JOURNAL OF COMPUTATIONAL PHYSICS 89, 462–482 (1990)

A Vector-Efficient and Memory-Saving Interpolation Algorithm for PIC Codes on a Cray X-MP

GIUSEPPE PARUOLO

CINECA, Inter-University Computing Center,
Via Magnanelli 6/3, 40033 Casalecchio di Reno, Bologna, Italy

Received April 18, 1989; revised August 3, 1989

The performance of particle-in-cell (PIC) programs executing on Cray X-MP supercomputers is highly dependent on a critical region of the code: the collection of grid quantities from particle quantities, which cannot directly be vectorized. To this purpose methods allowing for either partial or total vectorization have been proposed; these methods either require very large additional storage or are limited to processing short vectors. After the illustration in three dimensions of the different methods devised and used up to now, a new scheme for the vectorization of the interpolation on the grid is proposed. This method requires a small amount of extra storage, is completely vectorized, and makes it possible to take full advantage of the intrinsic vectorizability of the physical system. A comparison using a tridimensional example, representing a vast range of simulations, shows a relevant speedup for the scheme proposed over the other methods in existence up to now, up to a factor of 3 when compared to the original scalar code.

INTRODUCTION

Particle-in-cell (PIC) codes are used for the computer simulation of physical phenomena in many fields of science, from astrophysics to fluid dynamics, to the physics of plasma, chemistry, biology, electronics, and others [5]. There is a rapid increase in the use of these programs, as well as in the dimensions of the problems considered and the duration of the simulations requested.

These codes require a considerable amount of resources, both in terms of computational power and memory size, and this explains why they are particularly suited to execution on a supercomputer. In order for them to fully exploit the capabilities of the hardware, optimization must be dealt with. In the case of the Cray X-MP this basically means: (1) vectorizing the code (possibly without increasing by much the amount of memory needed); (2) making it processable on several processors in a parallel manner.

In a typical PIC code there is a basic cycle which is repeated at every time step of the simulation. This cycle may be divided into four steps. For example, if we use the terminology for a gravitational simulation in astrophysics, in which the particles represent the stars in a galaxy or galaxies in a part of the universe, then the four steps in the basic cycle may be described as follows:

0021-9991/90 $3.00

1. Solution of the field equations and calculation of the forces on the grid points.

2. Interpolation to calculate the forces acting at the particle positions.

3. Solution of the motion equation to update the positions and the velocities of the particles.

4. Interpolation to calculate the mass density in the grid points starting from the new positions of the particles.

In other types of simulation the equations for movement and field have to be substituted with the laws relative to the physical phenomenon being considered. For example, in the simulation of an electromagnetic field, the first step will solve the Maxwell equations and the fourth will calculate the charge and current densities at the grid points.

On a Cray X-MP, steps 1, 2, 3 are vectorized (the vectorization of step 2 is made possible by the presence of hardware vector instructions managing gather/scatter operations—see Horowitz [2]). Instead, vectorization of step 4 is anything but easy. If step 4 is carried out in a scalar mode, it typically absorbs almost 50% of the total time of execution, that is, in terms of CPU time, it has the weight of the other three steps put together [6]. For this reason, it is important to define an algorithm which allows vectorization, at the same time attempting not to pay too high a price in terms of the additional storage requirement.

Solutions to this problem have up to now been proposed by Nishiguchi, Orii, and Yabe [1], Horowitz [2], and Schwarzmeier and Hewitt [3]. Not all of these solutions were presented on computers of the Cray X-MP series, but they are at any rate suited to implementation on them.

It is the purpose of this paper to propose a new method and to compare it with methods already in existence. The basis for a comparison is provided by an astrophysical tridimensional simulation, where step 4 calculates the mass density at the grid points, starting from the mass and the positions of the particles [4].

Calculation of Density at Grid Points

To assign the density to the mesh, piece-wise polynomials of different order can be used. With linear splines, the mass of each particle is divided between the eight grid points which are vertices of the cell in which the particle is contained. The weight pertinent to one grid point is the fraction of volume of the cell included between the projections of the particle on the sides of the cell and the opposite vertex. For example, in the bidimensional case illustrated in Fig. 1, the contribution of the particle to the mass density at the grid point (I, J) is proportional to the area $A1$. In the tridimensional case, in place of the four areas A1, ..., A4, we have eight volumes V1, ..., V8.

