(t) = \emptyset$, then for every $e_{(j)} \in \Phi_k(t)$, $e_{(j)}$ arrives at processor k later than t, and Policy III is identical to Policy II.

Part of the correctness proof for Algorithm 3 is similar to that for Algorithm 2. We borrow some results directly form Theorem 2, without formal proofs.

Consider Algorithm 3. From an inductive proof similar to the one for Hypothesis (a) in Theorem 2, we have:

LEMMA 8. *If $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3, then $T_k = max_{j \in P_k} \bar{\tau}(e_j^-)$, and $\Phi_k^* = \Phi_k(T_k)$.*

From Lemma 8 and from a proof similar to the one for Lemma 5, we have:

LEMMA 9. *Suppose that $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3. For all $j \in P_k$, if $e_{(j)}.\alpha < \infty$ then (i) $e_{(j)}.\alpha$ is the time when $e_{(j)}$ arrives at processor k, and (ii) $\tau(e_{(j)}) \geq max(e_{(j)}.\alpha, T_k)$.*

Consider a processor ξ, chosen as follows.

Definition 4. Suppose that $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3. Let

$$\Theta_k = \min_{\forall j \in P_k} \left[\max(e_{(j)}.\alpha, T_k)\right].$$

Let processor ξ be the processor such that $\Theta_\xi = \min_{1 \leq k \leq K} \Theta_k$.

Now we show how to select the next event to be executed at processor ξ after time T_ξ under Policy III. Let

$$\Upsilon_\xi^* = \left\{e \mid e.\alpha \leq T_\xi, \forall e \in \Phi_\xi^*\right\}.$$

We have the following lemma.

LEMMA 10. *If $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3, then (i) there exist at least one $e \in \Phi_\xi^*$ such that $e.\alpha < \infty$, and (ii) $\Upsilon_\xi^* = \Upsilon_\xi(T_\xi)$.*

PROOF. Two cases are considered.

—Suppose that $e_{(j)}.\alpha < \infty$ for all $j \in P_\xi$. Then $\Upsilon_\xi^* = \Upsilon_\xi(T_\xi)$ (from Lemmas 8 and 9).

```
    /* initialization */
1   for $k = 1$ to $K$ do $T_k = 0$;
2   for all $e$ pre-scheduled in the event list do $e.\alpha = 0$;
3   for $i = 1$ to $N$ do $Q_i = \emptyset$;
    /* the main loop */
    while not complete do
        /* Let $e$ be the next event to be executed */
4       execute $e$;
        /* Let $e.E$ be the set of events scheduled due to the execution of $e$ */
5       for all $e' \in e.E$ do
6           $e'.\alpha = \infty$;
7           insert $e'$ into the event list according to the timestamp order;
        end for
        /* Let $e$ be scheduled process $i$ */
8       $Q_i = Q_i \cup \{e\}$;
9       $T_s = T_s + \eta(e)$;
10      if $e$ is the last event to execute then
11          for $j = 1$ to $N$ do $Q_j = Q_j \cup \{z_j\}$; /* where $z_j.ts = \infty$ */
12          $T = \infty$;
13      else $T = e.ts$;
14      while $\theta(\Phi^*, T) = true$ do
15a         $l = \xi$; /* cf. Definition 4 */
15b         $e^* = \epsilon_l^*$; /* cf. (9) */
            /* Let $e^*$ be scheduled for process $m$ */
16          $T_l = \max(T_l, e^*.\alpha) + \eta(e^*)$;
17          for all $e' \in e^*.E$ do $e'.\alpha = T_l + \delta(e')$;
18          $Q_m = Q_m - \{e^*\}$;
        end while
18  end while
```

Fig. 8. Algorithm 3: A critical path analyzer for Policy III.

