Dielectric breakdown in a simplified parallel model

Howland A. Fowler,[a] Judith E. Devaney,[b] and John G. Hagedorn[c]
High Performance Systems and Services Division, Information Technology Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland 20899

Francis E. Sullivan[d]
Center for Computing Sciences, Institute for Defense Analyses, Bowie, Maryland 20715

(Received 5 February 1998; accepted 11 June 1998)

The growth of streamer trees in insulating fluids (a submicrosecond process that triggers high-voltage breakdown) has been simulated with a combination of parallel-coding tools. Large grids and arrays display well the multifractal, self-avoiding character of the streamer trees. Three physical cases have been approximated by different power-law weightings of the statistical growth filter: dense anode trees, in the uniform field; sparse cathode trees (a rarer experimental case); and ultrasparse anode trees (seen in some fluids of higher viscosity). The model is contained in a software package that is written in Fortran 90 with data parallel extensions for distributed execution. These extensions encapsulate an underlying, invisible message-passing environment, thus enabling the solution of memory-intensive problems on a group of limited-memory processors. Block partitioning creates processes of reasonable size, which operate in parallel like small copies of the original code. The user needs only to express his model in transparent array-directed commands; parallel interfacing between blocks is handled invisibly. Breakdown is performed in parallel in each of the local blocks. Results are presented for experiments run on eight and nine nodes of the IBM SP2, and four and eight nodes of the SGI Onyx and Origin, three examples of multiple-processor machines. Display is carried out in three dimensions. Timing of the growth can be shown by color banding or by frame animation of the results. The adequacy of the growth rules and size scaling are tested by comparing the simulations against snapshots from high-voltage discharge events. [S0894-1866(98)01805-7]

INTRODUCTION

The growth of fast streamer trees in liquid-dielectric insulation provides the precursor ''leader'' conduction path through which damaging flashover between electrodes can take place. The high-speed, variable nature of this phenomenon has made its detailed mechanism elusive. Nonetheless, a global description of the process may be useful for its characterization.

We have applied stochastic Laplacian growth as a model for fast streamer trees in liquid dielectrics. Filamentary dielectric breakdown has been extensively discussed by Pietronero and Wiesmann[1] and by others.[2–5] Here we construct a practical realization of the algorithm on a large Cartesian grid using boundary conditions that confine the electric field. We examine the effect of parameters (threshold voltage, choice of power law) on the fractal structure and timing of the growth process.

The calculation of the voltage field throughout the full volume, repeated after each stage of breakdown growth, is the major computational burden that calls for parallel methods.

The single-instruction-stream, multiple-data (SIMD) model fits our problem closely. We had tested an earlier machine-language version of our algorithm on the CM-2 Connection Machine.[6] The SIMD version there was synchronous and specific to the machine. The present method uses block partitioning under Message Passing Interface (MPI) control,[7] in a single-program, multiple-data (SPMD) approach, which runs asynchronously. For a data-parallel environment that is easy to use and transport, we have written the code in array-directed commands of Fortran 90, together with our library DPARLIB[8,9] that encapsulates the underlying MPI calls.[7] This will run in any environment that is Fortran 90 and MPI enabled, be it networks of workstations, symmetric multiprocessors (SMPs), parallel machines, or some combination of these.

Running Fortran 90 on a single processor does not provide access to the memory that large data-parallel programs require. Our software has been designed not only to fit into the Fortran 90 syntax transparently, but also to run across the nodes of a group multiprocessor. It extends the array-directed commands seamlessly, across block boundaries. For development purposes, the number of processors executing can be reduced to one, like a serial version of the code using only plain Fortran 90 commands. These features enable users to test small versions of their code on their workstations. Then, with only minor changes in the code, much larger versions can be run on machines like the SP-2,

[a]Guest researcher; e-mail: howland.fowler@nist.gov
[b]E-mail: judith.devaney@nist.gov
[c]E-mail: john.hagedorn@nist.gov
[d]E-mail: fran@super.org

taking advantage of the larger memory and fast communication network.

I. PHYSICAL MODEL AND ALGORITHM

The elements of a stochastic fractal model have been demonstrated in numerous examples.[10–16] The streamer tree is assumed to be fully conductive and electrically attached to the electrode from which it originates. The voltage field is defined over a set of points in a rectangular grid. The streamer tree is a connected set of those points.

After initialization, the algorithm alternates between a Laplacian convergence procedure and statistically weighted streamer tree growth. Our algorithm may be summarized as follows:

```
Initialize voltage field and anode/cathode configuration
Set the streamer tree to be the anode (including starter
   needle)
Run Gauss-Seidel over-relaxation on Laplace's equation
   for 200 iterations

 Do until streamer tree has reached cathode

      Set all sites on streamer tree to voltage 0.0
      Set all sites on cathode to voltage 1.0
      Solve Laplace's equation via Gauss-Seidel
         over-relaxation
      Find nearest neighbors to tree

      Do until the set of breakdown sites is not less than one
            Set Breakdown sites equal to the empty set
            Choose, locally in each block, red or black
               (at random)
            Set breakdown candidates equal to
               neighbors of the color
            Remove candidates with voltage below a
               specified threshold
            For each remaining site generate a weighted
               random number
            Remove breakdown sites whose voltage is
               below this number
      End Do

      Add breakdown sites to streamer tree

End Do
```

The two stages of this algorithm are the solution of Laplace's equation and the growth of the streamer tree. We now consider these in more detail.

A. Solution of Laplace's equation

Laplace's equation is fully solved throughout the interior region (filled with dielectric liquid) using the anode, starter needle, and streamer tree as one boundary at zero potential and the plane cathode as the counterelectrode. Sides of the cube-volume support periodic boundary conditions. The quasistatic field approximation (i.e., that the voltage field always ''catches up'' with the new boundary condition at each stage of the growth) is physically reasonable, since breakdown streamers are known to advance at supersonic velocities, still slow by comparison with the speed of light in the dielectric liquid.

A first-order, six-point-averaging Laplacian operator[17] on red-to-black and black-to-red checkerboard subgrids advances through Gauss–Seidel overrelaxation. Initially 200 iterations of convergence bring the smoothing precision below one part in 10,000, on a grid of $128\times128\times128$. (Nominal single iteration time is roughly 10 s.) After this first high-precision convergence, the epsilon is relaxed (to 0.001–0.005) in the interests of speed. Because each growth stage creates only small perturbations on the existing boundary conditions, the later cycles reconverge rapidly, in 5–10 iterations. The changes propagate in rapidly diminishing ripples, away from the newest growth links on the tree.

B. Growth of the streamer tree

Grid sites immediately adjacent to the tree are examined. If their voltages exceed a specified threshold (or cutoff) level, then the surviving voltages are compared against a weighted distribution of random numbers. Those that pass over the statistical hurdle are attached to the tree, and the Laplacian convergence is recycled for the new, perturbed boundary condition. Field and growth stages alternate until the counterelectrode is reached.

