SAND-90-0227C

DE90 007070

Architecture and Computing Philosophy of the QUICKSILVER, 3D, Electromagnetic, Particle-In-Cell Code

M. L. Kiefer, D. B. Seidel, R. S. Coats, J. P. Quintenz,

T. D. Pointon, and W. A. Johnson

Sandia National Laboratory

Albuquerque, New Mexico 87185

Overview of QUICKSILVER

Charged-particle simulations in three dimensions are now performed routinely in the Pulsed Power Sciences Directorate at Sandia with the QUICKSILVER suite of codes. QUICKSILVER is a multitasked, finite-difference, three-dimensional, fully relativistic, electromagnetic, particle-in-cell code developed at Sandia. It is targeted for use on current and near-term supercomputers, such as the Cray X-MP/416, which are characterized by large, shared central memories and multiple processors. QUICKSILVER has already been used to simulate ion diodes, magnetically insulated transmission lines, microwave devices, and electron beam propagation.

QUICKSILVER is actually a suite of codes; in addition to the main simulation code there are several support codes. The problem geometry is generated with a preprocessor and the simulation results are examined with one or more postprocessors. The MERCURY preprocessor assists the user in defining the mesh, boundary conditions, and other input parameters. The FLASH and AVS[1,2] postprocessors are used to examine a wide variety of simulation output, including 3D rendering of particle positions, conductor surfaces, and scalar and vector quantities. The PLOTPFF postprocessor displays 2D slices and 1D pencils derived from 3D scalar and vector quantities. Additionally, time histories of various simulation quantities can be examined and manipulated with the IDR[3] postprocessor. Each code in the QUICKSILVER suite may be run on the hardware platform for which it is best suited. Currently, QUICKSILVER is run on a Cray X-MP/416, FLASH and AVS on a Stardent 3D graphics workstation, and MERCURY,

---

IDR, and PLOTPFF on a VAX computer. A robust and functional computer network then ties the suite together. All codes in the suite adhere to standards as much as possible in order to ease porting to different hardware platforms.

Before describing the suite in detail, it is worthwhile to highlight some salient architectural features of the QUICKSILVER simulation code. All grid-related quantities (electromagnetic fields and current density, for example) are stored in central memory and particle information may be stored out of memory if there is not sufficient central memory. In order to squeeze the largest possible simulation into central memory, large conductor volumes in which fields are always zero must not be stored. This necessitates constructing the simulation region from "blocks" logically connected along arbitrary planes. The overhead required to accomplish this block connection in the field-solving routines uses a very small percentage of the total processing time. After the grid-related arrays have been allocated in memory, the remainder of the memory is used to cache particle information. When this cache is full, it is necessary to shuffle particle information in and out of the memory cache and onto disk or other out-of-memory storage.

The MERCURY Preprocessor

The process of generating input data for three-dimensional simulations is difficult and error-prone. MERCURY is a menu-driven preprocessor that is used in defining the variably-zoned, finite-difference mesh, the problem geometry, the boundary conditions, the dielectric volumes and other input parameters. MERCURY allows free-format input and provides on-line help. All input for a QUICKSILVER simulation is processed and checked for errors and inconsistencies.

QUICKSILVER uses a nonuniform, multiple-block, finite-difference mesh with staggered full and half grids. A nonuniform mesh defined by the relationship between the grid index i and the physical grid location x

$$x(i) = x_0 + ai + bi^2 + ci^3$$

is supported by MERCURY. Different regions of the mesh can have different descriptions of the relationship between i and x. Across any interface between such mesh regions, the function x(i)

must be continuous in order to retain accuracy in the field solution. The MERCURY mesh generator ensures that this condition is met as the mesh is produced. Logically connected blocks, i.e., multiple conformal regions of space with a local mesh, are also supported. MERCURY can also interactively plot the generated mesh. Cartesian, cylindrical, and spherical coordinate system meshes can be generated by MERCURY. Figure 1 shows a two-dimensional slice of a multiple-block, variably-zoned, finite-difference mesh. The MERCURY mesh generator is also used to generate meshes for other finite-difference simulation codes at Sandia.

