# Parallel diffusion-limited aggregation

Henry Kaufman,[1] Alessandro Vespignani,[1,*] Benoit B. Mandelbrot,[1,2] and Lionel Woog[1]
[1]*Department of Mathematics, Yale University, New Haven, Connecticut 06520-8283*
[2]*Thomas J. Watson Research Center, Yorktown Heights, New York 10598-0218*
(Received 21 March 1995)

We present methods for simulating very large diffusion-limited aggregation (DLA) clusters using parallel processing (PDLA). With our techniques, we have been able to simulate clusters of up to 130 million particles. The time required for generating a 100 million particle PDLA is approximately 13 h. The fractal behavior of these "parallel" clusters changes from a multiparticle aggregation dynamics to the usual DLA dynamics. The transition is described by simple scaling assumptions that define a characteristic cluster size separating the two dynamical regimes. We also use DLA clusters as seeds for parallel processing. In this case, the transient regime disappears and the dynamics converges from the early stage to that of DLA.

PACS number(s): 02.70.−c, 68.70.+w, 05.40.+j, 02.50.−r

## I. INTRODUCTION

A model of irreversible growth to generate fractal structures was the diffusion limited aggregation (DLA) model of Witten and Sander [1]. This model accounts for the origin of fractal structures in a great variety of processes: dendritic growth, viscous fingers in fluids, dielectric breakdown, electrochemical deposition, etc. [2,3]. Despite the simplicity of the rules that govern DLA, it shows unexpectedly subtle and complex properties and poses theoretical problems of new type [4,5]. To this day, the asymptotic properties of radial DLA are still not completely clear because of the discrepancies between the various measures of fractal dimension [2,6–10] and slow crossover to the asymptotic regime [11]. To investigate this and other issues in DLA, one seeks increasingly large DLA clusters, so that the microstructure becomes irrelevant and the asymptotic regime can be observed and analyzed. To that end, we propose the *parallel DLA* (PDLA), a variant of DLA, that uses a multiprocessor parallel computer. By parallelizing the cluster aggregation, we generated PDLA clusters of 100 million particles in 13 h on a 32-processor IBM power visualization system (PVS) computer. Therefore, the simulation time is no longer the limiting factor in generating very large clusters.

However, a subtle difficulty arises. The fact that more than one particle is diffusing at the same time introduces some interference effects in the dynamical process. The number of processors used in the simulation is the computational counterpart of the density of diffusing particles. This means that the early growth stages of a PDLA are in a multiparticle diffusion aggregation (MPDA) [12] regime that drifts to the DLA regime for larger sizes. A transient region is therefore present in the behavior of several quantities of the PDLA. From simple scaling assumptions we can characterize the transient region and find the behavior of the characteristic cluster size for which the dynamics become identical to DLA's dynamics.

To avoid the effects of the dynamical drift and to speed up the convergence towards the DLA regime we propose a combination of serial and parallel computing. We present large clusters simulated by a PDLA process growing on DLA cores. The cores have been generated on a serial machine and are used as the seeds of our larger clusters. In this way the PDLA process already starts in a low particle density regime corresponding to the usual (serial) DLA.

The study of PDLA clusters deserves further analysis, particularly to examine possible differences between the asymptotic structure of PDLA and DLA. Nevertheless, this method seems very promising in that it allows simulations to go beyond the present limitations of serial computers.

Section II introduces the algorithm to generate PDLA and discusses the details involved in creating and analyzing such large clusters. Section III presents the results from analyzing the clusters and differences between DLA and PDLA are discussed. Section IV describes the results obtained in the case of clusters obtained from PDLA growing on a DLA core. Finally Sec. V discusses results and perspectives of the method.

## II. PARALLEL DLA

DLA is simulated by placing a particle at a random location on a "birth" circle that is at some fixed distance from the maximum radius of the existing cluster. The new particle undergoes Brownian motion until it comes within a fixed "sticking distance" to the cluster (we use $\frac{1}{20}$ of a particle diameter for the sticking distance). At that point, the particle sticks to the clusters and a new particle is added on the birth circle, and the process continues. The original random walk DLA model is very simple but

*Present address: Instituut Lorentz, Leiden University, P.O. Box 9506, 2300 RA Leiden, The Netherlands.

biases due to approximation in the algorithm can introduce instabilities that dominate large scale properties of the structure [13]. These parameters are the random walk step size, the distance at which the random walk starts, etc. Our algorithm lets the particle take the largest step possible, without landing on the DLA. If the particle exits the birth circle, it is projected back with the "first-hit" probability distribution (see Appendix A) that models correctly the Laplacian boundary conditions of DLA. The particles thus wander "infinitely" until they first hit the DLA.

The above process is extremely time and memory consuming, even with the various efficiency schemes in practice, hence few people could generate clusters greater than 10 million particles. A single cluster of 100 million particles with our current algorithm would require a single computer for two to three weeks. This is possible, but very impractical since several clusters are required for reliable analysis.