By contrast with some of the stochastic fractal models cited earlier, which recalculate Laplace's equation after each single new-growth site is added, our algorithm considers simultaneous, distributed growth possibilities on all ''red'' or all ''black'' neighbors of the tree during each growth stage. Growth is favored near the tips of the tree, where electrostatic field lines converge strongly and the voltage gradients are largest. While the details of the mechanism for fast streamer growths are still to be determined from experiment, we ask, ''To what laws of shape and timing does the observed growth conform?''

Weighting of the growth-probability power law, as a function of electric field strength, determines the degree of bushiness (or fractal dimension) of the final structure. At one extreme, Sanchez *et al.*[18] have shown that a fourth-power (or higher) dependence on field strength produces a single, self-avoiding strand that resembles a directed random walk. Side branching is minimized. At the other limit, Witten and Sander[19] showed that a linear filtering against uniformly distributed random numbers results in a dense growth form, which is also found in diffusion-limited aggregation, from a different mechanism. In this linear regime, a mass of twigs advances with a ''front'' of thickly spaced growing tips. A recent review by Erzan, Pietronero, and Vespignani[20] considers ways of characterizing fractal dimension in this class of problems.

In Fig. 1 we see a log–log plot of breakdown probability (for a single link between grid vertices) versus power of the electric field strength, measured as the voltage difference between the conducting tree, at zero potential, and its immediate neighbors on the Cartesian grid. The power law determines the likelihood of growth from the tip, as against side branching. Recall that Laplace's equation is size-scalable without limit. By adopting a power-law probability, we may expect self-similar growth over large ranges of both size and field strength. This is not viewed as a microscopic description, but rather as a law controlling the global growth form.

Notice how minuscule the breakdown probability becomes for cube-law and higher powers on the left half of

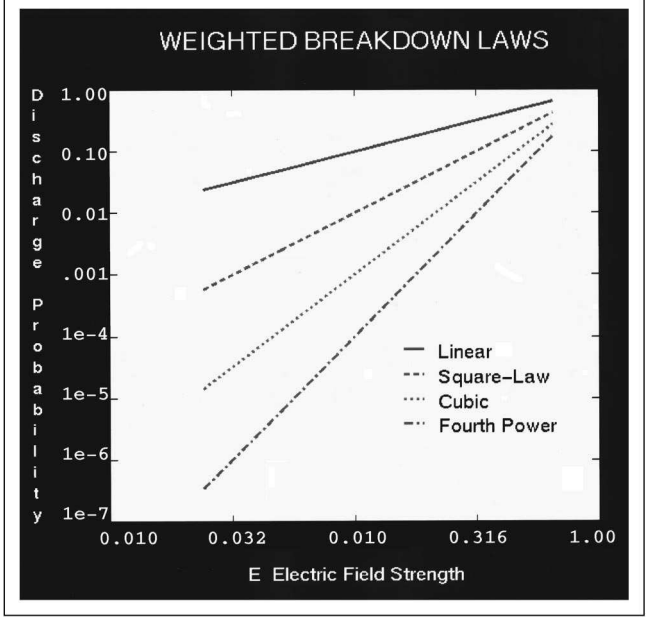

Figure 1. Log–log plot of breakdown probability (discharge probability for an individual link) versus electric field strength for power laws with integral exponents. The horizontal scale is electric field strength at the candidate links in the discretized step approximation (difference between the zero voltage of the streamer tree and the voltages at the neighbor grid nodes). The limit value 1.0, to which all fields are normalized, is the value that would be reached between flat-plate electrodes separated by one grid step.

the graph. The range shown, of field strengths, is that encountered in growth across a $128\times128\times128$ grid volume. As the streamer tip advances from anode to cathode, the diminishing gap leads to progressively higher fields around the leading branches, while electrostatic screening causes the fields around the base of the tree to drop back below threshold, so that growth is cut off there. Because the electric field is confined by limiting boundary conditions, and the growth process starts and stops at an abrupt threshold, we do not expect a ''pure'' fractal dimension throughout the growth. The tree is sparser as it leaves the concentrating needle on the primary electrode, and denser as it approaches the counterelectrode.

The algorithm must operate uniformly across a wide range of growth rates. For square-law and higher powers, the starting probability near the anode needle (where the field strength only slightly exceeds the threshold value of 0.015–0.10) is very small—tens or hundreds of empty statistical tries may occur between single breakdown events. By contrast, linear weighting leads to tens of events per growth stage, in the shorter gaps, approaching the cathode.

The uniform treatment of slow and fast stochastic processes assumes that the discretized physical model approximates a Markov-process master equation,[21,22] whose time variable is proportional to the number of trial instances.

II. IMPLEMENTATION

Our code was first implemented in a serial version, on Sun and SGI workstations, and was run interactively on small grids. This enabled extensive testing with visual tracking of detailed printouts to the screen. This serial code was implemented in standard Fortran 90 and exploits that language's array-oriented syntax and intrinsic functions. As examples of the powerful simplifications available, we describe some of the details.

The code makes extensive use of array arithmetic, the WHERE construct, array functions such as CSHIFT, MAXVAL, COUNT, and the dynamic-memory-allocation features of Fortran 90. Our program logic is expressed through combinations of real and logical arrays, the WHERE assignment mask of Fortran 90, and a selection of spreading and collecting operations.

For example, our Laplacian-operator subroutine makes vigorous use of Fortran 90's CSHIFT operation. The sum of neighbors calculation was implemented in the serial version with the following code:

```
blksum=cshift (blkphi,+1,1)+cshift (blkphi,−1,1) &
      +cshift (blkphi,+1,2)+cshift (blkphi,−1,2) &
      +cshift (blkphi,+1,3)+cshift (blkphi,−1,3)
```

CSHIFT is the standard Fortran 90 intrinsic function that performs circular shifts. The array (blksum) that results from this calculation contains at each grid point the sum of the six neighbors of that grid point in the original array, using periodic boundary conditions. The division of the three-dimensional grid into alternating/adjacent red and black subgrids is a feature brought over from statistical-mechanics simulations. Black-to-red and red-to-black averaging steps are carried out across all processes, at each loop. This routine is global over the entire internal voltage-field array.

We also made extensive use of the ''modules'' feature of Fortran 90. This simplified the debugging process by enabling the compiler to catch certain types of coding errors. In particular, the compiler checked that all subroutine calls had properly constructed calling sequences.

The statistical selection was coded by the following steps. By contrast with the Laplacian routine, these are local operations, executed independently in each block.

(1) Find all nearest neighbor sites to the growth tree within the process block. This procedure produces a logical mask array that is .true. at the neighbor sites and .false. elsewhere.
(2) Select, at random for each block, either the red or black neighbors within that block. The mask array of neighbors is modified to reflect this choice. Since we are considering simultaneous but independent growth across the full distributed array of neighbor sites, this red/black choice discourages selection of adjacent sites, which might lead to growth concentration or cause a locally enhanced perturbation of the voltage field.
(3) Allocate a one-dimensional REAL array with one element for each surviving neighbor. If a power law higher than linear is desired, then additional arrays of the same length are allocated.
(4) Fill the one-dimensional array(s) with uniformly distributed random numbers. The DPARLIB dp_uni is used to generate random numbers that are independent across all of the parallel processes.

