Super-criticality revisited

Sudhir Srinivasan
Paul F. Reynolds, Jr.

Computer Science Report No. CS-93-29
Issued May 25, 1993
Revised November 8, 1994

Super-criticality revisited

Sudhir Srinivasan
Paul F. Reynolds, Jr.

Contact:
Sudhir Srinivasan
Department of Computer Science
Olsson Hall, University of Virginia
Charlottesville, VA 22903
Email: ss7a@uvacs.cs.Virginia.EDU

Abstract

Critical path analysis has been suggested as a technique for establishing a lower bound on the completion times of all parallel discrete event simulations (PDES). A protocol is super-critical if there is at least one simulation that can complete in less than the critical path time using that protocol. Previous studies have shown that several practical protocols are super-critical while others are not. We present a sufficient condition to demonstrate that a protocol is super-critical. Also, we show that a condition used in a previous study is not sufficient but necessary for super-criticality. It has been claimed that super-criticality requires independence of one or more messages (or states) on events in the logical past of those messages (states). We present an example which contradicts this claim and examine the implications of this contradiction on lower bounds.

Appeared in the *9th Workshop on Parallel and Distributed Simulation*

1 Introduction

One of the techniques that has been suggested to derive a theoretical lower bound on the completion time of all parallel discrete-event simulations (PDES) is critical path analysis. This application of critical path analysis is particularly interesting because of the somewhat counter-intuitive result that it is possible for certain simulations to complete in less than the critical path time, a phenomenon we call *super-critical speed.* We say a protocol is *super-critical* if it is possible for at least one simulation using that protocol to complete in super-critical time. There are several practical protocols that are super-critical and several that are not [JeRe91(6)]. In this paper, we re-examine two issues concerned with super-critical speed: (i) a sufficient condition to demonstrate a protocol is super-critical, and (ii) the requirement for super-criticality, of independence of messages or states on events earlier in logical time.

Berry and Jefferson [BeJe85(2)] applied critical path analysis to PDES and argued that the critical path time is a lower bound on the completion time. In [Berr86(1), JeRe91(6)] it was shown that certain variants of the Time Warp protocol [Jeff85(5)] are super-critical. In particular, [JeRe91(6)] presented a criterion for super-criticality and used it to show that four protocols were super-critical. These results showed that the critical path time is not a lower bound for all PDES's. Lin and Lazowska [LiLa91(8)] proposed a lower bound that applies to all PDES's but is a very loose one since it requires that each LP guess all of its computation correctly. These early analyses defined inter-event dependence based on the timestamps of events and messages. This scheme has the disadvantage of incorrectly assuming some pairs of events to be dependent when in fact there is no semantic dependence between them. Recently, Gunter [Gunt94a(3)] has proposed an enhanced definition of dependence which attempts to overcome this limitation. Based on this definition, he contends that independence is necessary for super-criticality and derives a new lower bound which is tighter than that of [LiLa91(8)].

This paper makes two contributions:

i) We present a sufficient condition for a protocol to be super-critical (recall that a super-critical protocol *allows* the possibility of super-critical speed; it does not guarantee it). We show that the condition used in [JeRe91(6)] is necessary for super-criticality but not sufficient. The condition we present has been used in [Gunt94a(3)] to prove that super-criticality requires independence but was presented without apparent support [Gunt94b(4)]. In a result developed independently we establish the truth of the sufficiency condition.

ii) We show, by example, that Gunter's enhanced definition of dependence is not sufficient to capture all forms of super-criticality. Specifically, super-critical speed is also possible when an LP guesses correctly a dependence on a message that it has not yet received. Thus the claim that super-criticality requires independence is invalidated. This insight suggests that irrespective of how accurately we are able to capture the semantic dependence of events, it is still possible to be super-critical. While Gunter's lower bound does not hold for simulations where LP's guess the *existence* of dependence, it does hold for simulations where LP's only guess the *lack* of dependence. Since all known aggressive protocols allow LP's to guess both the lack and the existence of dependence, it seems unlikely that critical path analysis can improve upon the lower bound of [LiLa91(8)].

2 Critical path analysis

We assume the PDES consists of a set of logical processes (LP's), P_1, P_2 ... P_n and each LP executes on its own processor*. Each P_i represents a sequence of simulated events. In the case of aggressive protocols, critical path analysis applies only to committed events. The timestamp of an event *e* is denoted by V(*e*). Each LP may send messages to other LP's as a result of executing events. In addition to simulation-specific information, a message contains a send-time which equals the logical clock of the LP sending the message and a receive-time which is greater than or equal to its send-time. When the message is received, the receiving LP schedules an event with timestamp equal to the receive-time of the message. This model is called the *message-initiating* model. With this model, we define the following two relations on events:

- Event *e* is the *predecessor* of event *e'* (or *e'* is the *successor* of *e*) if: (i) *e* and *e'* are executed by the same P_i, (ii) V(*e*) < V(*e'*) and (iii) there is no other event *e"* in P_i such that V(*e*) < V(*e"*) < V(*e'*). We denote the predecessor of an event *e* as *pred*(*e*).
- Event *e* is the *antecedent* of event *e'* if the execution of *e* causes a message to be sent which schedules *e'*. Note *e* and *e'* may be executed by the same LP. We denote the antecedent of an event *e* as *ante*(*e*).