Direct vectorization of this process on all of the particles is not possible. In fact, different particles can contribute to the mass density of a same grid point, and if

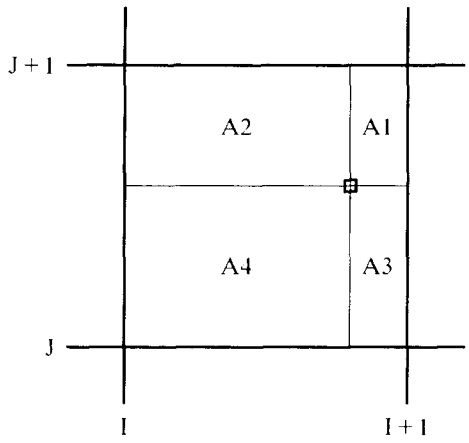

FIG. 1. Weights regulating the portion of the value of a physical quantity in the position of the particle to be deposed on the surrounding grid points, in a bidimensional case: (I, J) receive a contribution proportional to the area A1, (I + 1, J) to A2, (I, J + 1) to A3 (I + 1, J + 1) to A4.

this cycle were to be vectorized, in some cases some of these contributions, overwritten by others, would be lost. This consideration can be split into two different reasons which inhibit the vectorization:

- there may be several particles in a cell;
- a grid point may belong to eight different cells (and this is, in fact, so, unless we are dealing with the boundary of the grid).

To avoid the first of the two problems, we might decide to vectorize the calculations relative to groups of particles contained in distinct cells. Clearly, however, the distribution of the particles on the grid influences the performance of this approach, which will reach the maximum speed for the particles distributed in a homogeneous manner. When we speak of "intrinsic vectorizability" of the physical system, we are referring precisely to the degree of uniformity in the particle distribution.

THE SCALAR CODE

Let us consider a tridimensional grid made up of $NC1 \times NC2 \times NC3 = NCELL$ cells, that is by $(NC1 + 1) \times (NC2 + 1) \times (NC3 + 1) = NGRID$ grid points. Let NPART be the number of particles, and X, Y, Z the vectors containing the particle positions relative to the three coordinates, in terms of grid cells: we assume that $0 \leqslant X(N) < NC1$, $0 \leqslant Y(N) < NC2$, and $0 \leqslant Z(N) < NC3$ for all the particles N. Let D be the tridimensional matrix which contains the mass density at the grid points. The scalar code for the deposition of the particle mass quantities (Q) on the grid points is thus as follows:

```
      PARAMETER (NG1=NC1+1,NG2=NC2+1,NG3=NC3+1)
      PARAMETER (NCELL=NC1*NC2*NC3,NGRID=NG1*NG2*NG3)
      REAL      X(NPART),Y(NPART),Z(NPART)
      REAL      D(NG1,NG2,NG3)

      ... setting of D to zero ...
         DO 10 N=1,NPART                          ! scalar
            I = INT(X(N))+1
            J = INT(Y(N))+1
            K = INT(Z(N))+1
            ............
            calculation of volumes/weights
            ............
            D(I,J,K)       = D(I,J,K)       + V1*Q/V
            D(I+1,J,K)     = D(I+1,J,K)     + V2*Q/V
            D(I,J+1,K)     = D(I,J+1,K)     + V3*Q/V
            D(I+1,J+1,K)   = D(I+1,J+1,K)   + V4*Q/V
            D(I,J,K+1)     = D(I,J,K+1)     + V5*Q/V
            D(I+1,J,K+1)   = D(I+1,J,K+1)   + V6*Q/V
            D(I,J+1,K+1)   = D(I,J+1,K+1)   + V7*Q/V
            D(I+1,J+1,K+1) = D(I+1,J+1,K+1) + V8*Q/V
      10 CONTINUE
```

During the presentation of vectorizing methods only dimensioning relative to the work arrays will be indicated, while X, Y, Z, and D will be implied.

The Nishiguchi, Orii, and Yabe Scheme [1]

The basic idea is to process the particles in batches of length LVEC, and let their contributions be distributed on different elements of a temporary array, thus excluding illicit overwriting and permitting vectorization. After processing all of the batches of LVEC particles, the temporary array is compressed in D.

The code, applied to the case considered, is as follows:

```
      PARAMETER (LVEC=...)
      REAL      DT(NG1,NG2,NG3,LVEC)

      ... setting of D to zero ...
      ... setting of DT to zero ...

      DO 20 NN=1,NPART,LVEC
CDIR$ IVDEP
         DO 20 N=NN,MIN(NN+LVEC-1,NPART)    ! vector
            L = N-NN+1
            ............
            calculations relative to particle N
            ............
            DT(I,J,K,L)       = DT(I,J,K,L)       + V1*Q/V
            ....
            DT(I+1,J+1,K+1,L) = DT(I+1,J+1,K+1,L) + V8*Q/V
   20 CONTINUE

      DO 30 L=1,LVEC
         DO 30 I=1,NGRID                        ! vector
            D(I,1,1) = D(I,1,1)+DT(I,1,1,L)
   30 CONTINUE
```

The inner loops 20 and 30 are vectorized on lengths equal to LVEC and NGRID, respectively. In order to have access to the best performance, LVEC should be a multiple of 64 (which is the length of the Cray vector registers). On the other hand, the amount of additional storage required is equal to NGRID * LVEC memory words, so LVEC should be given the smallest value possible. The choice of LVEC = 64 is the least penalizing in terms of memory needed, among those that best make use of vectorization, but, at any rate, the amount of storage required is definitely large.