A parallel computer, however, accelerates the process linearly in the number of processors. We accessed the IBM power visualization system (PVS), whose architecture is well suited for simulating DLA in parallel. The PVS contains 32 CPU's. Each processor has 16 megabytes of "private" random access memory (RAM), as well as access to 512 megabytes of shared memory. This architecture is ideal for PDLA because the cluster can be stored in the shared memory and can be accessed and modified quickly by any processor. Furthermore, the simulation of PDLA is a "read mostly" process, because most of the time the particles wander around the PDLA, and access the PDLA structure without changing it. Only when a particle becomes close enough to stick to the cluster, is the PDLA modified by adding the new particle. This is an important characteristic, because, to insure the correctness of the PDLA data structure, only one processor may modify the shared PDLA data structure at any given time, whereas many may read it.

This mutually exclusive behavior is implemented as a semaphore. When a processor is ready to stick its particle to the PDLA, it sets a semaphore that insures exclusive write access. If another processor is currently modifying the cluster, this processor must wait until the other has released the semaphore (i.e., finished making its modification). This type of waiting often yields poor performance in parallel algorithms. Thus, the read mostly behavior of the DLA simulation is important, because time-consuming processor interferences occur rarely. Furthermore, because the wandering particles do not "see" each other in our simulation, there is very little interprocess communication overhead that could also reduce the efficiency of the parallelism.

The actual implementation we use is similar in spirit to that of previously developed algorithms [14] but with great attention to the efficiency of storage. The algorithm uses a data structure called a quad-tree [15]. A quad-tree is a hierarchical subdivision of a square region of the plane into smaller and smaller square subregions. Each level of the hierarchy contains squares that are half the width of the next level up in the hierarchy. Because a region is subdivided only if it contains particles, memory is allocated only for the regions of space occupied by the PDLA. We have found PDLA's to maintain fairly constant average density, so we preallocate a fixed amount of quad-tree memory for the entire PDLA, which is proportional to the number of particles to be generated.

A given region of the quad-tree is represented by a block of four pointers. The subregions of that region that are empty have null pointers, while the occupied subregions point to other blocks of pointers. At a fixed maximum subdivision level, the leaf pointers point to lists of particles. We have found, in practice, that a minimum region size of eight particle diameters works well in the tradeoff between particle list searching efficiency and memory usage. This requires us to store 14 levels of subdivisions to represent a region of width 128 K (1 K$=10^3$) particle diameters in order to adequately store a $10^8$ particle PDLA whose diameter is around 90 K diameters.

To further optimize memory usage, we strove to minimize the memory requirements for the particle list in the leaves of the quad-tree. In our previous implementation, particles were represented by four-byte floating-point numbers for each coordinate, and a four-byte "next" pointer, which pointed to the subsequent particle in its region. However, 12 bytes per particle prohibited reaching clusters of $10^8$ particles on any of the computers we had available. Furthermore, the subparticle accuracy of the coordinates would begin to drop as the PDLA radius increased to beyond $10^5$ particle units. The average maximum radius for our $10^8$ PDLA's is $4.5\times10^5$, which yields a subdiameter accuracy of $\frac{1}{256}$ in the position of the outer particles because 15 of the 23 bits in the float's fraction must be used to store the particle's integer position.

We observed that the traversal path of the quad-tree from the root node to the leaf regions encodes global position information. Thus, we store only local particle coordinates within the $8\times8$ leaf region with one-byte fixed-point values for each coordinate. This gives us acceptable position accuracy to within $\frac{1}{32}$ of a particle diameter. Using local coordinates enabled us to store particle positions with two bytes instead of eight bytes.

We also wanted to optimize the particle list data structure. By taking advantage of the fact that there are almost always several particles falling within a single region, we could optimize the use of the "next" pointer by storing particles in chunks with a single "next" pointer. We took density distribution data from several $10^6$ PDLA's to minimize the following discrete memory function of the particle chunk size $N_p$. The density distribution $N(s)$ is the distribution of the number of regions with density $s$,

$$N_{\mathrm{ch}}(N_p)=\sum_{s=1}^{\max s} N(s)\left\lceil \frac{s}{N_p} \right\rceil , \tag{1}$$

$$f_{\mathrm{mem}}(N_p)=N_{\mathrm{ch}}(N_p)(N_pS_p+S_h) , \tag{2}$$

where $f_{\mathrm{mem}}(N_p)$ is the total memory required, $N_{\mathrm{ch}}(N_p)$ is the number of particle chunks of size $N_p$ required to store all the particles, $S_p$ is the size of an individual particle position (2 bytes), and $S_h$ is the size of the "next" pointer (4 bytes). The minimum value for $f_{\mathrm{mem}}(N_p)$ optimizes the tradeoff between the cost of the 4 byte "next" pointer and the unused particle slots for densities that do not exactly fit into an integer number of chunks. For regions of eight particle diameters, the optimum bunch size is seven, at an average cost of 3.21 bytes per particle. In practice, we use six, so that the total chunk size is 16 bytes to effectively use the hardware's memory-aligned cache. This only increases the total memory usage by less than 1%. The total memory required for storing $10^8$ particles is thus 306 megabytes. Using our naive storage scheme, with floating point coordinates would have required 1200 megabytes, and 600 megabytes with the fixed point coordinates. The quad-tree requires about 58 megabytes for a $10^8$ particle cluster.