If defined, *pred*(*e*) and *ante*(*e*) are unique for a given event *e*. Also, an event can be the predecessor of at most one event but the antecedent of more than one event. We define the relation $\rightarrow$ as $e \rightarrow e'$ (*e* immediately affects *e'*) if either $e = pred(e')$ or $e = ante(e')$. Finally, the transitive closure $\Rightarrow$ of the relation $\rightarrow$ induces a partial ordering on the events in the simulation as follows: $e \Rightarrow e'$ (*e* influences *e'*) if there

* Obviously, the lower bound may change if multiple LP's execute on a processor since more than one LP may have an executable event at the same time. The issue of optimal scheduling when multiple LP's are assigned to a single processor is addressed in [Lin92(7)].

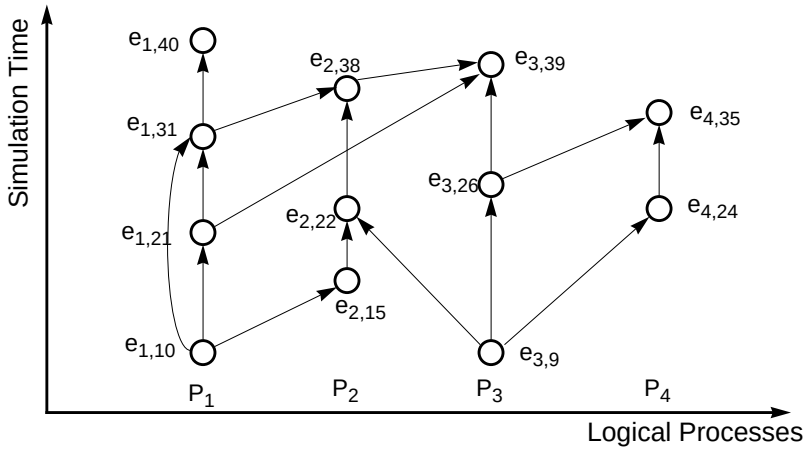


Figure 1 - Space-time diagram

exists a sequence of events $e = e_{(0)}, e_{(1)}, \ldots e_{(n)} = e'$ such that $e_{(i)} \rightarrow e_{(i+1)}$ for all $0 \leq i < n$. A particular parallel discrete event simulation *run* (an execution of a simulation program with a given set of input values) may be represented using a *space-time* diagram as in Figure 1. The points in this 2-dimensional plane represent events that were executed in the simulation run. The two co-ordinates of each event are the LP at which it was executed (the space co-ordinate) and its logical timestamp (the time co-ordinate). Arrows are used to represent the $\rightarrow$ relation among events. For example, $e_{1,10}$ is the predecessor of $e_{1,21}$ and the antecedent of $e_{2,15}$. Events that have no predecessors are called *initial* events. These are the first events to be executed at any LP. Further, initial events that have no antecedents (e.g., $e_{1,10}$ and $e_{3,9}$) are called *start* events.

With each event e, we associate an amount of real time required to execute that event, T(e). Also, for the sake of simplicity we ignore all overheads associated with inter-LP communication. These overheads can be incorporated easily into critical path analysis if required. If *start(e)* is defined to be the real time at which the execution of event e is started, then *complete(e)* = *start(e)* + T(e) is the time at which the execution of event e completes. The *critical time* crit (e), of each event e is defined as follows:

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

where the simulation is assumed to start at real time zero and crit (ante (e)) and crit (pred (e)) are defined to be zero if ante (e) and pred (e) are not defined respectively. crit (e) is the earliest time the event e can complete execution under the assumption that no dependences are violated. Consequently, the largest value of crit (e) among all of the events in the simulation run will give us the lower bound on the completion time of the simulation run under the same assumption. Events that have this maximum critical time are called

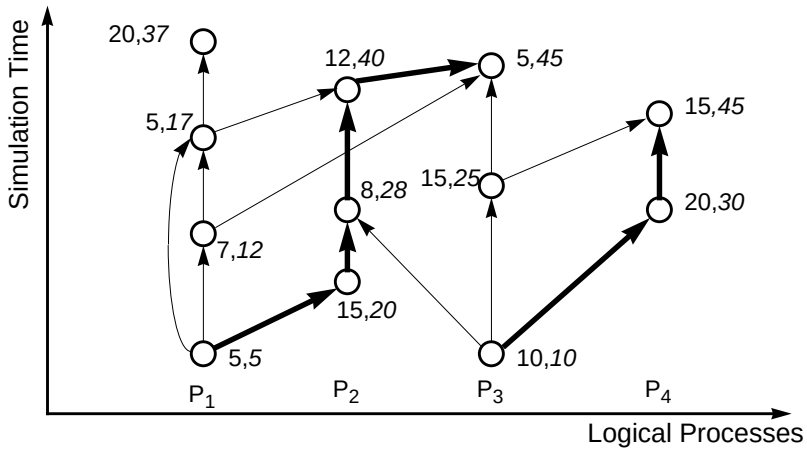


Figure 2 - Critical paths

final events. An example of the computation of critical times is shown in Figure 2. The first number beside each event *e* is T(*e*) and the second number is crit (*e*).

