Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds)

ON ANALYZING EVENTS TO ESTIMATE THE POSSIBLE SPEEDUP OF PARALLEL DISCRETE EVENT SIMULATION

Tapas K. Som
Bruce A. Cota
Robert G. Sargent
Simulation Research Group
Syracuse University
Syracuse, NY 13244

ABSTRACT

We describe in this paper a new event based approach for estimating the possible speed up in parallel discrete event simulation which is similar to Berry and Jefferson's method. This method calculates a critical path through a graph developed from a trace of a simulation and from constraints on the order of processing of event instances that ensure correctness of the simulation. Different ways of classifying event instances to determine required constraints are presented. A software system that uses this approach is described and examples are presented.

1. INTRODUCTION

Two basic approaches to parallel discrete event simulation based on model decomposition have evolved: conservative [Misra 1986] and optimistic [Jefferson 1985]. Recently many variants of these two approaches have been proposed in the literature. These variants try to use model specific knowledge to achieve maximum speed up. Their success should be measured by comparing the speed up actually achieved with the possible speed up for a given model, where **possible speed up** is the least upper bound on the amount of speed up that is possible in a given model.

To date, two methods have been suggested for estimating the possible speed up for a given model. The first method [Berry and Jefferson 1985] employs a critical path analysis of a trace from a given simulation. In the second method [Wagner and Lazowska 1989], the network of logical processes is viewed as a queueing network model. This queueing network model is then analytically solved.

Berry (1986) demonstrated that the first approach can underestimate the possible speed up. The second approach is limited in scope since queueing network models are analytically tractable only for a limited class of models.

In this paper we describe a new event based approach to estimate possible speed up which is similar to Berry and Jefferson's. This new approach can give a better estimate of the possible speed up than Berry and Jefferson's method. We also describe a software system to estimate the possible speed up based on this new approach.

2. AN EXAMPLE OF THE NEW EVENT BASED APPROACH

We adopt the event world view [Zeigler 1976] in modelling the behavior of a system in this paper. In this world view the behavior of a system is described by specifying a set of 'event types' and then describing how occurrences of one event type may lead to occurrences of other event types. In the subsequent discussion we distinguish between an **event type** 'e', and an occurrence of event type e at time t. The latter will be referred to as an **instance of event type e** and will be denoted by the tuple (e,t). (This notation assumes that multiple instances of the same event type do not occur at the same time.)

To illustrate our method we use the following example of three interacting objects. This example is similar to the one presented by Berry. Consider a physical system comprised of three objects 0_1, 0_2, and 0_3. Each of these objects has an internal counter, i.e. object 0_i has an internal counter C_i, and C_i is initialized to some numeric value. These objects interact by sending messages. An object can send a message to itself. A message is one of two types - **True** or **False**. A True message carries an instruction which is either **Increment** or **Square**.

An object does not do anything when it receives a False message. When an object 0_i receives a True message it takes the following actions A_1, A_2, and A_3:

A1: If the instruction is Increment then the object increments its counter by 1, i.e. $C_i \leftarrow C_i+1$. If the instruction is Square then the object squares its counter, i.e. $C_i \leftarrow C_i^2$.

A2: The following probabilistic decisions are made:
- i) Whether or not to send new messages.
- ii) If new messages are to be sent, then how many of them are to be sent.
- iii) Where these messages are to be sent.
- iv) Contents of these messages, i.e. True/False and Increment/Square.

A3: Send the messages decided on in A2.

Actions A1, A2 and A3 are instantaneous, i.e. the receipt of a True message and the completion of actions A1, A2 and A3 triggered by this message take place at the same instant of time. The delay between the time a message is sent and the time it is received is a random variable.

Since the receipt of a message by an object and the completion of resulting actions taken by the receiving object occur at the same instant of time, one can consider the receipt and resulting actions

Table 1: Messages for Example in Section 2

Received Message					Sent Message				
Message ID #	Object	At Time	Message Content		To Object	Receive Time	Message Content		Message ID #
			Type	Instruction			Type	Instruction	
1	0_1	0	True	Increment	0_2	1	False	—	2
					0_3	1	True	Square	3
					0_1	1	True	Increment	4
2	0_2	1	False	—	—	—	—	—	—
3	0_3	1	True	Square	0_2	4.5	True	Increment	5
4	0_1	1	True	—	0_3	2	True	Increment	6
					0_1	2	True	Square	7
6	0_3	2	True	Increment	—	—	—	—	—
7	0_1	2	True	Square	0_1	3	True	Square	8
8	0_1	3	True	Square	0_2	4	False	—	9
9	0_2	4	False	—	—	—	—	—	—
5	0_2	4.5	True	Increment	0_2	5.5	True	Increment	10
10	0_2	5.5	True	Increment	0_3	6	True	Increment	11
11	0_3	6	True	Increment	0_3	7	False	—	12
12	0_3	7	False	—	—	—	—	—	—

as a single event. It is therefore possible to describe the behavior of the above physical system in terms of the following six event types:

et_i, i=1,2,3 :	Receipt of True message by object 0_i and completion of resulting actions A1, A2 and A3.
ef_i, i=1,2,3 :	Receipt of a False message by object 0_i. (Note that there is no resulting action.)