(5) Calculate the MAX array of these random arrays. The resulting array contains a weighted distribution at one power lower than the desired breakdown power law. Step (7) below, the statistical filtering, is equivalent to a single order of integration for the distribution of voltages that will survive this hurdle.
(6) UNPACK the MAX array through the neighbor mask array. This creates a three-dimensional array with the weighted random numbers at the surviving neighbor sites.
(7) Compare the voltages to the weighted random array at the neighbor sites. WHERE they pass, add new sites to the existing growth tree. Write out their grid positions, trial numbers, and field strengths.
(8) If no sites have been added to the tree, return to step (2).
(9) If a site has been added to the tree, return to the Laplacian-convergence stage.

Note that the approach used in steps (3)–(6) substantially reduces memory and computational requirements by generating the weighted distribution of random numbers in minimally sized arrays. The resulting linear array is UNPACKed into the full-size three-dimensional array only when necessary.

The Fortran 90 array instructions that we needed were WHERE, CSHIFT, UNPACK, MAXVAL, MAX, MOD, COUNT, ANY, ALLOCATE, DEALLOCATE. In the block-parallel mode, some are used in local fashion by the individual processor, addressing only the portions of the major arrays seen within its own block.

III. CONVERSION TO PARALLEL OPERATION

The DPARLIB operations, which were required as substitutions or additions to enable parallel operation, were dp_cshift, dp_uni, dp_count, dp_any, dp_sync.[8,9] In addition, the dp_initialize and dp_query_layout routines were called at the start of the run to set up the division of domains between processors.

The adaptation of the serial code was accomplished through the use of the DPARLIB subroutine library. This is a Fortran 90 library developed at NIST to facilitate the transition of serial applications to a parallel-computing environment and the development of new parallel programs. DPARLIB is built on MPI, an industry-standard library for passing data and coordinating the activities of multiple processes in a parallel operating environment.

DPARLIB is designed to be used in the SPMD programming approach. In other words, multiple copies of the same program are running simultaneously, and each copy is processing a different portion of the data. In particular, DPARLIB provides simple mechanisms to divide very large arrays into blocks, each of which is handled by a separate copy of the program. In practice, this means that the researcher can write parallel code that looks almost identical to serial code. In our case, the code could be written as though addressed to a single active grid node and its immediate neighbors. Fortran 90, extended across block boundaries by DPARLIB, executed each instruction on all 2,000,000 sites of each array.

In this way, an existing serial program can often be converted to a parallel program with few changes by using DPARLIB, which plays the role of a high-level language for block parallelism.

DPARLIB's emphasis on array handling is designed to mesh with Fortran 90's array syntax and intrinsic array-handling functions. Much of DPARLIB consists of parallel versions of the intrinsic array functions such as CSHIFT and MAXVAL. Because DPARLIB is coded entirely in standard Fortran 90 and depends only on MPI, it is portable to any environment that provides those two resources.

The adaptation of the dielectric breakdown code was straightforward and involved the following steps.

(1) Add calls to DPARLIB housekeeping functions for initialization and specification of the mapping of data arrays to an array of processes. (dp_initialize and dp_query_layout).
(2) Make all distributed data arrays allocatable, and allocate the arrays based on information provided by DPARLIB calls.
(3) Change various Fortran 90 intrinsic calls to the corresponding DPARLIB calls. This simply involves changing the name of the called routine, e.g., cshift becomes dp_cshift and maxval becomes dp_maxval.

Once this conversion was complete, each process ran the same program with its own block of data. It used no explicit reference to data within other blocks, except as information was received through collecting, spreading, and shifting operations in DPARLIB. The parallel version could be run at once, on a network of workstations with the local area multicomputer (LAM) implementation of MPI.[23]

Our Laplacian operator is first-order and requires only the exchange of data for single layers between domain blocks.[24]

A convenient feature of DPARLIB is that communication between processes is handled automatically. Thus, the typical boundary-layer transfers of data, described by Gropp, Lusk, and Skjellum,[7] do not require explicit consideration by the code writer. dp_cshift includes—invisibly—the necessary ''shadow'' sites and cross-boundary transfers of data that must take place at the planes between process domains.[17,24] The user can ignore this complex and error-prone aspect of code writing.

There were, however, two crucial points in this program at which we had to take some care about the parallel execution of multiple copies of the code: writing the streamer tree sites to a disk file and testing for whether the streamer tree had reached the cathode. At these points, the application code must explicitly deal with the fact that multiple copies are executing in parallel.

At the end of the tree-growth part of the algorithm, the grid sites that have been added to the streamer tree are written to a data file. Because these sites will be found in multiple processes, we have these multiple processes trying to write to the same output file at the same time. This will result in errors on many systems, including the Sun, SGI, and IBM systems on which we were running these tests.

This problem can be solved by having each process write to its own output file. These separate output files could then be merged after the program is complete. Where multiprocessor access to a common file is possible, we instead implemented a ''round-robin'' solution that involves the synchronization of the processes using the DPARLIB rou-

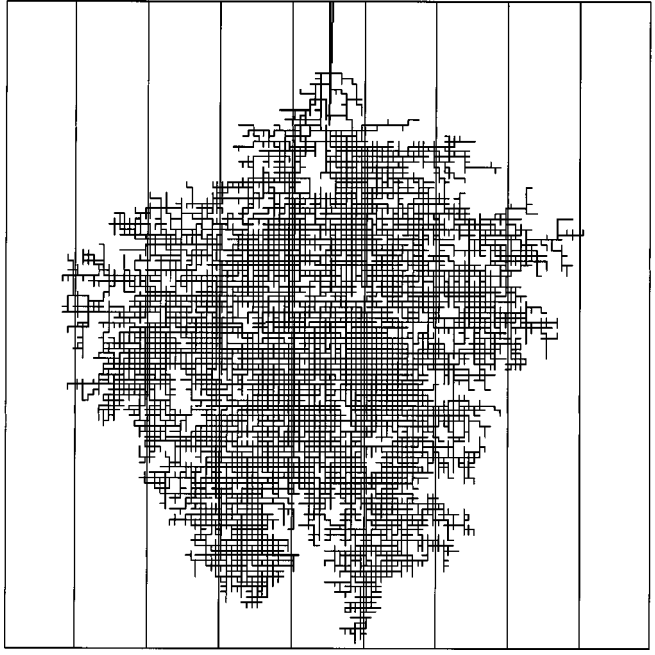

Figure 2. Lateral projection of a linear-weighted streamer simulation on a grid of 126×126×126, showing the boundary planes between the nine process blocks. (The dimension 126 permits division by 9, with an even number of planes per process, which is needed for the red/black subgridding division.) The centered starting needle extends 15 grid units from the upper (anode) plane; the cathode plane is at the bottom. Periodic boundary conditions are applied at the side walls of the cube. In this run, 597 statistical tries produced 27,203 discharged grid links. Cutoff (threshold) voltage was set at 0.05. Real time for the calculation on the IBM SP-2 was 5:17:58.

