1981 Winter Simulation Conference Proceedings
T.I. Ören, C.M. Delfosse, C.M. Shub (Eds.)

THE SIMULATION OF A PIPELINED EVENT SET PROCESSOR

John Craig Comfort
Florida International University
Department of Mathematical Sciences
Miami, Florida 33199

Anita Miller
Department of Mathematics
Ransom-Everglades School
Coconut Grove, Florida 33133

Abstract. The availability of inexpensive, sophisticated processing elements affords the computer system designer the opportunity to create tailored computer systems for specialized applications. In this paper, the authors present a design for a discrete event simulation computer, in which event set manipulation is performed by a pipelined set of microprocessors. A simulation model for the system is presented and the selection of optimal parameters for the system are discussed. The results of the simulation and suggestions for further research are presented.

1. INTRODUCTION

When a program is designed using a top down methodology, each step of the refinement process results in the specification of a set of subtasks, where each subtask has associated with it a set of input specifications, a set of output specifications, a process specification, and a designated environment in which it functions. With the availability of inexpensive, powerful microprocessing elements (such as the Motorola 68000 and the Intel 432), a system designer has flexibility in assigning selected subtasks to diverse processing elements.

Within the context of a discrete event simulation program, certain groups of related tasks may be identified. One of the most significant of these is the group associated with maintaining the event set. In a recent study by one of the authors (Comfort, 1981), the behavior of a moderately large simulation program was studied. In various executions of the program, it was observed that between 32 and 38 percent of all instructions executed by the processor were used for event set maintainence. In this paper, we present an algorithm for event set processing which is distributed over a set of processors and memory units.

Considerable work has been done in recent years in the design and analysis of event set algorithms and data structures (all cited references except Comfort and Coats, 1980). All of the referenced articles assume that a single processor performs both general simulation computation and event set processing. In the study mentioned above, a rough simulation was done of a simulation processor/event set processor system, indicating that further research into this problem is justified.

2. EVENT SET DATA STRUCTURES AND OPERATIONS

Every individual (here referred to as an "entity") which is represented in the simulation program will be assigned a unique set of data locations in the main memory of the simulation computer. This block contains information about the state of the entity, times used for the calculation of statistics, and various pointers to other entities in the system. Further, for those entities which have a defined time of next processing, an "event notice" will be created. This event notice will contain (minimally) the address of the block of the memory assigned to the entity and the entity's time of next processing. For the purposes of this study, event notice memory will be assumed to be physically disjoint from main memory. Further, in a multiple processor system, a separate event notice memory will be assigned to each processor.

Of the operations associated with event set processing, three may be considered primitive:

1) "schedule (time,entity)"-Insert an event notice with the given "time" for the given "entity" in the event set.

81CH1709-5/81/0000-0591$00.75

2) "next" - Return the "entity" with the smallest next processing "time" to the calling program. Update the simulation clock to "time".

3) "remove (entity, time)" - Destroy the event notice associated with "entity". Do not change the simulation clock. Note that the "time" is not necessary for this operation, as an "entity" may have at most one event notice associated with it. The specification of the time greatly facilitates the location of the event notice to be removed.

For the course of this discussion, our attention will be focused principally on the "schedule" and "next" operations. In simulations done by the authors, these operations accounted for over 98 percent of all calls on the event set programs (For a situation in which this is not the case, see Blackstone et.al. (1981)).

3. A PIPELINED EVENT SET ALGORITHM

Let us assume that the computer system being simulated consists of one simulation process (called SIMP) and its associated memory, and N event set processors (called EVSPs), each with its own memory. Each event set processor has associated with it two additional control parameters, called UNDER and OVER. The event set algorithm always attempts to maintain the event set size between these two bounds. In addition each processor maintains:

1) A doubly linked list of event notices anchored at each end by the cells PAST, with time of -1, and FUTURE, with time infinity (or an engineering approximation thereof).

2) The variable MAXTIME, containing the largest next processing time of an event notice in this processor.

In the situation where the SIMP wishes to schedule an event notice in EVSP(1), or when EVSP(n) attempts to schedule an event notice in EVSP(n+1), let the entity being referenced be "EN", and the time of processing "tsim". Then the SIMP or EVSP(n) will

1) Wait for processor EVSP(n+1) to become unbusy.

