NOTE

Which Algorithm Is Better?

D. C. Rapaport notes in [1], that the degree to which the discarded collision schedulings are repeated, is not studied in [2]. To answer the comment, I made the requested measurements. Under the conditions comparable with those suggested in [1] and discussed below, there are 8 to 13% more schedulings counted with multiplicity, than those counted only once. Also, among all the processed events, there may be up to 15% no-collision events [2]. This yields $(100/(100-15))-1=18\%$ overhead. Although the repeated collision schedulings caused by these 18% are already subsumed in the $8-13\%$ overhead, we generously multiply the upper bounds: $1.18\times 1.13=1.33$. The latter bound (computed for the specified conditions) can be spelled out as follows: *execution of a best implementation of* [2] *takes at most one third longer than execution of a best implementation of* [3]. The bound holds despite the fact that in this calculation we ignored computing time spent for the combinatorial data manipulations (choosing next event, maintaining linked-lists and binary trees, etc.), because these manipulations per processed event take longer for [3] than for [2].

If a user has to write a billiard-ball simulation without the assistance of the authors of the original algorithms, shortcuts are likely to be taken. Efficient but complex data manipulation structures in [3] might probably be substituted with simpler, but inefficient ones. As a result, the *actual implementation of* [3] *might be slower than that of* [2], because the latter algorithm is more robust with respect to the data manipulation inefficiency. That is the main message of my response to [1]. The other comments made in [1] are addressed below.

I believe the algorithmic difference between methods [3, 2] is assessed in [1] correctly, except for the statement made in [1] that the mechanism in [2] "does not guarantee monotonically increasing time between events." Indeed, if we ignore the effects of roundoff, then both algorithms under the same input and conditions would process the same collisions in the same time-increasing order, unless encountering degenerated and usually rare cases of multiple simultaneous collisions when both algorithms would experience similar difficulties. However, I have reservations about the premises on which [1] bases its conclusion that the algorithm in [3] "appears substantially faster" than the algorithm in [2].

I quote from [4]:

	[fixed code instance for] $n=100$	[best optimization effort for] $n=1000$
IBM 4381-13	1.2 Mflops	No data
DEC VAX 8550	0.99 Mflops	1.3 Mflops

We see that for the same $n=100$ task, IBM is 20% faster than DEC, contrary to the statement in [1] that equates both computers at 1.2 Mflops and is apparently based on a version of the same report by Dongarra.

The quote from [4] also illustrates a well-known fact that the choice of a task's parameters or code adjustment may substantially change the algorithm performance. One should not compare DEC performance for optimized code and $n=1000$ with IBM performance for fixed code and $n=100$. As to the algorithms in [3, 2], their performance is sensitive to the ratio of a particle diameter to the size of the subdividing cell. Not only [3] confirms this statement by thorough experiments, but it also stresses optimization of performance with respect to the parameters, including the mentioned ratio. Hence one should assume that speed quoted in [1], 670 collisions/s, is an optimized value. By contrast, experiments in [2] were not similarly optimized, because size of particles changed during computations. While the particles were growing, the cells were fixed, and since the final particle size was unknown, the cell size was based on an upper bound for the particle size.

Computations in [2] were arranged as a sequence of sessions; the performance range of 150 to 450 collisions/s quoted in [1] was produced over different sessions. Since the first session started with particles of zero size, the collision rate at first was low, about 150 collisions/s. As the experiment progressed the rate gradually grew. The small initial collision rate is not specific to [2]; the algorithm in [3] would also show small collision rate when particle sizes are sufficiently small. A statement in [1] seemingly to the contrary, namely, that [3] shows even faster speeds at "lower densities" should not be misunderstood.

On DEC VAX 8550, I followed the conditions described in [1] and made several runs with $N=2500$ disks of fixed equal and known size for densities (covering fractions) ranging from 0.698 to 0.876. (The above-mentioned scheduling overhead measurements were done under the same conditions.) The cell size each time was chosen to be the minimum possible. Note that these cell sizes are not necessarily optimal, as is argued in [3], but even for such a choice the range of collision rates appeared much smaller than quoted in [1], namely 400 to 470 collisions/s. Each run was repeated several times and it is interesting that the variations among repetitions of the same run were of the same order as the variations among different runs.

In [2], not only is memory size smaller than in [3], but is also *fixed* per particle. This latter advantage becomes important in parallel processing, where each processing element has a small and fixed memory. A parallel version of [2] is presented in [5, 6], where the simplicity of the data structure facilitates the mechanism of returning to an old state of computation (rollback). Greater convenience of scheme [2] as compared to scheme [3] in a parallel implementation can be seen in the following example. To find quickly the particle with the minimum next scheduled event time, both [2, 3] use a binary tree. However, in [2], for a small number N of particles, a straightforward examining of all N next-event times (recorded in a contiguous array, not via a linked-list as in [3]) is the simplest and most efficient way to perform the task. This is not so for [3], where even for a small N, some form of a list structure must be retained, because of variability of the size of data records. Hence, using [2] on a parallel machine, where each processing element holds only a few particles in its local memory, we can, without losing efficiency, abandon the binary tree in favor of the straightforward search.

It is hard to argue with the remark in [1] that the algorithm in [2] being non-natural, "requires a lengthy discussion to convince the reader of its consistency and correctness," while the algorithm in [1] is "natural." I tried to explain the algorithm in [2] to some colleagues who were not aware of the algorithms [3] or [7] and they could not even see the originality in my algorithm, to such an extent it was clear and "natural" to them. As to the "lengthy discussion" in [2], much of it equally applies to [3] and not all of the discussion is correctness proof. In my opinion, the algorithm in [3] needs a proof of correctness not in a smaller degree, than does the algorithm in [2].

Finally, I should apologize for being unaware of [3] when writing [2]. Moreover, I should concede that I learned about the pioneering algorithm in [7] and subsequent work, e.g., [8], only after my first serial code for billiards simulation was already operational, as I was initially motivated to this task, not by its use in computational physics: I was after an efficient serial simulation procedure appropriate to render it parallel. I think the algorithm in [2] is such a procedure which was also successfully used as a serial code [9, 10].

REFERENCES

1. D. C. Rapaport, *J. Comput. Phys.* **104** (1993).
2. B. D. Lubachevsky, *J. Comput. Phys.* **94**, 24 (1991).
3. D. C. Rapaport, *J. Comput. Phys.* **34**, 184 (1980).
4. J. J. Dongarra, Oak Ridge National Lab. and Comput. Sci., Univ. Tennessee Report CS-89-85, June 30, 1992 (unpublished).
5. B. D. Lubachevsky, in *Proceedings, 1990 Multiconference, Simulation Series*, Vol. 22, No. 2 (Society for Comput. Simulation, San Diego, CA, 1990), p. 194.
6. B. D. Lubachevsky, *Int. J. Comput. Simul.*, in press.
7. B. J. Alder and T. E. Wainwright, *J. Chem. Phys.* **31**, 459 (1959).
8. J. J. Erpenbeck and W. W. Wood, in *Modern Theoretical Chemistry* (B. J. Berne, ed.), Vol. 6B (Plenum, New York, 1977), p. 1.
9. B. D. Lubachevsky and F. H. Stillinger, *J. Statist. Phys.* **60** (5/6), 561 (1990).
10. B. D. Lubachevsky, F. H. Stillinger, and E. N. Pinson, *J. Statist. Phys.* **64** (3/4), 501 (1991).

Received August 3, 1992; accepted August 23, 1992

B. D. LUBACHEVSKY

AT & T Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974