Conducting surfaces and volumes and dielectric volumes are easily generated with MERCURY by adding or removing objects, such as a cylinder, from a library of simple, solid objects. MERCURY then fits the object to the finite-difference mesh. The generated surfaces can then be viewed interactively. Figure 2 shows a coaxial conductor generated on a Cartesian mesh.

MERCURY also allocates memory for arrays in QUICKSILVER so that only the minimal memory required for a simulation is used. This is accomplished using FORTRAN parameter statements and requires recompilation of some QUICKSILVER modules for each simulation. This memory-conservation feature reduces memory charges, which for some simulations, can exceed CPU charges.

The QUICKSILVER Fields Solver

The minimization of the central memory required to store grid-related arrays is further enhanced in the electromagnetic fields solver by storing each multiple-block array in a single, one-dimensional array. Each of these one-dimensional arrays is partitioned, by the use of simple offsets, into arrays containing the component for the various blocks. At the highest levels of QUICKSILVER, these one-dimensional arrays, which have three dimensions implicitly embedded, are used. At the lower levels of the field-solving routines, where the bulk of computations are performed, these grid-related components are referenced as local, single-block, three-dimensional arrays. This simplifies the coding in QUICKSILVER, and greatly improves its readability. After the grid arrays have been allocated in memory, the remainder of the memory is used to cache particle information. A buffer surface, one cell in thickness, is used for each mesh

block to ease connectivity of blocks in the field-solving algorithms. Any grid-related array not needed in a simulation (for example, charge density in a fields-only simulation) has its length set to one to further conserve memory.

The QUICKSILVER fields solver utilizes explicit[4] and implicit[5] finite-difference, leap-frog algorithms. Multiple dielectric volumes are allowed for regions with no particles. Available boundary conditions include conductors, inlet and outlet boundaries, mirror symmetry and periodic symmetry. Simulations can be performed in Cartesian, cylindrical or spherical coordinate systems. Currently, the inlet boundary condition will drive multiple, independent TEM modes and the outlet boundary condition will handle all modes with a fixed wave velocity. More general inlet and outlet boundary conditions are being developed. Conductor boundary conditions are implemented directly in the fields solver using bit-wise operations. The fields solver has been optimized for the vector hardware of the X-MP/416 which can overlap I/O from central memory as well as overlap floating point operations. No nonstandard FORTRAN is necessary for this optimization, only for the bit-wise operations. The algorithms operate at an average rate of 130-140 MFlops. This is slightly more than one floating point calculation per clock cycle, on average. The nesting order of do-loops is set internally for optimal vector length and to avoid memory bank conflicts.

To fully utilize the capability of a multiple-processor supercomputer such as the X-MP/416, multitasking of the fields solver is employed. The Los Alamos Autotasking Library[6,7] is used to implement multitasking on the Cray X-MP. This library is based upon a multitasking model in which indistinguishable tasks (processors) subdivide the work of a single program in a self-scheduling manner. All multitasking is incorporated into the code via directives to the Autotask pre-compiler. In the field-solve algorithms, processing is done block-by-block. All processors work on each block simultaneously using a "concurrent-outer-vector-inner" method (COVI) on the do-loops. In this parallelization method, the outer loop is performed in parallel and the inner loop is vectorized. To facilitate load balancing in the fields solver which has 3 levels of loops, the two outer loops are combined into one to increase the number of concurrent iterations. Another way to visualize this multitasking approach, is to to think of the three-dimensional field arrays in each block as two-dimensional arrays of vectors (1D arrays). The algorithm is parallelized by requesting that the available tasks individually and exhaustively process the entire array of vectors

in a self-scheduling manner. When all the vectors are processed in one block, the tasks are synchronized and then proceed to the next block. The only multitasking overhead required is the task synchronization between blocks. Algorithms for treating boundary conditions, which are over two-dimensional surfaces, employ an analogous COVI approach by treating 2D arrays as 1D arrays of vectors. Correspondingly, task synchronization is required between each surface processed.