2) Transfer "EN" and "tsim" to EVSP(n+1).

Viewed from EVSP(n+1), this processor will:

1) Accept "EN" and "tsim" from EVSP(n) or SIMP, and acknowledge the receipt.

2) If the processor has at least UNDER event notices, and "tsim" > MAXTIME, then

   2.1) Wait for EVSP(n+2) to become unbusy.

   2.2) Transfer "EN" and "tsim" to EVSP (n+2).

3) If the processor has fewer than UNDER event notices, or "tsim" $\leq$ MAXTIME, then schedule the event notice in the local event set memory of EVSP(n).

4) If the processor now has more than OVER event notices, then unlink the event notice with largest next processing time. Let it's time of next processing be "tmax" and let it refer to the entity "ENmax". Update MAXTIME to relect the largest simulation time of the remaining event notices. Then

   4.1) Wait for processor N+2 to become unbusy.

   4.2) Transfer "ENmax" to "tmax" to EVSP (n+2).

If the SIMP or EVSP(n) (n>1) has requested to perform a "next" operation, EVSP(n+1) must:

1) Unlink the event notice with smallest next processing time "tnext" and its entity number "ENnext". If there are no such notices, set "tnext" and "ENnext" to zero.

2) Transmit "tnext" and "ENnext" to EVSP(n) or SIMP.

3) If the number of event notices in EVSP(n) is now less than UNDER (but greater than zero), then

   3.1) Wait until EVSP(n+2) is unbusy.

   3.2) Initiate a "next" operation on EVSP (n+2), and wait.

   3.3) Retrieve the time, "t", and the entity, "EN", from EVSP(n+2).

   3.4) Link "t", "EN" into the event set on processor n+1.

4. THE SIMULATION EXPERIMENT

The simulation experiment consists of configuring a (simulated) computer system consisting of N EVSPs and the SIMP. This sytem is exercised using trace data taken from a real simulation program (described in Comfort and Coats, 1981). The primary dependent variable of the simulation is the percent of time that the SIMP must wait on the first EVSP in the pipeline - presumably all of the remainder of its time may be used in running simulations. Also of interest are the idle times of the EVSP's in the system. Timing data were obtained by counting the number of simulated machine instructions executed during and between calls on the event set routines. This information was gathered by inserting software probes into an already existing simulation program. Similar data were obtained for the event set routines by examining a compiled assembly language listing of the program. This metric was chosen instead of elapsed time because the Univac 1100 real time clock, which has a resolution of 200 microseconds, is far too coarse for the purpose of this study. In addition, the Univac system has a large cache memory - measured performance would have a substantial amount of unexplained variance due to the memory access patterns of other jobs being run at the same time.

The simulation program was written in University of Wisconsin Pascal, and designed using a top down methodology.

The parameters to the simulation program are:

1) N, the number of event set processors, with $0 \leq N \leq 3$.

2) The overflow and underflow limits (OVER and UNDER) for each processor in the pipeline.

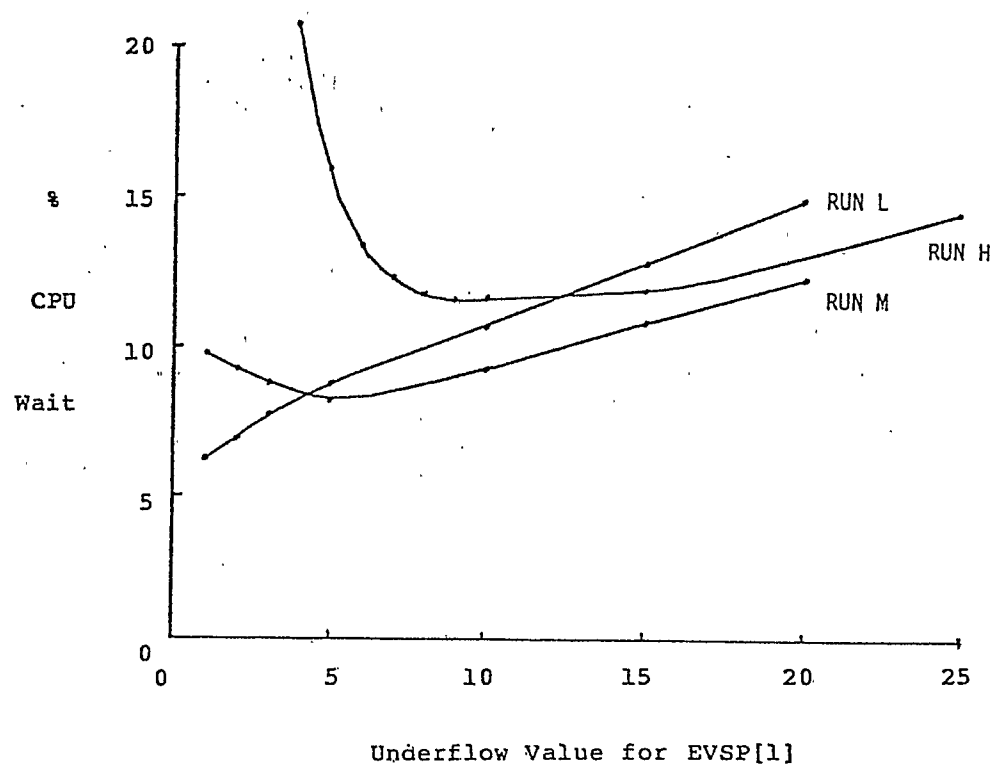
3) An external trace file, consisting of one