When a particle is added to the PDLA, the quad-tree is traversed from the root node. Any empty regions are allocated in the quad-tree, and finally the particle is added at the end of the last chunk of particles in its region, or a new chunk is added if the last chunk is full. The "next" pointer functions as a particle counter for nonfull chunks.

As the PDLA is being formed, the particles are written to a file. This way, the file contains all the particles, in the order that they first appeared. This order is important for some types of analysis. For example, to establish parent-child connectivity relationships between two connected particles, one needs to know which particle is older. Due to the ordering constraint in the file, we can not take advantage of the concise representation of the particles in the quadtree. The particles' absolute positions are written in the file as 4-byte fixed-point integers for each coordinate. This yields large files of $8\times 10^8$ bytes for $10^8$ particle clusters. We can thus only store one large PDLA on a disk at a time. In order to generate several DLA's, say, over a weekend, each PDLA is immediately written out to an 8 mm streaming tape as soon as it is complete. Each tape can store up to three $10^8$ particle clusters.

Many analyses that we have run simply involve sequentially reading each particle, outputting some intermediate values and then accumulating some final statistics at the end. This type of analysis is done for computed moments, max-radius, mass-radius histograms, center of mass, Laplacian relaxation, and printing the PDLA. Since the entire PDLA does not need to be stored in memory, these analyses can be run on any workstation by reading the PDLA directly off the tape. Other analyses require reading the entire PDLA into memory and building the quadtree hierarchy in order to perform efficient geometric query operations on the PDLA. These include on- and off-center mass-radius computation, crosscut analysis, gap measurements, box counting, and angle of incidence measurements (if the angle is computed as the particle is added to the PDLA from the file). These runs only require the position information, without any detailed order of arrival information. Thus the compact run-time representation described above may be used. Other analyses, however, require geometric and order information, so the particles may not be grouped in lists, but must reside separately, each with a next pointer. This representation is the most costly in terms of memory, since each particle occupies 6 bytes of storage. The order information is important in establishing ordered connectivity information, such as parent-child relations, or recursive traversal of the PDLA. The following analyses require this richer information: Horton-Strahler branch order measurements [9,16], number of offspring measurements, and branch mass measurements.

With these techniques it is possible to generate clusters up to 130 million of particles. The CPU time required is drastically diminished, and we need approximately 13 hours to generate a 100 million particle cluster. In this way we have generated twenty 100 million particle PDLA's as well as a single 130 million particle PDLA. At this point, the cluster size is only limited by memory requirements, and not simulation time.

Of course, PDLA is not equivalent to DLA. The fact that more than one particle is diffusing, generates some interferences that can affect the properties of the final structures. In particular, it is possible to study some finite transients in the fractal properties of the clusters. In the next section we will analyze these transients.

## III. FRACTAL ANALYSIS OF PDLA

Figure 1 shows a PDLA cluster of 100 million particles. To the eye, this cluster is extremely similar to those produced by the single particle process. In our technique, however, more than one particle is diffusing contemporaneously and interferences occur because two or more particles may be wandering simultaneously around the same area of the PDLA. A first kind of interference occurs when a particle sticks, and a second particle finds itself overlapping with the newly attached particle. The second kind of interference occurs when two particles might decide to stick to the same particle at the same time. Only one processor may modify the DLA at a time, so one particle, chosen arbitrarily, is allowed to

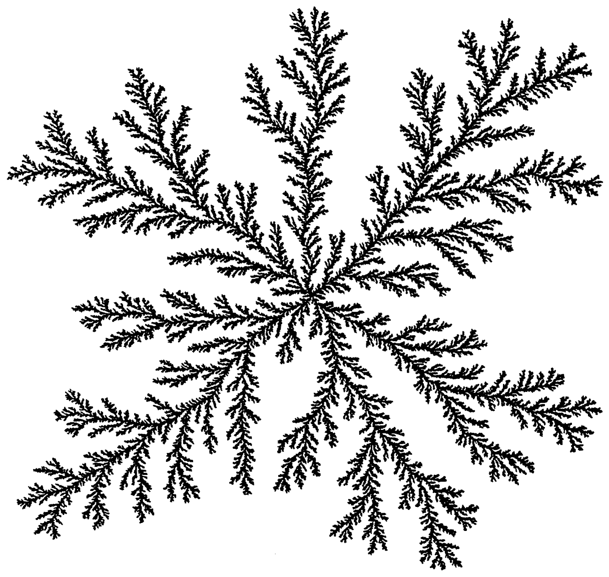
FIG. 1. Parallel-DLA aggregate of 100 million particles.

