Supercritical Speedup

David Jefferson
UCLA
Los Angeles, CA 90024

Peter Reiher
Jet Propulsion Laboratory
Pasadena, CA 91109

Abstract

The notions of the *critical path* of events and *critical time* of an event are key concepts in analyzing the performance of a parallel discrete event simulation. The highest critical time of any event in a simulation is a lower bound on the time it takes to execute a simulation using any conservative simulation mechanism, and is also a lower bound on the time taken by some optimistic methods. However, at least one optimistic mechanismis able to beat the critical path bound in a nontrivial way.

In this paper we make a systematic study of the meaning of the critical path in parallel simulation, and describe criteria that determine when a simulation is bounded by its length and when it is not. We show (again) that no conservative mechanism can beat the critical path, but that at least four known optimistic mechanisms are all capable of *supercritical speedup*. We give performance data for the JPL Time Warp Operating System showing two specially constructed applications using different methods to beat the critical path.

1 Introduction

The notion of the *critical path* of a parallel discrete event simulation has long been used as a standard of against which parallel simulation mechanisms have been compared. The length of the critical path in a simulation, calculated from the real time it takes to execute each event on a particular architecture and from the precedences induced among the events by causal dependencies, is a lower bound on the time it takes to execute a simulation using any conservative simulation mechanism on that architecture, and is also a lower bound on the time taken by some optimistic methods. It is well-known that at least one optimistic mechanism (Time Warp with lazy cancellation) is able to beat the critical path bound in a nontrivial way. Until recently this observation has not been studied in detail, and has been treated as something of a performance curiosity.

In this paper we make a systematic study of the meaning of the critical path in parallel simulation, and describe criteria that determine when a simulation mechanism is bounded by its length and when it is not. We show (again) that no conservative mechanism can beat the critical path bound, but we also show that at least four known optimistic mechanisms, Time Warp with lazy cancellation, Time Warp with lazy rollback, Time Warp with phase decomposition, and the Chandy-Sherman space-time family of mechanisms, all can do so. As a result, we say that those mechanisms are capable of *super-critical speedup*.

Supercritical speedup is, for now at least, of more theoretical than practical interest. We do not know of any cases where a realistic simulation of some useful model has ben shown to achieve a beginning-to-end supercritical speedup in practice. However, if an implementation contains one or more supercritical mechanisms, super-critical speedup should occasionally be realized for short periods during realistic simulations. We should thus expect that supercritical mechanisms might have an overall positive effect on a simulation's performance if their overhead is not too high.

Two of the supercritical mechanisms we discuss, lazy cancellation and phase decomposition, are both implemented in the JPL Time Warp Operating System (TWOS). In the last section of this paper we give performance data from two specially-constructed simulations to show that the critical path can be beaten in practice, at least for artificially constructed simulation models.

2 Critical Times and the Critical Path

The notion of the *critical time* of an event and the *critical path* of a simulation can be defined for any discrete event simulation that is statically decomposed into parallel processes that communicate by event message. For simplicity we will consider a rather well-structured model of parallel simulation, with several semantically troublesome cases excluded, though none of these exclusions have any affect on the conclusions of this paper.

Consider a simulation S decomposed into processes P_1 ... P_n, where each process P_i is in turn decomposed into m(i) events which, when ordered by increasing simulation time, we denote by $e_{i,1} \dots e_{i,m(i)}$. For any event e we denote by V(e) the virtual time (simulation time) at which e occurs. In our model of simulation an event occupies only a single instant of virtual time, although of course it takes a nonzero interval of real time to execute.

The execution of a simulation induces four causality relations on its events:

> Event e is the (immediate) *predecessor* of e', (or e' is the *successor* of e) if e and e' are in the same process P, and V(e) < V(e'), and there are no events e'' in P such that V(e) < V(e'') < V(e'). The predecessor of an event e (when defined) will be denoted pred(e).
>
> Event e is the *antecedent* of e' if e schedules e', i.e. if e posts the event notice for e', or (in distributed terminology) if e sends the event message that schedules e'. The antecedent of an event e (when defined) will be denoted ante(e).

Notice that e can be the antecedent of e' whether or not they are in the same process. For simplicity we assume that if e is the antecedent of e', then V(e) < V(e'), thereby excluding the case of "zero-delay" messages where V(e) = V(e'). We also assume that no event has more than one antecedent, i.e. that no two events are ever scheduled for the same virtual time at the same process.

> We define e <- e' (e *leads to* e') if either e is the predecessor of e' or e is the antecedent of e'.

Both the ante(e) and the predecessor of pred(e) are immediate causal ancestors of e, with a semantically symmetric relationship to e; but since in current technology there are different performance issues involved (e.g. the presence or absence of message communication delay, and the notion of same or different "process") we will observe the somewhat artificial distinction here. When the distinction is not relevant, we will use e <- e'

Finally we let the <- relation induce a partial ordering << on the set of events in the simulation.