This method obtains good vectorization and the only overhead introduced is the final compression of the temporary array DT in D. The storage requirement and the time needed to perform the final compression do not depend on the number of particles, and thus, once the grid has been fixed, performance increases as do the number of particles. Furthermore, this scheme is insensitive to the distribution of the particles on the grid.

The main problem is the size of extra storage: if we wish to decrease it, by lowering LVEC to values less than 64, the performance of the code is also lowered. It may also be observed that when the calculation of more than one quantity is required on the grid, the work memory requirement increases proportionally.

Therefore, the Nishiguchi, Orii, and Yabe scheme (from now on referred to as NOY) appears to be impracticable for use in large problems and instead suited to situations in which the size of the grid is small, only one quantity has to be calculated at grid points, and the particles to be processed are many more than the cells.

THE HOROWITZ SCHEME [2]

The problem of the presence of several particles within the same cell is solved with a preventive subdivision into groups of particles pertaining to distinct cells. The other inhibitor to vectorization (every grid point is vertex to eight different cells) is solved by introducing eight temporary copies of the D array, one for each vertex.

In these conditions, it is possible to vectorize the calculations for the particles of a same group. The code is as follows:

```
PARAMETER (IGNMAX=...)
INTEGER   NPIC(NCELL),NIG(IGNMAX)
INTEGER   IP(NCELL,IGNMAX),ISP(NPART)
REAL      D1(NG1,NG2,NG3),D2(NG1,NG2,NG3)
REAL      D3(NG1,NG2,NG3),D4(NG1,NG2,NG3)
REAL      D5(NG1,NG2,NG3),D6(NG1,NG2,NG3)
REAL      D7(NG1,NG2,NG3),D8(NG1,NG2,NG3)

IC(N) = INT(Z(N))*NC2*NC1+INT(Y(N))*NC1+INT(X(N))+1

... setting of NPIC to zero ...
... setting of NIG to zero ...
... setting of D1,D2,D3,D4,D5,D6,D7,D8 to zero ...
NSP = 0
```

```
      DO 40 N=1,NPART                          ! scalar
         IF (NPIC(IC(N)).LT.IGNMAX) THEN
            NPIC(IC(N))      = NPIC(IC(N))+1
            IGN              = NPIC(IC(N))
            NIG(IGN)         = NIG(IGN)+1
            IP(NIG(IGN),IGN) = N
         ELSE
            NSP      = NSP+1
            ISP(NSP) = N
         ENDIF
   40 CONTINUE

      DO 50 IGN=1,IGNMAX
CDIR$ IVDEP
         DO 50 M=1,NIG(IGN)                    ! vector
            N = IP(M,IGN)
            .............
            calculations relative to particle N
            .............
            D1(I,J,K)       = D1(I,J,K)       + V1*Q/V
            ....
            D8(I+1,J+1,K+1) = D8(I+1,J+1,K+1) + V8*Q/V
   50 CONTINUE
      DO 60 I=1,NGRID                          ! vector
         D(I,1,1) = D1(I,1,1)+D2(I,1,1)+D3(I,1,1)+D4(I,1,1)
     1              D5(I,1,1)+D6(I,1,1)+D7(I,1,1)+D8(I,1,1)
   60 CONTINUE

      DO 70 M=1,NSP                            ! scalar
         N = ISP(M)
         .............
         calculations relative to particle N
         .............
         D(I,J,K)       = D(I,J,K)       + V1*Q/V
         ....
         D(I+1,J+1,K+1) = D(I+1,J+1,K+1) + V8*Q/V
   70 CONTINUE
```

The collection of particles in the different groups is carried out in the scalar loop 40; the work arrays required for this phase are NPIC, which contains the number of particles present in each cell, NIG, which contains the number of particles contained in each group, the matrix IP, of which each column contains the list of particles in a group, and finally the vector ISP, containing the particles to be processed in scalar mode.

The parameter IGNMAX represents the maximum number of groups; within each group there may be at most NCELL particles. After the vectorial process of the first IGNMAX particles of each cell, those which may remain are processed in the scalar loop 70. The work area required to manage this phase is thus equal to IGNMAX * NCELL + NCELL + IGNMAX + NPART words.