TABLE 2

PERFORMANCE OF A TWO EVSP SYSTEM

	RUN H Mean Event Set Size 203.1			RUN M Mean Event Set Size			RUN L Mean Event Set Size		
UNDER [1]	% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy
25	14.2	16.3	19.3						
20				12.1	15.4	8.0	14.9	17.9	2.7
15	11.9	11.2	24.6	10.6	13.1	9.1	12.7	14.9	4.5
10	11.6	9.8	26.5	9.0	10.7	11.6	10.0	12.1	6.9
9	11.5	9.5	26.4						
8	11.7	8.9	28.7						
7	12.2	8.4	30.6						
6	13.3	7.8	33.3						
5	15.9	7.0	38.6	8.1	7.7	20.4	8.6	9.4	9.7
4	20.9	6.9	47.1						
3				8.7	5.8	34.2	7.7	7.9	14.5
2				9.2	4.8	42.5	6.9	6.6	21.4
1				9.7	4.8	47.8	6.2	5.2	29.6

FIGURE 1. PERFORMANCE OF A TWO EVENT SET PROCESSOR SYSTEM



record for each event set call made in the original program. Each record consists of:

a) the kind of call ("schedule" or "next")

b) the incremental simulation clock time

c) the instruction execution count between this call and the previous call on the event routines

d) the instruction execution count to perform the event set operation in the driving program.

4) The time required to communicate between two processors.

The program used to generate the trace files is an event driven simulation of a large specialized computer network (described in Comfort 1980). The program was implemented in FORTRAN on a Univac 1100 computer system. It was designed using a top down methodology, and no especial effort was made to minimize running time. The event set algorithm used was a variant of the adaptive indexed linked list algorithm (described in Comfort 1979 and Wyman 1975).

The results presented are chosen to represent conditions of light, moderate, and moderately heavy loading of the simulated computer system. They will be referred to as RUN L, M, AND H respectively.

5. RESULTS

The behavior of a system using the pipelined list event set in a system with one SIMP and one EVSP is shown in Table 1. By most standards, this performance is clearly unacceptable, as the CPU is idle (waiting on the EVSP) more than 20 percent of the time and more than 50 percent in a heavily loaded system. An analysis of the sequence of calls to the event set processors (performed in Comfort 1981) reveals that it is quite common to have a "next" operation immediately following a "schedule", requiring the CPU to wait for the completion of the "schedule". Further, in a one EVSP system, the pipelined algorithm reduces to the simple doubly linked list algorithm (Comfort 1979), which is quite inefficient in time. Thus, it is necessary either to employ a more sophisticated event set algorithm, or to add additional event set processors to the system.

