27
10/30/79
24 to 27 T/S

SAND-79-1748

UNLIMITED DISTRIBUTION

OPTIMIZATION OF THE PARTICLE PUSHER
IN A DIODE SIMULATION CODE

MASTER

M. M. THEIMER AND J. P. QUINTENZ

Sandia Laboratories

SAND-79-1748

UNLIMITED DISTRIBUTION

OPTIMIZATION OF THE PARTICLE PUSHER IN A DIODE SIMULATION CODE

M. M. Theimer and J. P. Quinterz
Plasma Theory Division - 4241

Sandia Laboratories
Albuquerque, New Mexico 87185

September 1979

ABSTRACT

The particle pusher in Sandia's particle-in-cell diode simulation code has been rewritten to reduce the required run time of a typical simulation. The resulting new version of the code has been found to run up to three times as fast as the original with comparable accuracy. The cost of this optimization was an increase in storage requirements of about 15%. The new version has also been written to run efficiently on a CRAY-1 computing system. Steps taken to affect this reduced run time are described. Various test cases are detailed.

DISCLAIMER

MASTER

TABLE OF CONTENTS

	<u>Page</u>
I. Introduction.	5
II. Second Order Versus Fourth Order Particle Pushers.	6
III. Vectorized Particle Pusher.	9
IV. Program Structure	11
V. Specialized Pushers	12
VI. Unsafe Optimization	13
VII. Instack Loops	13
VIII. Test Cases.	14
IX. Conclusions	16
Table 1: Run Statistics for Hollow Cathode Diode Test Case. Total Simulation Time was 0.3 ns.	18
Appendix A: Timing Estimates	19
References.	20

OPTIMIZATION OF THE PARTICLE PUSHER IN A DIODE SIMULATION CODE

M. M. Theimer and J. P. Quintenz
Plasma Theory Division - 4241

1. Introduction

The original particle-in-cell (PIC) diode simulation code developed at Sandia in the early 70's has undergone a continual evolution since its inception.¹⁻⁵ Both electron and ion diodes have been investigated with the code providing information on particle currents and flow patterns. A wide variety of diode configurations have been tested successfully. During the code's development an effort has been made to make the code general enough to treat the wide variety of applications which have arisen in the past and to anticipate some of the needs of the future. The increase in code generality has been coupled with improvements in the code structure which make use of the code much easier.

Recently, as simulations became more involved and time consuming, a need to improve the code efficiency, especially as measured in run time, was realized. Since most of the code's execution time is involved with particle pushing (advancing particles through their self consistent electric and magnetic fields), this was the logical part of the code in which to begin the optimization process. Changes which have been made to the code have been made with two goals in mind: to improve the overall run

time and to make a smooth transition in the eventual transfer of the code to a CRAY-1 computing system. It is the purpose of this report to describe these changes and the effects they have had on program run time, accuracy, and storage requirements.

The primary changes that have been made to the code are: replacement of the fourth order integration scheme with a second order one, vectorization and specialization of the integration and field interpolation code, and revision of the subroutine structure and decision making structure of the entire particle pushing algorithm. (This includes integration of the trajectory equations and checking and handling of illegal conditions and particle-surface encounters.)

In Section II various particle pushing algorithms are discussed and the most suitable one for this application is chosen. Section III describes the vectorization of the particle pushing routine. A revised subroutine and decision-making structure is described in Section IV. Sections V and VI describe specialized algorithms for pushing particles in various combinations of E and B field components and also the unsafe optimization feature available on CDC computers. Section VII describes an unsuccessful attempt to utilize the instack feature of CDC computers. Various test cases are described in Section VIII and conclusions are presented in Section IX.