stick first. When the second particle is allowed to stick, it discovers that it overlaps with the first particle. These two types of interference are readily observable by checking that the closest particle in the DLA is not overlapping with the wandering particle. An overlap would imply that another particle "suddenly" appeared, since particle steps in Brownian motion are calculated so that they never overlap with the DLA. When this type of overlap occurs, the particle is rejected, thus the DLA is still guaranteed not to have any overlapping particles, as in serial DLA. A third, and more subtle interference is possible: while the particles might not overlap, the trajectory that they undergo is different, since they do not "see" all the particles that were launched before them. This differs from DLA, where every wandering particle can be affected by all previously launched particles. In PDLA, a particle can only be affected by all particles except for the last 31, since they are still wandering when the new particle is released.

These interactions are more pronounced for small PDLA's and they have the effect of slightly increasing the density. For large PDLA however, the change is very small (only one particle in 10 000 overlaps as the cluster size approaches 100 million particles; see Fig. 2).

From a physical point of view, this situation corresponds to a kinetic fractal aggregation in a particle bath with a finite density $\rho$ of diffusing particles. This phenomenon, called multiparticle diffusive aggregation (MPDA) [12], has been studied in the past both numerically and experimentally. The difference is that in our case the particle density is not kept constant during aggregation. In fact, the number of diffusing particles is constant, namely, the number of processors we use, while the structure grows and the number of active sites exposed to growth processes is increasing. This implies

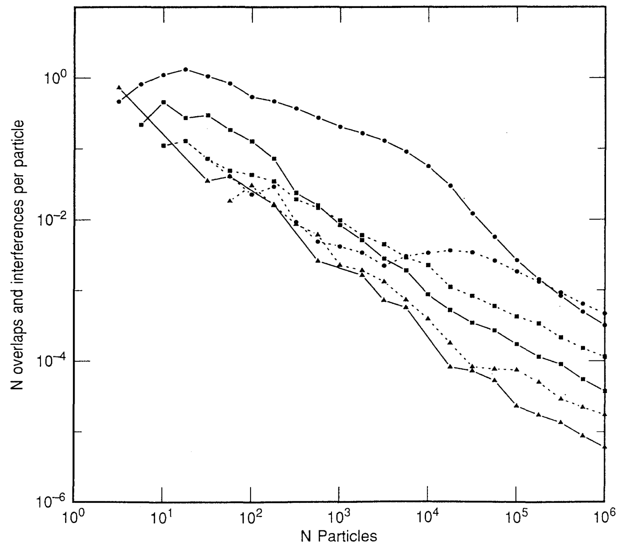

FIG. 2. Number of overlaps (solid lines) and interference rejects (dotted lines) per particle as the PDLA grows. Triangles, squares, and circles correspond to $n=4$, 16, 32 processors, respectively.

that as the active interface of the cluster increases, the density of the diffusing particles decreases. Therefore, particles around the structure have a smaller probability of interacting in the growing space of diffusion near the active region.

Thus we start with a MPDA regime for small cluster sizes but, as the structure grows, the dynamics approaches that of DLA. Since the dynamics does not approach the usual DLA process except for large sizes, we expect several quantities to show a transient region whose extent depends upon the numbers of processors used, i.e., the starting density of diffusing particles.

The best known and best established property of DLA is the mass-radius scaling relation

$$N(R) \sim R^D , \tag{3}$$

where $N(R)$ is the number of particles inside a circle of radius $R$ [17]. The exponent $D$ is the cluster's fractal dimension and it is extracted from the diagram of $\ln N(R)$ versus $\ln R$. For DLA this scaling behavior is confirmed on several decades, and for a number of particles $N$ up to $N \simeq 10^7$, $D=1.71$ as inferred since the very small clusters were used [1,6].

We compared the mass-radius scaling behavior of PDLA and DLA. Given the large amount of data stored for each cluster we developed a particular algorithm for obtaining the mass distribution efficiently. The algorithm is sketched in Appendix B. Figure 3 shows the doubly logarithmic plot of the number of particles versus the cluster's radius for the different number of processors used. For very large clusters, the mass-radius curve collapses onto DLA, even for the 32-processor PDLA. However, as more processors are used, it takes longer for the PDLA clusters to approach DLA behavior. This is indeed the transient due to the drift from the MPDA regime to the DLA regime. Figure 4 focuses on the small cluster region in order to show the behavior of the mass-radius plot in the transient region.