> We define e<<e' (e *influences* e') if there exists a sequence of events $e=e_0, e_1, e_2, \dots, e_n=e'$ such that e_i <- e_{i+1} for all 0 <= i < n. (Since 0 < n we are excluding e<<e.)

These relations can be summarized in Figure 1, where the events of a simulation are shown in a *spacetime diagram*. Event e_{11} is the predecessor of e_{12}, and e_{12} is the predecessor of e_{13}, etc. Event e_{11} is the antecedent of e_{21}, which in turn is the antecedent of e_{33}. Event e_{31} is both the predecessor and antecedent of e_{32}. And any two distinct events e and e' have the property e<<e' if there is a path from e to e' in the diagram that proceeds only upward, and never downward. Hence e_{11}<<e_{33}, and e_{31}<<e_{13}.

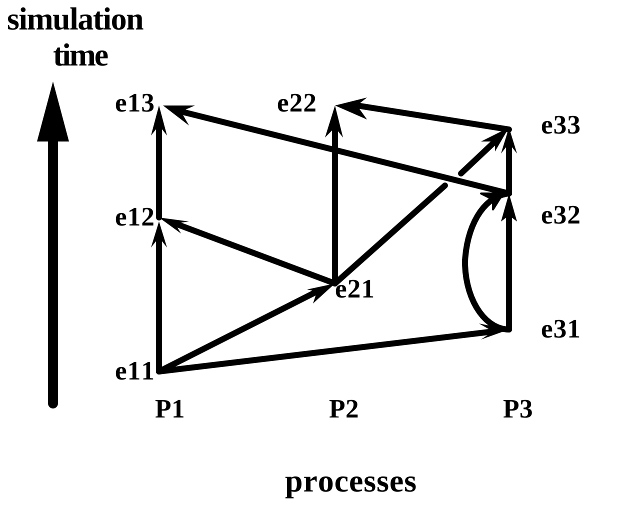

Figure 1: Precedence relations among events in the spacetime diagram of a simulation. Vertical arcs represent predecessor relationships; other arcs represent antecedent relationships.

Some events (e_{11}, e_{21}, and e_{31}) have no predecessors; we will call such events *initial*. Some initial events have no antecedents, e.g. e_{11}, and we call them *start events*. Nonstart events have exactly one antecedent, but an event may be the antecedent of any number of events, including zero.

The spacetime diagram of a simulation such as that of Figure 1 expresses apparent (but not always actual) precedence constraints on the order of event execution by showing the paths of (potential) information flow among events in the simulation. Any correct simulation mechanism must execute the simula-tion in such a way that it *appears* that the events were executed in a real time order consistent with these constraints; it must appear that if e<<e', then e finished execution before e' started. Of course, it is not actually necessary to obey those precedence relations strictly; it is only necessary that the output of the execution be *as if it were* executed that way.

Let us associate with each event e the amount of real time T(e) > 0 that it takes to execute e sequentially. We will call T the *timing function*. We assume that events are *atomic*, with no internal sequentiality, parallelism, synchronization, etc. Once started, an event e proceeds to completion without any delays, interrupts, context switches, rollbacks, etc., always taking exactly T(e) seconds. The assumption that events have no internal sequential structure implies that all messages sent by an event to schedule other events must be considered to be sent in the last instant of execution of that event, excluding the possibility that an event sends several messages, one at a time, at different points during its execution. Although most real parallel simulation mechanisms permit event messages to be sent sequentially during the execution of an event, properly modeling this feature would force us to consider an "event" to be a sequence of "microevents" each of which is either the start of the event, the sending of an event message, or the completion of the event. This would merely complicate our analysis without changing our results.

Once the simulation, the simulation method, and the timing function have been fixed, we can define start(e) as the moment at which the execution of event e begins, and complete(e) = start(e) + T(e) as the moment at which the last instruction of e is finished and commit(e) as the moment at which the execution of e is *committed*, i.e. the moment at which the result of e is considered irreversible, and all options for undoing any of its side-effects are relinquished. We now define the *critical time* for each event e recursively as

$$\text{crit}(e) = \max(\text{crit}(\text{ante}(e)), \text{crit}(\text{pred}(e))) + T(e),$$

where if ante(e) is not defined, then the term crit(ante(e)) is defined to be zero, and if pred(e) is not defined, then the term crit(pred(e)) is defined to be zero. Because they involve real time measurements, both T(e) and crit(e) are machine-dependent.