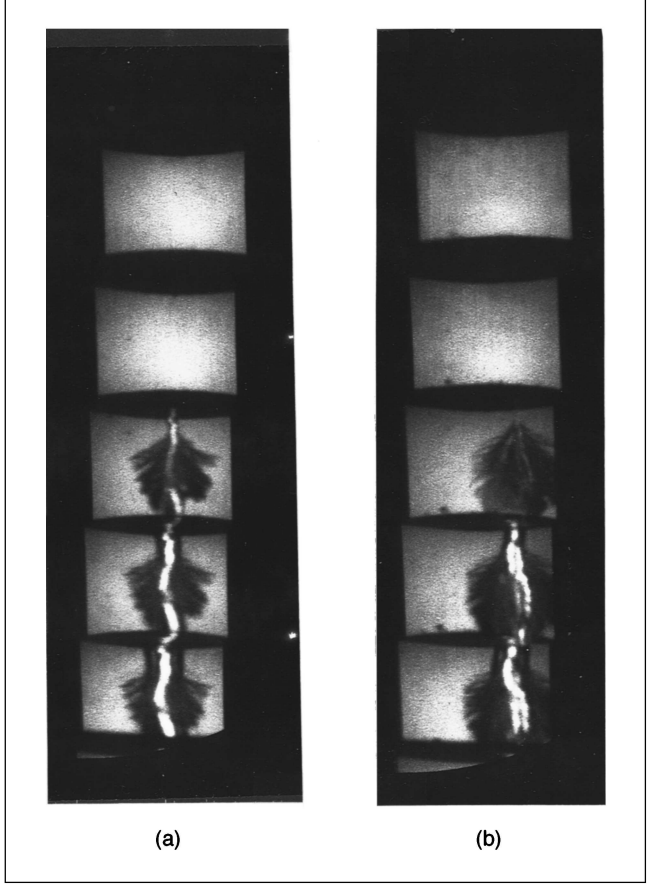

Figure 3. (a), (b) Two examples of fast anode streamer discharges, grown in n-hexane in less than 50 ns (Ref. 26). The gap between electrodes is 3 mm; the radius of the spherical electrodes is 1.27 cm. The anode is at the top, the cathode at the bottom. Peak voltage was 250 kV. The time sequence, is top to bottom, in 100-ns intervals. The growth occurs in less than 30 ns before the third frame shown in each sequence. The dark volume is filled with fine filaments, incompletely resolved here. Other experiments have shown the filaments to have individual diameters as small as 4 μm. They are composed of individual small straight linear segments, most being 30 μm or longer in length.

tine dp_sync. In this solution, each process in turn opens the common output file, appends the necessary information, and closes the file, so that no two processes are writing to the output file at the same time.

The other point in the algorithm at which the code must explicitly deal with the parallel execution of the code is in testing whether the streamer tree has reached the cathode. This is because only a portion of the tree and of the cathode may be present in each of the processes. The issue is handled by a simple test based on which part of the full data array has been assigned to each process. If a process contains a portion of the cathode, then the appropriate part of the streamer tree array that resides in that process participates in a test of whether the tree has reached the cathode.

IV. RESULTS AND INTERPRETATION

The crucial test was, ''Can streamer growth proceed seamlessly through the defined block boundaries?'' Figure 2 illustrates an example of a linear-weighted streamer simulation grown with nine process blocks. The block-partition boundaries were crossed as expected by the DPARLIB version of the program, which contained no explicit reference to boundaries in the array-directed commands. Neither the Laplacian nor the statistical-test subroutine faltered in crossing—meaning that data transfer had proceeded successfully.

Each run delivers an output list of vertices, recorded with the index number of the statistical (red–black) trial. Knowing that they form a singly connected tree, we find the links between sites by means of a backward-search algorithm. Because the statistical trial number—the clock tick for ''Monte Carlo time'' in a stochastic process[25]—has been recorded for each breakdown event, a simulation of time progression in the global growth is possible.

Display is carried out in three dimensions, using color banding to mark the time history of growth. Animation of frames is also readily achieved. Visual presentation of the data can be compared against high-speed photographs from experiments,[2,26] which are two-dimensional (2D) shadowgraphs taken in side view. (In examining the results of very large, dense fractal growths, the combination of color banding and three-dimensional dynamic rotation can often call

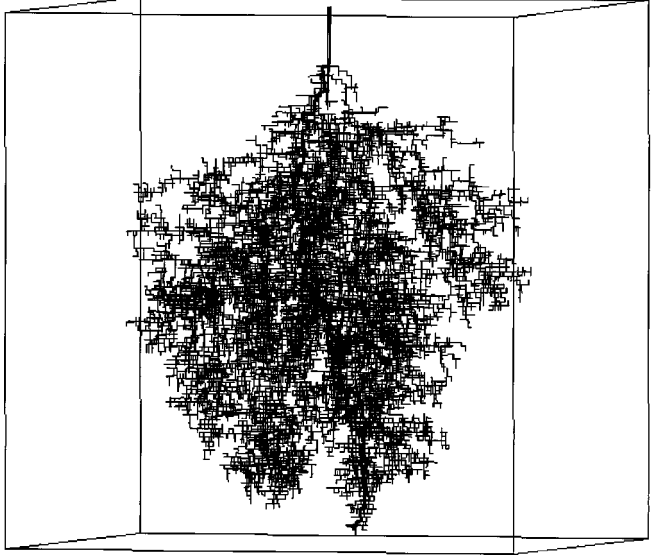

Figure 4. Isometric projection of the 3D dense anode-streamer simulation in Fig. 2. The branching structure in 3D, which was hidden in the Fig. 2 projection, becomes evident. The tree is dense, but the major branches remain self-avoiding because the electric field is screened out of the gaps between. Scaled to the dimensions of Fig. 4, the individual discharged links (each one grid interval) would have length of approximately 30 μm. The apparent visual density is dependent on the choice of linewidth in the graphical reconstruction. Compare against Fig. 3. The leader path connecting the electrodes is shown in heavy line weight.