A *critical path* is a path from a start event to a final event defined as follows:

i) Every final event is on a separate critical path.

ii) If *e* is on a critical path and crit (ante (*e*)) ≤ crit (pred (*e*)) then if pred (*e*) exists, it is on the critical path.

iii) If *e* is on a critical path and crit (pred (*e*)) ≤ crit (ante (*e*)) then if ante (*e*) exists, it is on the critical path.

We have highlighted the two critical paths in Figure 2. The maximum value of crit (*e*) is called the *critical path time*.

3 When is a protocol super-critical?

The critical path time defines a lower bound on the completion time of a simulation under the assumption that events are actually executed in the order specified by the dependence relation ⇒. However, it is only required that the overall effect of the simulation be the same as if the events were executed in that order. Therefore, if an LP "guesses" correctly, it may execute certain events out of order while keeping the simulation accurate. By guessing correctly on the critical path, it is possible to complete the simulation in less than the critical path time. This phenomenon was called *super-critical speed-up* in [JeRe91(6)] but we refer to it as *super-critical speed* since the critical path time is an absolute quantity. For a simulation run to be super-critical, the act of "guessing correctly" must occur on every critical path of the simulation.

It is important to note the distinction between a super-critical simulation run and a super-critical protocol. A super-critical run is a particular execution of a simulation (using some protocol) that completes

in super-critical time. A protocol is said to be super-critical if it is possible for a simulation run (at least one) using that protocol to be a super-critical run. By its very nature, a general PDES protocol cannot guarantee that LP's will guess correctly; it can only *enable* them to do so. Even so, it is desirable to be able to determine whether a protocol has this capability or not. To do so, we require a (sufficient) condition for super-criticality, **SC** such that if a protocol allows **SC** to be true in a simulation run, then the protocol permits the simulation run to be super-critical (i.e. the protocol is super-critical). We may then prove protocols to be super-critical by showing that they allow **SC** to be true in at least one simulation run. In [JeRe91(6)], the authors present one such condition. We show that their condition is merely necessary (i.e. super-critical protocols will satisfy it) and present the actual sufficient condition.

3.1 Condition for enabling super-critical speed

Recall that for any event *e*, the following are defined:

- start (*e*) : the real time at which the execution of *e* commences.
- complete (*e*) : the real time at which the execution of *e* completes.
- crit (*e*) : MAX {crit (pred (*e*)), crit (ante (*e*))} + T(*e*), which is the earliest time at which *e* can complete if no dependences are violated.

The super-criticality condition of [JeRe91(6)], which we call **C**, is stated below:

> **C**: There must be at least one pair of events *e* and *e'* on every critical path such that $e \rightarrow$ e' and complete (*e*) > start (*e'*)

We show by construction that the fact that **C** is true for a simulation run does not imply the run can complete in super-critical time.

Consider conceptually extracting a critical path and laying it on the real-time line as shown in Figure 3a. Each block of time (shaded rectangle) corresponds to the execution of an event on the critical path. Dots indicate that intermediate events have been left out for brevity. Note, since we have ignored the cost of inter-LP communication in this paper, we do not allocate any real time between events. Now consider the critical path of a particular simulation run in which the condition **C** is satisfied. This condition could have been satisfied in one of the two ways shown in Figure 3. In Figure 3b, **C** is satisfied because the execution of $e_{(i+1)}$ commenced on time (at crit ($e_{(i)}$)) but the execution of $e_{(i)}$ completed late (after crit ($e_{(i)}$)). Note $e_{(i)}$ may have completed late due to several reasons. For instance, it may be that just prior to this execution of $e_{(i)}$, the LP executing $e_{(i)}$ may have guessed incorrectly and consequently rolled back, causing

the delay of $e_{(i)}$. We have depicted this by a single delay (gap in the shading) at some point earlier than the execution of $e_{(i)}$. The LP executing $e_{(i+1)}$ guessed correctly so that the execution of $e_{(i+1)}$ was not rolled back when $e_{(i)}$ completed after crit ($e_{(i)}$). By only observing $e_{(i)}$ and $e_{(i+1)}$, the two events that satisfy **C**, we cannot conclude that the simulation can complete in super-critical time. In Figure 3c, **C** is satisfied because the execution of $e_{(i+1)}$ commenced early (before crit ($e_{(i)}$)) while the execution of $e_{(i)}$ was on time (completing at crit ($e_{(i)}$)). This occurs because the LP executing $e_{(i+1)}$ guessed correctly so that it is not rolled back when $e_{(i)}$ completes execution at crit ($e_{(i)}$). If every event on this critical path following $e_{(i+1)}$ is executed immediately after its preceding event completes (as depicted), the simulation will complete before crit ($e_{(n)}$), as shown (assuming this is the only critical path).