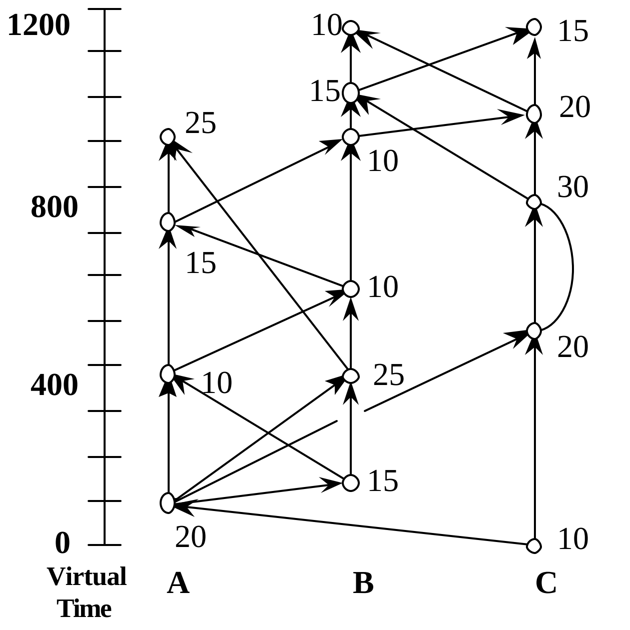

Figure 2: Simulation's computation times.

For a given simulation S and timing function T, there will a set of events, called *final events*, whose critical times are maximal. Critical paths through the simulation (not necessarily unique) are sequences of events from start events to terminal events defined most easily in reverse chronological order as follows:

1) All final events are on a critical path

2) If e is on a critical path, and crit(ante(e)) >= crit(pred(e)), then ante(e), if it exists, is on a critical path.

3) If e is on a critical path, and crit(pred(e)) >= crit(ante(e)), then pred(e), if it exists, is on a critical path.

As an example, Figure 2 shows a simulation whose events are annotated with the timing function T(e) that it takes to execute events. In Figure 3, we show the same simulation annotated with the critical times for each event, and display the critical path of the simulation (unique in this case) in bold. The event with the maximal critical time of 140 is the final event.

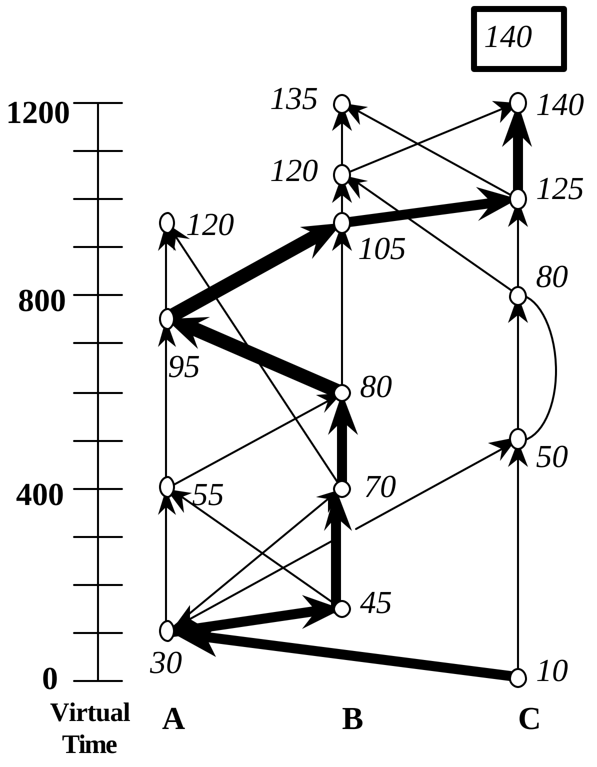

Figure 3: Critical times and critical path for the example simulation of Figure 2.

3 Conservative and Optimistic Mechanisms

A parallel simulation method is called *conservative* if it never uses any form of event undoing, abortion, or roll-back. Once an event e starts executing, it executes directly to completion in T(e) seconds, and is never undone or re-executed. An *optimistic* mechanism, by contrast, is one that is not conservative, i.e. it does use event rollback in at least some circumstances. Some events may be executed, and then rolled back, with their side-effects completely nullified. An optimistic mechanism can be thought of as *tentatively* executing an event and then deciding later whether to *commit* it, or *undo* it (roll it back). If an event execution ends up being committed, we will refer to it as a *real*, or *committed* event; if it is eventually rolled back we will refer to it as a *pseudo-event*. Hence, conservative mechanisms execute only real events, while optimistic mechanisms are characterized by the possibility of pseudo-events. Of course, all correct simulation mechanisms executing a particular simulation S decomposed into processes in a particular way will produce the same trace of committed events connected by the same antecedent, predecessor, <- and << relationships. But two optimistic mechanisms executing S (or the same one executing S twice) may differ on the set of pseudo-events. The relationships of antecedent, predecessor, <- and <<, and also the notions of critical time and critical path, will be interpreted as applying only to committed events; they do not apply to pseudoevents.

Even after completion of event e, an optimistic mechanism may retain the option to undo its side-effects until it can be ascertained that effects of all influencing events have been included. At some point that option is relinquished, and that moment is, by definition, the moment of commitment. Hence, for optimistic methods there is a distinction between the times of completion and commitment, and the relationship between them is start(e) < start(e) + T(e) = complete(e) <= commit(e), for all events e.