—Suppose that there exists $j \in P_\xi$ such that $e_{(j)}.\alpha = \infty$. Follow an argument similar to the proof for Lemma 7, we can show that there exists i such that $e_{(j)}$ cannot arrive at processor ξ before $\bar{\tau}(e_{(i)})$, where $e_{(i)}.\alpha < \infty$. Since $\max(e_{(i)}.\alpha, T_k) \geq \Theta_k \geq \Theta_\xi$ (definitions of Θ_k and ξ), there exist at least one $e \in \Phi_\xi^*$ such that $e.\alpha < \infty$, and (i) holds. Since $\bar{\tau}(e_{(i)}) > \max(e_{(i)}.\alpha, T_k) \geq \Theta_k$ (from Lemma 9 and the definition of Θ_k) and $\Theta_k \geq \Theta_\xi$ (definition of ξ), the arrival time of $e_{(j)}$ is later than $\Theta_\xi \geq T_\xi$. Thus, $\Upsilon_\xi^* = \Upsilon_\xi(T_\xi)$ and (ii) holds. □

THEOREM 4. *Suppose that $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3, and ξ is selected according to Definition 4. Let*

$$\epsilon_\xi^* = \begin{cases} e & \text{if } \Upsilon_\xi^* = \emptyset \textit{ and } e.\alpha = \min\limits_{e' \in \Phi_\xi^*} e'.\alpha \\ e & \text{if } \Upsilon_\xi^* \neq \emptyset \textit{ and } e.ts = \min\limits_{\forall e_{(j)} \in \Upsilon_\xi^*} e_{(j)}.ts. \end{cases} \tag{9}$$

Then $\epsilon_\xi^* = \epsilon_\xi(T_\xi)$, and $\epsilon_\xi^*.\alpha$ is the time when ϵ_ξ^* arrives at processor ξ.

PROOF. If $\Upsilon_\xi^* = \emptyset$, then $\Theta_\xi = \min_{e_{(i)} \in \Phi_\xi^*} e_{(i)}.\alpha$. Since $\Theta_k = \max[\min_{\forall e_{(j)} \in \Phi_k^*} e_{(j)}.\alpha, T_k] \geq \min_{\forall e_{(j)} \in \Phi_k^*} e_{(j)}.\alpha$, and $\Theta_\xi \leq \Theta_k$, we have $\min_{e_{(i)} \in \Phi_\xi^*} e_{(i)}.\alpha = \min_{e_{(j)} \in \Phi^*} e_{(j)}.\alpha$. In other words, $\epsilon_\xi^* = \sigma^*$. From a proof similar to the one for Hypothesis (c), Theorem 2, $\sigma^* = \sigma_\xi(T_\xi)$. This implies that $\epsilon_\xi^* = \epsilon_\xi(T_\xi)$. If $\Upsilon_\xi^* \neq \emptyset$, then directly from Lemma 10 and the definition of $\epsilon_k(t)$, $\epsilon_\xi^* = \epsilon_\xi(T_\xi)$. From Lemma 10 (i) and definitions of Υ_ξ^* and ϵ_ξ^*, $\epsilon_\xi^*.\alpha < \infty$, and is the time when ϵ_ξ^* arrives at processor ξ (Lemma 9). □

At line 15a in Algorithm 3, the processor $l = \xi$ is chosen based on Definition 4. At line 15b, the next event to be executed at processor l is selected based on (9). From Theorem 4 and a proof similar to the one for Theorem 3, Algorithm 3 is correct.

6. AN EXAMPLE OF CRITICAL PATH ANALYSIS

This section demonstrates an example of critical path analysis. We consider a simulation of N processes. Initially, μ events are prescheduled for every process. When an event e is executed at a process, an event e' is scheduled with timestamp $e'.ts = e.ts + \Delta_{ts}$, where Δ_{ts} is drawn from an exponential distribution. The event e' is scheduled for a process i, where i is randomly selected form the N processes with probability $1/N$. In other words, the network topology is fully connected. When $N > P$, the processes are assigned to processors in a balanced manner (e.g., if $N = 20$ and $P = 6$, then each of the first 4 processors are assigned 3 processes, and each of the remaining 2 processors is assigned 4 processes). We apply critical path analysis to this simulation application, and address the following issues.