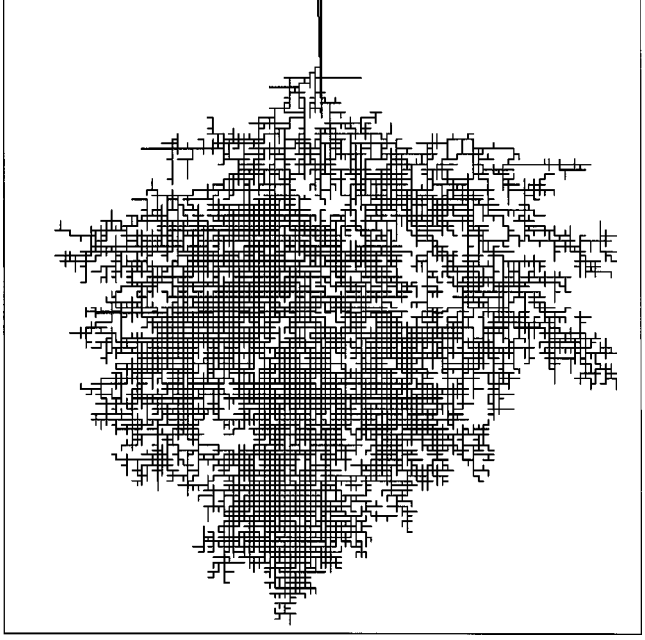

Figure 5. 2D projection of the simulation in Fig. 4, orthogonal to Fig. 2. Note the well-defined cone angle of the upper envelope, a feature corresponding to experiment.

attention to features of the growth that are not obvious from static projections.)

We include two examples of photography from experiment.[26] Figures 3(a) and 3(b) show two separate samples of dense anode-streamer formation, each growing in less than 30 ns in *n*-hexane, in a ''uniform'' field configuration (meaning that the growth commences close to the anode, in a small gap between large-radius spheres). The time sequence proceeds from top to bottom, at 100-ns intervals. The third frame in each sequence captures the start of the ''leader'' breakdown, with the central shock wave growing in diameter in the last two frames.

Figure 2 is one Cartesian projection of a three-dimensional (3D) dense growth, whose isometric view is displayed in Fig. 4. Figure 5 shows the orthogonal 2D projection. What we note with some interest is the well-defined cone-shaped upper envelope to the growth; Stricklett *et al.*[26] have called particular attention to this feature of the experimental observations. This cone represents the boundary at which further lateral growth is cut off by screening, causing the electric field to drop below the cutoff (threshold) value for discharge. This feature of the numerical experiment is undergoing further investigation.

For this case of growth with a uniform (linear) statistical filter, the major divisions of the tree have branched laterally to fill the intervening spaces with twigs. They remain self-avoiding, however. The heavily weighted track in Fig. 4 (and later in Fig. 8) is the first continuous connecting path between electrodes—it simulates the ionization path along which the ''leader'' flashover will occur.

Figure 6 displays a sparse fast cathode streamer, formed under similar conditions, but in the reverse polarity. The positioning of the electrodes has been reversed; the cathode is at the top, the anode at the bottom. Here the timing sequence of frames proceeds from bottom to top. In this Fig. 6 case, the fast ''secondary'' cathode streamer begins from a primary slow-growing bushy structure; Watson[27] and Fenimore[28] have described the primary stage by a bubble simulation. (The bipolarity of the anode and cathode fast-streamer processes presents a further challenge for physical interpretation: How is filamentary tree growth possible in both directions under like conditions?)

Figures 7 and 8 show growth with a square-law statistical filter for comparison against the fast cathode streamer in Fig. 6. In this case, we note that both experiment and simulation show a single major ''trunk,'' or column of growth, proceeding halfway across the gap, then dividing into three or four major branches, whose envelope lies within a more narrow angle than the dense anode streamers of Figs. 3(a) and 3(b). Here the forward-directed field is shaping the growth. The lateral twig growth from the major branches has not filled the intervening gaps; screening does not dominate.

Figures 9 and 10 provide examples of successive growth frames, as a function of cumulative statistical trial numbers. Development of the growth against ''Monte Carlo time'' shows notable acceleration as the gap is shortened and the resultant electric field (voltage gradient) increases. In fact, the lower half of the growth takes place in roughly 20% of the total elapsed duration.

The fronts, or growth-tip envelopes for both power-law weightings, show considerable variability from run to run. This is not surprising, if we recall that the statistical filters are weighted in favor of extreme (high) values of electric field, producing a runaway tendency. Note that the

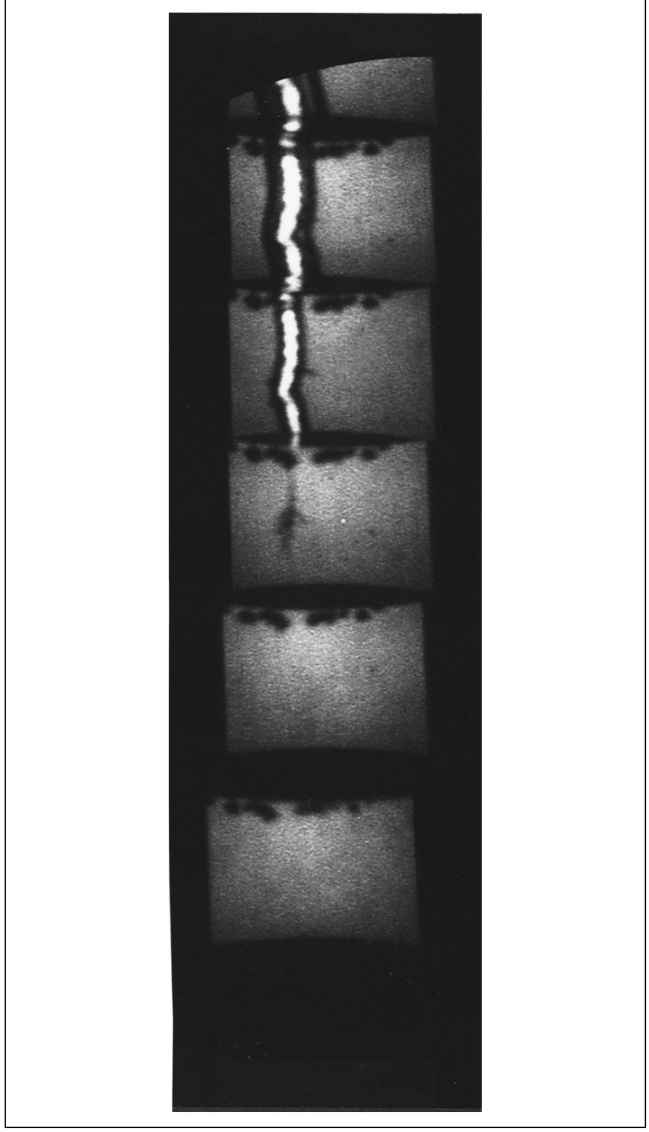

Figure 6. Fast (secondary) cathode streamer, produced under conditions similar to those in Fig. 3. The cathode is at the top, the anode at the bottom. The time sequence now proceeds from bottom to top. The tree structure is noticeably sparser. This event occurs with distinctly lower probability than the fast anode streamer. The bushy growth attached to the cathode at the top is the slower (primary) streamer. Both anode and cathode fast streamers provide the leader breakdown path that leads to high current flow and the formation of the shock wave.

''leader'' conduction path from needle to counterelectrode, shown in heavy line weight, resembles a directed random walk. In some instances, two branches of the leader may reach the counterelectrode on the same statistical try.