We define *elementary scheduling* to be any event scheduling mechanism such that for all *committed* events e and e', whenever e <- e', then complete(e) <= start(e'). Thus, an elementary scheduling mechanism enforces the principle that no committed event can start execution before its antecedent and predecessor committed events (if any) are completed. Notice that the definition of elementary scheduling says nothing about pseudoevents.

Theorem 1: For any simulation executed with elementary scheduling, crit(e) <= complete(e), for all committed events e.

Proof. By induction over committed events, using the <- relation. If e is a start event, then crit(e) = complete(e) trivially. If e is not a start event, then we will assume it has both an antecedent and a predecessor, i.e. there are two distinct committed events e' and e'' such that e' <- e and e'' <- e. (The case where there is only an antecedent can be handled similarly.) By definition

$$\begin{aligned}\text{crit}(e) &= \max(\text{crit}(e'), \text{crit}(e'')) + T(e),\\ &= \max(\text{crit}(e')+T(e), \text{crit}(e'')+T(e))\end{aligned}$$

By the induction hypothesis crit(e') <= complete(e') and crit(e'') <= complete(e''), so

$$\text{crit}(e) \le \max(\text{complete}(e')+T(e), \text{complete}(e'')+T(e))$$

Since e'<-e and e''<-e, we know by the hypothesis of elementary scheduling that complete(e') <= start(e) and complete(e'') <= start(e). Hence,

$$\begin{aligned}\text{crit}(e) &\le \max(\text{start}(e) + T(e), \text{start}(e) + T(e))\\ &= \text{start}(e) + T(e) = \text{complete}(e).\end{aligned}$$

end proof

Hence, we can conclude that no committed event can ever complete before its critical time unless the scheduling mechanism is nonelementary. If the arcs in the spacetime diagram are viewed as precedence constraints for purposes of event scheduling, then a simulation mechanism that can achieve supercritical speedup must violate at least one precedence constraint: there must exist two committed events e<- e' on the critical path that are scheduled in such a way that start(e') < complete(e).

Theorem 2: All conservative mechanisms are bound by the critical times of events, i.e. with a conservative mechanism, for all simulations S and all events e in S, crit(e) <= complete(e).

Proof: We show that all correct conservative simulation mechanisms must use elementary scheduling. Suppose a correct simulation mechanism does not use elementary scheduling, and that e and e' are two committed events in S such that e < e', with complete(e) > start(e'). We will show that any conservative mechanism might execute the simulation incorrectly.

Either e is the predecessor or the antecedent of e'. If e is the predecessor of e', then the last instruction of e might create a side-effect that affects the process state for event e'. If e' is started before e finishes, then it cannot execute in the context of the exact state produced by e; the last instruction of e might produce a state change upon which e' depends. Hence e' might execute incorrectly.

Likewise, if e is the antecedent of e', then any mechanism that would start to execute e' before finishing e must in effect be "guessing" that e' will be scheduled at the end of event e, and also "guessing" what the parameters from e to e' would be. (Recall our assumption that e can only send the event message to schedule e' at the very end of e's execution.) Since those guesses might be wrong, the simulation might be incorrect.
End proof

4 Supercritical Simulation Mechanisms

In the following analysis we will take Time Warp with aggressive cancellation and the Cancelback Protocol as our "standard" Time Warp mechanism denoted by TW. Other Time Warp mechanisms will be considered as variations on TW.

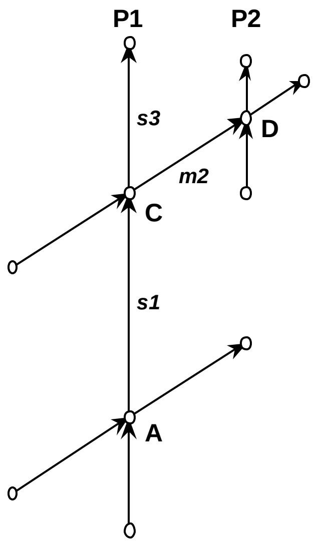

Figure 4a

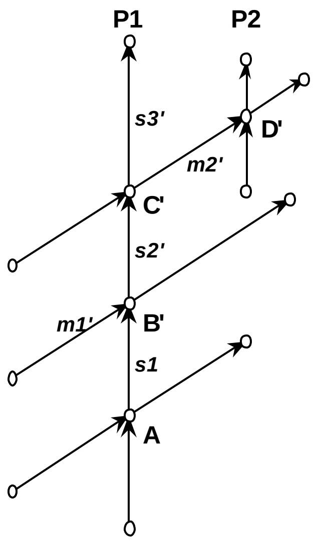

Figure 4b

From Theorem 2 we know that any simulation mechanism that is able to achieve supercritical speedup *must* be optimistic. But the converse is not true; not all optimistic mechanisms can be supercritical. For example, in our standard Time Warp, TW, if we extend our notation momentarily to apply to pseudoevents, we can see that for any pair of (pseudo)events e <- e', if event e is rolled back, then e' is also rolled back. Hence, if we confine our attention only to *committed* event executions, ignoring pseudo-event executions, we can see that for any pair e <- e', even though TW may execute e and/or e' more than once it always completes e for the last time (the committed execution) before starting e' for the last time. Hence, TW's scheduling is elementary; even though it is optimistic, it cannot achieve supercritical speedup.