Selection of event execution time η and communication cost δ. In many examples only a few types of events exist in the simulation. For each type of event, the execution times are fixed and can be easily determined through measurement. If the event execution time varies from one to another, then we need to sample the event execution times and determine an event execution time distribution to be used in critical path analysis. Sometimes an event execution time is too short to be measured. In such a case we may repeat the execution of an event several times, then find the average value. As a first approximation of critical path analysis, we may assume that η is a constant. The communication cost is usually obtained from the targeted architecture.

Number of events to be processes. A large number of events must be processed in critical path analysis before we can obtain a reliable speed-up figure. For steady state simulation, our experiments indicate that the reliable speed-up figure can be obtained only after the transient effect of the simulation disappears. Figure 9 illustrates the transient effect on speed up. We observe that a large P is more sensitive to the transient effect than a small P. The reason is that in the transient phase, the optimal parallel simulation experiences a bursty pattern of events available for execution, which has more effect on the instantaneous speed up of a larger P.

Load-balanced process assignment. Since the example we study is homogeneous, the issue of which process to be assigned to which processor is not

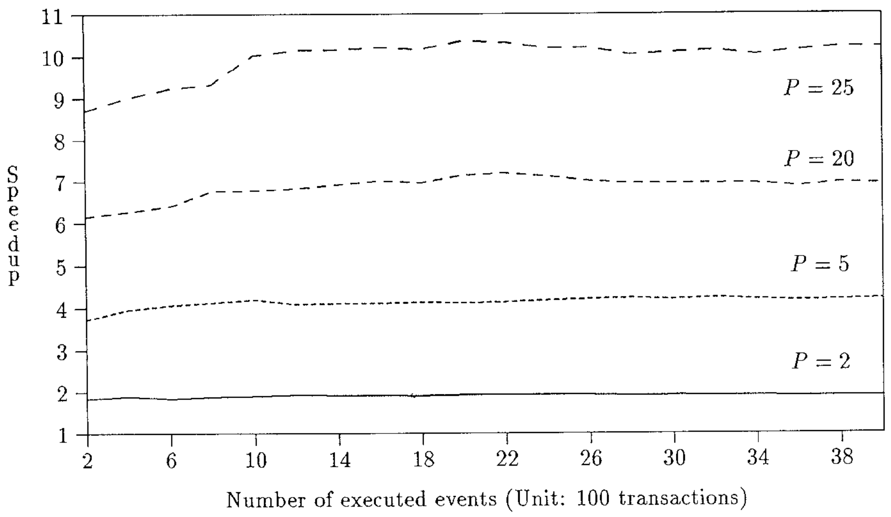

Fig. 9. Transient effect ($N = 20$, $\mu = 4$, $\eta = 1$, $\delta = 0$, Policy III).

important. It is more important to assign the same numbers of processes to every processor. Figure 10 shows the effect of load balancing. The circles represent the speed ups of the "balanced points" where P divides N (in such cases all processors are assigned the same numbers of processes). We observe that for a small P, the "distance" between two balanced points is short, and the unbalanced points in between are not significantly affected by the unbalanced load. For large P, it is most beneficial to add extra processors if it is close to the next balance point. Note that when $P = 11$, by adding 10 processors, the benefit is just the same as adding 2 extra processors when $P = 38$. In Figure 10, $\delta = 0$. The same phenomenon is observed when the δ value is small.

Interactions between the number of processors and the communication cost. The number of processors for parallel simulation must be selected to balance the effects of Constraints 1 and 2 in order to yield the maximum speed up. It is clear that if the communication cost δ is high, assigning extra processors to a parallel simulation may degrade speed up. Figure 11 shows how the interactions between P and δ affect speed up. The $\otimes$ symbols mark the maximum speed ups. We observe that when the communication cost is 20 times of an event execution time, the maximum speed up occurs when $P = 2$, and adding extra processors to the parallel simulation only degrades the performance.