In [JeRe91(6)], the authors consider only the scenario of Figure 3c and not that of Figure 3b. By itself, condition **C** specifies nothing about the absolute completion times of *e* and *e'*, which are essential to super-critical speed since the maximum critical time is an absolute quantity. Indeed, the scenarios of Figure 3b and Figure 3c were generated simply by moving events along the real time line while maintaining their relative timing relations. Clearly, crit (*e*) is the quantity which must link the condition for super-critical speed with the absolute timings of events since it defines whether an event is early, on time or late. Therefore, we claim that the actual sufficient condition for a protocol to be super-critical is:

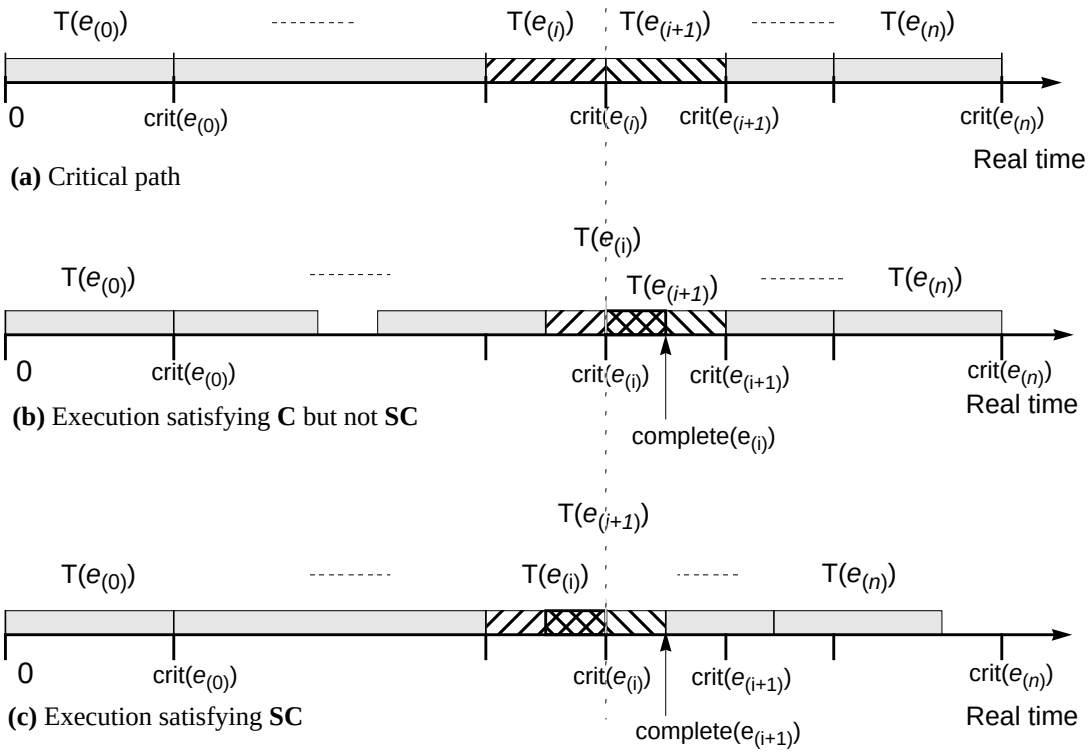


Figure 3 - Super-critical speed

SC:There must be at least one event *e* on every critical path such that

complete (*e*) < crit (*e*)

Note, this condition is satisfied in Figure 3c (complete ($e_{(i+1)}$) < crit ($e_{(i+1)}$)) but not in Figure 3b, thus distinguishing the two. The intuition behind this condition is very simple: at least one event on every critical path must complete before its critical time so that there is the possibility of all events following it on the critical path to complete before their critical times and thus for the simulation to complete before the maximum critical time. The key is that crit (*e*) (and not the starting or completion time of event *e* relative to its adjacent events on the critical path) defines the earliness or tardiness of event *e*. Thus we have shown by construction that the fact that a simulation run satisfies **C** does not imply that it can complete in super-critical time whereas the fact that a simulation run satisfies **SC** does imply that it can complete in super-critical time. Of course, even in the situation of Figure 3c, it is possible for the simulation to complete after crit ($e_{(n)}$), if at least one event following $e_{(i+1)}$ is sufficiently tardy.

If a protocol is super-critical, then by definition, there exists a simulation run using that protocol such that the final events on every critical path complete before their critical times. Trivially therefore, super-criticality implies that the protocol allows **SC** to hold for at least one simulation run. Thus **SC** is also a necessary condition for super-criticality.