TABLE 1

	RUN H	RUN M	RUN L
% SIMP Wait	53.78	31.08	24.05
% EVSP Busy	60.41	62.26	27.99

Following the second approach, a simulated three processor system (one SIMP plus two EVSP's was configured, and a set of simulation experiments was run. One of the first observations made was that the value of the OVER parameter is largely irrelevant. There are two mechanisms for transferring an event notice from EVSP(n) to EVSP(n+1). In the first, the new event notice has time larger than that of the most distant event notice in EVSP(n) and is passed on directly. In the second, the event notice is scheduled in EVSP(n), a local memory overflow occurs, and the most distant event is 'bumped' to EVSP(n+1). Apparently the first mechanism is used almost exclusively unless the OVER value is very close to that of UNDER. As a result, the OVER value was set to the maximum size for each processor's event set memory, and subsequently ignored.

In a two EVSP system, more nearly satisfactory results have been obtained (as shown in Figure 1 and Table 2). Under conditions of heavy system loading (labeled RUN H), an optomal CPU wait of about 11 percent is obtained with an underflow value of 9. Under moderate load, the optomal point is less well defined, and occurs when UNDER is 5. With light loading, best behavior is seen when UNDER assumes its minimum value of 1. Fixing the parameter value at that which optimizes performance in the heaviest system loading, CPU wait percentages of 9.81, 10.53, and 12.08 are obtained for the three runs.

It is interesting to observe that optimal behavior seems to occur when EVSP(2) is doing roughly three times as much work as EVSP(1). When the load ratio is greater than 3, the time that SIMP waits on EVSP(1) while that processor waits on EVSP(2) becomes significant. At a lesser load, EVSP(1) is doing too much scheduling, and the time that SIMP waits on EVSP(1) dominates.

If a two EVSP system were desirable for some a priori reason, it would be quite possible to design an algorithm in which the SIMP is able to dynamically alter the value of UNDER (1). The pragmatics of such an algorithm are beyond the scope of this paper, however.

By proceeding to a three EVSP system and varying the UNDER parmaters to EVSP (1) and (2), even better performance was obtained. Starting with the optomal value of UNDER (1) obtained in the two processor system and varying UNDER (2), the CPU idle percent is reduced to 8.38 (with an UNDER (2) value of 10). For this set of parameter values, the middle event set processor is fairly lightly loaded with respect to the front end processor. By analogy with the observed behavior of the two EVSP system, this suggested that a smaller value of UNDER (1) could be employed. Values of 5,3,2 and 1 were used for this parameter (see Table 3 and Figure 2). As expected, the value of one produces the best behavior, yielding a SIMP idle percentage of 5.30 and a relative loading ratio of 3.5:1.

Using the same parameter values for the moderately and lightly loaded system, similar values for SIMP idle percentage were obtained.

6. CONCLUSION AND DIRECTIONS FOR FURTHER RESEARCH

The simulation study here performed certainly indicates that a pipelined event set processor may result in a significant increase in the utilization of the central simulation processor and that the performance increase is nearly independent of processor loading. (see summary in Figure 4)

TABLE 3

PERFORMANCE OF A THREE EVSP SYSTEM (RUN H)

UNDER(2): 100

UNDER [1]	% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% EVSP3 Busy
9	8.51	9.92	10.79	3.29
5	8.35	7.80	16.83	3.61
3	11.09	5.03	32.60	3.78
2	13.34	4.55	42.90	3.75
1				

UNDER(2): 50

% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% EVSP3 Busy
8.36	10.08	4.46	15.21
7.16	7.88	7.82	15.36
6.48	6.27	15.74	15.69
6.10	5.20	23.10	16.50
6.40	4.36	27.90	16.89

UNDER(2): 25

UNDER [1]	% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% EVSP3 Busy
9	8.40	10.61	2.77	12.75
5	7.20	8.00	4.40	19.14
3	6.25	6.20	9.70	19.96
2	5.73	5.14	13.68	20.23
1	5.30	4.35	16.78	21.03

UNDER(2): 10

% SIMP Wait	% EVSP1 Busy	% EVSP2 Busy	% EVSP3 Busy
8.38	9.89	2.19	22.79
7.27	7.87	3.07	25.34
6.40	6.12	6.45	26.34
5.89	5.06	8.57	26.78
5.50	4.29	10.27	27.73