Increasing the problem size. For a fixed number of processors, if the problem size (i.e., N) increase, the inherent parallelism also increases. Figure 12 indicates that if $N \gg P$, speed up of P can be expected. This observation supports Nicol's conclusion [19] that a simple parallel simulation protocol can yield good speed up if the problem size is sufficiently large. Note

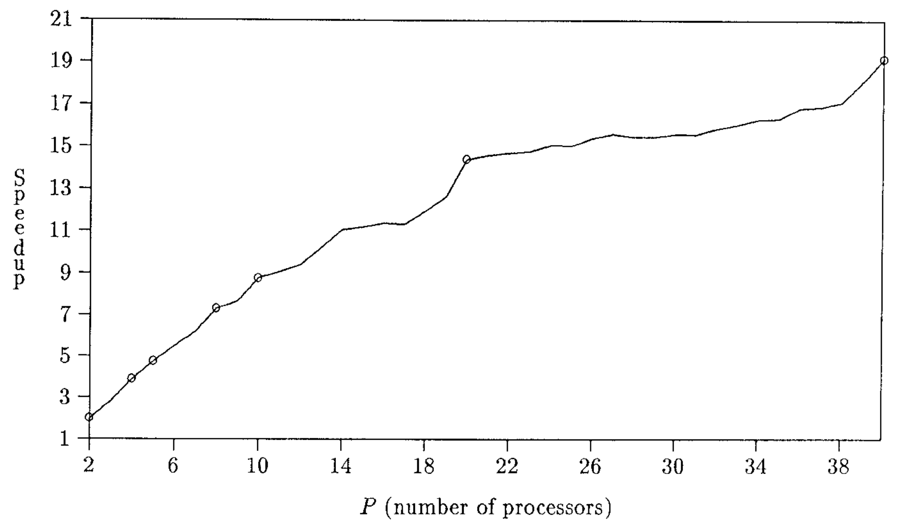

Fig. 10. Effect of load balancing on process assignment ($\eta = 1$, $\mu = 4$, $\delta = 0$, $N = 40$, Policy III).

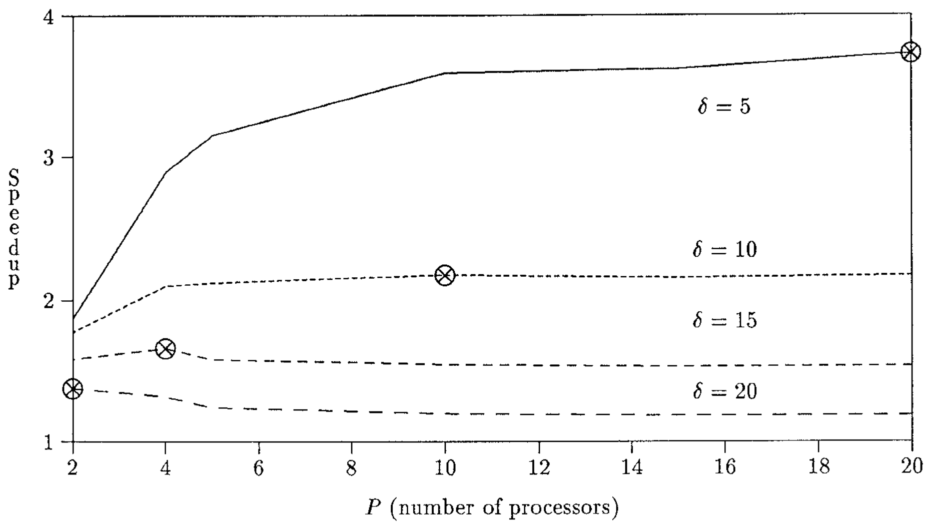

Fig. 11. Interactions between N and δ ($P = 20$, $\mu = 4$, $\eta = 1$, Policy III).

that when P is large, a much larger N is required to fully exploit processor power.

Increasing the problem size with fixed processes to processors ratio. Consider the case where both the problem size and the number of processors increase such that N/P is a constant. In this scenario the number of events

