

# technical correspondence

## On the Representation of the Simulation Event Set

□ In a recent paper Ulrich [3] reported on a new method, i.e. the 'converging lists' data structure, for maintaining the event set for discrete simulations requiring large numbers of events. We find it necessary to dispute two specific claims made in this paper: (1) superiority over the TL structure [1], and (2) superiority over the indexed list method [4] as it concerns event scheduling.

As to (1), Ulrich deduces from our paper [1], that "at least 100 machine instructions are required to schedule the typical event" and then states "With the algorithm described here, fewer than 30 instructions are required." This statement is to be supported by performance evaluations done by Phillips and Tellier [2]. We have two main observations to make. One, the performance evaluation in [2] measures the time to *schedule* an event, whereas both [1] and [4] measure *hold* time. Two, the results reported in [2] are obtained with an index table of size 512, whereas [1] and [4] use a table size of 30! Seemingly less than a fair comparison. In terms of algorithm analysis, none is given. Actually, the algorithm described in [3] has an  $O(n)$  worst case complexity compared with an  $O(\sqrt{n})$  worst case complexity in [1]. Furthermore, it should be observed that Ulrich, Phillips, and Tellier compare an implementation done in Pascal with an implementation done in BLISS, a system implementation language.

As to claim (2), we fail to find any conceptual difference between Ulrich's approach and the indexed list of [4]. On the implementation level the main differences seem to be:

(a) Ulrich postpones dynamic incorporation of events in the overflow

list into the indexed lists till the events in half the table have been executed.

(b) Ulrich uses the standard compiler optimization technique of loop unrolling and replaces the modulo computation by a shift-and-mask operation.

(c) Ulrich uses a singly linked list and an additional pointer to remember the last node visited in lieu of a doubly linked list.

Concerning point (a) above, Ulrich claims that by postponing the problem it will more or less go away since "Usually, there should be none or only a few" (nodes in the overflow list). How many events will be in the overflow list is strictly a function of the distribution in the event set and the table size and again the worst case is of  $O(n)$  in [3]. In addition we note that by postponing the reorganization, on the average 25 percent of the table will not be utilized and that just before the special events *th1* and *th2* are executed half the table will not be utilized.

Finally we would like to remark on the nonchalant use of large tables and the tacit assumption that in most simulations events occur only at integer multiples of a fixed time interval, which seems to permeate much of the discussion (e.g. "For example, the maximum length of the horizontal list can be forced to time units of 8, 4, 2, or 1. This means that the average scan for scheduling a new imminent event cannot encounter more than 4, 2, 1, or 1/2 [sic] of the already scheduled items.") As an example, consider the bimodal distribution in [4] with 500 events in the event set. A snapshot of the event set showed a range of event times between .5 and 10.0 with 4 percent of that range containing almost all event times. We estimate that Ulrich's algorithm would need a table

of size 2000 with approximately 1900 entries representing empty lists to give a performance comparable to [4], and it has to be remembered that it does take a comparison to find out that an entry in the table represents an empty list.

In summary, although Ulrich's paper contains many worthwhile points concerning the implementation of the event set, the two specific claims of superiority over existing methods do not hold up in light of the  $O(n)$  worst case complexity in [3], an absence of a reasonable performance comparison, and the lack of a clear statement as to what the improvements over existing methods are (beyond minor implementations improvements). It might be the case that Ulrich's method is indeed better than the TL algorithm in specific circumstances. But we believe that the one claiming superiority over an existing method should be the one to prove it—either through analysis or performance evaluation. If a performance evaluation is done it should be a fair one (as was the case for the comparisons with earlier work in [1]); that is, both methods should be implemented on the same machine using the same language, and the same optimization techniques should be employed.

W. R. FRANTA

K. MALY

University of Minnesota  
Minneapolis, MN 55455

1. Franta, W.R., and Maly, K. An efficient data structure for the simulation event set. *Comm. ACM* 20, 8 (Aug. 1977), 596-602.
2. Phillips, D.N., and Tellier, J.G. Efficient event manipulation—the key to large scale simulation. Proceedings, Semiconductor Test Conference, Nov. 1978, pp. 266-273.
3. Ulrich, E.G. Event manipulation for discrete simulations requiring large numbers of events. *Comm. ACM* 21, 9 (Sept. 1978), 777-785.
4. Vaucher, J.G., and Duval, P. A comparison of simulation event list algorithms. *Comm. ACM* 18, 4 (April 1975), 223-230.