The two dynamical regimes are characterized by the density of diffusing particles in the active region of the cluster, namely, the growing interface of the cluster. This growing interface is formed by $N'$ points where

$$N' \sim \frac{dN}{dR} \sim R^{D-1} \sim N^{(D-1)/D} , \tag{4}$$

where we used the fact that $N \sim R^D$. We can define the effective density of diffusing particles as

$$\rho \sim \frac{n}{N'} \sim \frac{n}{N^{(D-1)/D}} , \tag{5}$$

where $n$ is, in our case, the number of processors used in the cluster generation. The serial DLA regime is recovered in the limit when $\rho \ll 1$, and the transient is characterized by a density $\rho \simeq 1$. We can define, therefore, a characteristic length by

$$\rho_c \sim \frac{n}{N_c^{(D-1)/D}} \simeq 1 . \tag{6}$$

Here $N_c$ gives the characteristic cluster mass discriminating the aggregation regimes, and its dependence on the

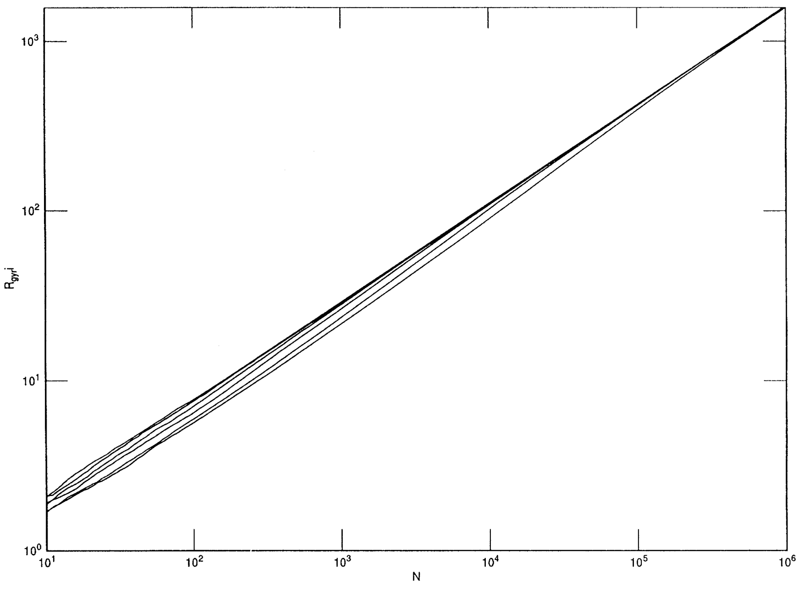

FIG. 3. Doubly logarithmic plot of the mass-radius scaling behavior, i.e., the radius of gyration $R_{\rm gyr}$ with respect to the number of particle $N$. The plots of PDLA with $n=1$, 2, 4, 8, 16, and 32 (from top to bottom) processors collapse onto the DLA one ($n=1$ processor) for $N \simeq 10^6$. The radius unit length is the particle's diameter; $i$ denotes the number of processors.

number of processors used is

$$N_c \sim n^{\theta}, \tag{7}$$

with $\theta = D/(D-1)$. This relation gives us an indication of the extent of the transient region as a function of the number of starting processors. If $N_c$ is the only characteristic length introduced in the PDLA, then the mass-radius relation should be described by the following scaling behavior:

$$R \sim N^{1/D} f\left[\frac{N}{N_c}\right], \tag{8}$$

where $f(x) \sim$ const for $x \gg 1$, and $f(x) \sim x^{\alpha}$ for $x \ll 1$. According to this scaling assumption the data obtained for the mass-radius relation for various values of $n$ should collapse onto the same universal curve $f(x)$, when $\ln(R/N^{1/D})$ is plotted against $\ln(N/n^{\theta})$ using the correct value of $D$. By using the value $D=1.7$ we obtain the best data collapse for $\theta \simeq 2.7$ (Fig. 5). That is in reasonable agreement with the value estimated by our scaling assumption $\theta \simeq 2.4$.

This analysis implies that PDLA can be considered as joining the DLA dynamical regime for cluster size $N \gg N_c$, with $N_c$ increasing with $n$ like a power law with exponent $\theta$. Therefore, we have to generate larger clusters as we increase the number of processors if we want a good approximation of DLA clusters.