V. TIME COMPRESSION FOR HIGHER POWERS

The extremely low probabilities shown in the lower half of Fig. 1 complicate the trial of cube-law and higher powers. For low values of initial field strength, many thousands of empty statistical trials would be required to secure a very few steps of growth. Our solution is to reweight the statistical test, after the manner of the algorithm of Bortz, Kalos, and Lebowitz[25] so that the *a priori* likelihood of at least one discharge event is increased to one.

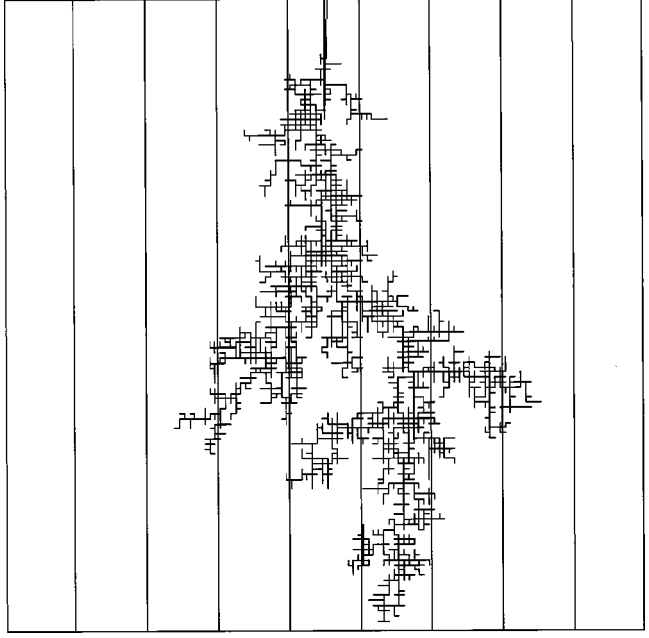

Figure 7. Lateral projection of a square-law-weighted streamer simulation on a grid of 126×126×126. 4227 statistical tries produced 4432 discharged links. The threshold voltage was set at 0.015. To compensate for the very low starting probability, all voltage values on neighbor sites were raised by 0.020 when comparing against the statistical filter. Real time was 13:32:37.

The changed routine for higher powers proceeds as follows, after step (3) in the sequence that was described earlier in Sec. II.

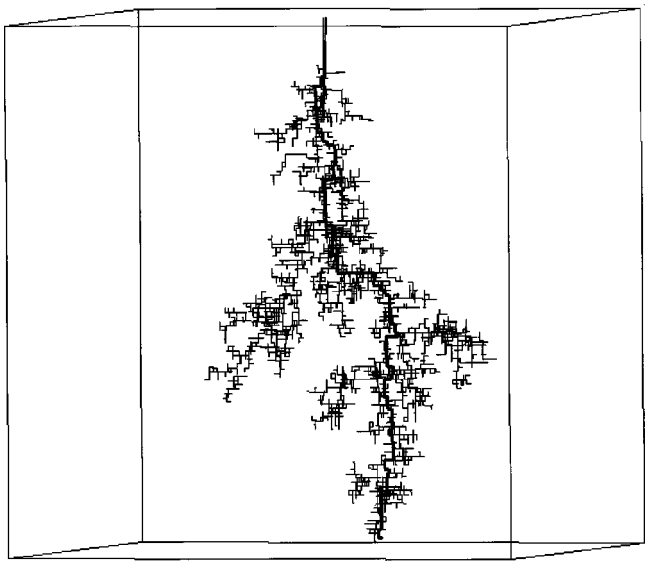

Figure 8. Isometric view of Fig. 7. Compare against Fig. 6. The cathode is now at the top, the anode at the bottom.

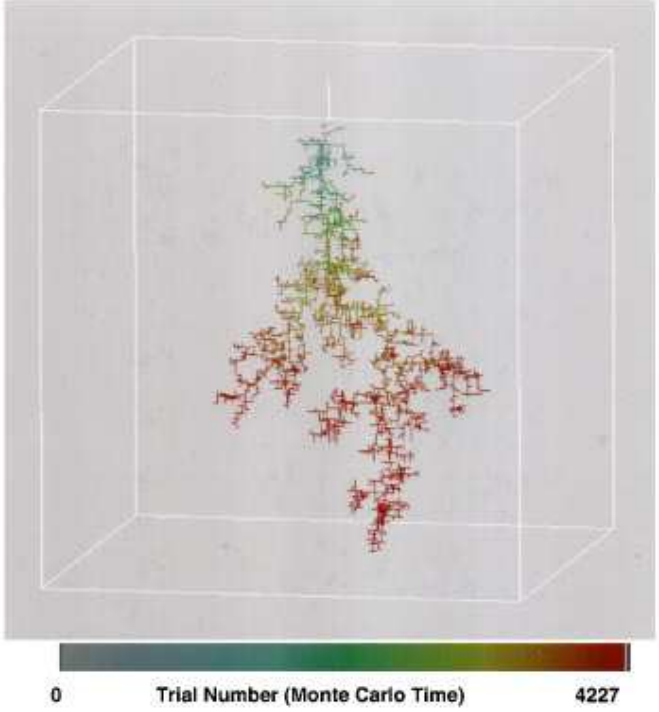

Figure 9. Color-banded version of the streamer tree shown in Fig. 8. The color palette is a linear representation of cumulative statistical tries, a measure of Monte Carlo time.

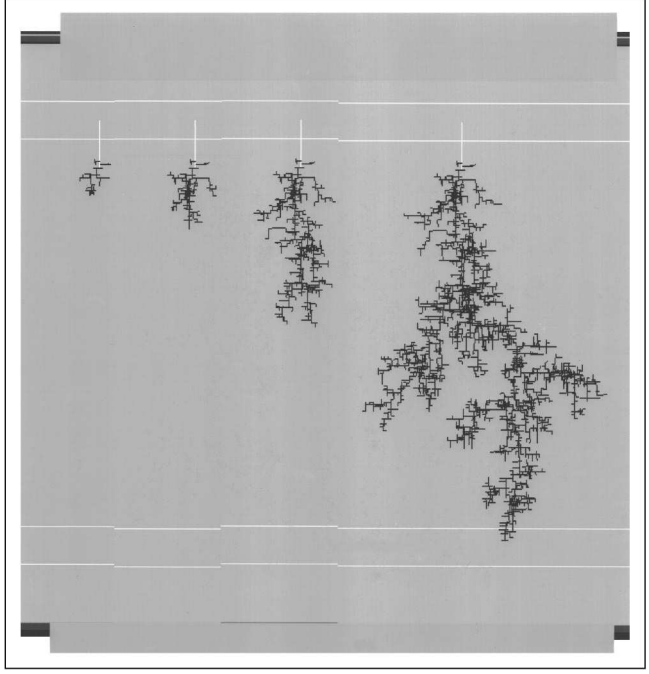

Figure 10. Animated screen development of Fig. 8 against cumulative statistical tries (Monte Carlo time). The acceleration of growth across the second half of the gap is worth noting. Note that the duration of the full growth would correspond to a period of around 30 ns or less than the interval between frames in Figs. 3 and 6.

