JOURNAL OF COMPUTATIONAL PHYSICS **126,** 152–164 (1996)
ARTICLE NO. 0127

Casting Pearls Ballistically: Efficient Massively Parallel Simulation of Particle Deposition

BORIS D. LUBACHEVSKY,*,[1] VLADIMIR PRIVMAN,†,[2] AND SUBHAS C. ROY‡,[3]

**AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, New Jersey 07974, †Department of Physics, Clarkson University, Potsdam, New York 13699-5820, and ‡Department of Computer Science, College of William and Mary, Williamsburg, Virginia 23185*

Received February 2, 1995; revised October 6, 1995

We simulate ballistic particle deposition wherein a large number of spherical particles are "cast" vertically over a planar horizontal surface. Upon first contact (with the surface or with a previously deposited particle) each particle stops. This model helps material scientists to study the adsorption and sediment formation. The model is sequential, with particles deposited one by one. We have found an equivalent formulation using a continuous time random process and we simulate the latter in parallel using a method similar to the one previously employed for simulating Ising spins. We augment the parallel algorithm for simulating Ising spins with several techniques aimed at the increase of efficiency of producing the particle configuration and statistics collection. Some of these techniques are similar to earlier ones. We implement the resulting algorithm on a 16K PE MasPar MP-1 and a 4K PE MasPar MP-2. The parallel code runs on MasPar computers nearly two orders of magnitude faster than an optimized sequential code runs on a fast workstation. © 1996 Academic Press, Inc.

1. INTRODUCTION

Simulation is the principal way to study the morphology of amorphous layers growing on planar substrates, the subject of interest to material scientists who investigate the adsorption and sediment formation [1, 5–8, 10, 14, 16]; see Section 6 for a discussion.

We simulate a ballistic deposition model [10]. In this model, continuous coordinates X and Y of centers of unit-diameter spherical particles are generated randomly independently and uniformly over a substrate area, one particle at a time, while coordinate Z is initially very large. After a particle is generated, it is "cast" or "dropped"; that is, it moves straight down (X, Y are fixed, Z is decreasing) until it attaches itself to the obstacle met first. The obstacle may be either the planar surface of the underlying substrate (this substrate plane is placed at $Z = 0$), or it may be the surface of a previously dropped particle.

In Fig. 1 (left) we show a sample configuration of 100 particles. Here for expository simplicity we eliminated the Y-coordinate thereby reducing this process to two dimensions, X and Z, and then we generated the deposition over a segment of length 10 particle-diameter units. Unlike this example, interesting simulations are in 3D. In order to yield statistical confidence, the actual runs involve many millions of particles and a substrate area in millions of square units. The 19 balls shown in Fig. 1 (right) is a tiny fragment cut out of such a large configuration.

The main interest in these simulations arises in the field of deposition of submicron particles on substrates. References to the underlying physical systems in colloid chemistry, in biology, and in other fields can be found in [10, 13, 17]. Numerical studies of these and similar systems in 3D and 2D are surveyed in Section 6. All previously developed numerical models and algorithms for studying deposition are to be executed on a sequential computer. The present article is devoted to the details of the numerical procedure which was further developed since the results reported in [10] and became amenable to execution on parallel computers. We also outline the new data available but do not attempt their analysis in the framework of theoretical ideas presented in [10], because this work is devoted to the simulational aspects only, specifically, to techniques of parallel simulations.

However, the presented model of deposition is apparently sequential. Thus, we rendered this process parallel, in order to simulate it on a parallel computer. The way we did it may be of interest by itself. Namely, we changed the notion of time. The sequential deposition process, in which particles are deposited at deterministic discrete instances $t = 1, 2, ...$, is recast into a process where particles are deposited at asynchronous random instances in continuous time $t > 0$. We show that despite the change, the resulted configurations are statistically the same. The new model can be efficiently executed in parallel.

The introduced continuous time can be viewed in two ways: (1) as an artificial mechanism that facilitates computing in parallel the final particle configuration, or (2) as the actual process time during which the deposition is taking

[1] E-mail: bdl@research.att.com.

[2] E-mail: privman@craft.camp.clarkson.edu.

[3] Subhas Roy's research was carried out while he was with AT&T Bell Laboratories. E-mail: subhas@cs.wm.edu.

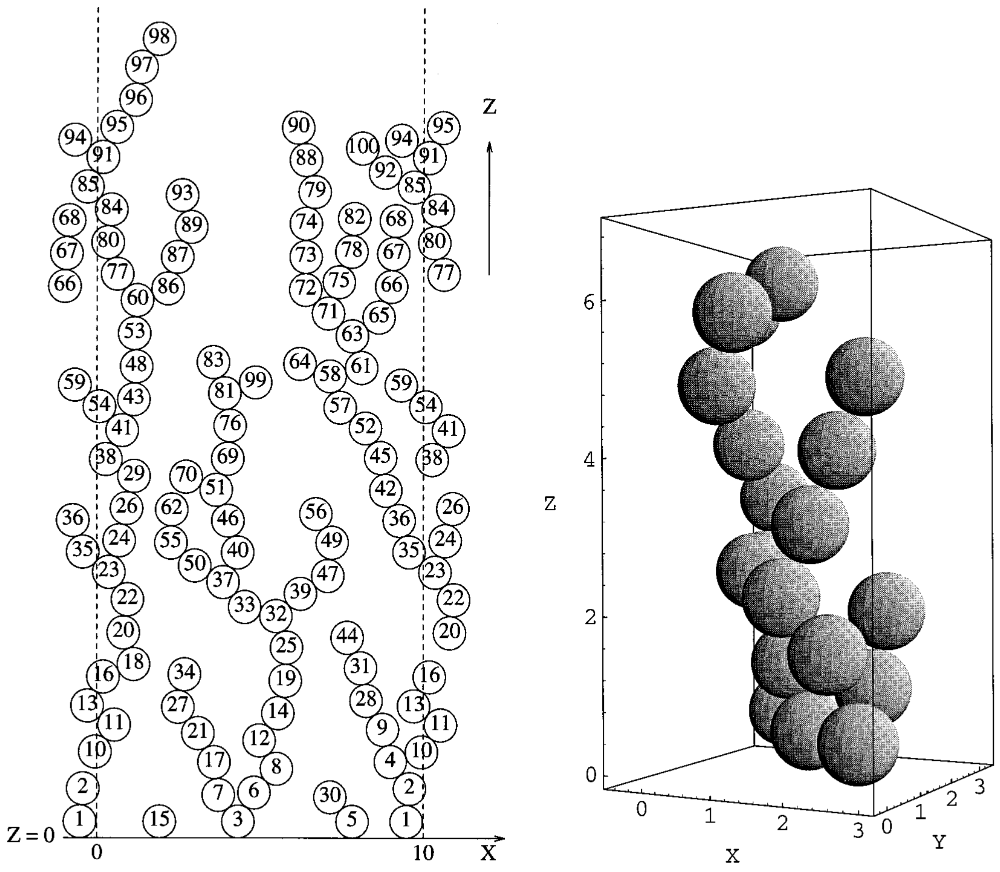

FIG. 1. *Left.* First 100 unit-size disk particles deposited over a horizontal segment of length 10. Particles are indexed with their deposition order numbers so that particle 1 is deposited first, then particle 2, then 3, and so on. The endpoints of the segment are "glued" together to form a periodic substrate, that is, a circle, so that the deposition space becomes the surface of a cylinder. Because of periodicity, some particles are drawn twice, e.g., particle 66. The dashed vertical line at $X = 0$ is also periodically repeated at $X = 10$. *Right.* A 19-ball fragment cut out of a large 3D deposition configuration.

place. The second interpretation is helpful as a natural and convenient base for expressing and evaluating transient, time-dependent properties of the deposition process. Indeed, in the physical process the particle transport to the substrate is via a fixed flux, and the assumption of the sequential model that the incoming particles are noninteracting is only valid for small fluxes, when the particle beam is "dilute."