Several other quantities could be affected by this transient, even for larger cluster sizes. In fact, circular DLA shows a very complex behavior with several departures from self-similarity [7,11,10]. This scenario implies an

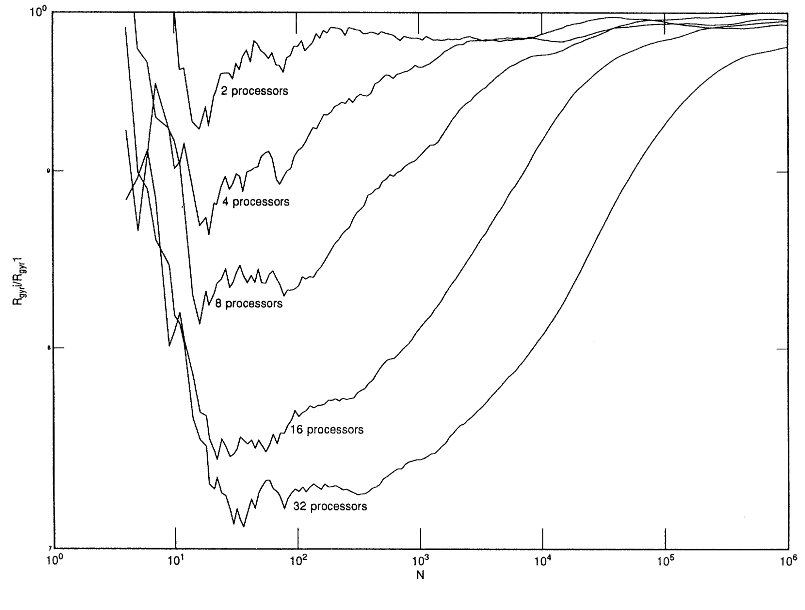

FIG. 4. Plot of the mass-radius relation for PDLA of $n=2$, 4, 8, 16, and 32 processors. To visualize the transient region, the radius of gyration of PDLA's is normalized with respect to the radius of gyration of DLA ($n=1$).

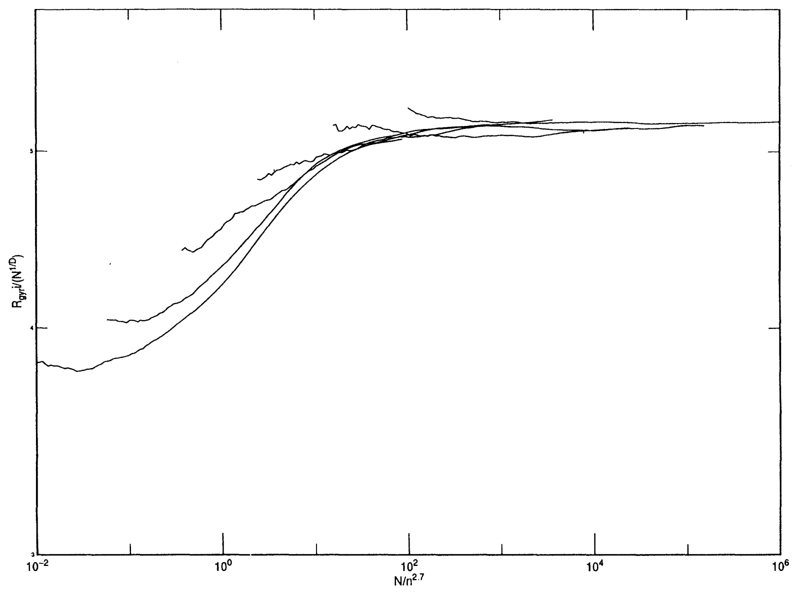

FIG. 5. Data collapse for the mass-radius relation from various values of $n$ by using the scaling functions of Eq. (8).

unusual approach to asymptotic self-similarity, in which different numerical characteristics of DLA have different crossover to self-similarity [7]. The origin of this "drift" deviating from simple self-similarity can be found in the dynamical aspects of the phenomenon. In particular, the old part of the cluster, i.e., the frozen region that will not grow in the future, preserves a memory of the dynamical process by which it has been generated. In this perspective, the fact that the PDLA changes the dynamical properties in the early growth stage might have several consequences on the future fractal properties of the resulting structure. For this reason the analysis we have shown on the mass-radius scaling behavior does not take into account many other geometrical and morphological properties of the generated structure. Further analysis is in progress and will be presented elsewhere.

## IV. PDLA GROWN FROM A DLA CORE

The PDLA algorithm yields very large clusters in a reasonable amount of time. However, the dynamical transient described in Sec. III reduces the effectiveness of this method for studying ordinary DLA. In fact, the more processors we use, the larger the aggregate has to be before we can consider the two aggregation processes as indistinguishable. To reduce this effect we begin the simulation with an existing DLA "core." Of course, if this cluster seed is large enough, the PDLA starts in the usual dynamical regime (DLA regime) and no transient should be detected in the scaling behavior. In this way the cluster does not have a memory of a different dynamical regime and the aggregates obtained are not affected by the multiparticle interaction effects. Thus, we use the PDLA process only after the finite density effects are negligible.

To test this method, we repeated the analysis on the mass-radius behavior for clusters grown with DLA cores of $N=10^4, 10^5, 10^6$ particles. Figure 6 shows the transient region of PDLA with and without a DLA core. It is possible to see that for clusters grown with a DLA seed, the transient disappears and the mass-radius relation does not show discrepancies with respect to DLA. We thus have combined the DLA with our PDLA in order to generate vary large clusters not affected by the transients that are present in the early stage of PDLA growth. From a computational point of view, this method allows us to have a major improvement in the CPU time needed to generate the cluster, while avoiding the drift induced by the initial multiparticle dynamics. Nevertheless, as already mentioned the parallel diffusion of particles might induce other effects on DLA geometry and a more careful study is required to have the certainty that the two processes are asymptotically identical.