The most pessimistic measure of multitasking speedup is defined as the CPU time for a unitask job with no multitasking constructs divided by the CPU time for the multitasked job. Using this measure, speedup of the fields solver on a Cray X-MP with 4 processors is 3.2. This speedup takes into account overhead due to serial calculations as well as system overhead for implementing multitasking. Measuring speedup as CPU time for a multitasked job using one processor divided by CPU time for a multitasked job using multiple processors, the speedup on a Cray X-MP with 4 processors is 3.66. This implies that over 97% of the calculations in the fields solver are performed in parallel. This also implies that the autotasking library introduces approximately 10-15% intrinsic overhead to coordinate the scheduling of tasks to do parcels of work. On an Alliant computer using 4 processors, speedup was 3.4 by either measure.

The QUICKSILVER Particle Handler

The physics features of the QUICKSILVER particle handler include advancing particle positions with three-dimensional, fully-relativistic kinematics. Multiple particle species with particle creation via preloading, beam injection and space-charge-limited field emission are allowed. The particle handler supports the same boundary conditions and coordinate systems as the fields solver. For most applications of the code, the particle handler will consume by far the most resources; consequently, it has been written to be fast and efficient while retaining as much clarity and ANSI-standard coding as possible. Particle I/O to and from central memory employs multiply-buffered memory caching for high efficiency. In addition, a low overhead scheme for packing particle information is used in order to reduce the large amount of storage required for particle data. The particle handler operates at an average rate of 35-40 MFlops on an X-MP.

Multiple algorithms for charge conservation will be available, including a local exact conservation algorithm and a pseudo-current algorithm[8] which diffuses errors in the charge to the simulation boundaries.

In the particle-handling algorithms of QUICKSILVER, multitasking is less straightforward to implement than in the fields solver, primarily for two reasons. Every particle can be advanced ("pushed") independently of all others. Thus, at the most basic level of multitasking, the entire set of particles is segregated into a large number of bins typically with a only a small number of these bins in central memory at one time. Then each processor pushes all particles in the bin, allocates their motion to the current density array, allocates their charge to the charge density array, and then puts them into a cache for output back to disk. The first multitasking difficulty arises because accessing particle data is complicated when much of the data are stored out-of-memory. This introduces problems because the inflow and outflow of particles to and from disk is not synchronized due to particle attrition during the particle-pushing stage. The second difficulty is that shared memory in a Cray X-MP computer introduces fundamental obstacles to rigorous multitasking; multiple tasks processing independent particles can potentially attempt to increment the same location in the charge and/or current density arrays simultaneously (collide), with the result that all but one of the particles' contributions will be lost.

The first difficulty mentioned above can be circumvented by forcing synchronization of particles moving to and from disk. This requires implementing one additional memory cache layer in the particle-caching algorithm. Several solutions to the second difficulty suggest themselves; we have chosen to implement four in QUICKSILVER. First, particles can be partially sorted in such a way that particles processed by simultaneous tasks cannot collide in the density arrays. The sort used is an efficient bucket sort which scales linearly in the number of particles. Although this approach preserves simulation integrity, we incur the overhead of the sort and the possibility that we can't sort enough to safely utilize all of the available processors. A second solution is to provide separate current density arrays for each task, and then sum them after all particles have been pushed. This approach is guaranteed to safely utilize all available processors but often entails prohibitive memory costs. A third solution is to ignore collisions in the hope that they are rare and thus have a negligible effect. While this is certainly the most efficient approach, it has inherent integrity problems. Finally, the fourth solution is to allow all

processors to push particles, but only allow one to allocate the current and charge density arrays. This entails a fixed memory overhead, which for a large grid can be much less than providing separate current density arrays, while eliminating the need for sorting. This method, which is not yet implemented, is an alternative to the second one and the choice of this method or the second one would be dictated by memory cost and availability.