Now consider a simulation of the three object physical system. The simulation generated the behavior shown in Table 1. Using the six event types defined above and our convention of denoting an instance of event type e at time t by the tuple (e,t), we can say that the following event instances occurred during the simulation:

$(et_1,0)$, $(ef_2,1)$, $(et_3,1)$, $(et_1,1)$, $(et_3,2)$,
$(et_1,2)$, $(et_1,3)$, $(ef_2,4)$, $(et_2,4.5)$,
$(et_2,5.5)$, $(et_3,6)$, $(ef_3,7)$ (1)

$(et_1,0)$ in (1) corresponds to the receipt of message ID #1 of Table 1 and actions taken by object 0_1. Similar correspondences exist for other event instances in (1).

In a sequential simulation these event instances are processed in order of their time of occurrence, e.g. $(et_1,0)$ must be processed before $(ef_2,1)$, etc. Event instances having the same time of occurrence e.g. $(ef_2,1)$, $(et_3,1)$, $(et_1,1)$, etc. must be processed in an order prespecified by the modeller whenever the order of processing matters. Otherwise they can be processed in any order.

A parallel simulation does not follow the above order of processing of event instances. However, some other (weaker) ordering constraints need to be observed to guarantee correctness of the simulation. It is obvious that $(et_1,0)$ must be processed before $(et_3,1)$ since the former **causes** **(schedules)** the latter. Therefore our first constraint on the ordering in which event instances are processed is that if event instance (e_1,t_1) schedules event instance (e_2,t_2), $t_2 \geq t_1$, then (e_1,t_1) must be processed before (e_2,t_2). Constraints of this type will be referred to as **scheduling constraints**.

It is noted that this is identical to sequential constraint 2 in Berry and Jefferson. Figure 1 shows the scheduling constraints imposed on the processing of the event instances in (1). Nodes in Figure 1 represent event instances and there is a directed arc from (e_1,t_1) to (e_2,t_2) if and only if (e_1,t_1) schedules (e_2,t_2).