## V. CONCLUSIONS

This paper presents a method to generate very large DLA clusters by using a parallel computer. This allows us to obtain a linear speed improvement as a function of the number of processors, and at this point the cluster size is only limited by run-time memory usage and output storage space, and not by simulation time. This method makes it possible to reach cluster sizes beyond the usual serial computer's capacity, but introduces some interference effects due to the simultaneous diffusion of particles. From a physical point of view, this corresponds to a drift from a multiparticle diffusion regime for small sizes to a DLA regime for larger sizes. This transient is present in the early stages of the growth and clearly depends on the number of parallel processors used. From simple scaling assumptions it is possible to characterize the behavior of the characteristic cluster size that discriminate between the two regimes and its scaling with respect to the number of processors.

Finally, we present the results for PDLA clusters grown on a seed defined by a DLA cluster. By using this

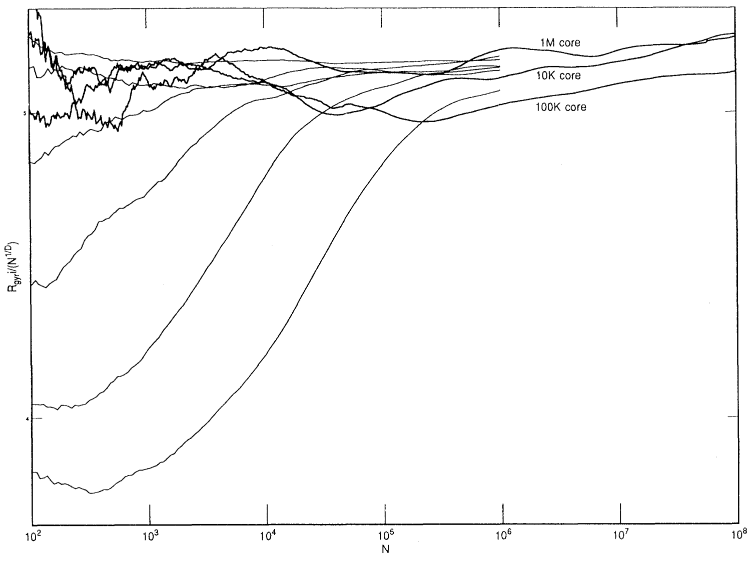

FIG. 6. PDLA's mass-radius behavior with (bold curves) and without DLA core. The radius of gyration is normalized with the DLA one. In the case of PDLA with existing DLA core, the transient region disappears. 1 M denotes ($10^6$) particles and 10 K denotes $10^4$ particles.

method we can avoid the problems due to the dynamical drift and speed up the convergence of the PDLA towards the usual DLA.

## ACKNOWLEDGMENTS

We are indebted to Lewis Siegel for help in figuring out the PVS and for organizing and running all the simulations. We thank Yuval Peres for suggesting the Moebius transformation described in Appendix A. We also wish to thank IBM Yorktown for generously allowing us to use the PVS and the disk storage for large DLA files. The investigations carried out at Yale University were supported by the Office of Naval Research under Grant No. N00014-90-J-1026, and were performed on two IBM RS/6000 RISC computers, made available at Yale by IBM's Technical Computing Systems Group (Mathematical Sciences Area). H.K. is grateful to Harvey Blech who started him on the path to physics.

## APPENDIX A

Often particles that wander far away from the cluster are "killed" at some killing radius. However, we bypass killing; in fact, we project the particle back to the birth radius with the Poisson kernel which is the correct first-hit probability distribution for Brownian motion. Here is an elegant way to implement this kernel by a Moebius transformation. Normalize the particle distance $H$ from the birth circle of radius $R$ to $d=H/R$, the distance to the unit circle. The first-hit distribution for a particle at distance $d$ from outside the unit circle is equivalent to that of a particle on the inside of the circle at distance $r=1/(1+d)$ from the origin. A point is generated with the proper distribution taking a uniform random unit complex number $z$, and performing the Moebius transformation

$$w=\frac{z+r}{1+rz}\ . \qquad \text{(A1)}$$

The Moebius transformation maps the complex unit circle on the complex unit circle, with fixed points at $z=1$, and $z=-1$. The point $w$ will have the correct distribution for the first-hit location of the original point at distance $H$. Its peak at $w=1$ narrows as $r$ approaches 1, as expected since the particle has a higher probability of reentering nearby, when it is close to the circle.