In tests of the particle multitasking algorithms, we have made several interesting observations. We have found that the overhead required to partially sort particles is typically less than 1%. However, if the grid is not sufficiently large, it is often not possible to sort enough to safely use all available processors in a load-balanced fashion. In our tests of the option that ignored particle collisions in the density arrays, we have never observed a resultant significant error. In fact, the errors we have seen were in all cases smaller than variations introduced by standard statistical techiques for allocating current and charge. We have made an initial measurement of speedup in the particle handler on a Cray X-MP using 4 processors and found it to be 3.0 by the more pessimistic measure. In that simulation, the number of particles was quite small compared to the number expected in a production multitasked simulation. Also, the number of emitted particles per timestep was unrealistically large compared to the total number in the simulation. This added atypical serial overhead since the emission algorithm does not execute concurrently.

Given the inherent difficulties described above in multitasking the particle handler, it might seem best to just forego concurrency in favor of the simpler, serial algorithms. However, there are two basic reasons that multitasking is necessary. First, multitasking when a single job can acquire dedicated use of more than 1 processor will give the user an answer as quickly as possible. Second, if a single job requires all of a resource on a computer, then multitasking will reduce the cost. Such a monopolizing job will undoubtedly be charged for use of the entire machine since no one else can make simultaneous use of it. It is for this second reason that multitasking has been implemented in QUICKSILVER. QUICKSILVER simulations will often require nearly all of the memory on a computer and so multitasking will reduce charges, as well as reduce the amount of time the machine is monopolized.

Postprocessors and Data Visualization

"Visualizing" (that is, comprehending through visual renditions) the data produced by QUICKSILVER has proven difficult, mainly because the data are defined on the four-dimensional domain of space and time. These data are a nontrivial extension over what has been regarded in the past as three-dimensional data, namely a scalar variable on a two-dimensional grid. At the extreme, QUICKSILVER data can be as complex as three-vector data on the four-dimensional domain. The software we use consists mainly of two applications: FLASH, a graphics workstation based postprocessor written at Sandia as an integral part of the QUICKSILVER suite; and AVS, a scientific visualization, three-dimensional graphics application written by Stardent Computer, Incorporated. Both are built upon the PHIGS ANSI-standard graphics package, the proposed ANSI-standard PHIGS+ extensions, and the market-standard X Windows software, so they are fairly portable. Our techniques demand very fast computer hardware. We are currently using a Stardent graphics workstation as our hardware platform.

Data from QUICKSILVER have five elementary forms, each with time-dependence: scalar values on the grid, such as charge density; vector values on the grid, such as current density; unordered points with no grid dependence, such as particle positions; unordered vectors with no grid dependence, such as particle momenta; and complex surfaces which are conformal with the grid, such as conductor boundary conditions. The first difficulty in comprehending objects in three-dimensions is representing the distance from the object to the viewer (depth) using visual information. We use the following methods to represent depth: perspective-viewing; depth-cueing, where the color of objects fades to the background color as depth increases; and z-buffering, where an object will mask portions of objects behind it in the line-of-sight. Stereo-viewing, where a stereoscopic image is produced using a color monitor and special viewing glasses, will be employed in the future. Further, manipulation of objects with a high degree of interactivity is required for the viewer to completely understand the three-dimensional structure of the object. Real-time rotations, translations, and scaling utilizing a mouse or set of dials are employed for this purpose. Finally, time dependence is displayed by showing consecutive time snapshots of the same data type at a fast rate, typically 3 to 4 frames per second.

The display of particle positions in space, unordered points with no grid dependence, is perhaps the easiest to render. We display them as dots or as 3D spheres. Spheres give more depth information, at the expense of reduced transformation speed, since they shrink as depth increases due to the perspective viewing. Different particle species, such as electrons and protons, are displayed with different colors. The particle density (the fraction of the total which are actually visible) can be modified to accentuate gross features (low density) or fine features (high density) of the particle field. By sorting the particle field so that only particles between two closely spaced, parallel planes are visible, it becomes possible to "look inside" the particle field by rapidly moving the location of the planes (like looking at each slice of a loaf of bread in rapid succession). We plan to display particle momenta, unordered vectors with no grid dependence, by attaching a line segment to each sphere. The orientation and length of a segment will correspond to the direction and magnitude of the particle's momentum.