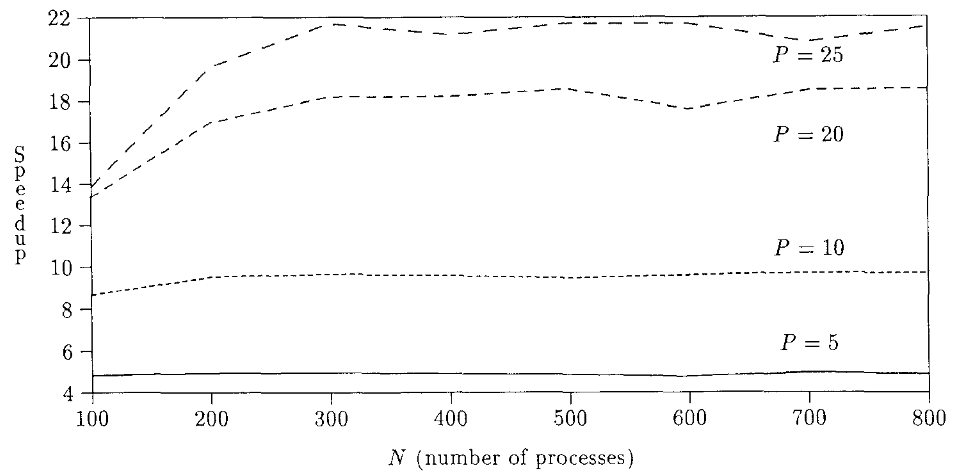

Fig. 12. Increasing the problem size ($\mu = 4$, $\eta = 1$, $\delta = 5$).

executed at a processor does not change statistically, but the number of processors to be communicated increases. Figure 13 shows that the speed up increases linearly if the problem size increases with a fixed N/P ratio. The above observation implies that the number of processors to be communicated with a processor does not affect the inherent parallelism. However, for conservative parallel simulation approaches such as the Chandy–Misra scheme, this type of parallelism is difficult to exploit [15].

Increasing message density. By increasing the number of messages per process, we increase the work load to the simulated system. In Figure 14 we observe that as μ increases, speed up increases and then (slowly) decreases. Similar phenomena were observed in [6] for conservative parallel simulation protocols.

7. CONCLUSIONS

Critical path analysis is a powerful tool for the study of inherent parallelism for parallel simulation. This paper proposed critical path analysis algorithms for three event-scheduling policies. The algorithms are easy to implement. They do not need event traces to compute parallel execution times. Instead, the algorithms are part of sequential simulation, and the critical path is computed as the events are processed in the sequential simulation. These algorithms can be integrated with sequential simulation problems written by users or be integrated with simulation languages. An example was given to show how useful information can be obtained from the proposed algorithms.

Since our algorithms can be easily inserted in sequential simulation programs, we can apply critical path analysis to existing sequential programs without difficulty. Then the results can be used to predict the performance of parallel simulation on similar applications. For example, the results of criti-

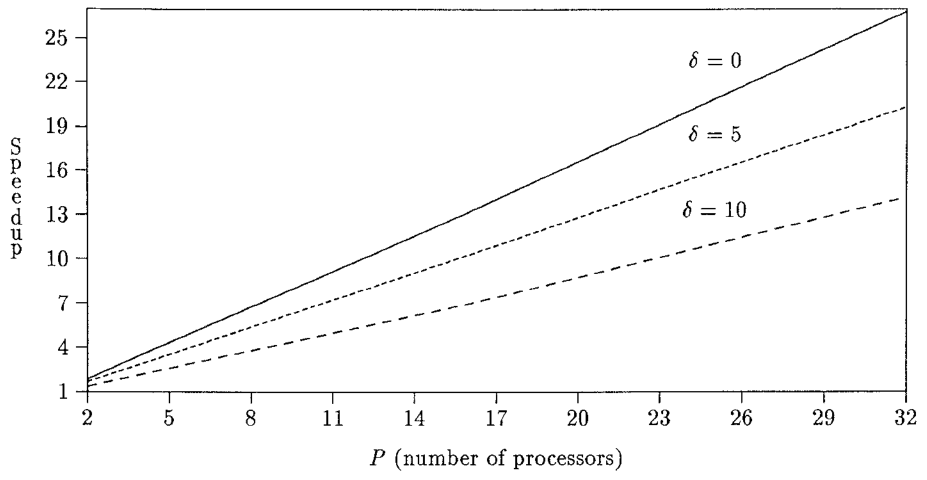

Fig. 13. Increasing the problem size with a fixed process to processor ratio ($N/P = 5$, $\eta = 1$, $\mu = 4$, Policy III).