FIGURE 2-EFFECT OF UNDERFLOW PARAMETER VALUES ON THE PERFORMANCE OF A THREE EVSP SYSTEM

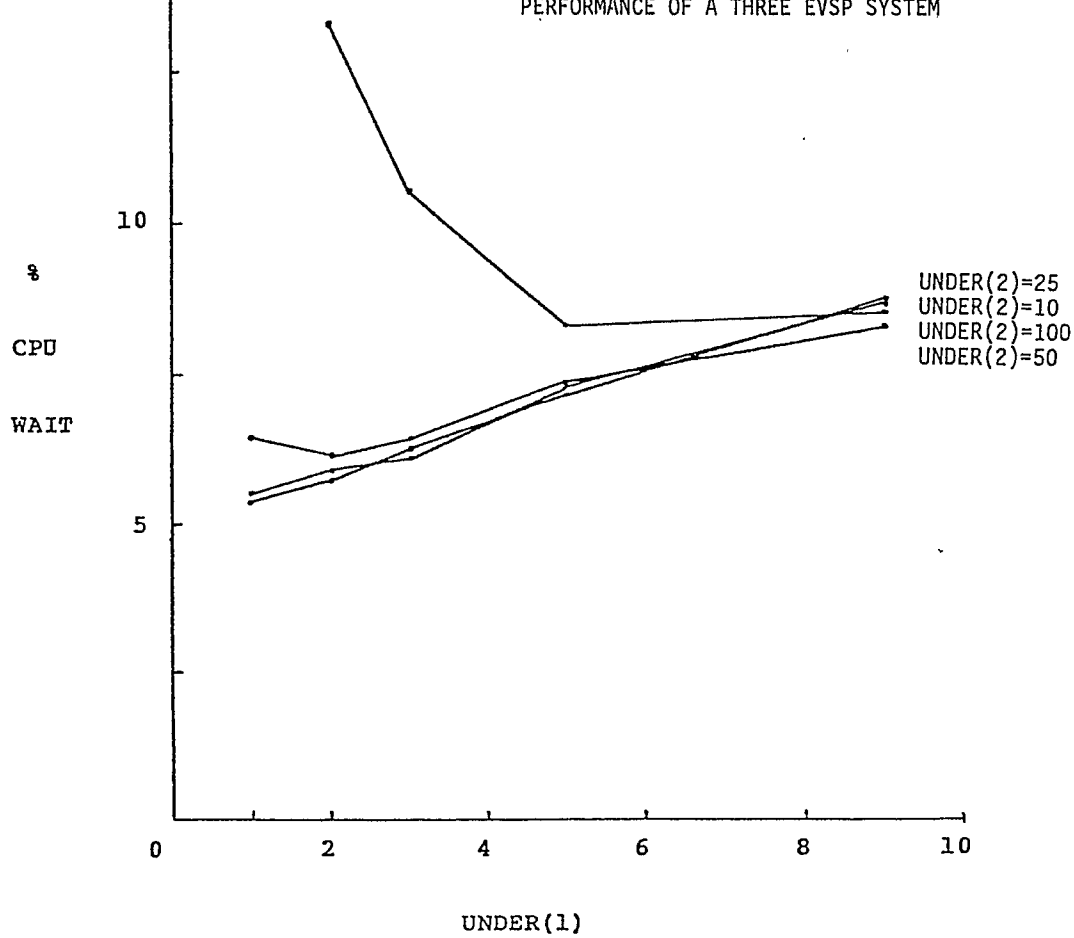
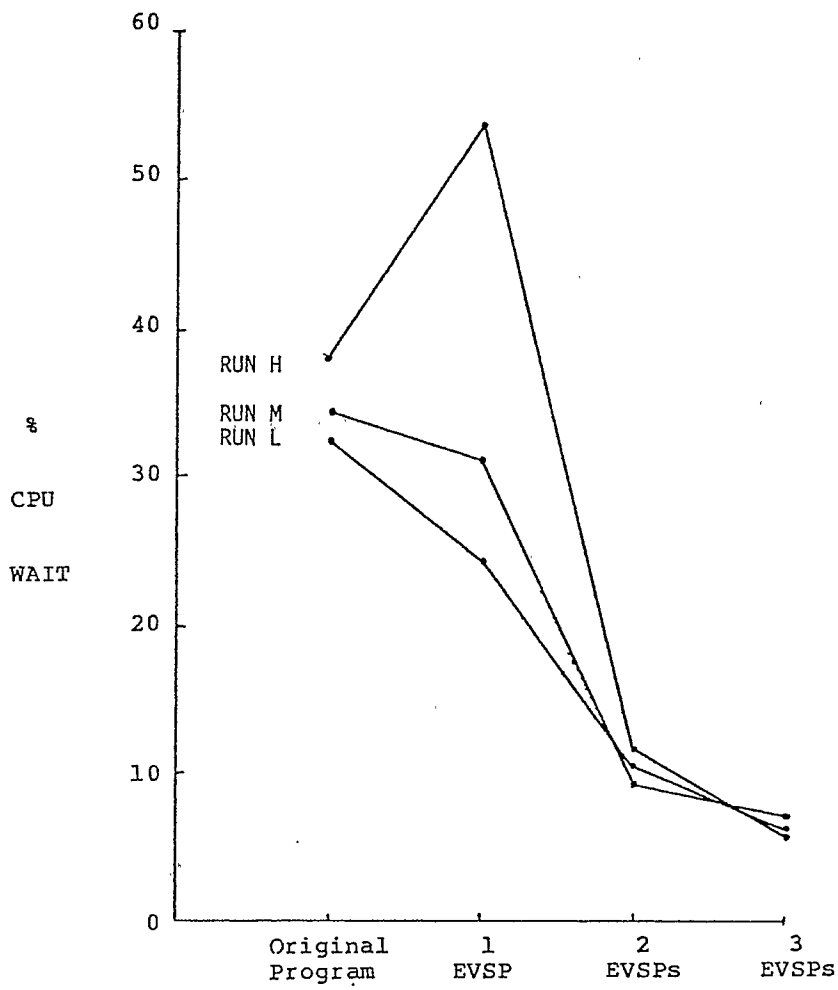