The scheme of synchronization and lookahead exploitation which we use for simulating the continuous time model in parallel is not unlike a previously described conservative scheme for simulating Ising spins [11, 12]. At an early stage of our research we tried this scheme and it resulted in a certain speedup compared with the execution of an optimized serial program on a fast workstation. However, the speedup (with respect to a workstation execution) was not particularly significant (two- to five-fold on MP-1) and neither was the utilization factor, i.e., the fraction of non-idling processors (around 9%).

In this paper we present both our earlier scheme and its augmentation with several techniques that are aimed at the increase of the efficiency of the simulation. One of those improvements capitalizes on *slackness* of event dependencies over short intervals of time, the technique described in [2]. The new technique of slackness exploitation is applicable in and would improve the performance of parallel simulations of Ising spins [11, 12]. We should mention that our application of the slackness was developed independently of [2]. When exploiting slackness we use what was called in [15] a *future event list.* Our techniques also relate to the methods discussed in [3] in the context of the use of uniformization for parallelization of simulating Markov chains, although, we do not necessarily require the model to possess the Markov property and we do not rely on uniformization.

When the task of simulation proper, that is, the task of *generating* the system trajectory, was sufficiently sped up, the task of collecting statistics, that is, that of *evaluating* the simulation trajectory, became a bottleneck. A difficulty in both statistics collection and configuration generation is the need to handle large particle sets. To accumulate a large set of previously deposited particles, a large memory storage may be needed. This memory may be unavailable. We combine several methods that make it possible to efficiently collect statistics. Among these methods is a "roof" computation technique that overcomes the difficulty of large deposits by identifying unneeded data and then cleaning the storage of it.

The enhancements that we made do not drive our algorithm out of the class of the conservative simulations. This algorithm can efficiently utilize a wide class of existing parallel architectures, but it is an especially good fit for massively parallel SIMD computers. We report an implementation of our parallel algorithm on a 16K processor MasPar MP-1 and a 4K processor MasPar MP-2. The simulation on these MasPar machines runs nearly two orders of magnitude faster than an optimized sequential version when executed on a workstation and the utilization factor raises to 80%.[1] Moreover, on the MasPar machines we have been able to run problems of substantially larger sizes which enabled us to make detailed and more accurate numerical simulation of the deposition process.

The rest of this paper is organized as follows: in Section 2 we describe a sequential model and algorithm for the problem. Then in Section 3 we explain the way we recast the discrete time model in continuous time. Section 4 contains a comparison of the continuous time model with the model of asynchronous Ising spins. In Section 5 we describe enhancements to the original Ising spins simulation algorithm which boost efficiency of the task of producing the simulation trajectory in our model. Section 6 outlines the experiments we did with the ballistic deposition model. Finally, in Section 7, we summarize the present status of the problem of numerical simulation of ballistic deposition and outline a new challenge we see in the parallel simulation of deposition models.

2. SEQUENTIAL MODEL AND ALGORITHM

As described in the Introduction, spherical particles of unit diameter are cast over a planar area. We choose this area to be an $L \times L$ square with periodic boundary conditions, $L \geq 1$. The boundary periodicity means that a particle close to a boundary of the square can land on a particle close to the opposite boundary. In 2D, an example of such a periodic deposition is particle 13 in Fig. 1. The particle is close to the right boundary, but it lands on particle 11 and the latter is near the left boundary. The 3D case has to be obvious also from this example.

Suppose that n particles are already deposited and the coordinates of their centers are $\{(X_i, Y_i, Z_i)\}$, where $1 \leq i \leq n$. To find the coordinates of the next, $(n+1)^{\text{th}}$ particle, we first determine its landing coordinates X_{n+1} and Y_{n+1}. They are generated independently, randomly, and uniformly in the square $L \times L$. Then we compute the attachment coordinate Z_{n+1} of particle $n+1$ as

$$Z_{n+1} = \max\left\{\frac{1}{2},\ \max_{i \in M_n}\left[Z_i + \sqrt{1 - (X_i - X_{n+1})^2 - (Y_i - Y_{n+1})^2}\right]\right\}, \tag{1}$$

where the index set

$$M_n = \{1 \leq i \leq n \mid (X_i - X_{n+1})^2 + (Y_i - Y_{n+1})^2 \leq 1\} \tag{2}$$

represents the particles which might obstruct the free fall of particle $n+1$. In the outer maximization in Eq. (1), the term $\frac{1}{2}$ is the particle radius. $Z_{n+1} = \frac{1}{2}$ if the particle lands on the substrate. Note that Eqs. (1) and (2) are only valid when $1 \leq X_{n+1}, Y_{n+1} \leq L - 1$. We omit here simple modifications to these equations for a particle dropped close to a boundary.

The straightforward coding of this model produces an inefficient program. While simulating the deposition of particle $n+1$, this program wastes order of n computations during the phase of selecting M_n according to Eq. (2) and then it wastes up to order of n computations during the phase of checking all the particles $i \in M_n$ according to Eq. (1).

Two improvements make the computations efficient and feasible for large n. The first one is dividing the substrate area into sectors and dividing the set of all deposited balls into the corresponding subsets, subset i consisting of all the spheres whose (X,Y) coordinates belong to sector i. We choose an integer m, $1 \leq m \leq L$, and using $m - 1$ equally spaced straight lines in the X-direction and, similarly, $m - 1$ lines in the Y-direction, we divide the $L \times L$ square into m^2 equal square sectors. Thus, when selecting M_n, the improved program checks not all the n particles but only the particles that belong to the nine sectors in the vicinity of the fall site; see Fig. 2.

The second improvement is to order the particles in each sector according to the magnitude of their Z coordinates. When the inner maximization in Eqn. (1) is being performed, the particles i with larger Z_i are accessed earlier than the particles i with smaller Z_i. We achieve this ordering by using a linear linked list for each sector. The ordering reduces the search as follows: when a candidate obstacle $i = i^*$ in Eq. (1) is found and the corresponding candidate

[1] By utilization here we mean the average fraction of nonidling PEs during the execution. We indicate both speedup and utilization because a change in speedup is not necessarily proportional to that in utilization. The increase in speedup will be lower than that in utilization, if the PEs do additional work, or it may be higher if code improvements unrelated to utilization are made.

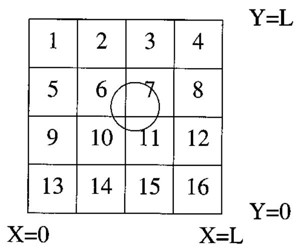

FIG. 2. An $L \times L$ square split into $m^2 = 16$ sectors. When a particle is being deposited in sector 7 only previously deposited particles in sectors 2, 3, 4, 6, 7, 8, 10, 11, and 12 have to be checked as possible obstacles. In real simulations, where $m > 100$, the saving generated by such sectorization becomes substantial.

$Z^*_{n+1} = Z_{i^*} + \sqrt{1 - (X_{i^*} - X_{n+1})^2 - (Y_{i^*} - Y_{n+1})^2}$ is computed and when a particle i such that $Z_i < Z^*_{n+1} - \frac{1}{2}$ is detected, no checking beyond particle i in the list for the given sector is needed.

The contact particle i, i.e., the one that delivers the maximum in Eq. (1), is more likely to be found in the list of that neighboring sector which is closer to the deposition site (X_{n+1}, Y_{n+1}). We exploit this circumstance by offering the neighboring sectors to the checking procedure in the order of the increase of their distance from the deposition site. This ordering further reduces the search for the contact particle. For the deposition example shown in Fig. 2, the sectors are ordered as follows: 7, 6, 11, 10, 3, 8, 2, 12, and 4.

The sequential program which implements these improvements works reasonably fast. When a sector size is equal to the particle diameter, the program, on average, checks only about five particles as candidates in maximization for each particle deposition. Of course, the program also incurs a certain overhead for maintaining sectors and lists, but this overhead, on average, is only a handful of operations per one particle deposition update. Still, the need to produce statistically valid data forces one to run this and similar optimized sequential codes for weeks of CPU on a workstation [1, 5–8, 10, 14, 16]. One could use an obvious replication technique and run multiple deposition processes with differently seeded random number generators on multiple workstations. However, in order to be able to run larger samples and collect time-dependent statistics which did not fit the workstation memory (as opposed to the asymptotic large-time statistics collected in our earlier simulations), on only one MasPar computer, we chose a more interesting and promising method where we rendered each run parallel. The method is described in the following sections.