Revisiting Figure 3c, if **SC** is true on a particular critical path, we can show by contradiction that **C** is true for $\hat{e}$ and *e* where *e* is the earliest event for which **SC** is true on the critical path and $\hat{e} \rightarrow e$. Thus, **SC** $\Rightarrow$ **C**. Since **SC** is necessary for super-criticality, **C** is also a necessary condition for a super-critical protocol.

In [JeRe91(6)], the authors use **C** to show that the four protocols:Time Warp with Lazy Cancellation, Time Warp with Lazy Reevaluation, Time Warp with Phase Decomposition and Space-Time Simulation are capable of super-critical speed. In the appendix to this report, we have shown that all of these protocols satisfy **SC** and therefore are indeed, super-critical.

It is possible to extend **SC** to a stronger condition which is sufficient to *guarantee* super-critical speed:

SCG: The final event on every critical path satisfies **SC**

Clearly, **SCG** $\Rightarrow$ **SC** $\Rightarrow$ **C**. No known protocol satisfies **SCG**.

Finally, a common misconception seems to be that a protocol that allows LP's to guess events is super-critical. A distinction must be made between all events and the ones that are actually committed. Time Warp with aggressive cancellation allows LP's to guess events but does not commit events which have started prior to the completion of previous events (i.e. it does not allow LP's to guess committed events). Accordingly, it has been established [LiLa91(8)] that Time Warp with aggressive cancellation is not super-critical. In words, this is what **C** claims: if a protocol allows LP's to guess committed events correctly, then it is super-critical. However, that is not sufficient. We have shown that the protocol must allow LP's to guess committed events correctly *and* before their critical times.

4 Super-criticality and independence

We make the following key observation:

Observation: There are two phenomena that may result in super-critical speed:

i) when LP's correctly guess that no messages will arrive that affect an event

ii) when LP's correctly guess the effect of one or more messages that have not yet arrived on an event

In the first case, super-critical speed is possible because the dependence relation $\rightarrow$ is defined based on timestamps of events (and messages). Under this definition, an event *e* at some P_i is dependent on a message *m* if *m* schedules an event *e'* at P_i such that $V(e') < V(e)$. However, it is possible that the execution of *e'* affects only a part of the state of P_i upon which the execution of *e* does not depend, i.e., the outcome of executing *e* is independent of whether *e'* is executed or not. Recognizing this intra-process concurrency, Gunter [Gunt94a(3)] formalizes the notion of independence of states and messages on events which we reproduce here for convenience:

Definition: A state (message), denoted S (*m*), created by event *e* is said to be independent of a set of events **E** if **E** is a subset of the set of all events **E'** such that $e \Rightarrow$ **E'** and the S (*m*) resulting when all of the events in the simulation are properly executed is the same as the S (*m*) when all of the events in **E** are not executed.

In particular, a message *m* is independent of an event *e* if the same message *m* is generated irrespective of whether *e* is executed or not. It follows that a message *m* is dependent on an event *e* if *m* is not independent of *e*. Using this enhanced definition of dependence, Gunter derives two results: (i) independence is necessary for super-critical speed, and (ii) a new definition of the critical path and hence a new lower bound which is tighter than that of [LiLa91(8)]. By capturing semantic dependences accurately, this new definition of

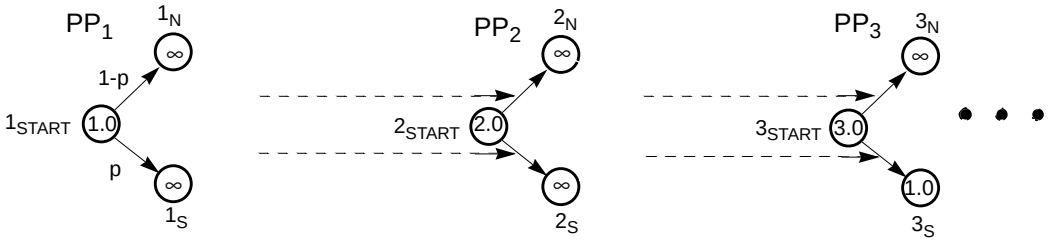


Figure 4 - A super-critical simulation without independence

dependence accounts for super-critical speed that occurs when LP's guess only the *lack* of dependence. However, it fails to capture super-critical speed that may occur when LP's guess the *existence* of dependence.

To understand how super-critical speed may occur when LP's guess the existence of dependence correctly, consider the physical system shown in Figure 4. The system consists of n physical processes, PP_i, each of which begins in a starting state i_{START}. At time 1.0, PP_1 transitions to state 1_S with probability p or to state 1_N with probability 1-p. At time 2.0, PP_2 transitions to state 2_N if PP_1 has transitioned to state 1_N earlier or to state 2_S if PP_1 has transitioned to state 1_S earlier. Note there is no default action for PP_2: it knows that PP_1 has transitioned earlier. Similarly, at time 3.0, PP_3 transitions to state 3_N if PP_2 has transitioned to state 2_N earlier or to state 3_S if PP_2 has transitioned to state 2_S earlier. The remaining processes also behave similarly. Pictorially, the dependences among the transitions are shown by the dashed lines in Figure 4.