In this section we present the main theoretical results of our paper: that four known optimistic mechanisms, Time Warp with lazy cancellation ($TW_{lazy\text{-}can}$), Time Warp with lazy rollback ($TW_{lazy\text{-}roll}$), Time Warp with phase decomposition (TW_{phase}), and the Chandy-Sherman space-time family of mechanisms (SpaceTime) are all capable of supercritical speedup. Three of these mechanisms are Time Warp variants, and are not mutually exclusive; in principle all three variations can be combined into a single Time Warp variant that could be supercritical more often than any of the individual variants. Of these results, only the one for $TW_{lazy\text{-}can}$

has been specifically noted. The fact that the possibility of supercritical speedup is so common among optimistic mechanisms suggests that there may be a deeper significance to optimism than has heretofore been understood.

For a simulation mechanism to produce supercritical speedup it is essential that there be at least one occasion when there are two committed events e and e' such that e<-e' and complete(e) > start(e'). We begin our discussion of supercritical mechanisms by examining Figures 4a and 4b, which show two stages in an optimistic execution of a fragment of the critical path of a simulation involving two processes P1 and P2. Figure 4a shows the apparent relationship among events as it has been computed up to a certain moment in real time by an optimistic mechanism. Events A and C are apparently successive events in process P1, with C scheduling event D in process P2. At the moment shown in this snapshot event A and pseudoevents C and D have all executed, and the processes P1 and P2 have proceeded ahead to higher simulation times.

Figure 4b shows the spacetime diagram describing the true behavior of the same simulation as reached by the optimistic execution some time later. All events, states, and messages in Figure 4b are committed. The differences between Fig. 4a and 4b all derive from the fact that message m1' requesting event B' has (finally) arrived at process P1 at a simulation time between that of A and C. The state s1 that had been input to C has now been used by event B', producing a new state s2'; event C, which had s1 as input to it, now has s2' as input, and since that presumably makes a difference, C has been relabeled as C'. The message m2 that C sent to schedule event D may have been affected as well, so m2 and D have been relabeled m2' and D' to indicate the possible difference. At the moment shown in the snapshot in Fig. 4b we assume that events A, B', C', and D' have all completed and have been committed, and processes P1 and P2 have executed ahead to higher simulation times, never to return to revise this portion of the computation again.

As mentioned before, for a simulation mechanism to produce supercritical speedup it is essential that there be at least one situation in which there are two committed events e and e' on the critical path such that e<-e' and complete(e) > start(e'). Let us now suppose that in Figure 4b, events A, B', C', and D' are all along a critical path, and examine some of the known optimistic execution mechanisms in such cases.

4.1 Time Warp With Lazy Cancellation

Let us denote by $TW_{lazy\text{-}can}$ our standard Time Warp mechanism TW but with lazy cancellation substituted for aggressive cancellation. The fact that $TW_{lazy\text{-}can}$ can achieve supercritical speedup was first reported in [1] and studied further in [2], [3], [4], [5], and [6]. For completeness we repeat that result here with a different proof.

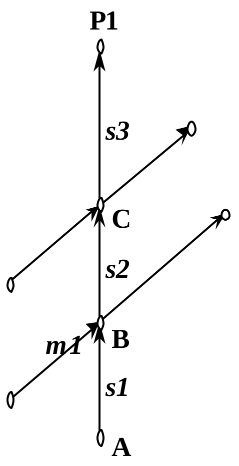

Figure 5a

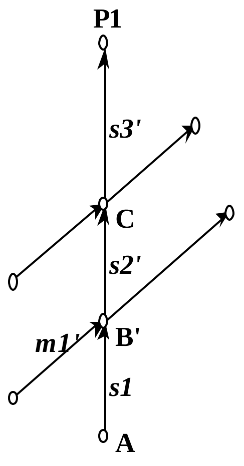

Figure 5b

Suppose in Figs. 4a and 4b that m2 = m2', so that event C operating from state s1 sends exactly the same message to process P2 that C' does from state s2'. Then under $TW_{lazy\text{-}can}$ the following sequence of events might occur to take the simulation from the situation in Fig. 4a to that of Fig 4b.

1. Message m1' arrives, causing process P1 to roll back to state s1.

2. Event B' executes, producing state s2'.

3. Event C' executes, producing output message m2'.

4. The lazy cancellation mechanism notes that m2' = m2, and suppresses its transmission to P2. As a result, event D', which is the same as D, is not re-executed, and has a completion time before the start of C'.

As a result of this analysis we have identified two critical path events C'<-D', such that start(D') < complete(D') < start(C') < complete(C').