3. RECASTING DISCRETE TIME DEPOSITION PROCESS IN CONTINUOUS TIME

The area of the underlying substrate is L^2. Let $\lambda > 0$ be an arbitrary constant. The continuous time $t > 0$ is associated with the process of generating the deposit as follows. The value of t is incremented with each dropped particle. Let t_n be the value of time at the instance of depositing particle n, where $n = 1, 2, \ldots$ and we set $t_0 = 0$. Denote $\Delta t_n = t_n - t_{n-1}$ for $n = 1, 2, \ldots$. We assume that the values Δt_n are i.i.d. random variables distributed exponentially with mean $1/(L^2\lambda)$. This construction amounts to attaching to the model a Poisson arrival process with rate $L^2\lambda$ and assuming that the instances of particle depositions are the instances of the arrivals. The augmented model can be thought of as representing a fixed incoming particle flux of λ particles per unit area per unit time.

Because in the new model the process of deposit generation is unchanged, the statistical properties of the deposit are identical between the two models. An advantage of the new model is that it can be thought of in terms of concurrency. The key observation is the following divisibility property. Suppose the $L \times L$ area is arbitrarily split into measurable subareas $S_1, S_2, \ldots, S_k$, where $S_1 + S_2 + \cdots + S_k = L^2$. (We use the same symbol S_i for the *subset* of the $L \times L$ square and for the *area* of this subset.) This partition induces a partition of the particle stream into k separate substreams, where the ith substream is formed by the particles whose centers fall in subarea S_i. Then the time component of each substream inherits the property of being a Poisson process. The rate of arrivals for the ith substream is λS_i. The space component of each ith substream inherits the uniformity of the distribution of fall cites (projections of particle centers) in its subarea S_i. The substreams for different i are mutually stochastically independent.

In our scheme of parallel simulation each substream i is hosted and simulated by a separate processing element, PEi, that samples arrivals of the particles in the ith substream, i.e., their times and (X, Y)-projections. For the sampling the PE uses its own independent random number generator. The times are a sample of a Poisson process with rate λS_i and the projections are samples from the uniform distribution on the subarea S_i. The resulting configuration is stochastically the same independently of the partition into subareas. Specifically, it is independent of the shape of S_i, $i = 1, 2, \ldots, k$, their number k, or their sizes.

Note that the fact that interarrival times of the cumulative process and its subprocesses are distributed exponentially is essential. If the interarrival times of the component processes were distributed differently, for example, uniformly, even with the same means (i.e., maintaining the same rates of particle deposition) as in the exponential case, the texture of the deposit and its statistical properties would have been different. Moreover, the properties of the deposit would not have been invariant of the way the substrate area is split into subareas.

Finally, it is important to emphasize that the breakdown into separate areal streams of arriving particles yields processes which are only statistically independent as far as the flux of incoming particles is concerned. The adhesion events themselves in different subareas are, of course, interrelated at the area boundaries as will be further explored in the next sections.

```
1. wait_until time(i) ≤ min{time(j) : j ∈ neighbors(i)}
2. state(i) ← next_state(state(i), state(neighbors(i)), random_experiment)
3. time(i) ← next_event_time(i, time(i), random_experiment).
```

FIG. 3. The basic conservative simulation algorithm. Each processor PE*i* executes this code repeatedly, until reaching a termination condition, such as time(*i*) exceeding a given threshold.

4. COMPARISON OF THE BALLISTIC PARTICLE DEPOSITION MODEL AND MODEL OF ISING SPINS

The Poisson substreams arriving at different subareas S_i are independent and hence the *times* of particle arrivals are independent of each other. However, different PEs will not be able to simulate different substreams fully independently of each other, because the processing would also involve update of the *states*. What should be considered the *state* in this deposition model? Let us take as the global state at time t the set of all the particles deposited up to this time. That is, the state is the set of coordinates (X, Y, Z) of (centers of) all previously deposited particles. The global state so defined is naturally decomposable into the partial states for all the subareas. The state of the subarea S_i at time t consists of all the particles deposited over S_i by t, that is, those particles whose (X, Y) coordinates of the centers belong to S_i and are deposited by time t.

If a particle with the center coordinates (X, Y) is being deposited over S_i, then to know the landing Z of the particle center, one may need to know, even if partially, the states of those subareas S_j that have a nonempty intersection with the circle at (X, Y) of a unit radius. The latter circle is the locus of possible projections of centers of particles that might obstruct the free fall of the arriving particle. Given that the center (X, Y) of the dropped particle is in S_i, we define the set of all relevant S_j's, excluding S_i, as the set of *neighbors* of subarea S_i, neighbors$(S_i) = \{S_{j1}, S_{j2}, ...\}$.

Now we note that in the scheme of simulating Ising spins [11, 12] there are spin variables S_i located at fixed positions in space (say, on a plane, like in our deposition case). Each spin has a fixed set of neighbors and can be in any of a specified set of discrete states. The state of a spin can be changed at discrete time instances and these changes can be put in correspondence with a Poisson point arrival stream, the streams for different spins being independent. Upon an arrival, the new state of a spin is a function of the current states of this spin and its neighbors with, typically, some randomness (generation of a random number) involved.

The coordination mechanisms of the parallel simulation of Ising spins and of particle deposition are analogous, with a spin S_i being analogous to a subarea S_i, and the state of spin S_i at time t analogous to the set of particles deposited by time t over the subarea S_i. The role of the random experiment in the model of Ising spins, in the deposition model is played by the random choice of the cast particle coordinates (X, Y) in the subarea S_i. The role of the neighboring spins for the given spin S_i in the model of Ising spins in the deposition model is played by the neighboring subareas for the given subarea S_i.

The analogy is between coordination mechanisms of parallel simulations, not between the processes themselves and their dynamics. In particular, a spin value, on one hand, and the set of particles deposited over a subarea, on the other, play similar roles in the organization of parallel calculations. But they are quite different objects and behave differently in their respective models. For example, spin value is a discrete variable while the coordinates of the set of deposited particles belong to a continuum.

The basic program in Fig. 3 simulates either one of the two models. It is assumed in this code that one spin or one subarea S_i is mapped into one PE*i*, with the corresponding "next-event-time" time(*i*) being the time of the next attempt to change spin S_i or the next instance of a particle deposition in subarea S_i. Set neighbors(S_i) is the set of neighbors of the spin or of the subarea as defined above. The task of PE*i* is to execute this code, until reaching the termination condition; e.g., time(*i*) > end_time, where end_time is an input parameter.

The code is correct because PE*i* stalls until the next-event-time of all its neighbors exceeds its own next-event-time. (Equal next-event-times introduce some non-determinism but do not impact correctness. Equality of times of arrivals from two independent Poisson streams has probability 0.) At that point, the neighbors are stalled and their current states, together with the state carried by PE*i* itself, determine the update (line 2 in Fig. 3) by PE*i* associated with the current event. The computation does not deadlock because at least one PE, the one with the current global minimum of next-event-times, does not stall. This PE is guaranteed to advance its next-event-time and to update one spin or to deposit one particle. (This guaranteed worst case performance is substantially exceeded in an average case; see discussion below.)

On line 3 in Fig. 3, PE*i* increments time(*i*) by a sample of the exponentially distributed random variable. In the Ising spins, the variables for all spins i are identically distributed, with mean, say $1/\lambda$. In our deposition case, the mean of the variable is $1/(\lambda S_i)$.

The code of Fig. 3 may be executed asynchronously, e.g., on an MIMD computer, with no coordination between processors beyond that implied by the wait_until directive of line 1. In an asynchronous setting, the wait_until can be implemented by having each PEi cyclically poll the values time(j) of its neighbors, at each cycle recomputing the minimal value. The system trajectory is guaranteed to be independent of the timing and polling order, as long as no two of the next-event-times tested in an execution of line 1 are equal.