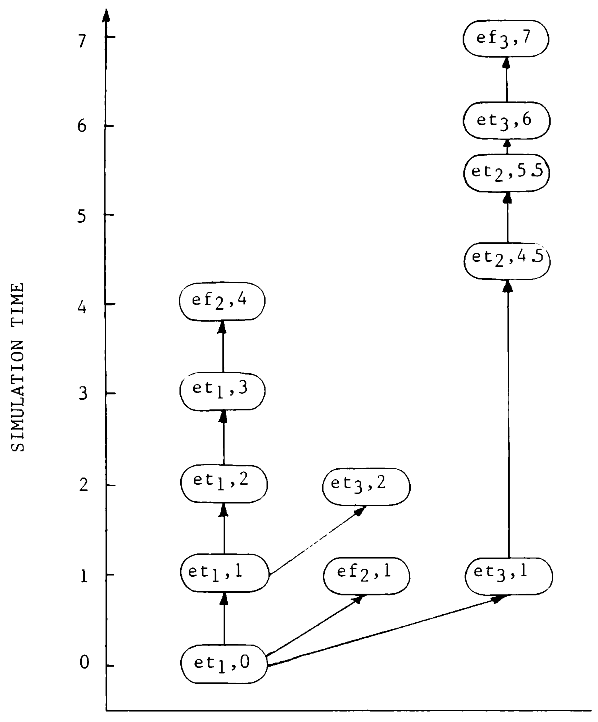

Figure 1: Graph of a Trace for Section 2 Example

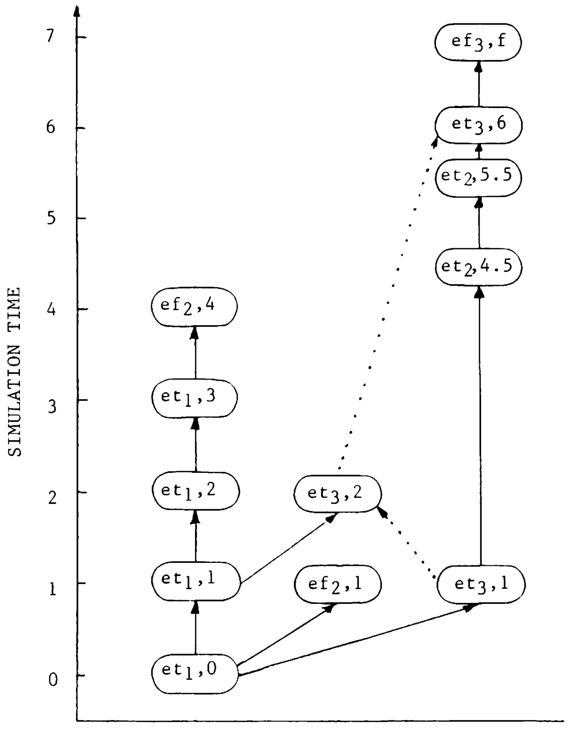

Figure 2: CEG for the Example in Section 2

What other constraints do we need to impose on the order of the processing of event instances? Assume for example that C_3 is initialized to 1 (i.e., the counter of object O_3 is initialized to 1 at the start of the simulation). It is evident that to obtain the correct value of C_3 at the end of simulation one must impose the order of processing $(et_3,1) \rightarrow (et_3,2) \rightarrow (et_3,6)$. We shall use the term **safety constraint** to denote the constraints of this type. Constraints on the order of processing of event instances required by the safety constraints and not by the scheduling constraints are added as dotted arcs to the graph in Figure 1. The resultant graph is shown in Figure 2.

We shall use the term **Constrained Execution Graph (CEG)** to describe graphs like Figure 2. In a CEG nodes denote event instances and directed arcs denote constraints on the order of processing of events (in this example scheduling and safety constraints).

After a CEG has been constructed, an estimate for the possible speed up can be made in the following way:

i) Assume a constant processing time for each event instances (= t_c, say) and assign the weight 't_c' to each node in the CEG representing the computation time for that node. Also assign a zero weight to each arc in the CEG.

ii) Estimate the execution time for a sequential simulation as $T_{seq} = Nt_c$, where N = total number of nodes in the CEG.

iii) The weight of the longest weighted path (the critical path) through the CEG is an estimate for the minimum parallel simulation time T_{par} [Berry and Jefferson]. The critical path and its weight can be found by many standard algorithms [Even (1979)].

iv) An estimate for the maximum speed up 'Su' is given by Su = T_{seq}/T_{par}.

It is easily seen from Figure 2 that for the three object example $T_{seq} = 12t_c$. The critical path in Figure 2 is $(et_1,0) \rightarrow (et_3,1) \rightarrow (et_2,4.5) \rightarrow (et_2,5.5) \rightarrow (et_3,6) \rightarrow (et_3,7)$ and has a weight equal to $6t_c$. Therefore $T_{par} = 6t_c$ and our estimate for possible speed up is $Su = 12t_c/6t_c = 2$.

3. SAFETY CONSTRAINTS

In the last section we described a method for estimating the possible speed up that can be achieved by parallelizing a particular simulation. To automate this method we need a general method for developing safety constraints. That is, we need a method for finding constraints on the order of simulation of event instances that will ensure the correctness of the simulation. Note that the weaker these constraints are, (that is, the fewer constraints that exist) the greater the estimate of the possible speed up. We will assume that it is computationally infeasible to decide for each pair of event instances whether a safety constraint is necessary. Instead, we will consider ways of classifying event instances and use those classifications to develop safety constraints.

3.1. Safety Constraints Based on Event Types

Event instances can first be classified according to their event types and then pairs of event types can be identified that "interfere" with one another. Whenever two event instances belong to types that interfere with one another, a safety constraint can be defined between those two event instances.

It is easy to determine when two event types interfere with one another [Cota and Sargent 1989]. Suppose X is a state variable [Zeigler]. X is an **input variable** of event type e if the effect of processing some instance of e depends on any way on the value of X. If the processing of some instance of e changes the value of X, then X is an **output variable** of event type e. Two event types, e_1 and e_2, are said to **interfere** with one another if there is a state variable that is an output variable of e_1 and is either an input variable or an output variable of e_2 or vice versa. For example, it is easy to see that in section 2, C_i is both an input and output variable for et_i. Thus et_i interferes with itself.

It is not always necessary for a CEG to contain an edge between two event instances whenever those two instances belong to types that interfere with one another (e.g. there is no edge between $(et_3,1)$ and $(et_3,6)$ in Figure 2, even though et_3 interferes with itself). Whenever every arc in a CEG has equal weight (possibly zero) it is unnecessary to have an arc between nodes (e,t) and (e',t') if there already exists a path from (e,t) to (e',t'). This simplifies the CEG and the critical path computation without affecting the weight of the critical path.