II. Second Order Versus Fourth Order Particle Pushers

The most recent versions of the diode code have used a fourth order integration scheme in the particle pusher. The basis for the choice of a fourth order scheme over a simpler second order one was the fact that in

pushing a single particle in analytic fields, the fourth order scheme achieved the same accuracy as several commonly used second order pushers in much less computer execution time.⁶ The reason for this is that a much larger time step, Δt , is permissible with the fourth order scheme than with the second order scheme to achieve the same accuracy.

In the diode simulation code, however, several thousand particles are advanced during each time step. Advancing this many particles involves repetition of the same operations many times and lends itself well to vectorization (as discussed in Section III). The desire to vectorize the particle pusher suggested a new look at pushers and their implementation. As will be seen later in this report, the second order schemes are easier to vectorize and require much less extra storage.

Pushing a single particle in known, analytic fields provides a good test for any particle pushing routine. In a diode simulation, however, the accuracy of the final particle position depends not only upon the time step, Δt , but also upon the accuracy to which the fields are known. PIC codes with self consistent fields must compute fields from particle and mesh information and in the diode code these are usually known only to second order in the mesh spacing. In many cases the degree of accuracy of the particle push can be limited by the inaccuracy in the field information. For these reasons it was felt that a new comparison between the second order and fourth order schemes was in order. A test case simulating a typical diode configuration was run; once employing a second order integration scheme and once employing a fourth order scheme. The convergence rates to steady state were essentially the same for both and

all important simulation quantities did not vary by more than 10% between the two runs.

On the basis of this test run and because of the desirability of vectorization, it was decided to replace the fourth order particle pushing scheme in the diode code with a second order one. Several integration schemes were considered as replacements, of which three appeared most promising. These were a self-starting predictor-corrector scheme,¹ a non-self-starting leap-frog method,⁷ and a specialized leap-frog scheme which takes into account the specific nature of the Lorentz force being integrated.⁸

Previous work done at Sandia⁶ indicated that the predictor-corrector and leap-frog schemes ran about equally fast, with the latter being perhaps 10% faster in some cases. However, the non-self-starting nature of the leap-frog scheme would have increased storage requirements by an additional 25% and would have required revision of all the surface emission and reflection sections of the code. This method was therefore not pursued further.

The specialized integration scheme was implemented and run on the diode test case mentioned previously. This scheme was originally written by Boris⁸ for a simulation code which calculated E field and B field information at alternating half time steps. Integration is performed by integrating the electric field components only through half a time step, performing a rotation to account for the magnetic field contribution, and then integrating the electric field components through the remaining half time step. (An implicit assumption of spatially slow-varying fields is made.) The half-time step shift in field evaluations presented a severe

difficulty in implementing the scheme in Sandia's diode code, which calculates E and B fields simultaneously. The problem was solved by basically ignoring the half time step difference between evaluation times. However, the results of the test case run yielded steady state numbers which varied by as much as 50% from those obtained with the original fourth order diode code. The same test case yielded differences of less than 10% when the predictor-corrector scheme was compared to the original code. Since the steady state convergence rates and run time of the specialized scheme were comparable to those obtained with the predictor-corrector code, it was decided to halt further investigation of this scheme and replace the fourth order integration scheme in the diode code with the second order predictor-corrector one.

III. Vectorized Particle Pusher

The new integration scheme and the field interpolation code have been vectorized. The remaining sections of the particle pusher are conditional in nature and cannot be vectorized.

Vectorization was performed primarily in anticipation of eventually transferring the code to a CRAY-1 computing system. A second consideration was a possible increase in the addressing efficiency of variables in memory. This improvement stems from the fact that vectorization usually produces longer code sequences (a code sequence is a section of code containing no branching instructions or statement labels) which allow an optimizing compiler to more efficiently overlap computational and memory access instructions. However, this type of improvement is highly machine and code

dependent and the amount of reduction in run time achieved through increased code sequence length in this code has not been investigated in detail.