(a) PACK the surviving neighbor-voltage values onto the first linear real array.
(b) square, cube, or otherwise exponentiate this array onto a second linear array. Sum the values of the elements in this array to a value we call ''sumsq.'' It represents the *a priori* probability of at least one discharge event.
(c) Fill a third linear array, of the same length, with uniformly random numbers. Multiply these by the coefficient ''sumsq''—this is equivalent to ''compressing'' Monte Carlo time. It is also equivalent to the normalization used in many previous examples of algorithms where only the fractal shape, and not the timing, were sought.
(d) WHERE the second array fails to exceed the random array, weighted by the coefficient ''sumsq,'' set the values of the first linear array equal to zero.
(e) UNPACK the revised first array through the position mask of the original neighbors.
(f) WHERE the unpacked 3D array is now greater than zero, add new sites to the existing tree.
(g) Update the statistical trial number (Monte Carlo time tick) by an integer equal to FLOOR((log(rho)/log(sumsq))+1), where rho is a random number.[25] This estimates rather than counts the number of empty trials before a successful trial in which growth is added.
(h) Return to the Laplacian-convergence subroutine.

Again, Fortran 90 has furnished powerful array-directed operations for composing the program logic. The additional computing burden is small, since these operations are performed on few elements, numbering only in the thousands.

The modified algorithm has been successfully run for cube-law and fourth-power examples. Figure 11 illustrates simulation of a cube-law streamer, run on a $128\times128\times128$ grid, with a short anode needle, two grid steps in length. The spareness, restricted lateral branching, and directed alignment are characteristic of growth with this exponent in the power law.

Recent reports by Miyano and collaborators[29] display fast anode streamers in D40 perfluoropolyether and FC10 perfluorocarbon. Photographs of them show strong similarities to Fig. 11.

Thus, we note that physical counterparts to our power-law simulations with exponents 1, 2, and 3 are found in the experimental observations on fast filamentary streamers.

VI. LIMITATIONS

There are several clear limits in the present realization of the model.

(1) Diagonal breakdown paths between grid nodes have not been included in the simulations shown here. Increased flexibility of directional choice, at each event, would be a step towards greater physical realism, albeit imposing a greater bookkeeping burden. (This capability is being added.)
(2) The Laplacian-convergence accuracy has been relaxed in a manner that may tend to favor tip-growth versus

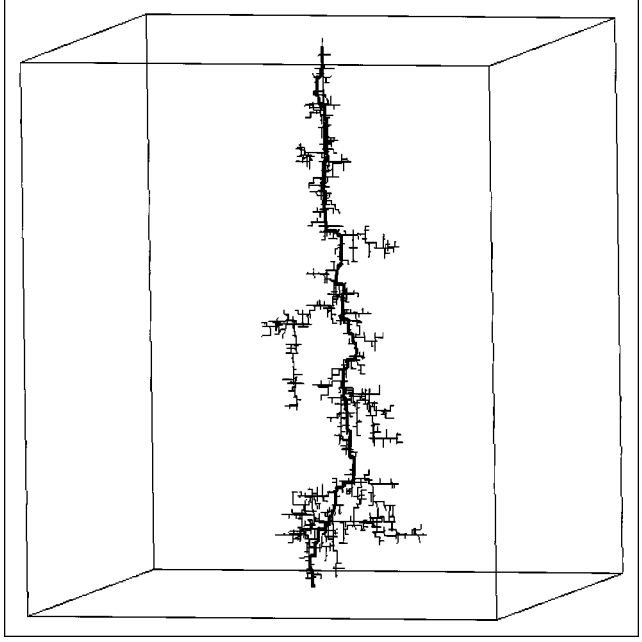

Figure 11. Cube-law streamer simulation on a grid of 128×128×128. The concentrating needle is two grid steps in length. The spare appearance of the filament tree, limited lateral branching, and directed alignment along the uniform field are characteristic of growth simulations with this power-law exponent. This simulation was produced with the time-compression version of the algorithm. It resembles experimental observations by Miyano et al. (Ref. 29).

side branching to a modest extent. This was considered acceptable at the present semiquantitative condition of comparison against experiment.

(3) Visual comparison against experiment is affected by details of the visual perception, rendering of linewidth weight, and 3D obscuration. (Thus, comparing Fig. 2 with Fig. 4, we see that the projection along the Cartesian axis looks distinctly more transparent.)

(4) The restriction to a standard unit length of discharged link does not specify how this implied ''averaging of behavior'' relates to a detailed model of microscopic breakdown.

VII. SUMMARY

The completed program is a robust and flexible tool for investigations of physics. All communications and coordination between copies of the program are hidden within calls to DPARLIB routines. The code is expressed and compiled in standard Fortran 90 using DPARLIB routines that closely emulate standard Fortran 90 intrinsic functions.

The MPI complexity is invisible. Thus, DPARLIB has filled in the needed elements for a high-level language of block parallelism.

The use of parallel methods has afforded improvements in physical realism: large array domains, parallel breakdown across the entire tree structure, and clocking of Monte Carlo time.

The grid size of larger than 100×100×100, scaled to an experimental gap of roughly 3 mm, would represent individual discharged links of 30-μm length. This is comparable to observed segment lengths in experimental streamers. The large grid also permits us to see the effects of a 40:1 increase in local tip–field strength during the course of single runs. Clocking of Monte Carlo time (estimated in the cases where breakdown probability is very low) is considered in this model for the first time; thus the distribution, directions, and rate of growth are followed together, as they are controlled by the evolving voltage field.

Many problems in the physical sciences are well adapted to such a SPMD parallel treatment. Large arrays corresponding to a spatial domain can be partitioned across many processes. Such algorithms can often be simply expressed in Fortran 90, and DPARLIB enables a quick transition from a serial to a parallel environment, without the need to learn complex communications techniques between processes.

For portability, the code was run under SGI (MIPSpro) and IBM (xpf) compilers. The size of the executable code depends on the choice of grid bounds. The executable version on the SGI Onyx, for a grid of 128 ×128×128, was roughly 110 Mbytes for each of four processes. This was easily within the 3-Gbyte memory capacity of the machine. Speed, using the R10000 MIPS processors with a 195-MHz rate, was sufficient for completion of most runs within 24 h in the absence of time-sharing competition.

A desirable feature for this type of Fortran 90 programming is the ability to handle a large number of very-large-sized arrays within the local process memory. Often a salient issue in physics computing is space, not time. To easily spread a problem over many compute nodes can mean the difference between being able to study a problem and not being able to do it at all.

Comparative timings have been carried out for the routine running on the IBM SP2 and on two SGI multiprocessors (Onyx and Origin). These are for the linear statistical comparison, running on a (small) 64×64×64 lattice. Because the choice of random-number seeds varies with the number of processes, the actual executions are for slightly different problems. The numbers give a general idea of speed on dedicated multiprocessors.