4.2 Time Warp With Lazy Rollback

Let us denote by $TW_{lazy\text{-}roll}$ our standard Time Warp mechanism with the lazy rollback mechanism added. $TW_{lazy\text{-}roll}$ does not necessarily include lazy cancellation; the two are independent mechanisms. Lazy rollback (also called lazy reevaluation and jump forward) was first described, implemented and studied by Darrin West in [7]. It is perhaps most easily described as the state-message dual of lazy cancellation, i.e. it does for states what lazy cancellation does for messages.

Figures Figs. 5a and 5b represent two snapshots of an optimistic execution similar to those in Figs. 4a and 4b. In Fig. 5a the computation has executed to the situation shown, but in Fig 5b a rollback has caused message m1 to be cancelled and replaced with m1', causing event B to be rolled back and re-executed as B'.

Assume that events A, B', and C are all on the critical path of the simulation, that the virtual time of B' is the same as the of B, and that s2 = s2'. Then under $TW_{lazy\text{-}roll}$ the following sequence of events might occur to take the simulation from the situation in Fig. 5a to that of Fig 5b.

1. Message m1' arrives, causing process P1 to rollback to state s1.

2. Event B' executes, producing state s2'.

3. The lazy rollback mechanism notices that s2' = s2, and suppresses the re-execution of C, which was already correctly executed.

As a result of this analysis we can see that committed event C is completed before committed event B', and we have identified two critical path events B'<-C, such that start(C) < complete(C) < start(B') < complete(B').

4.3 Time Warp With Phase Decomposition

Let us denote by TW_{phase} our standard Time Warp mechanism with the addition of phase decomposition (but without either lazy cancellation or lazy rollback, though it is compatible with both). Time Warp with phase decomposition was first described in [8], and is a rather dramatic departure from other Time Warp mechanisms. Previously the fundamental unit into which simulations were decomposed was the process (or object). But with this mechanism a process can be further decomposed temporally into phases, where a phase is the execution of a process during an interval of simulation time. For example, a process P, might be decomposed into phases P[0..100), P[100..500), and P[500..8). The three phases might reside on different processors, and can even execute in parallel in some circumstances.

Whenever a phase such as P[0..100) completes execution, it produces a final state which must also be the first state of the next phase, P[100..500). Hence, part of the phase decomposition mechanism involves sending the final state of P[0..100) to the processor where P[100..500) resides and installing it as the first state of P[100..500). If at that moment P[100..500) has executed beyond time 500, it must roll back to 500. The feature that makes phase decomposition behave supercritically is this: if P[0..100) transmits a state s to P[100..500), and then later rolls back and re-executes its final event, but produces the identical final state s the second time as it did the first, then the second transmission of s to P[100..500) is suppressed, and P[100..500) need not roll back. Because this involves comparison of states for equality, phase decomposition is similar to lazy rollback permitted only at phase boundaries.

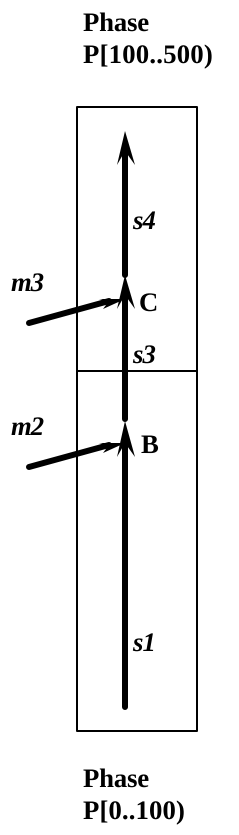

Figure 6a

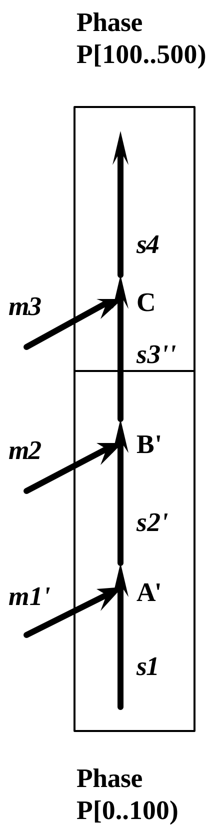

Figure 6b

Phase decomposition is illustrated in Figures 6a and 6b. Two phases of process A are shown, P[0..100) and P[100..500). Fig. 6a shows a snapshot of the optimistic execution in which phase P[0..100) has tentatively completed and transmitted state s3 to phase P[100..500), which has executed beyond event C. Figure 6b shows the final, true behavior of the simulation as reached later in the optimistic execution. The difference is that a message m1' has arrived for phase P[0..100), causing it to rollback and to re-execute event B (now called B') in state s2' instead of s1.

Suppose that events A', B', and C are on the critical path, and that for some reason event B' produces the same output state, s3, as event B did, i.e. s3' = s3. Then the phase decomposition mechanism might behave as follows:

1. Message m1' arrives, causing process P[0..100) to rollback to state s1 and execute event A', producing state s2.

2. Event B' executes, producing state s3'.

3. s3 is the final state of a phase, but the phase decomposition mechanism notices that s3' = s3, and suppresses the transmission of s3' to A[100..500), which was already correctly executed.