Loop 50 processes the particles within each group and loop 60 compresses the temporary arrays in D, both in vector mode. Additional 8 * NGRID words of work memory are needed for this phase.

This scheme vectorizes most of the calculations, but it executes in scalar mode

the initial grouping of particles, as well as the final processing of the remainders; a further overhead is represented by the compression of temporaries into D. The performance is influenced by the degree of uniformity in the distribution of the particles.

The amount of work memory needed, although less than that of the NOY scheme, is in any case relevant and dependent on many factors: the number of cells, the number of particles, the dimension of the space in which simulation takes place, the number of the quantities to be calculated on the grid, and, last but not least, the value chosen for IGNMAX.

Therefore, the method proposed by Eric Horowitz (hereafter EH) represents a step forward as compared to the NOY scheme because of the smaller amount of additional memory required, but it, at any rate, remains elevated, and vectorization is incomplete; furthermore, the dependence of the performance and of the storage requirement on many factors does not favor adaptability to different situations.

THE SCHWARZMEIER AND HEWITT SCHEME [3]

Considering loop 10, relative to the scalar code, it is observed that the obstacles to vectorization are in the updating of array D, and not in the calculation of the contributions of the Nth particle to the vertices of its cell. Furthermore, once a particle has been fixed, the eight elements of D represent the vertices of a single cell, thus they are distinct: thus, the updating of these eight values can be vectorized.

These considerations, which are independent of the pecularities of the physical problem, are basic to the method presented by Schwarzmeier and Hewitt (which will be referred to as SH from now on). They developed it for a bidimensional code which simulates an electromagnetic field, in which it is necessary to calculate the current and charge density on the grid, represented by four bidimensional arrays: thus, the vectorization of the update takes place on a loop equal to 16 in length.

The SH method provides the following code for the example considered:

```
      PARAMETER   (NBLK=...)
      INTEGER     LL(9,NBLK)
      REAL        WW(9,NBLK),A(NGRID)
      EQUIVALENCE (A,D)
      ILOC(I,J,K) = I+NG1*(J-1)+NG1*NG2*(K-1)
      ... setting of D to zero ...
      DO 90 NN=1,NPART,NBLK
         DO 80 N=NN,MIN(NN+NBLK-1,NPART)     ! vector
            L = N-NN+1
            ............
            calculations relative to particle N
            ............
            WW(1,L) =  V1*Q/V
            LL(1,L) =  ILOC(I,J,K)
            ....
            WW(8,L) =  V8*Q/V
            LL(8,L) =  ILOC(I+1,J+1,K+1)
80       CONTINUE
```

```
        DO 90 M=1,MIN(NBLK,NPART-NN+1)
CDIR$ IVDEP
            DO 90 L=1,8                          ! short vector
               A(LL(L,M)) = A(LL(L,M))+WW(L,M)
  90 CONTINUE
```

The particles are processed in batches of length NBLK; the contributions relative to the vertices of the cells containing the NBLK particles considered are calculated in loop 80; then, for each particle, in loop 90 these contributions are added to the appropriate elements of D, using vectors which are 8 long.

The monodimensional vector A, which shares the storage locations of D, allows for vectorization on a single index of loop 90. The work array LL is used to contain the positions of the eight grid points surrounding the particle (by means of the ILOC function which maps the tridimensional coordinates of grid points to one dimension), while WW contains the contribution to the vertices. The first dimension of LL and WW is given by 9 (instead of 8, which would be sufficient) in order to avoid memory bank conflicts, as loop 80 vectorizes on the second index. As loop 80 processes vectors equal to NBLK in length, it is useful to let NBLK be a multiple of 64.

A minimum amount of work memory is needed: exactly 18 * NBLK words. Both the work area and the performance are independent of factors such as the size of the grid, the number of particles, and their distribution. The code is completely vectorized and, as it processes the particles one at a time, it provides results which are equal up to last bit to those of scalar execution. The only overhead introduced is in terms of transfers of LL and WW elements between vectors registers and memory: this is the price that must be paid in order to split the deposition loop in two parts.

The factors on which both the storage requirement and, in particular, the performance depend are the dimension of the space in which the simulation takes place and the number of quantities to be calculated at grid points. In fact, the length of the vectors that are processed in loop 90 depend on these values. Nonetheless, a real problem rarely makes this length equal to 64, using the full length of the vector registers.