Conducting surfaces, which are quite complicated in the typical QUICKSILVER simulation, are at the next degree of difficulty for visualization. We use multiple light sources and smooth shading of a surfaces to show the interrelationships of the conducting surfaces. Figure 3 shows a plot of a complex conductor. In addition, the viewer often wishes to display data, such as a particle field, with the conductor data for a better frame of reference. Figure 4 shows a plot of about 10,000 electrons in a simulation represented as spheres overlaid on the conductor shown in Figure 3. When overlaying data with conducting surfaces, the user can make the surfaces nearly transparent. This allows the viewer to "see through" any surface that might otherwise occlude data, such as particles, while leaving enough of the conductor image to provide orientation information.

Scalar data defined on the finite-difference grid, for example, charge density, is displayed using small objects at the locations of the data on the grid. These objects may be crosses, dots, or spheres. The hue color assigned to each object is then tied to the corresponding scalar value. Values at the lower end of the scale are blue, changing to cyan, green, yellow, and finally red for values at the upper end of the scale. In this manner, a field of objects in three dimensions is presented to the viewer to represent the scalar field. Unfortunately, it is difficult to "look inside" the field, as well as determine iso-surfaces (arbitrary surfaces upon which the scalar field is constant). To "look inside" the field, the viewer can display individual planes of the scalar field.

Iso-surfaces of the data are rendered by determining a set of points with scalar values equal to the iso-surface value and then "tiling" the set of points with triangles to represent the iso-surface[9].

Vector data defined on the grid, for example, current density, can be displayed using the scalar data techniques above if one wishes to view only one component of the vector field or its magnitude. Visualizing all the information in the three-vector field at once on a color monitor is extremely difficult since there are now three physical quantities at each point on the three-dimensional grid. One technique we use is to display each vector quantity using short, equal-length line segments at each grid point with the orientation of each segment corresponding to the direction of the vector quantity. The vector magnitude is incorporated as the hue of each segment instead of variations in the length of the segments. This is because long segments interfere with others in the 3D vector field rendition. Also, the use of equal-length line segments allows the viewer to apply the relative foreshortening of the line as a visual cue to a vector's orientation in space. To "look inside" the volume of vector data, one can again pick out individual planes of these line segments.

Many hardcopy formats from the Stardent workstation are available. We have the capability to convert the video from the workstation to NTSC format, either composite, Y/C, or RGB. These signals can then be recorded onto video laser disks or videotape. The laser disk units can be controlled by the workstation for unattended, frame-by-frame recording. Hardcopy can be produced on 300 dot per inch black and white or color plotters or as very high quality 35mm or 8×10 slides on a Dicomed system.

The IDR postprocessor is a VAX computer-based application that allows manipulation and display of the following time history data from QUICKSILVER: electromagnetic field values, line, surface and volume integrals of field quantities, particle counts, charge and energy by species, and various performance monitors. IDR has extensive tools for manipulating time history data. These tools include addition, subtraction, multiplication, division, differentiation, integration, filtering, Fourier transformation, time shifting, and overlays.

The PLOTPFF postprocessor is also VAX-based and allows manipulation and display of two-dimensional slices and one-dimensional pencils of scalar grid data and magnitudes and components of vector grid data. PLOTPFF also can apply Fourier transforms to these data, integrate the data, display two-dimensional contour plots, and extract one-dimensional pencils

from the two-dimensional slices. In the future we will merge the capabilities of PLOTPFF into the AVS software and discard the VAX version.

Inter-Machine Transfer of Postprocessing Data

The transfer of the large data files generated on the Cray X-MP to the Stardent workstation or VAX computer for postprocessing requires a file format that is portable among machines with vastly different word structures and that is minimal in size. It is quite easy to generate millions of 64-bit words of data from a QUICKSILVER job that must be postprocessed. To address this, the Portable File Format (PFF) and supporting library of modules has been developed. The PFF module library consists of low and high level routines which write data into a file (and of course, read it back) in a format which is easily passed between machines. Typically, only a few of the low level routines must be modified when porting to new machines. The high level routines, which are the only ones directly called by a user writing an application, are completely portable. Both FORTRAN and C versions of the PFF library are available. Using the library, all data are written to the PFF file as two-byte integers. The format of a PFF file is extensible to accomodate most any data type desired. For instance, besides supporting the types of QUICKSILVER data described in the section above, image-processed data formats are also supported. PFF is currently undergoing a major modification to enhance its extensibility.