As a result, we can identify events B' and C on the critical path of the simulation such that B' $<\!\!\cdot$ C, but start(C) < complete(C) < start(B') < complete(B').

Supercritical speedup from phase decomposition is similar to supercritical speedup from lazy rollback, but has one other characteristic. Unlike lazy rollback, phase decomposition permits two events for the same object to be performed simultaneously on different nodes of a processor, thereby allowing supercritical speedup under some circumstances when lazy rollback could not achieve it.

4.4 Chandy-Sherman Space-Time Method

Chandy and Sherman invented a very general theory of parallel simulation called the Space-Time approach, first described in [9] but then studied further in [10] and [11]. They imagine that a simulation has a space axis (roughly corresponding to the parallel threads of activity in the model being simulated) and a temporal axis (simulation time). Their simulation method requires that the programmer partition his model, not into processes (which would correspond to vertical strips of space-time), but into arbitrarily-shaped regions of space-time. Each region of space-time is simulated in parallel and "communicates" with neighboring regions of spacetime, including those directly ahead and behind in simulation time. Since the behavior of a model in a region of space-time depends generally on the behavior of neighboring regions, Chandy and Sherman propose a general relaxation scheme that repeatedly simulates each region as long as its inputs from neighboring regions are changing. Much of the performance of this method is dependent on efficient techniques for detecting convergence.

The Space-Time approach is not so much a single mechanism as a family, so general that with suitable specialization it can emulate all other known parallel simulation mechanisms. For example, Time Warp with phase decomposition is similar in spirit to the Space-Time method restricted so that the regions of space-time must be rectangular, and one process wide in the spatial dimension. Hence, each of the examples given in Sections 4.1-4.3 could occur in principle in a Space-Time simulation, and thus it too is capable, at least in principle, of supercritical speedup.

5 Empirical results

Both lazy cancellation and phase decomposition are implemented in the Time Warp Operating System (TWOS) that runs at JPL on and 84-node BBN GP1000 [12]. In this section we describe two artificial benchmarks that achieve supercritical speedup under TWOS, and give the performance measurements we have made to demonstrate it.

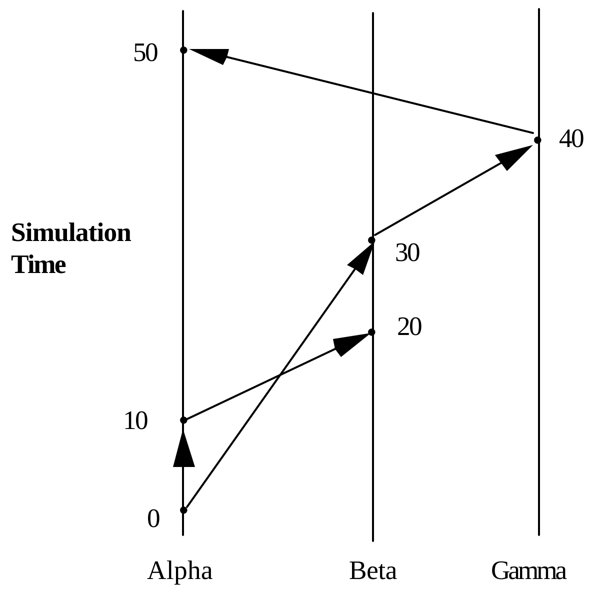

Figure 7: LazyCrit's Basic Cycle

The Lazycrit application achieves supercritical speedup through lazy cancellation. It consists of three objects that repeat a cycle of events until the cutoff time is reached. Figure 7 shows one cycle of this simulation. In this diagram, each arrow indicates a message sent by an object. Object Alpha starts out with an event at time 0. This event causes two messages to be sent, one to itself at time 10, another to Beta at time 30. The second event at Alpha, at time 10, causes one message to be sent to Beta, at time 30. Beta's event at time 20 sends no messages, and its event at time 30 sends a message to Gamma at time 40. Gamma sends a message to Alpha at time 50, restarting the cycle at a higher set of virtual times.

With appropriate delay loops in each event, Beta will execute the event at time 30 and send the message to Gamma earlier in real time than Alpha sends the message for time 20 to Beta. When that message does arrive at Beta, Beta rolls back and executes at time 20, then re-executes at time 30. However, the message sent to Gamma is precisely the same regardless of whether the event at time 20 had been performed or not, so lazy cancellation does not resend that message. Gamma executes its event at time 40 in parallel with Beta's execution, despite Beta's event preceding Gamma's on the critical path. This violates the rules of elementary scheduling, and permits the overall simulation to obtain supercritical speedup.

The actual value of the speedup depends on the lengths of the delay loops in the various events. For the following speedups, the delay loops in the events at time 0 and 30 took 2 milliseconds, the delay loops for the events at time 10 and 20 took 20 milliseconds, and the delay loop for the event at time 40 took 40 milliseconds. For this set of delay loop values, the critical path speedup for Lazycrit was 1.39, while Time Warp version 2.4.1 obtained a speedup of 1.47. The pattern of events shown in figure 7 was repeated 750 times for the speedups just quoted.