Because of its characteristics, the SH method is very efficient when used in tridimensional problems with many quantities to be calculated on the grid. The minimum request for work memory also makes its use interesting in cases in which vector loop 90 is short. For example, when the physical problem is intrinsically not vectorizable, that is, the distribution of particles on the grid is highly non-uniform, or when other methods are not efficient, the SH scheme is nonetheless preferable to scalar execution.

A New Method

The new method we introduce requires two work vectors: IAP of the size NPART + 1, to represent the particles, and IAG of the size NCELL, to represent

the grid cells. At the beginning, conventional values are placed in the vector IAP, to indicate that all of the particles are to be processed. The method can be summarized as follows:

1. The purpose of the first step is to collect a particle for each cell, if it contains any.

1.1. IAG vector is cleaned, placing a conventional value in all its elements.

1.2. The index of each particle still to be processed is placed in IAG, in the element corresponding to the cell that contains it; this loop is executed vectorially, without dealing with overwriting generated by particles contained in the same cell, because what is important is that at the end of the loop for each cell there be a particle (or none).

1.3. The particles saved in IAG will be processed in the second step and are thus to be eliminated from successive iterations; this occurs with a vectorized cycle that places a conventional value in the IAP position relative to each particle present in IAG.

How this step operates is shown in Fig. 2, in a very simplified example.

2. In the second step the masses of particles selected during the previous one are accumulated on the grid points. Each particle is in a different cell, thus, the only inhibitor to vectorization is the fact that each grid point is vertex in eight different cells. Instead of defining some temporary arrays, a particular numerical order of the cells is used, so that neighbouring cells are as far away as possible in the order in which they are considered in the loop.

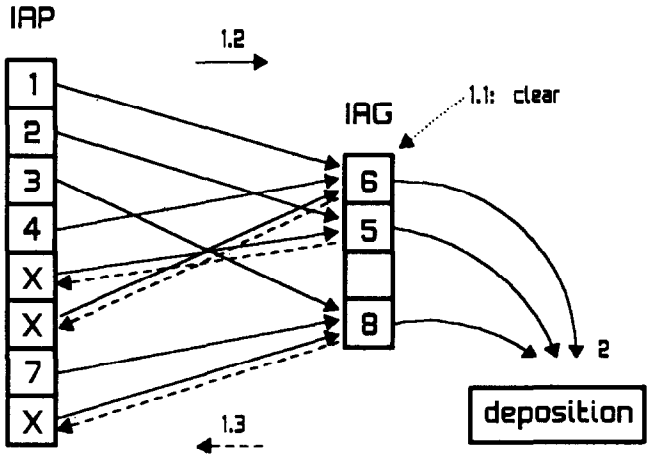

FIG. 2. The collection of a group of particles pertaining to different cells, as carried out in step 1 of the proposed method, in a very simplified example. After the cleaning of the IAG vector (1.1), particles are put in the corresponding cells (1.2), then particles chosen are eliminated from successive iteration (1.3) and are processed in the deposition loop (2). In this example the first iteration process particles 5, 6, 8; the second 2, 4, 7; and the last 1, 3.

In fact, the mapping of the tridimensional grid on a monodimensional vector, carried out by the ICELL function, makes it so that neighbouring cells are at least

$$\text{MINDIST} = \frac{\text{NC1}}{2}\frac{\text{NC2}}{2}\left(\frac{\text{NC3}}{2} - 1\right)$$

elements away (we assume that NC1, NC2, and NC3 are even numbers, as usually happens).

If MINDIST $\geqslant 64$ (the length of vector registers in the Cray X-MP) then the deposition loop is vectorizable without requiring any temporary vector. MINDIST $= 64$ corresponds, for example, to a $8 \times 8 \times 10$ grid: for a dimension less than this, vectorization is certainly not a critical factor, and if it were, the NOY method would be suitable.

Thus, in the hypothesis of MINDIST $\geqslant 64$ the loop is vectorized on a length which is always equal to NCELL (the data are not compressed; this is essential to vectorize). This vectorization of the deposition loop makes extensive use of the hardware feature "compressed index" and is efficient thanks to it.

3. In the third step we decide whether to continue to process the particles in the manner illustrated above or not. The number of particles collected in IAG decreases at each iteration, or at most it remains constant, thus the number of elements just processed constitutes an upper limit for the successive iteration. This number and that of the elements which remain to be processed are compared with a certain value IPMIN. If one of these two values falls under the IPMIN level then the remaining elements are processed with step 4; otherwise we go back to 1.

4. The processing of remaining particles, if there are any, is carried out in this step.

4.1. Particles remained to be processed are compressed vectorially.

4.2. An obvious possibility is to finish with a scalar loop. An interesting alternative is instead the use of the SH scheme.