Acknowledgements

We would like to acknowledge Alex Siegel, presently at Cornell University, and Dan Shawver, presently at the University of New Mexico, for their contributions to the development of the FLASH postprocessor. Also, much of the background work for generating video recordings at Sandia was done by John Mareda and the software for generating Dicomed plots from the Stardent workstation was written by Constantine Pavlakos.

This work was supported by the U.S. Department of Energy under Contract No. DE-AC04-76DP00789

References

1. C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, and F. Moss, IEEE Computer Graphics and Applications, "The Application Visualization System: A Computational Environment for Scientific Visualization", (July 1989).

2. Application Visualization System User's Guide and Developer's Guide, Stardent Computer Inc., Newton, Massachusetts, 1989.

3. W. B. Boyer and L. B. Bishop, "IDR -- Operations Manual for the Interactive Data Reduction Program," Sandia National Laboratory, May 1988, unpublished.

4. O. Buneman, Relativistic Plasmas, O. Buneman and W. Pardo, eds. (Benjamin, New York, 1968), p. 205.

5. B. B. Godfrey, Proc. 9th Conf. on Numerical Simulation of Plasmas, Northwestern University, paper OD-4 (1980).

6. F. W. Bobrowicz, "Autotasking on Cray-XMP Supercomputers," Los Alamos National Laboratory, LA-UR-88-3269 (1988).

7. F. W. Bobrowicz, S. H. Dean, D. A. Mandell, and W. H. Spangenberg, "LANL Multitasking Overview," Los Alamos National Laboratory, LA-UR-87-759 (1987).

8. Barry Marder, J. Comp. Phys. 68, 48 (1987).

9. W. Lorenson and H. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Computer Graphics (Volume 21, Number 4, July 1987).

X Windows is a trademark of MIT.

Figure 1. An example of a nonuniform, two-block, finite-difference mesh generated by MERCURY