Assume this system is simulated by n logical processes, P_i, one for each PP_i. After a state transition, each P_i (except P_n) sends a single message m_i to P_{i+1} indicating the direction in which it has transitioned (North or South). Recalling the definition of dependence above, clearly each m_i is dependent on m_{i-1} and the event that caused m_{i-1}. Now consider an execution of this simulation using Time Warp with Lazy Cancellation. Each P_i, ($2 \le i \le n$) has an event which will cause it to transition out of i_{START}. Since these P_i's have not yet received the transition messages (m_i) from their predecessors, they have to guess the next state to which they must transition. Let us assume that each P_i decides to transition to state i_S and generates m_i informing its successor of the transition. At the same time, P_1 executes its only event, decides (probabilitistically) to transition to state 1_S and sends m_1 accordingly. Since every P_i has guessed its incoming message correctly, no antimessages are generated (because of Lazy Cancellation) and the simulation is complete when all m_i have been received. The entire simulation run takes O(1) time whereas, under the new definition of dependence, the critical path time is O(n). Thus, this simulation run has

completed in super-critical time. In this example, the super-critical speed comes from processes P_2 through P_n. The events they execute and the messages they generate are all dependent on other events and messages (in other words, there are no indpendent messages or states). Thus, Gunter's claim that super-critical speed requires independence is refuted.

From our example, it is evident there are simulations where Time Warp with Lazy Cancellation may complete in super-critical time, even with the new definition of the critical path in [Gunt94a(3)]. It follows that the lower bound derived in [Gunt94a(3)] based on this critical path is *not* a lower bound for Time Warp with Lazy Cancellation (in fact for any super-critical protocol). Our insight suggests that irrespective of how accurately we capture the semantic dependence among events, protocols such as Time Warp with Lazy Cancellation (and others listed in [JeRe91(6)]) still have the capability to complete in super-critical time. Under the best circumstances, each LP can guess all of its dependences correctly (as is the case in our example above), completing its execution in an amount of time equal to the sum of the execution times of all of its events. We are thus led to the conclusion that the lower bound of [LiLa91(8)] (which was stated as a lower bound only for Time Warp with Lazy Cancellation but applies to all protocols):

$$LB = MAX_{P_i}\left(\sum_{e \in E_i} T(e)\right) \text{ where } E_i \text{ is the set of events executed by } P_i$$

remains the best known general lower bound for all PDES's. Moreover, it suggests that critical path analysis will not be able to improve upon this general lower bound. However, this lower bound is of limited significance because it is nearly impossible to achieve in practice for any realistic simulation. Consequently, it is important to note that Gunter's lower bound, which is a tighter one, applies to *particular simulation runs* using super-critical protocols (under the old definition of the critical path) in which LP's do not guess the existence of dependence*.

It is natural to ask if the example we have presented is realistic (i.e. has any practical counterparts). We believe it is because one can imagine a simulation where LP's can determine statistically that the probability of transitioning along a particular arc (*p* in Figure 4) is high and consequently, guess that the next transition will be along that arc. Finally, we note our example also serves to demonstrate that while super-criticality is possible, it may not be observable in practice because it may occur only in brief phases during the simulation but not for the entire simulation. Specifically, if only a subset of the P_i ($2 \le i \le n$) guess their computation correctly, the rollbacks induced could cause the simulation to take longer than the critical path time even though that subset of the processes completed their execution in super-critical time.

* It seems difficult to incorporate this restriction into the protocol itself, because the nature of an LP's guessing depends on the application and cannot be specified in a protocol.

5 Conclusions

A protocol is said to be super-critical if there exists at least one simulation which *can* complete in less than the critical path time using that protocol. Since several implemented protocols have been shown to be super-critical, a sufficient condition for super-criticality is desired which may be used to determine whether protocols of interest are super-critical or not. We have argued that one such condition used in a previous study to demonstrate the super-criticality of four protocols is not sufficient but necessary. By the same argument, we have established a sufficient condition for super-criticality. Our observation that super-critical speed is possible when LP's guess correctly the dependence of some events on unreceived messages disproves the claim in [Gunt94a(3)] that super-critical speed requires independence. We have studied the implications of this contradiction on lower bounds on completion times of simulations.