Further on, we will see the possible implications of this choice; an immediate consequence is, at any rate, that when particle distribution is highly non-uniform, and so the number of particles to be processed in this step is relevant, the performance flattens on that of the SH method and not on that of the scalar code. If, instead, the distribution is quite homogeneous, the remainders to be processed are very few as compared to the total number of particles, and thus the difference is truly minimal.

The Program

The Fortran code which carries out the method illustrated, in the hypothesis of a tridimensional grid with an even number of cells in each dimension and such that

$$\frac{\text{NC1}}{2}\frac{\text{NC2}}{2}\left(\frac{\text{NC3}}{2} - 1\right) \geqslant 64$$

is as follows:

```
      PARAMETER (IPMIN=...)
      PARAMETER (ICOEF=(NC1/2)*(NC2/2)*(NC3/2))
      PARAMETER (ICOX2=1-2*ICOEF)
      PARAMETER (ICOY2=(NC1/2)-4*ICOEF)
      PARAMETER (ICOZ2=(NC1/2)*(NC2/2)-8*ICOEF)
      INTEGER   IAP(NPART+1),IAG(NCELL)

      ICELL(N) = ICOEF*(INT(X(N))+INT(Y(N))*2+INT(Z(N))*4)
     1           +INT(Z(N))/2*ICOZ2+INT(Y(N))/2*ICOY2
     2           +INT(X(N))/2*ICOX2+1

      ... setting of D to zero ...

      DO 100 N=1,NPART                          ! vector
         IAP(N) = ICELL(N)
  100 CONTINUE

      ICTDN = NPART

  200 CONTINUE

      DO 300 I=1,NCELL                          ! vector
         IAG(I) = NPART+1
  300 CONTINUE

      DO 400 N=1,NPART                          ! vector
         IF (IAP(N).NE.0) IAG(IAP(N)) = N
  400 CONTINUE

      DO 500 I=1,NCELL                          ! vector
         IAP(IAG(I)) = 0
  500 CONTINUE

      ICN = 0
CDIR$ IVDEP
      DO 600 II=1,NCELL                         ! vector
         IF (IAG(II).LE.NPART) THEN
            ICN = ICN+1
            N   = IAG(II)
            ............
            calculations relative to particle N
            ............
            D(I,J,K)       = D(I,J,K)       + V1*Q/V
            ....
            D(I+1,J+1,K+1) = D(I+1,J+1,K+1) + V8*Q/V
         ENDIF
  600 CONTINUE

      ICTDN = ICTDN-ICN
      IF (ICTDN.EQ.0) RETURN
      IF (ICN.GT.IPMIN.AND.ICTDN.GT.IPMIN) GOTO 200

      NSCA = 0
CDIR$ IVDEP
      DO 700 N=1,NPART                          ! vector
         IF (IAP(N).NE.0) THEN
            NSCA      = NSCA+1
            IAP(NSCA) = N
         ENDIF
  700 CONTINUE

      ..... processing of the particles contained
            in IAP  (in positions from 1 to NSCA)
            in scalar mode or  with the SH scheme .....
```

• The ICELL statement function carries out the 1-to-1 mapping of the indexes of the cells from three to one dimension, in the way required to be able to vectorize the deposition loop. If $x_n = |X(N)|$, $y_n = |Y(N)|$, $z_n = |Z(N)|$, $h_1 = NC1/2$, $h_2 = NC2/2$, $h_3 = NC3/2$, then ICELL provides the same values of the function

$$h_1 h_2 h_3 (x_n \bmod 2 + 2(y_n \bmod 2) + 4(z_n \bmod 2)) + \frac{z_n}{2} h_1 h_2 + \frac{y_n}{2} h_1 + \frac{x_n}{2} + 1.$$

This was, in fact, the original formulation of the ICELL function, which was then rewritten only in terms of the function INT and not of the functions MOD or AMOD for reasons related to vector efficiency of DO 100.

With this mapping, neighbouring cells are at least $h_1 h_2 (h_3 - 1)$ elements apart. To apply this method to a bidimensional situation, the ICELL function needed is deducible from the given formula setting $z_n = 0$ and $h_3 = 1$, giving

$$h_1 h_2 (x_n \bmod 2 + 2(y_n \bmod 2)) + \frac{y_n}{2} h_1 + \frac{x_n}{2} + 1.$$

In this case, neighbouring cells are at least $h_1(h_2 - 1)$ elements apart in the mapping. The reverse mapping, given the cell c, is as follows:

$$x_n = 2((c-1) \bmod h_1) + \left(\frac{c-1}{h_1 h_2}\right) \bmod 2$$

$$y_n = 2\left(\frac{(c-1) \bmod h_1 h_2}{h_1}\right) + \left(\frac{c-1}{2h_1 h_2}\right) \bmod 2.$$

In a similar way we can deal with other dimensions. We note explicitly that all the divisions reported in the preceding formulas are integer divisions.