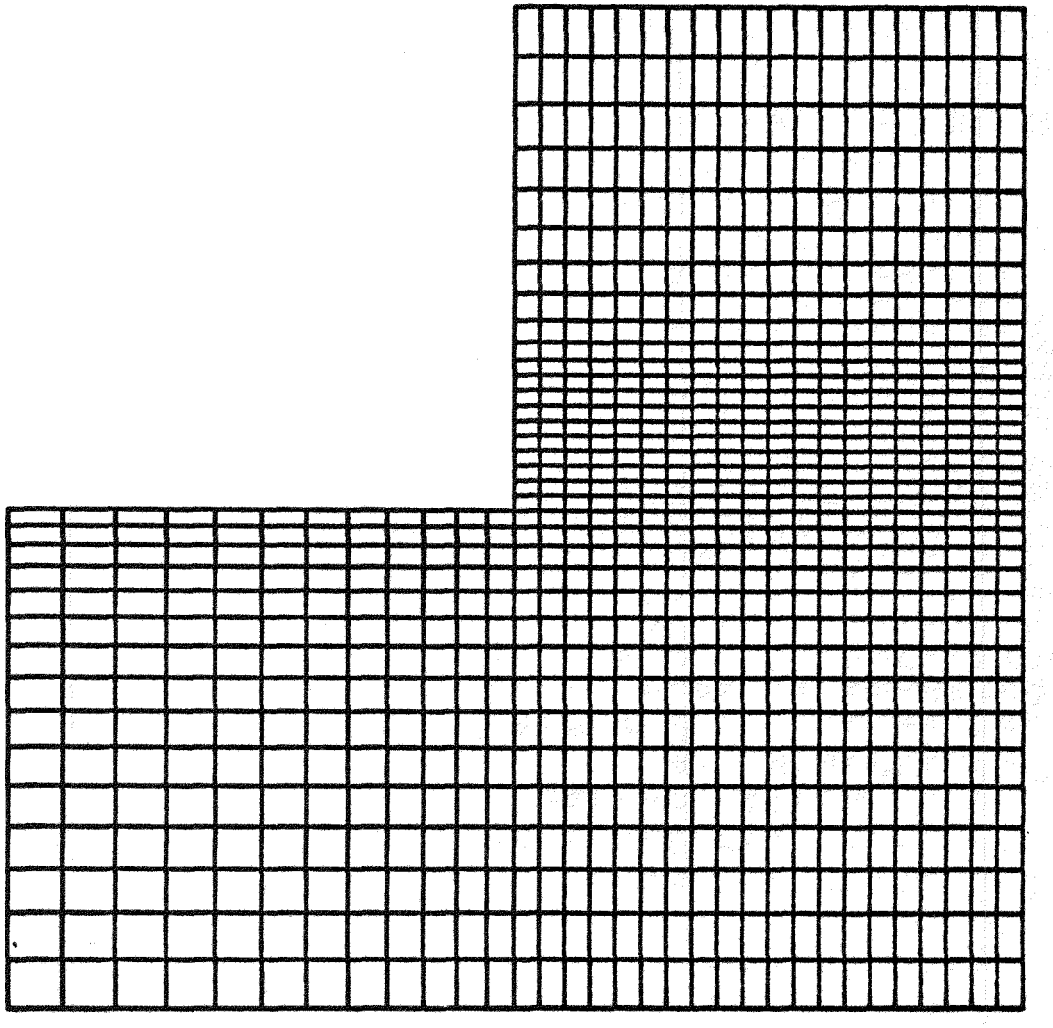

Figure 2. Sample MERCURY output showing a coaxial conducting structure overlaid on the mesh