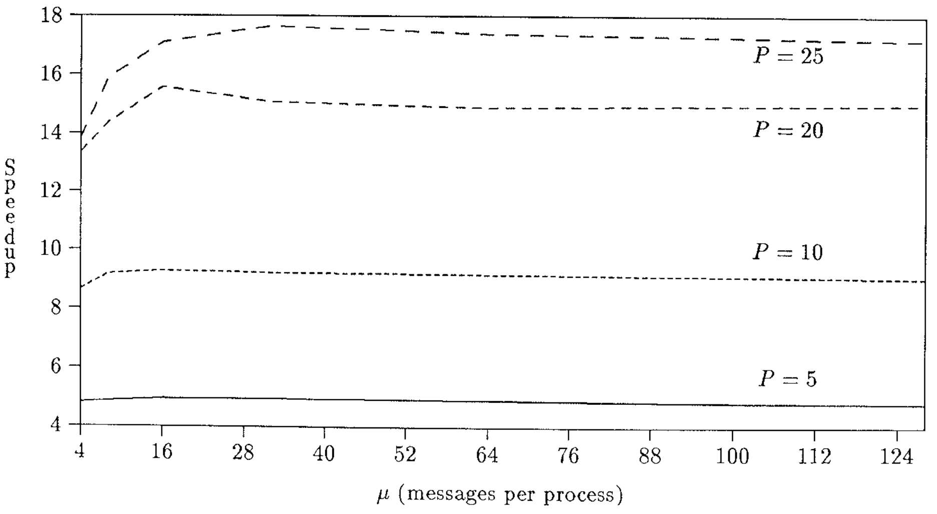

Fig. 14. Increasing message density ($N = 100$, $\eta = 1$, $\delta = 5$, Policy III).

cal path analysis on torus simulation [17] can be used to predict parallel simulation for mobile radio systems [14].

Critical path analysis can also be tailored to study a specific parallel simulation protocol. A simplified critical path analyzer for Chandy–Misra simulation of acyclic networks was used in [15]. One of our future directions is to construct critical path analysis algorithms for specific parallel simulation protocols such as Time Warp or Chandy–Misra.

APPENDIX: NOTATION

This section serves as an index to the notation. A notation followed by [**v**] represents a variable used in Algorithms 1, 2, or 3.

$\delta(e)$	the time to schedule the event e. If g_e and e are scheduled at different processes, then $\delta(e)$ represents the message-sending delay.
$\epsilon_k(t)$	the next event to be executed at processor k after time t under Policy III. $\epsilon_k(t) = \begin{cases} \sigma_k(t) & \text{if } \Upsilon_k(t) = \emptyset \\ e & \text{if } \Upsilon_k(t) \neq \emptyset \text{ and } e.ts = \min_{\forall e_{(j)} \in \Upsilon_k(t)} e_{(j)}.ts \end{cases}$
ϵ_k^*	[**v**] variable used in Algorithm 3. $\epsilon_k^* = \begin{cases} e & \text{if } \Upsilon_k^* = \emptyset \text{ and } e.\alpha = \min_{e' \in \Phi_k^*} e'.\alpha \\ e & \text{if } \Upsilon_k^* \neq \emptyset \text{ and } e.ts = \min_{\forall e_{(j)} \in \Upsilon_k^*} e_{(j)}.ts \end{cases}$
e^*	[**v**] a variable. In Algorithms 2, $e^* = \sigma^*$, and in Algorithm 3, $e^* = \epsilon_\xi^*$.
e_i^-	[**v**] the last event deleted from Q_j when θ is true at line 14 of Algorithms 2 and 3.
$e.ts$	the timestamp of event e.
$e.\alpha$	[**v**] a variable. If $e.\alpha < \infty$, then $e.\alpha$ is the time when e arrives at processor k, where e is scheduled for process $i \in P_k$.
$e.E$	[**v**] the set of events scheduled due to the execution of e. In other words, for all $e' \in e.E$, $e = g_{e'}$.
$\bar{e}_i(t)$	an event scheduled for process i such that for all events scheduled for process i, $\bar{e}_i(t)$ is the next event to be executed after time t.
$e_{(i)}$	[**v**] the event with the smallest timestamp in Q_i; i.e., $e_{(i)} \in Q_i$ and $e_{(i)}.ts = \min_{\forall e \in Q_i} e.ts$. $e_{(i)} = \phi$ if $Q_i = \emptyset$.
$\eta(e)$	the execution time for the event e.
g_e	an event such that its execution results in the scheduling of event e.
N	the number of processes.
P	the number of processors.
p_e	an event. Both events e and p_e are scheduled at the same process, and the execution of e_p is followed by the execution of e.
P_k	the set of indexes of processes mapped into processor k (i.e., $i \in P_k$ if process i is mapped to processor k).
$\Phi_k(t)$	$\Phi_k(t) = \{\bar{e}_j(t), j \in P_k\}$ is the set of events available for execution at processor k after time t.