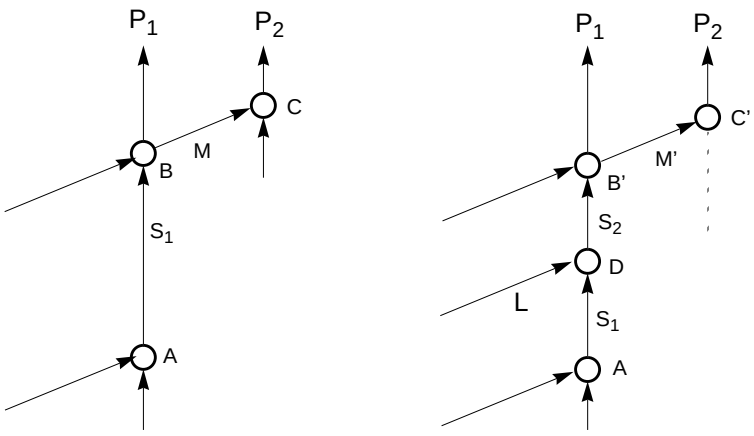
APPENDIX: Super-critical protocols

We examine the four protocols described in [JeRe91(6)] as being super-critical and show that they are indeed super-critical using the condition **SC** presented in Section 3. Note these protocols have already been shown to satisfy the condition presented in [JeRe91(6)], which we call **C**. This is in accordance with our earlier result that **SC** $\Rightarrow$ **C**. We assume familiarity with general PDES terminology and the Time Warp protocol [Jeff85(5)] in particular. The messages and events of interest in the examples are assumed to be on the critical path as required by **SC**.

I. Time Warp with Lazy Cancellation

In this protocol, when an LP rolls back, it does not immediately send out antimessages for all messages sent out with timestamps greater than the rollback time. Instead, it completes the rollback and resumes execution of events. During this resumed execution, it sends out antimessages for previously sent messages which are determined to be incorrect. The rationale behind this scheme is that if the previously sent message turns out to be correct despite the rollback, then the antimessage is not required and some time is saved. Intuitively, this is where this scheme "guesses". If the guess is correct, then it is possible to beat the critical path time.

The scenario satisfying **SC** is shown in the space-time diagrams of Figure 5 which is the same as Figure 4 of [JeRe91(6)]. A vertical arrow (predecessor dependence) is labeled with the state of the LP between the execution of the two events. For example, S_1 is the state at the end of the execution of event *A* and is the state on which event *B* is executed in Figure 5a. A slanted (or horizontal) arrow (antecedent relation) is labeled with the message that caused it. For instance, the execution of event *B* caused message *M* to be sent to P_2 which caused the scheduling of event *C*. We see that P_1 receives message *L* with a timestamp



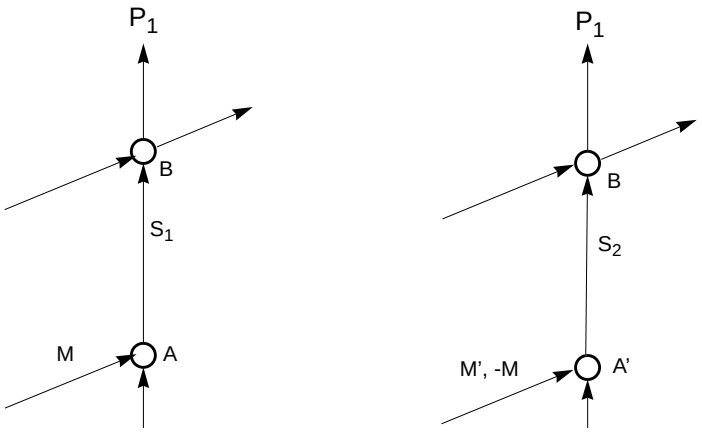
(a) Before rollback **(b)** After rollback

Figure 5 - Lazy cancellation

less than that of event *B* after it executes *B* and sends message *M* to P_2. This causes P_1 to roll back to event *A* and process the message *L*. Event *B* is then re-executed as event *B'* which causes message *M'* to be sent to P_2. If *M = M'*, *M'* will not be sent and therefore, event *C* will not be re-executed. Assuming event *B'* completes at its critical time and is committed, we see that event *C* will complete before its critical time (since it has started execution before its earliest starting time). Thus event *C* satisfies **SC**.

II. Time Warp with Lazy Rollback

Lazy Rollback (or Lazy Re-evaluation) is the equivalent of lazy cancellation for states of LP's (i.e. lazy rollback works on states of LP's while lazy cancellation works on messages sent by LP's). Lazy rollback works as follows. Upon receiving a *straggler* (a message out of order), an LP does not immediately perform all of the actions of rolling back (state restoration, undoing events, etc.). Instead it executes the straggler message in its proper context (i.e. in the state it would have been executed in if it had arrived in the correct order) and examines the resulting state with the LP's previously computed state at that logical time. If these two are the same, rollback is inhibited and the LP continues execution of events. If they are not the same, the LP proceeds to re-execute events until it finds a state which matches a corresponding state before the arrival of the straggler. At this point it restarts its normal execution of events. Thus, the effect of this scheme is that the LP only re-evaluates as many states as required instead of all states in ordinary Time Warp. Of course, lazy rollback also relies heavily upon the capability of an LP to guess its state correctly. Figure 5 shows a scenario under lazy rollback which satisfies **SC**. In Figure 5a, P_1 has processed message *M* and thereafter event *B* in state S_1. In Figure 5b, the message *M* is annihilated by its antimessage -*M* and P_1 receives another message *M*'. It executes the corresponding new event *A*' resulting in state S_2 for event *B*. If