### Author's Response:

Franta and Maly are ignoring solid evidence provided in the paper by Phillips and Tellier. Comparing Figures 12 and 13 in that paper with Figure 5 in theirs clearly demonstrates that the Phillips/Tellier implementation of the converging lists algorithm is typically three to four times faster than the reported imple-

mentation of the TL algorithm. This suggests, of course, that the converging lists algorithm is generally superior to the TL algorithm.

Rather than acknowledging this quantitative evidence, Franta and Maly are crying "foul" over several minor issues, are clearly exaggerating the virtually negligible issue of worst-case performance, and are attempting to dismiss important conceptual aspects of the converging lists algorithm as mere implementation detail.

A few specific points are the following:

(1) It is highly unlikely that the use of BLISS represents a large advantage over using Pascal. Whether it does or not, it is absurd to demand use of the same language as a condition for drawing conclusions about the merits of these algorithms.

(2) It is, nowadays, unreasonable to demand that a certain table should be of the same size as a similar table used within another algorithm, particularly if the difference is only 480 words.

(3) Franta and Maly are correct in pointing out that their work was reported in terms of the "hold" (accessing plus scheduling) operation while I and Phillips and Tellier reported the results of scheduling only. However, this difference isn't worth arguing about. Event accessing is invariably done in two to four machine instructions, a negligibly small number in comparison with the 30 to hundreds of instructions required for scheduling with these various algorithms.

(4) There is no question that the TL algorithm could perform better for very large numbers of nonsimultaneous events within a bimodal event distribution when event clusters are far distant from each other. However, this rather contrived worst-case scenario is so improbable that it becomes virtually negligible. I regard this point as merely rhetoric because good performance for typical event distributions (such as the overwhelmingly typical negative exponential distribution) should far outweigh the consideration of extreme

cases of one particular unlikely event distribution.

(5) A good portion of the speed of the converging lists algorithm is due to using a time-wheel table of length  $2^n$ . This fact is vital and permits use of a shift-and-mask sequence rather than a modulo operation to enter the table. The attempt to dismiss this as a minor implementation detail is questionable.

(6) Finally and most importantly: A fundamental feature and basic advantage of the converging lists algorithm is the distinction between simultaneous and nonsimultaneous events. This distinction is absent within the TL and other predecessor algorithms. Franta and Maly must be aware of this difference, but appear to be unable to appreciate its conceptual and practical importance.

ERNST G. ULRICH  
Digital Equipment Company  
Maynard, MA 01754

---

## On Encoding of Line Numbers

□ In his article [1] Paul Klint gives a method to encode the line numbers of a high level language program in unused instructions or unused instruction fields in the translated machine code. This technique is fine when using abstract machine code but raises a problem when using real machine code.

When designing new computer models the manufacturer often makes them compatible with earlier models by using the same instruction codes and cramming new instructions into hitherto unused fields. This means that compilers using Klint's technique will not always be portable from one machine in a series to another even if all other software is. And the translated programs could start showing side effects due to the execution of instructions which had been intended as no-ops.

One could solve the latter problem by retranslating all programs using the changed compilers when shifting to a new computer (even though the new model is compatible

with the old). It would mean some work, however, and one cannot always be sure of having the source program available.

Another solution would be to have computer manufacturers promise to leave certain dummy fields "untouched" in future models, thereby dedicating them to line number-bookkeeping and similar purposes.

JAKOB NIELSEN  
Banegaardspladsen 10, 1. th.  
DK-8000 Aarhus C  
Denmark

I. Klint, P. Line numbers made cheap. *Comm. ACM* 22, 10 (Oct. 1979), 557-559.

### *Author's Response:*

Jakob Nielsen correctly remarks that there is a certain danger involved in employing unused instruction fields for maintaining the line number administration. There is, however, a difference between the use of defined instructions with zero effect (as in the examples I gave) and the use of undefined instruction fields. The former can be used safely, while the latter may have disadvantages as described by Nielsen.

His proposal to leave some unused fields untouched in all future models of a certain computer is unrealistic and does not solve the problem in a fundamental way. In fact, techniques as presented in my paper are a subterfuge and reflect the enormous gap between programming languages and current computer architectures. A better solution than the one proposed by Nielsen would be to integrate line numbers completely (and of course other entities related to the source text level) within the architecture of future computers.

PAUL KLINT  
Mathematisch Centrum  
1091 AL Amsterdam  
The Netherlands

---

*Letters intended for the Technical Correspondence department should be sent to the Editor-in-Chief, Robert L. Ashenurst, The University of Chicago, 5640 South Ellis, Chicago, IL 60637.*

---