TABLE 4

SIMP WAIT PERCENT AS A FUNCTION OF THE NUMBER OF EVSPS

	% CPU WAIT		
	RUN H	RUN M	RUN L
Original program	38.1	34.2	32.3
One EVSP	53.8	31.1	24.1
Two EVSPs	11.5	9.2	10.6
Three EVSPs	5.3	6.0	5.4

FIGURE 4 - COMPARATIVE PERFORMANCE OF THE ORIGINAL AND PIPELINED COMPUTER SYSTEMS



The results as presented are not "real" in the sense that they are based on instruction counts taken from high level compiled code on a computer whose use as an event set processor would be moderately infeasible. In the next phase of this research a verifiable system will be simulated using a PDP-11/44 as the SIMP, and Motorola 68000 computers as the EVSPs. We plan to use data from this composite computer sytem to validate the simulation program.

It seems reasonable that an extremely cost effective simulation computer system may be designed as a network of dedicated microprocessing elements. Extensions of the simulation program discussed here will be a major tool in the design and refinement of such a system.

ACKNOWLEDGEMENTS

The authors wish to thank Martin Miller for his assistance in the preparation of this manuscript.

REFERENCES

Blackstone, J.M., Hogg, G.L., and Phillips, D.T., "A Two-List Method for Synchronization of Event Driven Simulation" Proceedings of the Fourteenth Annual Simulation Symposium, March 1981, pp.95-101.

Comfort, J.C., "The Simulation of a Microprocessor Based Event Set Processor", Proceedings of the Fourteenth Annual Simulation Symposium, Mach 1981, pp. 17-33.

Comfort, J.C. and Coats, P.K., "Electronic Funds Transfer Networks: The Impace of Performance and Security Considerations", Proceedings of the Thirteenth Annual Simulation Symposium, March 1980, pp. 1-26.

Comfort, J.C., "A Taxonomy and Analysis of Event Set Management Algorithms for Discrete Event Simulation", Proceedings of the Twelfth Annual Simulation Symposium, March 1979, pp. 115-146.

Franta, D., and Maly, W., "An Efficient Data Structure for the Simulation Event Set", CACM 20, 8 (August 1977) pp. 596-602.

Gonnet, G.H., "Heaps Applied to Event Driven Mechanisms", CACM 19, 2 (July 1970), pp. 417-418.

Vaucher, J.G. and Duval, P., "A Comparison of Simulation Event Set Algorithms", CACM 18, 4 (April 1975), pp. 223-230.

Wyman, P.F., "Improved Event Scanning Mechanisms for Discrete Event Simulation", CACM 18,4 (April 1975), pp. 350-353.