3.2. Safety Constraints Based on Objects

An alternate classification of event instances is used by Berry and Jefferson to study object oriented simulations. Here a system is modelled as a collection of "objects", and each event instance in a simulation is associated with exactly one object. Every state variable is also associated with exactly one object and an event instance associated with a particular object can only access state variables that are associated with that object. However, it can schedule other event instances associated with different objects. To ensure the correctness of the simulation under these circumstances it is sufficient to ensure that any two event instances associated with the same object are simulated in correct order. That is, it is sufficient to ensure that whenever two event instances are associated with the same object, the earlier of the two instances is processed before the later of the two. Thus, event instances are classified according to the objects with which they are associated, and safety constraints are defined between all event instances belonging to the same class.

Note that the classification of event instances based on event types gives a strictly finer classification of event instances than the classification based on objects. That is, the set of event instances associated with a particular object can be partitioned into a number of event

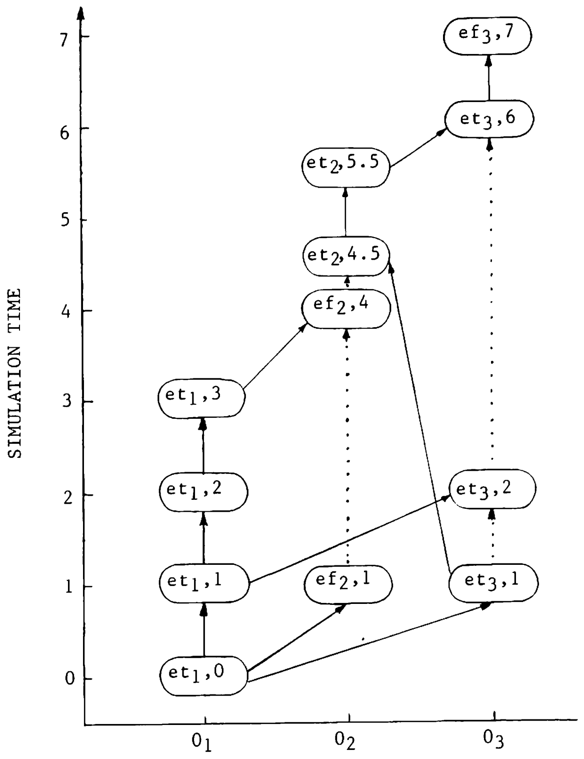

Figure 3: CEG Based on Objects

types, so that it is possible to weaken the safety constraints based on objects and obtain a better estimate on possible speed up.

Consider for example the physical system described in section 2 which consists of three objects O_1, O_2 and O_3. Behavior of individual objects can be described by the following event types: O_1 : et_1, ef_1; O_2 : et_2, ef_2; and O_3: et_3, ef_3. Figure 3 shows the CEG for event instances in (1) using safety constraints based on object decomposition. Scheduling constraints are the same as in Figure 2 but safety constraints are based on objects as discussed here.

Assigning, as in section 2, a weight 't_c' to each node in Figure 3 (and zero weight to each arc), one can easily calculate the maximum weighted path (critical path) which in this case is $(et_1,0) \rightarrow (et_1,1) \rightarrow (et_1,2) \rightarrow (et_1,3) \rightarrow (ef_2,4) \rightarrow (et_2,4.5) \rightarrow (et_2 5.5) \rightarrow (et_3,6) \rightarrow (et_3,7)$ having a weight $= 9t_c = T_{par}$. Therefore, $Su = 12t_c/9t_c = 1.33$. Thus, for this example, Berry and Jefferson's method underestimates the possible speed up. (This is consistent with an observation made by Berry.)

It is quite easy to see how an actual implementation may exceed the speed up predicted by Berry and Jefferson's method. Suppose object O_i is simulated on processor i. Consider the following scenario in a Time Warp [Jefferson] implementation. (See Figure 3.) Processor 2 receives the message corresponding to arc $(et_3,1) \rightarrow (et_2,4.5)$ before it receives the message corresponding to arc $(et_1,3) \rightarrow (ef_2,4)$. It processes $(et_2,4.5)$ and then $(et_2,5.5)$ and sends the message corresponding to arc $(et_2,5.5) \rightarrow (et_3,6)$. While processor 3 is busy processing $(et_3,6)$ and then $(et_3,7)$, processor 2 receives the message corresponding to arc $(et_1,3) \rightarrow (ef_2,4)$, processes $(ef_2,4)$, and finding no reason to nullify the earlier message to processor 3, does not send any antimessage. Thus processor 3's completion time is not affected by the unnecessary constraint $(ef_2,4) \rightarrow (et_2,4.5)$. This constraint is unnecessary since ef_2 and et_2 do not interfere with one another.