A second TWOS application achieves supercritical speedup through temporal decomposition. This application, called Cassandra, was described in [8]. Briefly, it consists of ten objects that send messages to themselves and each other. At the beginning of the simulation, every Cassandra object sends itself a flock of messages for all other integral simulation times up to the end of the simulation. Each event caused by the arrival of one of these messages causes the object to send a message to every other Cassandra object one time unit in the future, until the simulation end time is reached. The states of the objects never change, and the result of handling any message can be determined without having handled any other message, including the rest of the messages that will arrive for the same simulation time as this event.

Cassandra is essentially a time-stepped simulation run on an event driven simulation engine, with each of its ten objects performing an event at each integral simulation time step. Leaving aside some initialization and termination events, Cassandra is perfectly parallel without temporal decomposition. Cassandra has a critical path speedup of 9.2 for ten objects, indicating that every object can execute in parallel with every other object almost all of the time.

However, since the results of any event do not depend on anything other than the simulation time of that event and the identity of the object running it, temporal decomposition allows different phases of the same object to run simultaneously. Once a phase has a state to work with, it can correctly process any event for which at least one input message has arrived. On forty nodes, using temporal decomposition and splitting each object into four parts, an experimental version of TWOS achieved a speedup of 12.8.

6 Conclusions

There is as yet no final theory of the performance of parallel discrete event simulations; we do not even have a good nontrivial lower bounds, let alone an average-case performance methodology. In this paper we show that critical path theory, which at first glance seems to be a plausible candidate around which a lower bound theory might be built, does not provide a lower bound on the execution time for parallel simulations, at least as naively applied. Although it provides a lower bound to all conservative mechanisms, it does not generally apply to optimistic methods. It remains an open problem to give an alternate formalism that provides a lower bound for *all* parallel discrete event simulation mechanisms.

Acknowledgements

This work was funded by the U.S. Army Model Improvement Program (AMIP) Management Office (AMMO), NASA contract NAS7-918, Task Order RE-182, Amendment No. 239, ATZL-CAN-DO.

The authors thank Steven Bellenot, Mike Di Loreto, Brian Beckman, Fred Wieland, Phil Hontalas, and John Wedel for their work on TWOS and TWOS applications. We also thank Jack Tupman and Herb Younger for managerial support, and Harry Jones of AMMO, and John Shepard and Phil Lauer of CAA for sponsorship.

References

[1] O. Berry and D. Jefferson, "Critical Path Analysis of Distributed Simulation", *Proceedings of the 1985 SCS Conference on Distributed Simulation*, San Diego, January, 1985.

[2] O. Berry, "Performance Evaluation of the Time Warp Distributed Simulation Mechanism", Ph.D. thesis, Dept. of Computer Science, University of Southern California, May 1986.

[3] A. Gafni, "Space Management and Cancellation Mechanisms for Time Warp", Ph.D. Dissertation, Dept. of Computer Science, University of Southern California, TR-85-341, December 1985.

[4] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems", *Proceedings of the 1988 SCS Conference on Distributed Simulation*, Volume 19, No. 3, Society for Computer Simulation, February 1988.

[5] Y. Lin and E. Lazowska, "Optimality Considerations of 'Time Warp' Parallel Simulation", *Proceedings of the 1990 SCS Conference on Distributed Simulation*, Society for Computer Simulation, Volume 22, No. 2, San Diego, January 1990.

[6] P. Reiher, R. Fujimoto, S. Bellenot, and D. Jefferson, "Cancellation Strategies in Optimistic Execution Systems", Distributed Simulation Conference, San Diego, January, 1990.

[7] D. West, "Optimizing Time Warp: Lazy Rollback and Lazy Reevaluation", M.S. Thesis, Dept. of Computer Science, University of Calgary, January 1988.

[8] P. Reiher, S. Bellenot, and D. Jefferson, "Temporal Decomposition of Simulations under the Time Warp Operating System", Parallel and Distributed Simulation (PADS), San Diego, February, 1991.

[9] K. Chandy, and R. Sherman, "Space-Time and simulation", *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, Society for Computer Simulation, Volume 21, No. 2, Tampa, Fla., March 1989.

[10] R. Bagrodia, and W. Liao, “Maisie: A Language and Optimizing Environment for Distributed Simulation,” *Proceedings of the 1990 SCS Conference on Distributed Simulation,* Society for Computer Simulation, Volume 22, No. 2, San Diego, January 1990.

[11] R. Bagrodia, W. Liao, “Parallel Simulation of the Sharks World Problem”, Winter Simulation Conference, New Orleans, Dec. 1990.

[12] D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLoreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel, H. Younger and S. Bellenot, “Distributed Simulation and the Time Warp Operating System”, *11th Symposium on Operating Systems Principles (SOSP)*, Austin, Texas, November 1987.