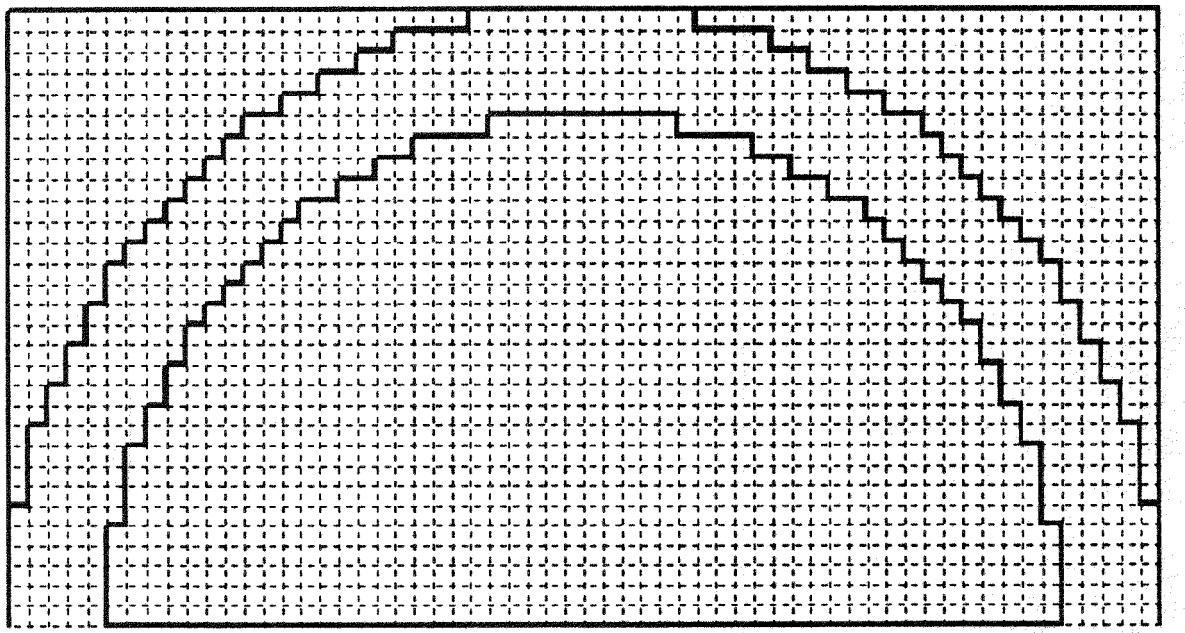

Figure 3. A sample plot of a conducting surfaces produced by the AVS software

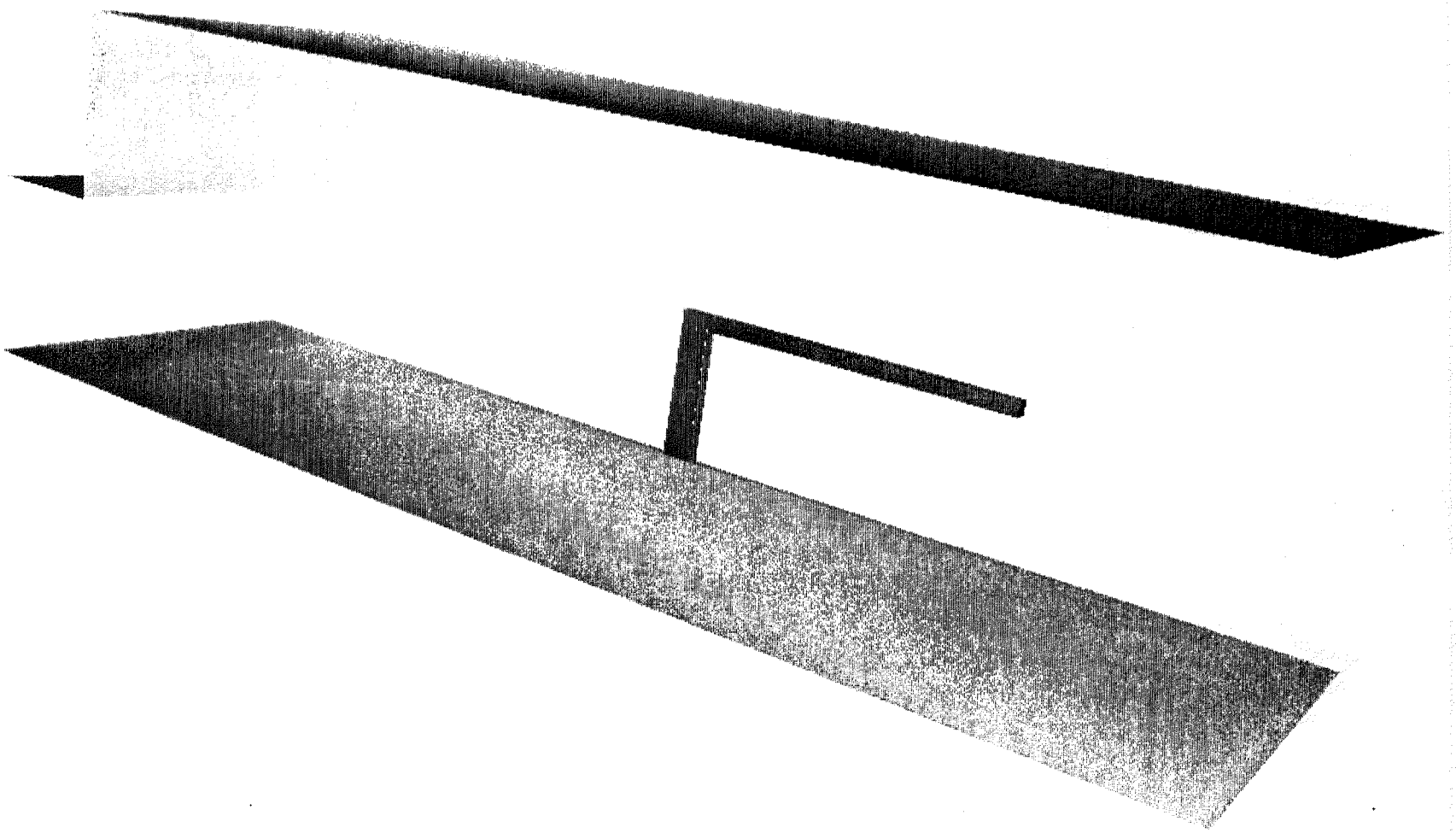

Figure 4. A sample plot produced by the AVS software showing particle positions overlaid with conducting surfaces