3.3. Dynamically Computed Safety Constraints

In many cases, event type based constraints are also stronger than necessary. Consider, for example, a simple, closed queueing network consisting of three servers with FIFO queueing disciplines. Suppose that the service time at each server is a random variable, and that after a service is completed, a customer randomly selects the server that it will require service from next. We can model this system by using three integer variables to represent the length of each queue and three boolean variables to indicate whether each server is idle. Letting 'q[i]' denote the length of the i'th queue (for i between one and three), and letting 'idle[i]' denote the boolean variable that indicates whether the i'th server is idle we can describe the behavior of the i'th server by the following event types:

```
arrival[i] (arrival of new customer to server 'i'):
   if idle[i]
   then do
      idle[i] ← false
      generate a random variate 't'
      schedule end[i] after t
   else q[i] ← q[i]+1

end[i] (end of a service period at server 'i'):
   select server 'j'
   schedule arrival[j] after 0
   if q[i] = 0
   then idle[i] ← true
   else do
      q[i] ← q[i]-1
      generate a random variate 't'
      schedule end[i] after t
```

'q[i]' and 'idle[i]' are output variables of both 'arrival[i]' and 'end[i]'. Furthermore, 'q[i]' is an input variable of 'end[i]' and 'idle[i]' is an input variable of 'arrival[i]'. Therefore event type arrival[i] interfers with event type end[i]. However, it is not always necessary to enforce safety constraints between events of type 'arrival[i]' and type 'end[i]'. To see this suppose that 'q[i]' is greater than zero and that 'idle[i]' is false. A pair of event instances of type 'arrival[i]' and 'end[i]' can then be processed in any order or in parallel (assuming that incrementing and decrementing q[i] is done atomically). No matter in which order the event instances are processed, the resulting value of 'q[i]' and 'idle[i]' will be the same, and the 'end[i]' event instance will schedule a new 'end[i] event instance. However, if q[i] is zero, then the order in which two such events are processed decides which of the two event instances schedules a new 'end[i]' event instance. This can affect the time of occurrence of the new 'end[i]' event. Thus, it is necessary to process instances of 'arrival[i]' and 'end[i] in correct sequential order if 'q[i]' is zero. Unfortunately, there is no way to express this constraint by identifying pairs of event types, as described in the subsection 3.1.

We can obtain safety constraints that are weaker than those obtained by analyzing event types by using information given by the modeller to classify event instances in a different way. To do this, we allow an event instance to belong to any number of classes, and we define a safety constraint between every pair of event instances only when they belong to a common class. The modeller would have to supply a procedure to determine the classes to which each event instance belongs. As a (sequential) simulation is being carried out, the procedure would be used before the processing of each event instance to determine the classes to which that event instance belongs. This information can be recorded in the trace. It would probably be convenient for the modeller to supply a different procedure for each event type. The major disadvantage of this approach is the reliance on the modeller to identify all necessary classifications.

For example, consider the queueing network described above. For each possible value of 'i', we would provide a class for events of type 'arrival[i]' and a class for events of type 'end[i]'. This would define safety constraints between event instances of the same type. (It might be possible to weaken these constraints.) In addition, we would provide an additional class for each 'i'. An event instance would belong to this additional class whenever the value of 'q[i]' is zero immediately before or immediately after the simulation of that event instance. This defines safety constraints between some event instances of type 'arrival[i]' and some event instances of type 'end[i]' but not between all such instances.

Allowing events to be processed in an order not corresponding to their time of occurence, even when it is "safe" to do so, may make data collection more difficult in some situations. Consider for example q[i]. The sequence of successive values assigned to q[i] during the parallel simulation would not necessarily reflect how q[i] changes with respect to time. We will not discuss this issue further, except to observe that the ease with which a simulation can be parallelized depends, in part, on the data to be collected by that simulation.

4. SOME RELATED ISSUES

4.1 Assigning Weights to Nodes in the CEG

It has been implicitly assumed in the last two sections, that an identical amount of time is required to process each event instance. This may not be true. It might therefore be useful to assign different weights to different nodes of a CEG to reflect the amount of time required for the processing of each event instance and to use these weights to compute T_{seq} and T_{par}. One way of doing this would be to assign a weight to each event type. All instances of a given type would then be assigned the same weight. This seems reasonable, since events of a given type are simulated by carrying out a particular procedure. The weight assigned to that event type then reflects an estimate of the amount of time required to execute that procedure.