(a) Before straggler **(b)** After straggler

Figure 6 - Lazy rollback

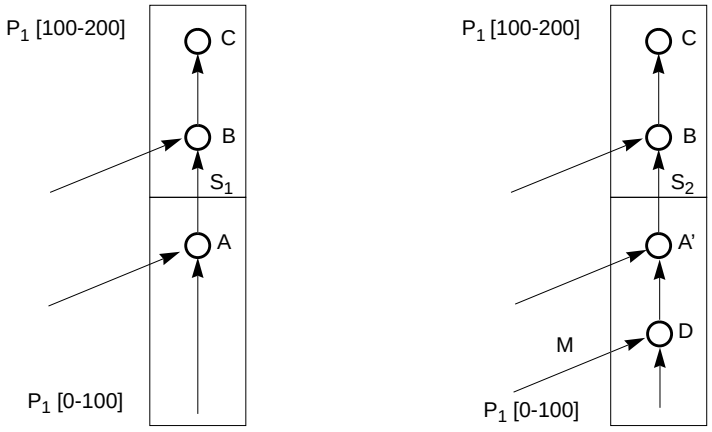
$S_1 = S_2$, then B is not re-executed. Assuming event *A'* is committed (not rolled back later) and completes on or after its critical time, then event *B* has completed before its critical time, thus satisfying **SC**.

III. Time Warp with Phase Decomposition

In this scheme, LP's are decomposed into *phases*. Each phase simulates the PP corresponding to the LP for a segment of logical time. Different phases simulate mutually exclusive and exhaustive segments of time. Each phase assumes an initial state for the LP. At the end of its time segment, each phase forwards its final state to the phase simulating the next contiguous time segment. This is where the phases guess. If a phase guesses its initial state correctly, it is capable of completing before its critical time. Using lazy rollback, this scheme can be made more efficient by having a phase re-evaluate its computation lazily if it receives an initial state which is different from the one it assumed. Figure 5 shows a scenario where Time Warp with phase decomposition achieves super-critical speed. P_1 has been decomposed into two phases, one simulating from time 0 to time 100 and the other from time 100 to time 200. In Figure 5a, P_1[0-100] has completed simulating its time segment and has sent state S_1 as its final state to P_1[100-200]. At a later time (Figure 5b), P_1[0-100] receives the message *M* and therefore has to rollback and re-evaluate event *A* resulting in a final state S_2. If $S_1 = S_2$, then P_1[100-200] will not have to re-evaluate its state and assuming event *A'* is committed and that it has completed on or after its critical time, event *B* will have completed before its critical time and will therefore satisfy **SC**.

IV. Space-Time simulation

Time Warp with phase decomposition is a specific case of a more general method called Space-Time simulation. The space-time method partitions a simulation's space-time diagram into mutually



(a) Before straggler **(b)** After straggler

Figure 7 - Phase decomposition

exclusive and exhaustive regions which are then simulated by processes. Thus, Time Warp with phase decomposition is a space-time partitioning in which each partition has unit width in the space dimension. Space-time simulation is capable of super-critical speed for the same reason that phase decomposition is; viz. if a process guesses its initial state correctly. Since phase decomposition is an example of space-time simulation, the scenario of Figure 5 serves to satisfy **SC** for space-time simulation as well.

ACKNOWLEDGMENTS

The concept of **SCG**, the condition which guarantees super-critical speed is due to a referee of an earlier version of this paper. This work was supported in part by the National Science Foundation (grant CCR-9108448, Aug. 91, number 48), MITRE Corporation (Academic Affiliates Program) and Mystech, Inc. (Academic Affiliates Program).

REFERENCES

1. Berry O., “Performance evaluation of the Time Warp distributed simulation mechanism”, Ph.D. thesis, University of Southern California, 1986.

2. Berry, O. and Jefferson, D., “Critical path analysis of distributed simulation”, *Proceedings of the 1985 SCS Conference on Distributed Simulation*, January 1985, 57-60.

3. Gunter, M., “Understanding supercritical speedup”, *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, July 1994, 81-87.

4. Gunter, M., Private communication, November 1994.

5. Jefferson, D., “Virtual time”, *ACM Transactions on Programming Languages and Systems*. Vol. 7, No. 3, July 1985, 404-425.

6. Jefferson, D. and Reiher, P., “Supercritical speedup”, *Proceedings of the 24th Annual Simulation Symposium*, April 1991, 159-168.

7. Lin, Y-B., “Parallelism analyzers for parallel discrete event simulation”, *ACM Transactions on Modeling and Computer Simulation*, Vol. 2, No. 3, July 1992, 236-264.

8. Lin, Y-B. and Lazowska, E.D., “A study of Time Warp rollback mechanisms”, *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 1, January 1991, 51-72.