• The assigning of particles to the cells is carried out in DO 400; for each particle it is thus necessary to know the cell containing it.

In order not to use the ICELL function to perform the calculation of the cell at each iteration, it is effective to calculate the cell containing each particle once at the beginning and to store it in a temporary array.

This is in fact done in DO 100, and to save storage the IAP vector is used. Thus, a value in IAP greater than 0 indicates that the particle is still to be processed, and it also represents the number of the cell that contains it.

• The DO 300 initializes the IAG elements to the value NPART + 1: this means that the cell does not contain a particle, and in fact this is tested in DO 600.

This value is also used in the DO 500, when the particles that are collected in IAG are excluded from successive calculations putting their corresponding position

in IAP to 0. In order to avoid testing whether each cell contains a particle or not, the IAP vector is dimensioned to NPART + 1, and for each cell that does not contain a particle, the last element is reset (which does not appear in any calculation).

- The DO 600 is the vectorized deposition loop.
- The variables ICNDN and ICN are counters that respectively represent the total number of particles that remain to be processed and the number of those processed in the last execution of DO 600. They are tested in order to decide whether to continue the processing of particles with this scheme, going back to label 200, or to process the remainders using a different method.
- The DO 700 carries out the compression of the particles not processed before. In fact, whether the remaining particles are processed in scalar mode or with the SH scheme, for reasons of efficiency it is preferable to have their indexes occupying adjacent positions. In order to save storage the same IAP vector is used to contain the compressed data; this makes it necessary to force the vectorization.
- If we choose to process the remaining particles using the scalar method then no extra storage is needed. If instead we use the SH method it is necessary to allocate vectors LL and WW. Their storage requirement is nevertheless reduced (in our case, if NBLK = 64, 1152 words would suffice); furthermore, the locations relative to the vector IAG which is no longer needed may be reused. In the example presented, if the grid is of dimensions $16 \times 16 \times 8$ or greater, then the vectors LL and WW do not require any additional storage.

Considerations

In the new method proposed the code is completely vectorized, and the particles are processed in groups; when the number of elements contained in the group to be processed falls below a certain level, then the remaining particles are processed in scalar mode or with the SH method.

The more uniform the distribution of the particles on the grid, the more efficient the code, but even when there is dishomogeneity the algorithm is flexible enough to make use of all of the possibilities for vectorization offered by the physical problem and then fall back on a method independent of these.

The only overhead introduced is the collection of particles (300, 400, and 500 loops) which involves all of the particles at each iteration, but the code is simple and vectorized, so the time required is not relevant.

The storage requirement is quite small: NCELL + NPART + 1 words. It is also independent of the number of quantities to be calculated at grid points. Furthermore, the work vectors have sizes of the order of the number of particles and of the cells in the grid: these are characteristic of the problem, similar vectors are usually required in other regions of the PIC code and thus, it is possible to share locations. Therefore, we may affirm that this algorithm, like the SH scheme, and unlike the

EH and NOY methods, requires no increase in the storage needed to execute the simulation.

Criteria Adopted in the Comparison

The algorithm proposed (hereafter GP) was compared with the scalar code and with the NOY, EH, and SH schemes; the physical problem used for the comparison is that described by A. Messina and P. Londrillo in [4].

Timing results were obtained using the Cray X-MP/48 at CINECA, equipped with the CIGS harware feature and operating with COS 1.16. The compiler used was CFT version 1.15 bugfix 3. All the tests were run solo on the machine, in a dedicated environment, to avoid conflicts with other jobs in the memory bank access.

The runs were carried out varying the three parameters: the dimension of the grid, the number of particles, and the distribution of the particles on the grid.

- We considered grids from $10 \times 10 \times 10$ up to $96 \times 96 \times 96$, that is, in order of size of the number of grid points varied from 10^3 up to 10^6.
- The number of particles, always at least equal to that of the grid points, was also varied on the order of 10^3 to 10^6.
- In order to obtain different kind on particle distribution on the grid, three methods were followed:
  - locating the particles in the center of each cell the maximum possible grade of homogeneity in the distribution is obtained;
  - using the pseudo-random number generator provided by the intrinsic function RANF( ), a nearly homogeneous distribution but with a considerable degree of perturbation (depending on the not excellent characteristics of RANF( ) as random number generator) is produced;
  - to obtain high dishomogeneity and, at the same time, to quantify it, direct filling of the cells is used, limiting it to a pre-established subset of the grid cells.