The cost of vectorization has been an increase in storage requirements of about 16,000 words or about 5% of the original storage used. This was needed to store intermediate results from one loop to the next. The occurrence of several loops was due to the fact that not all parts of a program loop can be vectorized (e.g., conditional statements and subroutine calls), which implies that loops must be broken up into vectorizable and non-vectorizable parts.

The problem of additional storage was greatly alleviated by revising temporary storage arrays as much as possible. This was best achieved by structuring the particle pushing code such that only 1000 particles would be processed at a time. An additional outer loop was then used to break up the total number of simulation particles into groups of 1000.

In summary, at the cost of about a 5% increase in storage requirements, a vector version of the particle pushing code has been created which can be expected to run between 5 and 10 times as fast on a CRAY-1 as the original scalar version would and which has a program structure which is more easily optimizable by the optimizing compilers available on most large computers today. An additional improvement was obtained from the process of vectorization due to the fact that such a major revision of the program eliminated or revised several sections of code which had become obsolete or inefficient during the evolution of the diode code to its present state.

IV. Program Structure

The subroutine structure and decision-making structure of the diode code's particle pusher has been almost completely revised in order to streamline the algorithm and also facilitate vectorization. Two changes yielded the greatest improvement in efficiency: minimization of the number of subroutine calls made and a revised method of checking for particle-surface encounters. Minimizing the number of subroutine calls is also necessary for purposes of vectorization.

The original code called several subroutines for each particle during each time step. By rewriting these subroutines so that they performed their function on all simulation particles in the system at once rather than just on one particle per call, the overhead of performing numerous subroutine calls with their associated parameter setup and transmissions was almost completely eliminated.

Revision of the particle-surface encounter check yielded similar benefits. By creating a two-dimensional array which contains all surface information (encoded in integer form) pertinent to each mesh zone of the simulation region, it was possible to reduce the check down to a single memory access and comparison instruction. This one-statement check was then placed outside the surface encounter subroutine so that the subroutine was called only if a surface had actually been encountered. (Less than 1% of all particles in the system encounter a surface during any given time step in most simulation runs.) The cost for this revision was an additional storage requirement of about 5%.

Several other minor changes were also made to the code. In order to avoid conditional statements (non-vectorizable and slow) the integration, field interpolation, and grid location routines have been duplicated so that ions and electrons can be handled separately. This also has certain other advantages which will be pointed out later on. In a similar manner, handling of illegal particle conditions (such as a particle moving more than one Δx or Δz in one Δt) has been delayed and made redundant in order to enable vectorization of the main particle pushing loops. This was done by removing "illegal" particles from the system only at the end of each time step. Since illegal conditions occur less than 0.1% of the time, the added overhead of unnecessarily keeping a particle in the system is negligible. Finally, the use of packed arrays to store the much-accessed 2-m grid coordinates of each particle has been discontinued at the expense of an additional 5% in storage requirements.

V. Specialized Pushers

Currently, of the six E and B field components which can be treated with the 2-1/2-D diode code, three are used in less than 10% of all simulation runs. Similarly, the $\vec{v}$ component of velocity for ions is very rarely needed. In order to avoid unnecessary computation and reduce storage requirements, the new version of the code has been implemented with specialized pushers which include only those fields which are being used in the current simulation case. Since ions and electrons have separate pushing routines, this was carried even further by deleting all v_z terms from the ion pushing routine.

The cost of this specialization is fairly small since less than 20

lines of actual Fortran code are affected by it. These lines have been marked and update packages written to provide several common configurations of included field and velocity terms.

VI. Unsafe Optimization

CDC computers provide an Unsafe Optimization (UO) compiler option which prefetches array contents from memory for the next iteration of a DO loop before the current loop termination check is performed. The benefit of this is a more efficient overlapping of computation and memory accessing. However, on the last iteration of a loop this can result in an attempt to retrieve a non-existent array location. If the array is not at the end of the program's allotted memory field length, then the machine will merely retrieve a "garbage" value from the subsequent variable(s) in memory. However, if the requested memory address is beyond the program's memory field length (or the array index used is less than 1) then an execution-time error will occur. Fortunately, this problem can almost always be circumvented by studying the program's load map and appropriately rearranging and enlarging the affected arrays. This has been done for the particle pushing routines (excluding the surface encounter handling routine) in the diode code and has yielded an improvement of about 2-3% in run time.