The work of simulation is done in line 2, which must be customized for each specific model, either that of Ising spins, or that of particle deposition. For instance, in the simplest, nearest-neighbor 2D Ising model, the neighborhood of each spin consists of one spin in each of the four directions: North, East, South, and West. An explicit function (one-line formula; see an example in [11]) of the five spins (including the spin attempting the change) and of a random number drawn from a specific distribution realizes the next_state.

On the other hand, in our deposition model, assuming that $k = m^2$ PEs are available for the task, we split the $L \times L$ substrate area into $k = m^2$ equal subsquares each with sides of length L/m. The $m \times m$ square processor grid of Maspar computers (128×128 grid in a 16K PE MP-1 and 64×64 grid in a 4K PE MP-2) suits ideally for hosting the $L \times L$ substrate, where the PE with grid coordinates i, j, $0 \leq i, j \leq m - 1$, hosts square $iL/N \leq X < (i + 1)L/N$, $jL/N \leq Y < (j + 1)L/N$.

We will also assume that $L/m \geq 1$. Then the neighborhood of a subsquare consists of eight subsquares: four in the North, East, South, and West directions as in the simplest Ising model, and the other four in the North–East, South–East, South–West, and North–West directions. Communication of a PE with its neighbors, which is required in the algorithm, is effected via the so-called *X-net*. The net provides a fast concurrent access to the memories of the eight neighboring PEs.

Function *next_state* here involves first sampling the X, Y for the incoming particle and then the algorithm of selecting the obstacle (a previously deposited particle or the substrate plane) met first. The selection algorithm is comparably lengthier than the one-line formula for *next_state* in the case of Ising spins. At the least, and as a particular case, it includes the entire sequential algorithm (described in Section 2).

The most obvious way to tailor the code in Fig. 3 to an SIMD execution, e.g., to the execution on a MasPar computer, is to execute each line in Fig. 3 in lockstep. Then the effect of line 1 is to simultaneously mask out those PEs whose *times* are not minimal in their neighborhood. The remaining PEs execute lines 2 and 3 in lockstep.

What would be the utilization in such an algorithm, i.e., the average fraction of active PEs at an iteration? (An active PE is the one that advances its next-event-time and updates one spin or deposits one particle.) Experiments

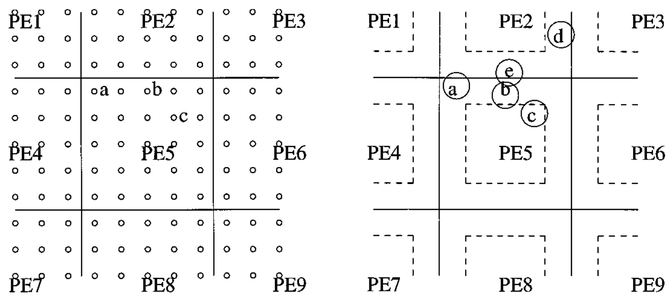

FIG. 4. Fragments of simulation area in the model of Ising spins at the left and in the deposition model at the right. The area is split among the PEs, nine of which are shown in each case. Depending on the site of the event, a, b, or c, different sets of neighboring PEs are to be polled by PE5. No PE has to be polled for spin change or a particle deposited at c. When PE5 is depositing particle b, even if the time of dropping particle d by PE2 is smaller than that of b, PE5 might not need to wait, because particle d cannot obstruct the free fall of particle b.

with the simplest Ising spins in 2D [11, 12] yield 12%. The utilization in our deposition case has to be lower because each subarea has eight neighbors to wait for in line 1, not four neighbors as in the case of Ising spins.

A method to raise utilization by aggregation is discussed in [11]. If each PE hosts a block of neighboring spins, then the set of neighbors PEj with which PEi negotiates the time of a coming event is reduced, if compared with what line 1 in Fig. 3 suggests in the one-spin-per-one-PE case. This technique of reducing the set of neighbors is exemplified in Fig. 4 on the left. Here each PE hosts 25 spins. When PE5 attempts to update spin a, it may check only local times of PE2 and PE4 and if both times are greater than the arrival time for update of spin a, then the update can be safely performed. For updating spin b it suffices to check only the local time of PE2. The update of spin c needs no negotiations with neighboring PEs; the set of neighbors with which PE5 should negotiate the next event time is empty.

A similar method can be used in the deposition model. Suppose we assign a 5×5 square sector for each PE, as shown on the right in Fig. 4. To deposit particle a, PE5 checks the time of three neighboring PEs: PE1, PE2, and PE4. To deposit particle b, it is enough to check time with PE2. To deposit particle c, no negotiation with the neighboring PEs is needed.

Note that unlike Ising spins, where we can only incorporate in a block an integer number of spins, in the particle deposition model we may take subareas of arbitrary sizes. We can increase or decrease a subarea by an arbitrarily small quantity. While one PE carries at least one spin in the Ising model, there is no limit on how small a subarea carried by a PE should be in the particle deposition model. It is not recommended, though, for a PE to carry a subarea smaller than a unit square, because then the subarea has too many neighbors and utilization is reduced. Shapes of subareas can also be arbitrary. The optimum shapes for subareas are known to be equal regular hexagons. Sacrificing the exact optimality, we cover the substrate area with equal square subareas. Squares yield a simpler indexing scheme than hexagons, a relief for the programmer.

We implemented this neighborhood reduction technique in an earlier version of our parallel deposition simulation. On 128^2 PEs of MP-1, we simulated deposition on a $L \times L$ square substrate area with $L = 256$, so that each PE hosted a 2×2 square. The obtained speed improvement with respect to a fast workstation was five-fold, and the utilization factor (an average fraction of active PEs per iteration) was 9%. The utilization factor increased when we took larger L, and this is easy to explain looking at Fig. 4; the case of particle c, whose deposition needs no coordination with the neighbors, became more frequent. However, the absolute speed improvement with respect to a serial workstation execution decreased with the increase of L. For $L = 1024$ estimated speed improvement with respect to a workstation would be only about two-fold.[2] In the following section we explain this speed degradation and describe the algorithm modifications that counter it.

5. BOOSTING THE PERFORMANCE OF THE PARALLEL CODE

We made several program modifications which significantly improved the performance as compared to our earlier version. The improvement holds over the range of sizes L of the substrate area; for an interesting value $L = 1024$, the improved code works nearly two orders of magnitude faster than an estimate for the best serial code executing on a workstation.[2] Those improvements are discussed below.

5.1. *Exploiting State Slackness*

Line 1 of the basic algorithm in Fig. 3 requires PEi to negotiate the safety of processing the next deposition at time(i) with *all* its neighboring PEj. Section 4 explained how the set of neighbors involved in this negotiation can be reduced if we take into account the specifics of the upcoming state change by PEi.

It turns out that we can further relax the requirement of the time negotiation between the PEs, if we take into account the upcoming state and *time* update by the neighbor PEj. The key observation is that some types of changes in the state maintained by PEi are independent of the state maintained by PEj even if, in general, the dependence takes place. The independence or dependence, in turn, depends on the specific current states carried by both PEs. This phenomenon was named *state slackness* in [2], where it was demonstrated in the context of cellular updates for wireless communications.

We discovered and exploited state slackness in the context of our deposition model. Note that in exactly the same way state slackness manifests itself in the model of asynchronous Ising spins [11] (which was not noticed in [11]), and the algorithms in [11] can be improved by exploiting this property in the same way as discussed below.

An example of state slackness is given in Fig. 4, on the right. Here we suppose that PE5 is about to deposit particle b. Because b is close to the area hosted by PE2, deposition of b might be affected by particles deposited by PE2 earlier. Thus, when PE5 compares its time(5) with time(2), if time(5) $\leq$ time(2), then PE5 can safely deposit particle b. However, even if time(5) $>$ time(2), it might still be safe for PE5 to deposit particle b. For instance (see Fig. 4), suppose (A) time(2) = $t(d)$ is the time of depositing particle d and d is far from b and (B) the time of the arrival following $t(d)$ exceeds time(5); this next arrival may be, for example, that of particle e.