Φ^*	[**v**]$\Phi^* = \{e_{(i)}, \forall i\} = \bigcup_{1 \le k \le K} \Phi_k^*$.
Φ_k^*	[**v**] $\Phi_k^* = \{e_{(i)}, \forall i \in P_k\}$.
ψ	$\psi = n$ represents the nth time when $\theta(\Phi^*, T)$ is true at line 14 of Algorithm 2.
Ψ_k	the set of events scheduled for process k in the simulation.
Q_i	[**v**] a queue maintained in Algorithms 2 and 3. If an event e is scheduled for process i and $\tau(e)$ has not been determined in the algorithm, then e is inserted in Q_i.
S	$S = \bigcup_{1 \le i \le N} S_i$ is the system state.
S_i	the state of process i.
σ^*	[**v**] a variable used in Algorithm 2. When θ is true, $\sigma^* = \sigma_k(T_k)$, where σ^* is executed at processor k. $$\sigma^* = \begin{cases} e & \text{if } e.\alpha = \min_{e' \in \Phi^*} e'.\alpha \\ \phi & \text{otherwise} \end{cases}$$
$\sigma_k(t)$	the next event to be executed at processor k after time t under Policy II.
$\tau(e)$	the time when e's execution starts in the optimal parallel simulation.
$\bar{\tau}(e)$	the time when e's execution completes in the optimal parallel simulation.
$\theta(\Phi^*, T)$	[**v**] a predicate used in Algorithms 2 and 3 that returns *true* if and only if (i) for all i, $e_{(i)} \ne \phi$, and (ii) for all i, $e_{(i)}.ts < T$. In other words, θ is a predicate to test if the timestamp of $e_{(i)}$, for all i, is less than T.
Θ_k	[**v**] $\Theta_k = \min_{\forall j \in P_k}[\max(e_{(j)}.\alpha, T_k)]$ when $\theta(\Phi^*, T)$ is true at line 14 in Algorithm 3.
T	[**v**] the timestamp of the event processed at lines 4–13 in Algorithms 2 and 3.
T_i	[**v**] In Algorithm 1, $T_i = \bar{\tau}(e)$ after line 1b is executed. In Algorithms 2 and 3, $T_i = \bar{\tau}(e^*)$ after line 16 is executed.
T_p	the execution time of optimal parallel simulation.
T_s	the sequential execution time.
$\Upsilon_k(t)$	$\Upsilon_k(t) = \{e \mid e \text{ arrives at processor } k \text{ earlier than } t, e \in \Phi_k(t)\}$. In other words, at time t, $\Upsilon_k(t)$ is the set of events already arrived at processor k, which are available for execution.
Υ_k^*	[**v**] a variable used in Algorithm 3. $\Upsilon_k^* = \{e \mid e.\alpha \le T_k, \forall e \Phi_k^*\}$
$\varphi(e)$	the iteration when e is processed in Algorithm 1.
ξ	[**v**] a variable used in Algorithm 3. ξ is selected such that $\Theta_\xi = \min_{1 \le k \le K} \Theta_k$.
z_i	[**v**] an artificial event used in Algorithms 2 and 3. z_i is inserted in Q_i for all i when the last event is processed in the algorithm.