The timings, in seconds, using 1, 2, 4, and 8 processors are as follows:

(1) IBM SP2: 12,445, 5336, 3256, 1770;
(2) SGI Power Onyx: 7767, 3590, 1588, 824;
(3) SGI Origin: 5467, 2310, 1328, 690.

Gains in timing are clearly achieved when adding processors, until the communication burden (between boundary planes on adjacent process blocks) becomes an important fraction of the total effort.

Full run times on the larger 128×128×128 lattice are roughly 10 times as long, since the larger problem involves a longer path from anode to cathode, as well as greatly enlarged Laplacian convergence tasks. The execution of Fig. 11, for example, was carried out on four processors of a shared SGI Power Onyx, with overnight turnaround time.

DPARLIB is based only on standard Fortran 90[30] and on the MPI.[7] MPI is an industry-standard library that implements message passing in parallel-computing environments. Both Fortran 90 and MPI are available on a variety

of computers and workstations. LAM is an open cluster environment for workstations or multiprocessors.[23]

Our dielectric-breakdown code, named CADMUS, will be available through our Web site http://www.itl.nist.gov/div895/sasg/.

ACKNOWLEDGMENTS

The authors are indebted to many colleagues for their useful participation and help. The authors' first ideas for solving the problem came from the ultra-high-speed photographic experiments of Ed Kelley, Ken Stricklett, and collaborators in the NIST EEEL Applied Electrical Measurements group. Suggestions from Jacques Amar helped them understand Monte Carlo time, and Jack Douglas has contributed valuable insights about Laplacian fractals. Jim Sims rendered major assistance with workstations, Fortran programming, and graphics (the routine FRAMES). Tere Griffin contributed color and animation to the results. Howard Hung performed the porting and timing comparisons. Larry Davis and Mitchell Murphy, University of Maryland Institute for Advanced Computing Systems, helped them greatly with their earlier SIMD version of the algorithm, adapted for the CM-2 version of the connection machine. Charles Fenimore has been a constructive critic and collaborator. One of the authors (H.A.F.) expresses his warm thanks to the NIST Information Technology Laboratory for access and guest-researcher technical privileges granted during 1996–1997.

Certain commercial equipment and software may be identified in order to specify or describe adequately the subject matter of this work. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment or software is necessarily the best available for the purpose.

REFERENCES

1. L. Pietronero and H. J. Wiesmann, Z. Phys. B **70**, 87 (1988).
2. W. G. Chadband, IEEE Trans. Electr. Insul. **23**, 697 (1988).
3. W. G. Chadband and T. M. Sufian, IEEE Trans. Electr. Insul. **20**, 239 (1985).
4. S. Satpathy, *Fractals in Physics* (Elsevier Science, Amsterdam, 1986), p. 173.
5. T. J. Lewis, in *The Liquid State and its Electrical Properties*, edited by E. E. Kunhardt, L. G. Christophorou, and L. E. Luessen, Proceedings the NATO Advanced Study Institute, 5–17 July 1989, Sintra, Portugal (Plenum, New York, 1988), pp. 431–453.
6. Larry Davis and Mitchell Murphy of the University of Maryland Institute for Advanced Computing Systems provided access to the CM-2.
7. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI* (MIT Press, Cambridge, 1994), especially pp. 60–80.
8. J. E. Devaney and J. Hagedorn, National Institute of Standards and Technology internal paper, May 7, 1996.
9. J. Hagedorn and A. Heckert, DPARLIB User's Guide, draft of NIST software library documentation, March 18, 1997.
10. L. Niemyer, L. Pietronero, and H. J. Wiesmann, Phys. Rev. Lett. **52**, 1033 (1984).
11. L. Pietronero and H. J. Wiesmann, J. Stat. Phys. **36**, 909 (1984).
12. H. J. Wiesmann and L. Pietronero, in Ref. 4, p. 151.
13. L. Pietronero, C. Evertsz, and H. J. Wiesmann, in Ref. 4, p. 159.
14. M. Murat, in Ref. 4, p. 169.
15. E. Arian, P. Alstrom, A. Aharony, and H. E. Stanley, Phys. Rev. Lett. **63**, 2005 (1989).
16. P. D. Beale and P. M. Duxbury, Phys. Rev. B **37**, 2785 (1988).
17. J. J. Modi, *Parallel Algorithms and Matrix Computation* (Clarendon, Oxford, 1988), pp. 135–143.
18. A. Sanchez, F. Guinea, L. M. Sander, V. Hakim, and E. Louis, Phys. Rev. E **48**, 1296 (1993). In these treatments, idealizations such as radial growth to a remote spherical-shell boundary, with normalized probabilities, favor growth having a simple, well-defined fractal dimension.
19. T. A. Witten and L. M. Sander, Phys. Rev. Lett. **47**, 1400 (1981); Phys. Rev. B **27**, 5686 (1983).
20. A. Erzan, L. Pietronero, and A. Vespignani, Rev. Mod. Phys. **67**, 545 (1995).
21. M. M. Clark, L. M. Ruff, and H. L. Scott, Comput. Phys. **10**, 584 (1996).
22. K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics* (Springer, Berlin, 1988), especially, pp. 28–29.
23. D. Burns and R. B. Daoud, Supercomputing Symposium '94, June 1994, Toronto, Canada; code available at http//www.osc.edu/lam.html.
24. For a similar treatment of a second-order operator, see, for example, A. Villareal and J. A. Scales, Comput. Phys. **11**, 388 (1997), especially p. 392.
25. I. Beichl and F. E. Sullivan, Comput. Sci. Eng. **4**, 91 (1997); see also A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, J. Comput. Phys. **17**, 10 (1975) especially Appendix B; J. G. Amar and F. Family, Phys. Rev. Lett. **74**, 2066 (1995).
26. K. L. Stricklett, E. F. Kelley, H. Yamashita, H. Kawai, and C. Fenimore Conference Record, IEEE International Symposium on Electrical Insulation, 1990, pp. 161–164. These data were taken by means of a high-speed ''optical storage line'' technique which limited the available spatial resolution. Dr. Stricklett made his original data available to the authors.
27. P. K. Watson, Conference Record of the 10th International Conference on Conduction and Breakdown in Dielectric Liquids, Grenoble, France, 10–14 September 1990, pp. 22–28.
28. C. Fenimore, Conference Record, IEEE International Symposium on Electrical Insulation, Cambridge, MA, 5–8 June 1988, pp. 27–30.
29. K. Miyano, K. Yamazawa, M. Pompili, C. Mazzetti, and H. Yamashita, 1997 Annual Report of the Conference on Electrical Insulation and Dielectric Phenomena, Minneapolis, 19–22 October 1997, Vol. II, pp. 640–643 (available from IEEE Service Center, Piscataway, NJ). We refer especially to Figs. 3 and 4 in this paper. The authors associate the properties of the observed streamers with the viscosity of the perfluoro liquids used.
30. J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener, *Fortran 90 Handbook* (McGraw–Hill, New York, 1992).