The algorithm exploits the state slackness in a systematic way. The procedure is as follows. A positive integer q is chosen and fixed. In its future event list each PEi keeps[3] q consecutive future particle arrivals: $(X_{i1}, Y_{i1}, t_{i1}) \rightarrow (X_{i2}, Y_{i2}, t_{i2}) \rightarrow \cdots \rightarrow (X_{iq}, Y_{iq}, t_{iq})$, where $t_{i1} < t_{i2} < \cdots < t_{iq}$ are the arrival times, and X_{ir}, Y_{ir} ($r = 1, ..., q$) are planar projections of the corresponding arriving particle centers. At each cycle, PEi attempts to process the earliest arrival (X_{i1}, Y_{i1}, t_{i1}) from the list; that is, it attempts to determine the corresponding Z_{i1}. This determination is done on the same footing as is done in the algorithm in Fig. 3; that is, particle is deposited at time(i) $= t_{i1}$ if it is safe to do so. However, the safety criterion is substantially relaxed as compared with the criterion in line 1 of the algorithm in Fig. 3.

Here is the procedure to determine the safety of the deposition: at first, PEi determines (not the full, but) the reduced set of neighbors $\{j\}$ that corresponds to the position (X_{i1}, Y_{i1}) of the particle to be deposited; this was explained at the end of Section 4. Then for each PEj in the reduced set of neighbors, PEi computes flag safe(j), safe(j) = yes or safe(j) = no:

1. If for some r, $1 \leq r \leq q$, we have $t_{j1} \leq t_{j2} \leq \cdots \leq t_{j,r-1} <$ time(i) $\leq t_{jr}$ and all $r - 1$ positions (X_{j1}, Y_{j1}), (X_{j2}, Y_{j2}), ..., $(X_{j,r-1}, Y_{j,r-1})$ are at a distance more than the particle diameter (a unity) from the position (X_{i1}, Y_{i1}), then safe(j) $\leftarrow$ yes.
2. If there exists an r, $1 \leq r < q$, such that $t_{jr} <$ time(i)

[2] Case $L = 1024$ did not fit the workstation memory, so we had to extrapolate the workstation performance in this estimation.

[3] The future event list construct was so named in [15]; it was also discussed in [11].

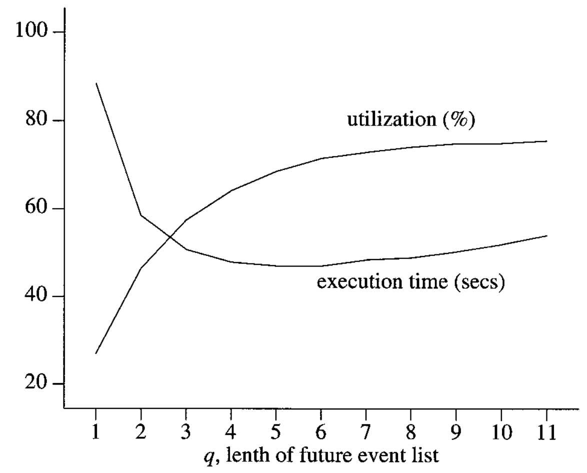

FIG. 5. Utilization and execution time as functions of future event list length q in an experiment with deposition of 5,000,000 particles over a 256 × 256 substrate area executed on a MP-2 with 4K PEs.

and the corresponding position (X_{jr}, Y_{jr}) is at a distance less than the particle diameter from the position (X_{i1}, Y_{i1}) or if $t_{jq} < \text{time}(i)$, then safe(j) ← no.

It can be seen that cases 1 and 2 above exhaust all the possibilities and have an empty intersection.

Now, if either the reduced set of neighbors is empty, or safe(j) = yes for each neighbor PEj in the reduced set, then it is safe to deposit particle at position (X_{i1}, Y_{i1}) for time(i) = t_{i1} using the currently known state, i.e., the set of deposited particles stored in the memory of PEi and its neighbors PEj. In such a case, PEi computes the landing Z_{j1}, presimulates the next arrival (the one that follows the last arrival in the list t_{iq}), makes it the last in the list, and eliminates the just-processed arrival t_{i1} from the list. (The list is implemented as a q-slot circular buffer.) The new list still has q items with arrival times ordered as before. On the other hand, if for some neighbor PEj in the reduced set of neighbors, we have safe(j) = no, then PEi stalls, waiting for the next cycle to start the checking again.

As an exercise, the reader may verify that for $q = 1$ the described procedure of checking degenerates to the original algorithm of simulating Ising spins with the reduced set of neighbors, as described at the end of Section 4. For $q = 2$ this is a truly different algorithm, and its utilization is substantially higher than that of the algorithm with $q = 1$. As q increases, the utilization keeps its increase too. However, we do not want to have too large a q, because of the overhead computations needed for maintaining the future event list, the larger the q the more the overhead. This overhead begins to outweigh the advantage of further increase in the utilization after $q = 5$ in the timing experiment presented in Fig. 5. In this experiment 5,000,000 particles are deposited over a 256 × 256 substrate area. The experiment is exercised on a MP-2 with 64 × 64 processor grid. The 4 × 4 sector hosted by each PE is split into 16 subsectors as described in the next section.

5.2. *Subsectorization*

Speed degrades in our earlier version of the parallel program when L increases, because with larger L each PEi has a larger subarea S_i to host. When a particle is being deposited over S_i, PEi scans the set of particles previously deposited over the entire subarea S_i, thereby introducing the same inefficiency that existed in the straightforward serial program (Section 2).The remedy is the same as in the serial code: subsectorize.

In the new program version we subdivide each of the m^2 square subareas S_i into square subsectors with side not smaller than the particle diameter. For each subsector a separate list of particles is maintained. These particles are deposited over this subsector and ordered by the magnitude of their Z-coordinate. Timing experiments presented in Fig. 6 clearly indicate the best way for the subsectorization; the side of a subsector should be as close as possible to particle diameter. With the side equal to the particle diameter, only nine subsectors and, on average, only about five candidate particles are to be checked when a particle is being deposited, like in the serial algorithm.

In all the runs presented in Fig. 6 the utilization factor is 78.5%. The utilization is independent of subsectorization because the number of the particle deposited, while it may be different at different iterations of the same run, is the same for each iteration number across the runs with different subsectorizations. Indeed, the same particles are deposited across the runs at each iteration number. This is so

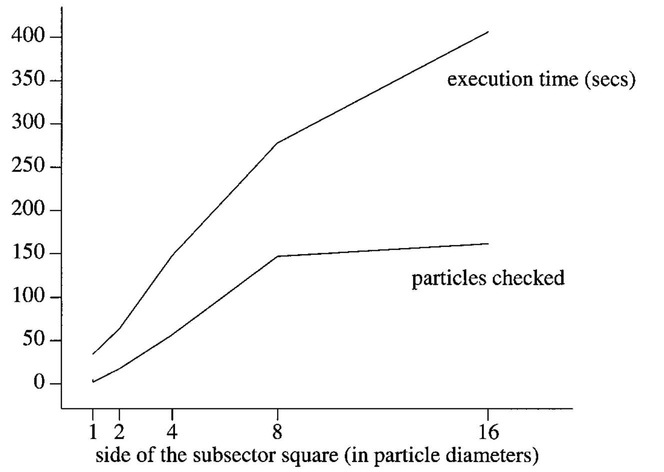

FIG. 6. The average number of particles checked when a new particle arrives and the total execution time as functions of the subsector square side in an experiment with deposition of 5,000,000 particles over a 1024 × 1024 substrate area. The experiment is run on a MP-2 with 4K PEs. Parameter of slackness $q = 4$.