VII. Instack Loops

Another feature provided on CDC machines to achieve quicker execution times is an instruction queue. This queue is used to retain several

instructions so that short loops can be executed without continually having to re-fetch the loop instructions. Significant reductions in the execution time of the loop are achieved by this "instack" capability. In an attempt to increase the utilization of this feature, a version of the particle pusher code was written which broke up large loops into several smaller ones, all of which could fit into the instruction queue of a CDC-7600. However, the added overhead of additional loop control instructions together with a reduction in compiler optimization due to reduced code sequence lengths more than offset any gains achieved by converting all loops to instack lengths. Several different loop combinations were tried and all yielded essentially the same results, indicating that large loops with long code sequences are probably the most preferable format. (It should be noted, however, that artificially inflating the length of a loop in many cases can inflict other inefficiencies which more than offset the gains accrued from the additional code sequence length.)

VIII. Test Cases

The integration and field interpolation routines used in the new version of the code have been tested by pushing a single particle through three different analytic field configurations: a constant, E field; a constant B field; and crossed, constant E and B fields, all at relativistic velocities. The accuracies obtained were similar to those obtained in a previous study done at Sandia⁶ on particle pushing schemes. As was already expected, the only troublesome area in terms of accuracy occurred in cases involving large β velocity values at points near the axis

of symmetry (i.e., small r position coordinate values). The problem arises from the centripetal (v_z^2/r) and Coriolis ($v_r v_z/r$) accelerations.

The entire new version of the simulation code has been tested on a flat-plate diode case which can be described by Child's law and the diode test case mentioned earlier. The Child's law case consisted of a low voltage, low current, one-dimensional monopolar diode. Electrons were the only species considered and the current density is given analytically by

$$j = \frac{4}{9} \sqrt{\frac{2e}{m}} \epsilon_0 \frac{V^{3/2}}{d^2} \quad (1)$$

where e is the electron charge, m is the electron mass, d is the anode-to-cathode separation distance, ϵ_0 is the free space permittivity, and V is the applied voltage. In the test case, $V = 1$ volt and $d = 1$ mm, which results in a current density, j , of 2.33 amps/m^2 . The test simulation calculated an average j of 2.31 amps/m^2 at the anode with r and z variations of less than a few percent. In the simulation there were 42 r and z uniform zones and Δt was 3.0×10^{-11} seconds. The comparison with the analytic result was quite good and could be expected to improve with more zones or a smaller time step, Δt .

The other diode test case, consisting of a hollow cathode opposite a planar anode, was run with three progressively smaller time steps and correspondingly larger number of simulation particles. The same test cases were also run with the original diode code as a comparison. In all cases the steady state convergence rates were essentially the same and all important simulation quantities varied by about 10% or less. Table I

lists the time steps, steady state particle numbers, average currents, and execution times of each of these test cases. As can be seen, the new version of the diode code runs between two and three times as fast as the original version does. At even larger particle numbers this factor may be even bigger.

IX. Conclusions

Revision of the particle pusher in Sandia's diode code has reduced the program's execution time for a typical simulation case by a factor of two to three from that of the original version while still retaining comparable accuracy. The cost of this optimization has been an increase in storage requirements of about 15%. The new version has also been structured to run efficiently on a CRAY-1 computing system in anticipation of a possible additional factor of five to ten decrease in execution time over that attainable on non-vector machines such as a CDC-7600.