4.2. Event Granularity

Franta [1977] defines an event as an "instantaneous change in the values of one or more state variables". This allows a number of different interpretations of what exactly an event consists of in a given simulation. For example, in the queueing network model discussed in subsection 3.3, event types were defined for the end of a service at each server and for the arrival of a customer to each server. However, each end service event instance is immediately followed by an arrival event instance, and each arrival event instance is immediately preceded by an end service event instance. We can therefore consider each pair of end service and arrival event instances to be a single event instance, which we will refer to as a "departure" event instance. We can describe departure events as instances of the following type:

```
departure[i]
   select server 'j'
   if q[i] = 0
   then idle[i] ← true
   else do
      q[i] ← q[i]-1
      generate a random variate 't'
      schedule end[i] after t
   if idle[j]
   then do
      idle[j] ← false
      generate a random variate 't'
      schedule end[j] after 't'
   else q[j] ← q[j]+1
```

The event type 'departure[i]' has been defined by simply appending the code for event type 'arrival[j]' to the code for event type 'end[i]' and modifying the scheduling of event instances. Each departure event instance is the result of **aggregating** an end service instance and an arrival instance. (See Sargent [1988] for methods of combining event types.) We will say that a departure event instance is of "higher granularity" than the corresponding end service instance and arrival instance.

Whenever there is a directed path in the CEG from one event instance to another, we will say that the first event instance **precedes** the second event instance and that the second event instance **succeeds** the first event instance. Note that "precedes" and "succeeds" are transitive in the sense that if (e_1,t_1) precedes (e_2,t_2), which precedes (e_3,t_3), then (e_1,t_1) also precedes (e_3,t_3) and (e_3,t_3) succeeds (e_1,t_1).

Now consider the effects of event aggregation and event granularity on a CEG. The aggregation of two events will change a CEG by merging the nodes in the CEG corresponding to instances of these two event types. The single node resulting from this combination will have every arc entering and leaving it that either of the original two nodes had. Suppose that (e_1,t) and (e_2,t) are aggregated to form (e,t). Then every event instance that precedes (e_1,t) or (e_2,t) will also preceed (e,t). Similarly, every event instance that succeeds

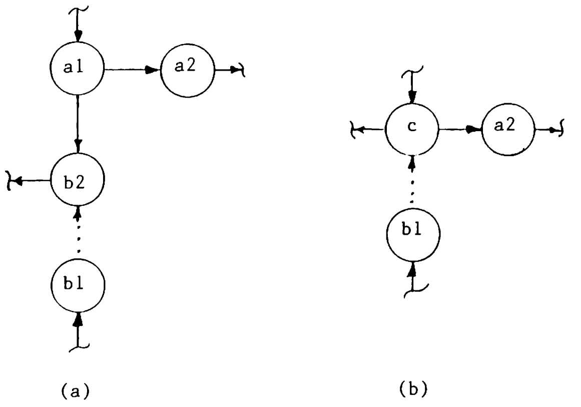

Figure 4: Event Aggregation

(e_1,t) or (e_2,t) will also succeed (e,t). Thus, every event instance that preceeds (e_1,t) will also precede any event instance that succeeds (e_2,t), and vice versa. This means that the aggregation of (e_1,t) and (e_2,t) may have introduced many more possible paths through the CEG, and so may have increased the length of the critical path through the CEG.

For example, in Figure 4a, a small portion of an arbitrary CEG is shown. Event instance 'a1' precedes event instance 'a2', and event instance 'b1' precedes event instance 'b2'. We have used solid arcs to denote scheduling constraints and dotted arcs to denote safety constraints. We have also assumed that 'a1' schedules 'b2', so that 'a1' precedes 'b2'. Now suppose that the time delay on this scheduling operation is zero, so that 'a1' and 'b2' occur at the same point in (simulated) time. Assume that we can aggregate 'a1' and 'b2' to form, say, event instance 'c'. The new configuration of constraints is shown in Figure 4b. The point is that since 'b1' must precede 'b2' to ensure correctness of the simulation, a safety constraint is required between 'b1' and 'c'. Because of this, 'b1' now precedes 'a2', and so a new path has been added to the CEG that could form part of the critical path.

In general, if the weight of (e,t) is defined to equal the sum of the weights of (e_1,t) and (e_2,t), then the weight of the critical path of the CEG after aggregation will be greater than or equal to the weight of the critical path of the CEG before aggregation. This effect will be illustrated by an example in section 5.

4.3. Computer Architecture

We can study the effects of computer architecture on the possible speed up obtained by a parallel simulation by assigning each event instance to a specific processor. The result of processing some event instance on one processor may have to be communicated to some other processor before some other event instance can be processed on the second processor. In this case there will be a safety constraint between those two event instances. The arc in the CEG that represents this constraint can be given a positive weight that reflects the cost of communication between the two processors involved. This weight would be used when computing the weight of the critical path. Also when a number of event instances are assigned to the same processor there has to be a total ordering on the sequence of processing of these event instances. One possible ordering corresponds to the time of occurrences of event instances. Additional arcs may have to be added to the CEG to reflect this total ordering and they may have a weight of zero. These additional arcs will be called "computer architecture arcs".

5.0 SOFTWARE SYSTEM

In this section we first describe a software system that runs on a SUN Workstation. This software creates a CEG from inputs given to it and then finds the CEG's critical path(s). Then an example is presented to illustrate the use of this software and to demonstrate that event granularity can affect the possible speed up in discrete event simulations.