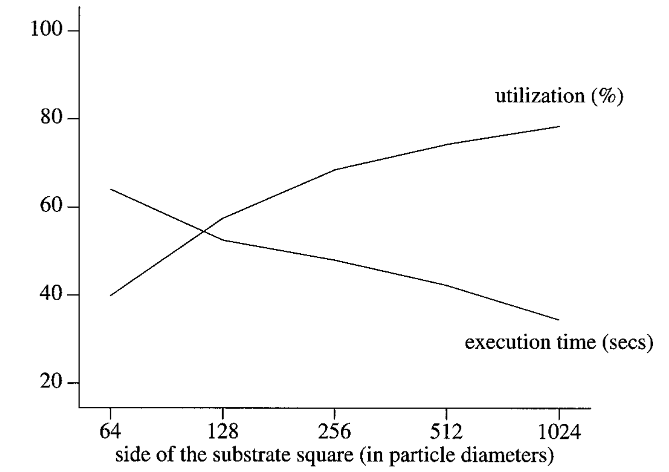

FIG. 7. Utilization and execution time as functions of substrate size in an experiment with deposition of 5,000,000 particles executed on a MP-2 with 4K PEs. The subsector side is equal to particle diameter and parameter of slackness $q = 4$.

because whether or not a particle can be safely deposited is defined by the set of particles scheduled for the deposition including their scheduled times. This set, in turn, is defined by the sequence of pseudorandom numbers that feed the experiment. The latter is the same in the presented runs. For a different random sample, the utilization factor is different, but close to 78%; similarly close is the shape of the dependences.

Note that some of the neighboring sectors/particles might be hosted by the neighboring PEs. Those PEs, even if they are not advancing their local times at some iterations (that is, they are idling, according to our definition of utilization) may indeed be doing useful work, like helping their neighbors to check the particle lists. (This list checking is unavoidable even in an efficient serial simulation.) In this sense the utilization we provide should be considered a lower bound.

Figure 7 presents a series of runs with diminishing substrate sizes. In these runs, the previously considered improvements are implemented, i.e., subsectorization with the optimum subsector size and the use of slackness with the best parameter q. The graph shows that increase in utilization with the increase of the substrate size is accompanied by the improving performance. This behavior of the improved algorithm is in contrast with that of the earlier algorithm version, where the increase in utilization is accompanied by the degrading performance (see the discussion at the end of Section 4). Note that the runs presented in Fig. 6 and Fig. 7 do not include statistics collection or roof computations discussed in next sections.

5.3. *Reclaiming Memory Using the Roof Computation*

While simulating the deposition of new particles, the algorithm also "deposits" new data into the memory and allocates extra space. The limit of memory space puts a limit on the time length of the simulation run and on the thickness (in dimension Z) of deposit that can be generated. The limit is decreased when the statistics collection is introduced—the latter needs substantial memory space too. The memory restriction is especially unpleasant in the SIMD environment when each PE has its own predefined memory space. As soon as one PE exhausts all its memory, for example, because of a locally dense configuration, the run is stopped, despite the fact that the memories of most other PEs may have plenty of space available.

Particles deep down in the deposit are unlikely to obstruct the free fall of new drops and one might hope to safely discard those, thereby alleviating the memory shortage. Specifically, let T be the set of particle deposited so far. We say that a given particle (X_i, Y_i, Z_i) is *exposed* in set T if there exists a position (X, Y) such that a new arriving particle at (X, Y) attaches itself to particle (X_i, Y_i, Z_i). A particle that is not exposed in set T is called *screened* in set T.

After needed statistics are computed, the screened particles can be safely eliminated from further simulation and the exposed particles will constitute the *roof*. Unfortunately, it is not easy to compute whether a given particle is exposed or screened. Instead, we adopt an approximate but more operational definition of the *roof*. However, the roof in the new definition may be nonminimal; it consists of all exposed particles and may include some screened particles.

In this definition, we cover the substrate area by a mesh of equal square sectors (a priori, these sectors have nothing in common with the sectors we considered earlier). Given a set T of particles deposited so far, for each sector s we search for the particles $(X, Y, Z) \in T$ with projections (X, Y) in sector s. Let P_s be the set of such particles. If $P_s \neq \varnothing$ for all s, then each P_s delegates one particle to the roof, the one with the largest Z coordinate among the particles in P_s. The minimum of coordinates Z of the delegated particles is computed among all sectors s; let it be $Z_{\min}$. The roof consists of all particles in T with $Z \geq Z_{\min}$. If $P_s = \varnothing$ for at least one sector s, the roof is not defined; in other words, the algorithm of roof computation returns no roof.

For the roof to be impenetrable, the sector mesh should be sufficiently fine. Let h be the mesh spacing; that is, h is the side length of a square in the mesh. How small should this h be to assure a "quality" roof? Figure 8 shows a fragment of a mesh with the largest possible "hole." It follows from the figure that the largest inserted circle e has radius $r = h\sqrt{2} - \frac{1}{2}$. Because the falling particles have a unit diameter, to assure that the roof stops all of them, the condition $r < \frac{1}{2}$ has to be imposed. This implies $h < 1/\sqrt{2}$.

Recall that we have already subsectorized each area

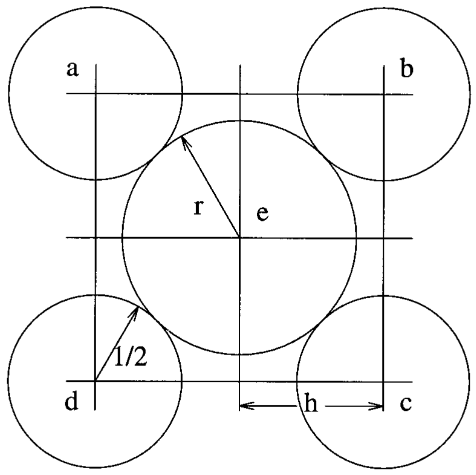

FIG. 8. A fragment of a plane that is split into equal square sectors with side h. Each square holds inside a center of a unit-diameter circle, so there is one-to-one correspondence between sectors and unit circles. The unit circles may overlap. We are trying to place an additional circle e with the largest possible diameter so that e would not overlap with the unit circles. While fitting in, circle e pushes four neighboring unit circles a, b, c, and d into the opposite corners of the corresponding four neighboring sectors.

hosted by a PE into squares of a unit side. So if we choose $h = \frac{1}{2}$, this, on one hand, satisfies the requirement $h < 1/\sqrt{2}$ and, on the other hand, it conveniently agrees with the already existing subsectorization. Namely, we subdivide each unit-side subsector into four equal roof-mesh square sectors with side $\frac{1}{2}$.

There is no need to maintain separate particle lists for roof-mesh subsectors and the roof computation and memory reclamation proceed as follows. Memory limits are set for each PE. Once any PE attempts to exceed the limit, all the PEs interrupt their further simulation advancement and begin the roof computation, wherein each PE scans in turn each of its unit-side subsector lists.

The lists are scanned in the order of the decrease in Z-coordinate of the particles. For each particle in the list, the PE marks as "occupied" a roof-mesh subsector. The subsector marked is the one (one of the four) on which the center of the particle projects. Initially all roof-mesh subsectors are "non-occupied"; the mark "occupied" can be placed repeatedly into a roof-mesh subsector. However, when mark "occupied" is placed for the first time, which is detected as the change from "nonoccupied" to "occupied," the particle that makes this change is marked as the "delegate" particle. Once all four roof-mesh subsectors are marked, the PE goes on to the next main subsector. The PE is done when all its roof-mesh subsectors are marked "occupied." After (and if) all the roof-mesh subsectors are marked "occupied" by all the PEs, the minimum of all Z coordinates of all the delegate particles, $Z_{\min}$, is computed, and then all the particles which have $Z < Z_{\min}$ are discarded and the memory is reclaimed.

We note that $Z_{\min}$ is somewhat analogous to the familiar notion of *global virtual time* in *time warp* [4]. In particular, whereas no rollback can propagate back in time beyond the GVT in the TW algorithm, in the deposition algorithm, no particle can fall below $Z_{\min}$. The latter property is helpful in collecting "final" statistics in items 1 and 2 in the following section 6.

We shall also note that it would be incorrect to limit the roof to delegate particles only, one from each sector s. Such a depleted roof would be able to stop any upcoming particle at some $Z > 0$, but this attachment value Z might be incorrect (too small).