The primary aspect which dominated almost the entire revision process was vectorization. Although selection of a new integration scheme, a new particle-surface encounter check, and specialization of the pushing algorithms probably yielded more savings in run time; the dominant feature in the new version of the particle pusher is its vector nature. This can be attributed to the fact that vectorization imposes rather stringent requirements on the structure of a program and also because the possible future gains obtainable (on the CRAY-1) make its usage extremely desirable. It is also the most generally applicable form of optimization since it is

independent of the exact nature of the program. (In essence, vectorization transforms a program in which individual lines of code manipulate individual items of a system into a program in which blocks of code manipulate the entire system--be it a system of 10,000 simulation particles or of six L and B field components.) The other modifications performed were specific to individual aspects of the diode code and could not affect the entire program.

Of these modifications, replacement of the fourth order integration scheme with a simpler second order one probably yielded the greatest savings in run time for a typical diode simulation. However, the new method of checking for a particle-surface encounter may have been the more important innovation in the long run, since an expansion of the code's capabilities to handle slanted or even general surfaces is envisioned. The current code can in general handle only vertical or horizontal surfaces situated along grid lines of the mesh. Specific slanted surfaces can be treated if they pass through mesh points, but they are cumbersome to include in the present code. Flat surfaces are fairly easy to handle compared to a general surface (not necessarily passing through grid points) which might be encountered by a particle at any point in the simulation region. Since the new check works on a zone basis rather than a grid coordinate basis, the fact that a surface will no longer be situated on a grid line is no longer a real difficulty and can be handled almost as efficiently. (The additional checking that is needed to determine if a particle actually has crossed a surface in its zone is only needed in the 10% or less of all cases in which a particle enters a zone which contains a surface.)

TABLE I

Run statistics for hollow cathode diode test case.
Total simulation time was 0.3 ns.

Version	Time Step (10^{-13} sec)	Steady State Particle Number	Av. Electron Current (10^6 Amp)	Av. Ion Current (10^6 Amp)	Execution Time (sec)	Ratio of Old-to-New Version Execution Times
Old New	3.00 3.00	4540 4755	.10836 .10885	.011713 .012080	2450 1142	2.15
Old New	1.50 1.50	9280 9160	.10920 .11030	.013509 .013531	8170 3284	2.49
Old New	0.75 0.75	18470 18920	.10712 .10883	.013960 .014330	35966 11691	3.08

APPENDIX A: Timing Estimates

This appendix has been written in order to provide a rough estimate of the various savings gained from each of the individual optimizing revisions made to the diode code. No actual study has been made to determine how accurate these estimates are since the various items are closely interrelated and a careful analysis would be very time consuming. Of the factor two to three improvement in run time, roughly 40% can be attributed to the change in integration scheme, 25% to vectorization and associated revisions, 13% to the new particle-surface encounter check, 20% to specialization of the particle pushing algorithms, and about 2% to use of the unsafe optimization feature of CDC computers.

References

1. J. W. Poukey, J. R. Freeman, and G. Yonas, *J. Vac. Sci. Technol.* 10, 954 (1973).
2. J. W. Poukey, *J. Vac. Sci. Technol.* 12, 1214 (1975).
3. J. P. Quintenz and J. W. Poukey, *J. Appl. Phys.* 48, 2287 (1977).
4. J. P. Quintenz, *J. Appl. Phys.* 49, 4377 (1978).
5. J. P. Quintenz, M. M. Widner, and J. A. Halbleib, Sr., to be published.
6. J. P. Quintenz, "Particle Pushing Schemes for 2-D Codes," Sandia Internal Memorandum US 5241/092, February 1976.
7. A. Ralston and H. S. Wilf, "Mathematical Methods for Digital Computers I," John Wiley and Sons, 98 (1960).
8. Proceedings of the Fourth Conference on Numerical Simulation of Plasmas, ed. by J. P. Boris and R. A. Shanny, p. 21, U. S. Govt. Printing Office, Washington, DC 20402, Stock #0851 00059 (1970).