The timestamp of z_i is ∞. These artificial events are only used to compute the parallel execution time of last events. They are not processed in the sequential simulation.

REFERENCES

1. BAILEY, M., AND LIN, Y.-B. Synchronization strategies for logic-level simulation. To appear in *Int. J. Comput. Simul.*, 1992.
2. BERRY, O. Performance evaluation of the Time Warp distributed simulation mechanism. Ph.D. dissertation, Univ. of Southern California, May 1986.
3. BERRY, O. Private communication. 1992.
4. BERRY, O., AND JEFFERSON, D. Critical path analysis of distributed simulation. In *Proceedings 1985 SCS Multiconference on Distributed Simulation* (Jan. 1985), pp. 57–60.
5. CHANDY, K. M., AND MISRA, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng. SE-5*, 5 (Sept. 1979), 440–452.
6. FUJIMOTO, R. M. Lookahead in parallel discrete event simulation. In *Proceedings International Conference on Parallel Processing*, Vol. III, 1988, pp. 34–41.
7. FUJIMOTO, R. M. Parallel discrete event simulation. *Commun. ACM 33*, 10 (Oct. 1990), 31–53.
8. HEIDELBERGER, P. Discrete event simulations and parallel processing: Statistical properties. *SIAM J. Sci. Stat. Comput. 9*, 6 (Nov. 1988), 1114–1132.
9. JEFFERSON, D. Virtual time. *ACM Trans. Program. Lang. Syst. 7*, (July 1985), 404–425.
10. LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM, 21*, (July 1978), 558–565.
11. LIN, Y.-B., AND LAZOWSKA, E. D. Optimality considerations for Time Warp parallel simulation. In *Proceedings 1990 SCS Multiconference on Distributed Simulation* (Jan. 1990), pp. 29–34.
12. LIN, Y.-B., AND LAZOWSKA, E. D. A time-division algorithm for parallel simulation *ACM Trans. Model. Comput. Simul. 1*, 1 (1991), 73–83.
13. LIN, Y.-B., AND LAZOWSKA, E. D. Effects of process scheduling in parallel simulation. *Int. J. Comput. Simul., 2*, 1 (1992).
14. LIN, Y.-B., AND MAK, V. K. On simulating a large-scale mobile radio system. Submitted for publication, 1992.
15. LIN, Y.-B., LAZOWSKA, E. D., AND BAER, J.-L. Conservative parallel simulation for systems with no lookahead. In *Proceedings 1990 SCS Multiconference on Distributed Simulation* (Jan. 1990), pp. 144–149.
16. LIVNY, M A study of parallelism in distributed simulation. In *Proceedings 1985 SCS Multiconference on Distributed Simulation* (Jan. 1985), pp. 94–98.
17. LUBACHEVSKY, B. Efficient distributed event-driven simulations of multiple-loop networks. *Commun. ACM 21*, 2 (March 1989).
18. NANDY, B., AND LOUCKS, W. M. An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers. In *Proceedings 6th Workshop on Parallel and Distributed Simulation* (Jan. 1992), pp. 139–146
19. NICOL, D. M. The cost of conservative synchronization in parallel discrete event simulations. Tech. Rep., Dept. of Computer Science, College of William and Mary, 1989.
20. REIHER, P., FUJIMOTO, R., BELLENOT, S., AND JEFFERSON, D. Cancellation strategies in optimistic execution systems. In *Proc. 1990 SCS Multiconference on Distributed Simulation* (Jan. 1990), pp. 112–121.
21. SOKOL, L. M., MUTCHLER, P. A., AND WEISSMAN, J. B. The role of event granularity in parallel simulation design. In *Proceedings 6th Workshop on Parallel and Distributed Simulation* (Jan. 1992), pp. 178–185.
22. WAGNER, D. B., AND LAZOWSKA, E. D. Parallel simulation of queueing networks: Limitations and potentials. In *Proceedings 1989 ACM SIGMETRICS and Performance '89 Conference* (1989), pp. 146–155.

Received March 1992; revised October 1992; accepted October 1992