It is, in principle, possible for some PE not to be able to "occupy" all its roof-mesh subsectors (for example, if memory available to each PE is too small in comparison with the problem size or if, as the run continues, the deposit structure becomes less uniform). Then the simulation would have to stop. It is quite easy to design an adversary deposit texture which would defeat in this manner our memory reclamation scheme for any given size of the PEs memory. (For example, this texture: a single column of particles that are being deposited one on top of the other.) On MP-1, where each PE had at least 32K bytes of memory (extensible to 48K bytes), the scheme has been never defeated during our experiments. (And we used limit parameters that were just fractions of full available memory sizes.) We ran the deposition on 2048×2048 substrates in excess of "thickness" $Z = 1000$. The number of particles deposited in between two consecutive memory reclamations were of the order of 10^7. Thus, we conclude that the described memory reclamation mechanism apparently eliminates dependence on the memory size in the deposition experiments. With the memory reclamation, the algorithm slowed down about twice.

5.4. *Combining Lists*

Consider the following programming fragment which is executed concurrently by all or some of the PEs of the parallel computer:

```
1. for each subsector $s_{ij}$ in a set $I_j$ for the given PE$j$ do {
2. for each particle in the list of subsector $s_{ij}$ do work }
```

A PEj that executes this fragment selects a subsector s_{ij} out of the set I_j of subsectors in line 1. In line 2 for each particle in set $P_{s_{ij}}$ of particles found in s_{ij} the PE does *work*. The *work*s performed by different PEs are assumed to be independent of each other. The independency may be assured, for example, by the safety mechanism in line 1 of Fig. 3. In that instance *work* consists of checking whether or not the particle whose deposition is being attempted by PEj can land on a particle being scanned in subsector s_{ij}. The PEs whose *work*s are not safe to do simply stall.

Because of independence, there is no need for synchronization while executing this phase. In particular, a PE that has exhausted its current set $P_{s_{ij}}$ does not need to wait for other PEs to exhaust their sets $P_{s_{ij}}$. A straightforward implementation of this construct on an MIMD computer automatically assures that different PEs do not wait needlessly. In contrast, in a straightforward SIMD implementation, lines 1 and 2 are executed in lock-step and this entails unnecessary idle waitings.

With a little sophistication it is possible to emulate the no-wait MIMD-style execution on an SIMD computer. Namely, each executing (nonmasked) PEj at the end of each *work* checks whether or not this *work* was for the last particle in the current $P_{s_{ij}}$ and if it was the last one, PEj switches to the next $P_{s_{ij}}$. This switch is possible to implement on SIMD computers despite the fact that different PEs must be executing the same instruction, if nonmasked. Indeed, the operands in the instruction may have different addresses with respect to the local memories of the PEs. Differently modifiable addresses have been supported in current generations of SIMD computers: the MasPar MP-1 and higher versions and the Thinking Machine CM-2 and higher versions. The SIMD flavor is not fully eliminated; a PEj_1 that does not switch would wait for a PEj_2 that does switch, but the switch itself takes very little time.

To evaluate the benefit of the improvement, consider a model of this situation. In the model we assume that the switch/nonswitch addition to *work* is negligible, N PEs are executing (say, $N = 16{,}384$), and there are n sectors to be checked by each PE (say, $n = 9$). Let w_{ij} be the sum of all *work*s to be performed for sector s_{ij}. This w_{ij} can be proportional, for example, to the number of particles checked in sector s_{ij} and it is reasonable to model all w_{ij}'s as i.i.d. random variables. The expected cost of execution in the SIMD fashion is

$$C_{sm} = E \sum_{1 \le i \le n} \max_{1 \le j \le N} w_{ij}, \tag{3}$$

whereas the expected cost of execution in the MIMD fashion is

$$C_{ms} = E \max_{1 \le j \le N} \sum_{1 \le i \le n} w_{ij}. \tag{4}$$

We are to evaluate the improvement ratio C_{sm}/C_{ms} which tells how many folds the expected cost of the MIMD-fashion execution is faster than that of the comparable SIMD-fashion execution.

Clearly, the improvement depends on the law of distribution of w_{ij}. For example, there is no improvement, i.e., $C_{sm}/C_{ms} = 1$, if the distribution is singular; $w_{ij} = \text{const}$. Let us evaluate the improvement for two nonsingular distributions: uniform on the interval (0, 1) and exponential with mean 1. In the computations we use the following result in [9]:

Let $W_1, \ldots, W_N$ be i.i.d. random variables with common distribution function F and let $W_{\max} = \max_{1 \le i \le N} W_i$. Under certain mild restrictions, that hold in both cases, of exponential and uniform distributions, $EW_{\max}/F^{-1}(1 - N^{-1}) \to 1$ as $N \to \infty$.

Thus, we evaluate $\max_{1 \le j \le N} w_{ij}$ in Eq. (3) by replacing it with equivalent large-N expression $F^{-1}(1 - N^{-1})$. In the uniform case, where $F(x) = x$, this yields $1 - N^{-1}$ and in the exponential case, where $F(x) = 1 - e^{-x}$, this yields $\log N$. Thus, C_{sm} in the uniform case evaluates to $n - n/N$ and in the exponential case it evaluates to $n \log N$, both evaluations holding asymptotically for large N.

$\sum_{1 \le i \le n} w_{ij}$ in Eq. (4) is a random value. In the uniform case its distribution function is $F(x) = x^{n-1}$ for $0 \le x \le 1$. The distribution is concentrated on the interval $0 \le x \le n$ and it is symmetric with respect to the middle point at $x = n/2$. Hence we have $F(x) = 1 - (n - x)^{n-1}$ for $n - 1 \le x \le n$ and the solution to the equation $F(x) = 1 - N^{-1}$ is $x = n - 1/N^{1/(n-1)}$. This is what C_{ms} evaluates to asymptotically for large N. The improvement ratio is $C_{sm}/C_{ms} = (1 - 1/N)/(1 - 1/nN^{1/(N-1)})$ in the uniform case. The ratio is only slightly larger than 1, i.e., no improvement. For example, for $N = 16{,}384$ and $n = 9$ we have $C_{sm}/C_{ms} = 1.03$.

In the exponential case the distribution function of $\sum_{1 \le i \le n} w_{ij}$ in Eq. (4) is $F(x) = 1 - e^{-x}(1 + x/1! + x^2/2! + \cdots + x^{n-1}/(n - 1)!)$. We find the solution of the equation $F(x) = 1 - N^{-1}$ numerically in this case. The improvement here is significant. For $N = 16{,}384$, and $n = 2, 3, 4, 5, 6, 7, 8$, and 9 the improvement C_{sm}/C_{ms} found from the model in Eqs. (3)–(4) is, respectively, 1.58, 2.01, 2.35, 2.64, 2.88, 3.09, 3.28, and 3.45.

As mentioned earlier, in many instances the variable W_{ij} is proportional to the number of particles deposited in subsector s_{ij}. The distribution of such W_{ij} is determined by the texture of the growing deposit, and it seems to resemble the exponential distribution. We systematically use the technique of combining lists and this speeds up the computations significantly.

6. DEPOSITION EXPERIMENTS

In this section we briefly review the literature on ballistic deposition and report some new simulation results. It is important to emphasize, however, that the main purpose of this paper has been a systematic exposition of simulation methodology. Thus, we do not attempt data analysis within the materials science application framework, nor do we