## APPENDIX B

Our mass distribution computations use the quad-tree as a way to efficiently measure the mass in large regions. A naive approach to computing the mass within a region could be to compare every particle in the cluster with the region and only count those particles that fall within the region. To avoid this approach, we take advantage of the hierarchical nature of the quad-tree. To each node of the quad-tree we add a value indicating how many particles fall within that node's square. This value is easily computed for every node by a single depth-first traversal of the quad-tree, since each node's value is simply the sum of its four children's values. To compute efficiently how many particles fall within a given circle, the quad-tree is traversed from the root and the square enclosing each node is compared with the query circle. Nodes which fall completely within the circle are counted by simply adding the number stored in the node, and no further traversal is required. A node which falls completely outside the circle is rejected, and similarly not traversed any more. If a node overlaps with the boundary of the circle, its four children are traversed recursively. If the tree has been traversed until a leaf node containing actual particles is reached, and it is still on the boundary, then every particle within that node is checked against the circle, and counted only if its center falls within the circle. This last case only occurs for a small percentage of the total circles that are counted, so the algorithm is very fast. Major time savings are achieved for large nodes that fall completely inside or outside the circle, because all of the particles are essentially checked with only one square-circle comparison. In the average case of randomly distributed points in the plane, the number of leaf nodes checked is proportional to the circle's radius $R$. Since DLA is fractal, however, the number of leaves checked grows as $R^{D-1}$. For the largest circles in a $10^8$ particle cluster, the number of particles actually checked is on the order of a few thousand.

This algorithm has been extended to process arbitrary boolean combinations of circles. This is necessary for mass within annuli computations (the difference between two circles), or off-center mass-radius computations which require counting the particles that fall within the intersection of two annuli. This method also works for computing the particles that overlap with a circle at some radius. An annulus is constructed whose boundary is one particle radius from the given circle on either side, and the particles whose centers fall within this annulus overlap with the given circle. This measurement is required to compute gap angles between particles that are at some radius from the center particle and for cross-cut dimension calculations.

The basic idea for the boolean combinations is to consider in advance all the possible cases when one should accept, reject or traverse the quad-tree further. For example, when considering the difference of two regions, if a node falls completely within the "negative" region, or completely outside the "positive" region, it can be rejected immediately. If the node falls completely within the "positive" region and outside the "negative" region, then it can be accepted. Otherwise, the quad-tree must be traversed further. All the boolean relations can be evaluated in this way recursively, by forming a tree of operations whose nodes represent the boolean operations, and whose leaves are the basic geometric query shapes (circles or rectangles). The final geometric query method is thus doubly recursive. The quad-tree is traversed recursively, and each node is checked recursively against the boolean operations tree. Though this method may seem to be extremely thorough, it was not very difficult to implement. Once the infrastructure was in place, it was easy to add new query shapes for new kinds of analyses (e.g., rectangles). Furthermore, the method runs very quickly in practice. In fact, for most of our analyses, the time is dominated by reading the PDLA off the tape.

[1] T. A. Witten and L. M. Sander, Phys. Rev. Lett. **47**, 1400 (1981).
[2] T. Vicsek, *Fractal Growth Phenomena* (World Scientific, Singapore, 1992).
[3] For a recent review see *Fractals in Natural Science,* edited by T. Vicsek, M. Schlesinger, and M. Matsushita (World Scientific, Singapore, 1994).
[4] L. Pietronero, A. Erzan, and C. Evertsz, Phys. Rev. Lett. **61**, 861 (1988); R. Cafiero, L. Pietronero, and A. Vespignani, *ibid.* **70**, 3939 (1993).
[5] T. C. Halsey, Phys. Rev. Lett. **72**, 1228 (1994).
[6] P. Meakin, in *Phase Transitions and Critical Phenomena,* edited by C. Domb and J. Lebowitz (Academic, New York, 1988), Vol. 12, p. 335.
[7] B. B. Mandelbrot, Physica A **191**, 95 (1992).
[8] C. Evertsz, Phys. Rev. A. **41**, 1830 (1990).
[9] P. Ossadnik, Phys. Rev. A **45**, 1058 (1992); Physica A **195**, 319 (1993).
[10] C. Amitrano, A. Coniglio, P. Meakin, and M. Zannetti, Fractals **1**, 840 (1993).
[11] B. B. Mandelbrot, H. Kaufman, A. Vespignani, I. Yekutieli, and C. H. Lam, Europhys. Lett. **29**, 599 (1995).
[12] R. Voss, Phys. Rev. B **30**, 334 (1984).
[13] R. Voss, Fractals **1**, 141 (1993).
[14] S. Tolman and P. Meakin, Phys. Rev. A **40**, 428 (1989).
[15] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics, Principles and Practice* (Addison-Wesley, Reading, MA, 1990).
[16] I. Yekutieli, H. Kaufman, and B. B. Mandelbrot, J. Phys. A **27**, 275 (1994).
[17] With $R$ is usually indicated the gyration radius of the cluster (see Ref. [6]).