5.1. Description of Software System

Our software system consists of two parts and is used in two stages. An event-oriented simulator is first used to obtain an execution trace by running a simulation model. Then this trace, along with other appropriate information, is input into a software package that produces the CEG and calculates the critical path(s).

The event-oriented simulator [Chacko and Sargent 1989] is a simulator developed by our research group for research and teaching purposes. It uses event graphs [Sargent 1988, Schruben 1983] for model specification. One of the output options of this simulator allows one to obtain an execution which trace contains the information required to create the nodes and scheduling arcs in a CEG. Thus it is straightforward to obtain the execution trace and requires no additional work (except to build a simulation model if one does not exist).

The trace from our event graph simulator is input into a software package along with information on safety constraints, computer architecture constraints (if desired), and weights for the nodes and arcs. Our current software package allows only the following: (i) definition of safety constraints based on event types per subsection 3.1, (ii) one option for computer architecture arcs, and (iii) weights for different event types and different types of edges (e.g. scheduling, safety, or architecture). For the safety constraints based on event types, the user must list the pairs of event types that interfere with each other. The only option currently available to model the effects of computer architecture is to assign event instances of each event type to a separate processor. The software package then uses these inputs to create a CEG and calculate the critical path(s). The output consists of the critical path(s) and the corresponding weight which is an estimate of T_{par}. We note that the software package can easily be extended, e.g. to allow alternative ways of calculating safety constraints such as Berry and Jefferson's object based approach.

5.2. Example

Let us consider a two server cyclic queueing system where customers always go to the other server for service when they complete service at a server. The queue disciplines are FIFO and the service times are exponential. Let us specify a simulation model of this system by using different event types for an arrival to the queue, begin service, and end service for each of the two servers. These six different event types are given in Table 2 along with the input and output variables for each event type (we assume different random number generators are used for B(1) and B(2)) and the pairs of event type interferences required for the safety constraints per subsection 3.1.

We ran a simulation model of this queueing system with two customers and with the mean equal to one for the service times of each server on our event graph simulator. The execution trace of this simulation was input into our software package with a weight of one for each node type along with the list of pairs of event types that interface with each other. (We did not assign any weights to the arcs or use any computer architecture constraints.) We obtained an estimate of possible speedup of approximately 1.41 for this model.

We discussed in subsection 4.2 that event granularity can affect the estimate of possible speed up. If one modelled the above queueing

Table 2: Event Types and Interferences for Example in Section 5.2

```
event type A[i], i=1,2:
   Arrival[i]:
      q[i] ← q[i]+1
      if s[i] = idle
      then do
         schedule Begin[i] after delay 0
         s[i] ← busy

event type B[i}, i=1,2:
   Begin[i]
      q[i] ← q[i]-1
      generate service time, ST[i]
      schedule End[i] after delay ST[i]

event type E[i], i=1,2:
   End[i]
      if q[i] > 0
      then do
         schedule Begin[i] after delay 0
         s[i] ← busy
      else s[i] ← idle
   select j
   schedule Arrival[j] after delay 0
```

Event Type	Input Variables	Output Variables
A[i]	q[i],s[i]	q[i], s[i]
B[i]	q[i]	q[i], ST[i]
E[i]	q[i]	s[i]

Event Type Interferences

(A[i],A[i]), (B[i],B[i]), (A[i],B[i]), (A[i],E[i]), (B[i],E[i]) system using only two event types, one for the end of each service as in subsection 4.2, these two event types would interfere with each other as well as with themselves. Thus there would be arcs between each instance of each event type and between instances of the two different event types. This requires all event instances to be processed sequentially. Thus, there would be no parallelism and the speed up would be one.

6.0 SUMMARY

We described a new event based approach for estimating the possible speed up for a given simulation model which is similar to Berry and Jefferson's approach. The estimate of the possible speed up is determined by the constraints imposed on the order of processing event instances. For the simulation to be correct the order of processing must satisfy both scheduling and safety constraints. Scheduling constraints can be easily identified from the trace of a given simulation run. Identification of safety constraints is more difficult. Berry and Jefferson used an object based model decomposition to identify constraints which are stronger than necessary. We discussed an event type based approach that is capable of identifying safety constraints which, even though stronger than necessary, are strictly weaker than those identifiable by Berry and Jefferson's method. Consequently both Berry and Jefferson's method and our event type based method underestimate the maximum possible speed up but the latter gives a better estimate. We also discussed that a modeller may be able to define classifications of event instances other than those based on event types or objects. This can further improve the estimate of the possible speed up.

We described how definitions of event types used by the modeller can affect the possible speed up. We concluded that event types of smaller granularity are generally better for identifying possible speed up. Also, we pointed out that interprocessor communication costs and the effects of a specific computer architecture on speed up could possibly be addressed.