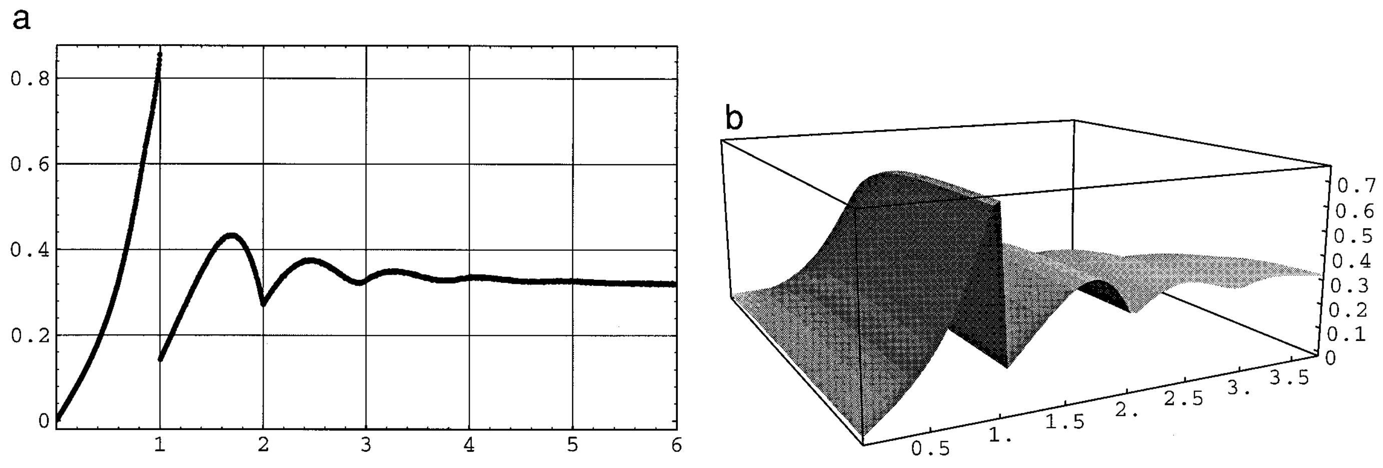

FIG. 9. (a) Particle density as a function of $H = Z - \frac{1}{2}$; H is changing horizontally. (b) Particle density as a function of both H and deposition time t. H is changing along the lower front horizontal edge of the box; t is changing along the lower left horizontal edge of the box. The rear facet of the box is the plane $t = 0$. Density is changing vertically in both (a) and (b).

review the appropriate theoretical models used in data interpretation. Numerical simulations of deposition processes have a long history, reviewed, e.g., in [13, 17]. Recent simulations of ballistic deposition, on both linear and planar substrates, were largely focused on the properties of growing surfaces far from the wall; these include the works [1, 5–8, 10, 14, 16]. Besides the simplest ballistic deposition considered here, some of these authors study various restructuring processes. Our work [10] also yielded some results on the deposit morphology near the wall.

The following are among the quantities a material scientist might want to obtain from this simulation:

1. In the final deposit, how the particle density changes with $H = Z - \frac{1}{2}$.
2. In the final deposit, how the density of particles, that have a specified number of contacts with other particles or with the substrate, changes with H.
3. Items 1 and 2 as a function of the deposition time. E.g., we should generate a function of two arguments $p(H, t)$, so that for a fixed $t_0 > 0$, $p(H, t_0)$ shows how the particle density changes with H in the deposit which exists at time t_0.
4. Items 1 and 2 in the limit $H \to \infty$.
5. Density and other statistics for particles that are in direct contact with the substrate.

Quantities in 1 and 2 were discussed in detail in [10]. Specifically, infinite-time asymptotic density and contact statistics have an interesting singular behavior near integer values of H. Our new results improve the statistics of the earlier study and also provide data on the time dependence. For instance, Fig. 9a displays statistics in item 1 and 9b displays statistics in item 3. However, detailed analysis of the data collected is outside the scope of this work and it will be reported elsewhere.

A performance figure: on a 16K PE MasPar MP-1216, our program takes 620 seconds to deposit 100 million particles on a 1024×1024 substrate. The run includes statistics collection and uses memory reclamation.

7. CONCLUSION

We believe that we have fully accomplished the task of efficient parallel simulation of ballistic particle deposition model. The new parallel programming techniques and concepts we introduced in solving the problem are:

- The sequential ballistic deposition model is recast in continuous time. This makes it amenable for parallel processing.
- We recognize that the simulation mechanism of Ising spins fits the continuous time model. We use this to exploit the Ising spins simulation paradigm as a base algorithm for the ballistic deposition model.
- We recognize and exploit the state slackness present in the model to boost the performance of the base algorithm.
- The use of efficient data organization (subsectorization, ordering in particle lists) to further increase the performance.
- A roof computation and memory reclamation mechanism that essentially eliminates the possibility of memory shortage during a simulation run. This, in turn, makes it possible to continue the simulation run practically indefinitely and facilitates the service tasks, such as collecting statistics.
- Combining lists reduces unnecessary busy wait in an SIMD execution.

Our techniques assure the efficiency on a data-parallel SIMD computer, a notably harsh environment for writing parallel simulations. On a MIMD computer, it should be only easier to write efficient parallel codes for the ballistic particle deposition model. In the MIMD environment we would naturally employ the techniques introduced in the SIMD case. Specifically, because of the large memories of the PEs, as contrasted with small PEs memories in the SIMD case, the frequency of interprocessor communication will be greatly reduced after placing large substrate subareas in the PEs. There will be no need for synchronization barriers in the main stream of execution: the lockstep synchronization in our implementation is imposed only by the SIMD discipline of the available Maspar computers, while the algorithm works even better as an asynchronous one. (Under an asynchronous execution the simulation remains correct; this was discussed in Section 4.) For the related model of Ising spins a high efficiency of an MIMD style asynchronous simulation was demonstrated in [11]. And that efficiency was reached even without the technique of slackness exploitation which was not discussed in [11]. Exploiting slackness is possible not only for the deposition model but also for the model of asynchronous Ising spins, where it should further improve the performance.

The feature of the model which greatly facilitates the solution is that horizontal particle motions are not permitted. Thus, the landing X, Y of an arriving particle can be known ahead of time as well as the time of the deposition. Only the landing Z needs to be determined. A useful extension for the deposition model would allow particles to make horizontal motions. For example, an arriving particle could roll over already stabilized particles in an attempt to occupy a possibly lower resting position. As a next challenge we consider the task of simulating in parallel the deposition model with such extensions.

REFERENCES

1. R. Baiod, D. Kessler, P. Ramanlal, L. Sander, and R. Savit, *Phys. Rev. A* **38**(7), 3672 (1988).
2. A. G. Greenberg, B. D. Lubachevsky, D. M. Nicol, and P. E. Wright, "Efficient Massively Parallel Simulations of Dynamic Channel Assignment Schemes for Wireless Cellular Communications," in *Proceedings, 8th Workshop on Parallel and Distributed Simulation* (*PADS '94*), *Edinburgh, Scotland, July 1994,* p. 187.
3. P. Heidelberger and D. M. Nicol, *IEEE Trans. Parallel Distrib. Systems*, **4**(8), 906 (1991).
4. D. Jefferson, *ACM TOPLAS* **7**(3), 404 (1985).
5. R. Jullien and P. Meakin, *Europhys. Lett.* **4**(12), 1385 (1987).
6. D. Y. K. Ko and F. Seno, *Phys. Rev. E*, **50**(3), R1741 (1994).
7. D. Y. K. Ko and F. Seno, *Phys. Rev. B* **50**(23), Part I, 17583 (1994).
8. J. Krug and P. Meakin, *J. Phys. A* **23**(18), L897 (1990).
9. T. L. Lay and H. Robbins, *Proc. Nat. Acad. Sci. USA* **73**(2), 286 (1976).
10. B. D. Lubachevsky, V. Privman, and S. C. Roy, *Phys. Rev. E* **47**(1), 48 (1993).
11. B. D. Lubachevsky, *Complex Systems* **1,** 1099 (1987).
12. B. D. Lubachevsky, *J. Comput. Phys.* **75**(1), 103 (1988).
13. A. J. McKane, M. Droz, J. Vannimenus, and D. E. Wolf (Eds.), *Scale Invariance, Interfaces and Non-Equilibrium Dynamics* (Plenum, New York, 1995).
14. P. Meakin and R. Jullien, *Phys. Rev. A* **41**(2), 983 (1990).
15. D. M. Nicol, "Parallel Discrete-Event Simulation of FCFS Stochastic Queuing Networks, in *Proceedings, ACM SIGPLAN Symp. Parallel Programming Experience and Applications, Lang. and Syst., New Haven, Conn., July 1988.*
16. P. Nielaba and V. Privman, *Phys. Rev. E* **51**(3), 2022 (1995).
17. P. Nielaba, V. Privman, and J.-S. Wang, "Irreversible Multilayer Adsorption," in *Computer Simulation Studies in Condensed-Matter Physics VI* (D. P. Landau, K. K. Mon, and H.-B. Schüttler, Eds.), Proceedings in Physics, Vol. 76 (Springer-Verlag, Berlin, 1993), p. 143.