A simple software system to obtain the critical path and an estimate of the possible speed up was described. This system first uses an event graph simulator to obtain the trace of a simulation. Then, this trace along with other user defined inputs are processed by a software package to estimate the possible speed up. The current version of our software package contains only a few alternatives; however, it can be easily extended to provide for others.

ACKNOWLEDGMENTS

This work was supported in part by the CASE (Computer Applications and Software Engineering) Center of Syracuse University. Scott Marcotte wrote the software package described in Section 5.

REFERENCES

Berry, O., and Jefferson, D. (1985). "Critical Path Analysis of Distributed Simulation", *Distributed Simulation, 1985* (Paul Reynolds, ed.), Simulation Series, 15, 2, 57-60, Society for Computer Simulation.

Berry, O. (1986). Ph.D. Thesis, Department of Computer Science, University of Southern California.

Chacko, J. and Sargent, R.G. (1989). BUBBLES, The Event Graph Simulator. Simulation Research Group, 441 Link Hall, Syracuse University, Syracuse, NY 13244.

Cota, B.A. and Sargent, R.G. (1989). "An Algorithm for Parallel Discrete Event Simulation Using Common Memory", *Proceedings of 22nd Annual Simulation Symposium,* (A.H. Raton, ed.), 23-31, March 1989, Tampa, FL.

Even, S. (1979). *Graph Algorithms.* Computer Science Press, Potomac, MD.

Franta, W.R. (1977). *The Process View of Simulation.* North-Holland, New York.

Jefferson, D. (1985). "Virtual Time", *ACM TOPLAS,* 7, 3, July 1985, 404-425.

Misra, J. (1986). "Distributed Discrete-Event Simulation", *ACM Computer Surveys,* 18, 1, 39-65.

Sargent, R.G. (1988). "Event Graph Modelling for Simulation with an Application to Flexible Manufacturing Systems", *Management Science,* 34, 10, 1231-51.

Schruben, L. (1983). "Simulation Modeling with Event Graphs", *Communications of the ACM,* 26, 957-963.

Wagner, D.B. and Lazowska, E.D. (1989). "Parallel Simulation of Queueing Networks: Limitations and Potentials," *Proceedings of the International Conference on Measurement and Modeling of Computer Systems,* 146-155, May 1989, Berkeley, CA.

Zeigler, B.P. (1976). *Theory of Modelling and Simulation.* John Wiley and Sons, New York.

AUTHORS' BIBLIOGRAPHY

BRUCE A. COTA is a Ph.D. candidate in the School of Computer and Information Science at Syracuse University. He has a B.S. degree in mathematics from Buffalo State College and an M.S. in mathematics from Syracuse University. His current research interests are in parallel processing and in discrete event simulation.

Bruce A. Cota
441 Link Hall
Syracuse University
Syracuse, NY 13244-1240
(315)443-2820

ROBERT G. SARGENT is a Professor at Syracuse University. He received his education at the University of Michigan. Dr. Sargent has served his profession in many ways. This includes being Department Editor of Simulation Modelling and Statistical Computer for the *Communications of the ACM* for five years; being chairman of the TIMS College on Simulation and Gaming; serving the Winter Simulation Conferences in several capacities such as being a member of the Board of Directors for ten years, Board Chairman for two years, General Chairman of the 1977 Conference, and co-editor of the 1976 and 1977 Conference Proceedings;

being a Director-at-large of the Society for Computer Simulation; serving as an ACM National Lecturer; and being a member of the Executive Committee of the IEEE Computer Society Technical Committee on Simulation. He has received four service awards, including one for long-standing exceptional service to the Simulation Community. Professor Sargent is the author of over seventy-five publications. His current research interests include methodology research in modelling and discrete event simulation, model validation, performance evaluation, and applied operations research. Dr. Sargent is a member of the AπM, New York Academy of Sciences, Sigma Xi, ACM, IIE, ORSA, SCS, and TIMS, and is listed in Who's Who in America.

Professor Robert G. Sargent
Simulation Research Group
441 Link Hall
Syracuse University
Syracuse, NY 13244-1240
(315)443-4348

TAPAS K. SOM is a Ph.D. candidate in the Computer and Information Science Program at Syracuse University. He holds a Bachelors degree in Mechanical Engineering from the University of Calcutta, India and a Masters degree in Industrial Engineering and Operations Research from Syracuse University. Before coming to graduate school, he worked as a Senior Project Engineer and was involved in the design and construction of fossil fired power stations. His current research interests are simulation modelling concepts and parallel discrete event simulation.

Tapas K. Som
441 Link Hall
Syracuse University
Syracuse, NY 13244-1240
(315)443-2820