For lack of space it is not possible to report the results of all of the experiments; thus, only some of the results are reported, those which sufficiently illustrate the conclusions that were drawn from the entire experimentation procedure.

Experimental Results

The first results are presented in Fig. 3, showing the CPU time in milliseconds for the execution of various tests. In these tests grids have the same number of cells (which is indicated as grid order, and varies from 32 to 64) in each of the three dimensions; number of particles precisely equals the number of cells in the grid.

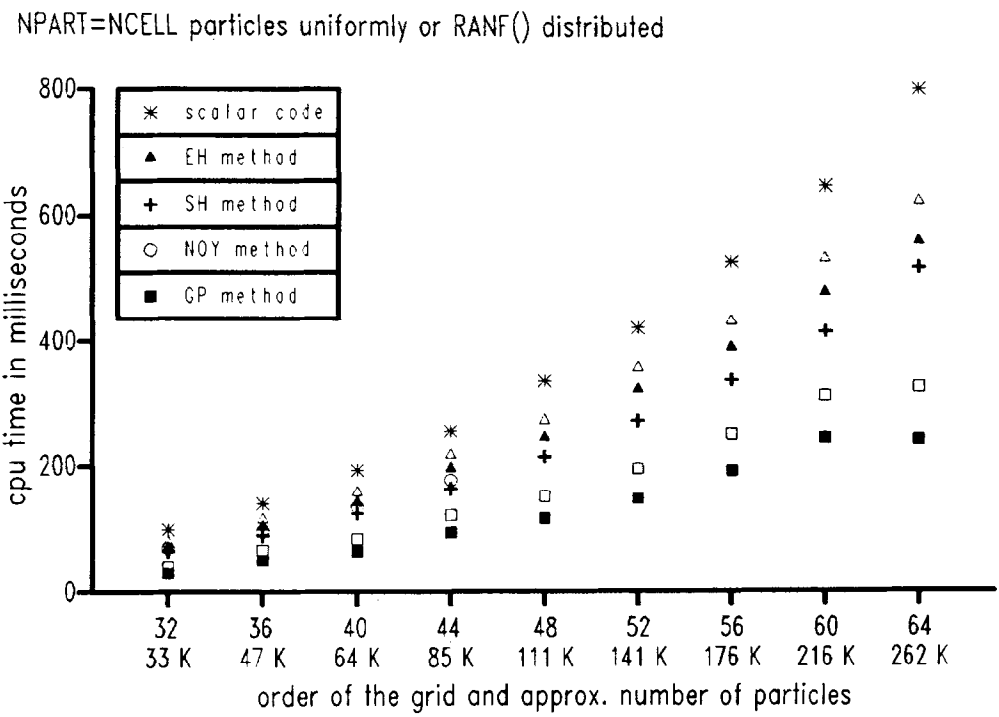

FIG. 3. Execution time with the various methods in the case of cubic grids, number of particles equal to the number of cells and with differing data distribution. The schemes NOY and SH, together with the scalar code, are not influenced by the distribution of the particles; the EH and GP methods have full symbols for totally uniformly distributed data, empty for data generated with RANF( ). IGNMAX = 2 was used in the EH method. NOY results for grids of order greater than 44 are missing, because their storage requirements exceed the memory size of Cray X-MP/48.

Two kinds of distributions are introduced: totally uniform and RANF( ) generated (with RANF( ) only 63% of the cells contains particles, and the maximum number of particles in the same cell varies from 7 to 9).

The performances of methods NOY, SH, and scalar code do not depend on the distribution of the particles, so the results for the totally uniform and for the RANF( ) distribution are the same. Methods EH and GP, instead, have different timings: the full symbols are relative to completely uniform distribution, the empty ones to RANF( ) generated data. The gap between these symbols provides an indication of the fluctuation of the performance of these methods with the varying of the particle distribution.

The results for the GP method refer both to the version that processes the remaining particles in scalar mode and to that which uses SH. In fact, the difference is small to the point that the symbols in the graph would be totally superimposed.

The parameters in the various methods are initialized at the following values:

▲ In the NOY method LVEC = 64 is given to have maximum vectorization, and compatibly with it, a minimum amount of work memory.

▲ In the EH method, IGNMAX = 2. In fact, the average for particles per cell is equal to 1, and 2 appears to be a suitable choice.

▲ In the SH method NBLK = 128 is given. This obtains vectors of full length and, although to minimize the amount of memory required we could choose the value 64, we prefer a greater value in order to eliminate part of the overhead due to the initialization of the loops; the amount of extra storage remains fully negligible.

▲ In the GP method we choose IPMIN = NCELL/8; this means that when the vectors to be processed in DO 600 are on average of less than eight elements long, we process the remaining particles with a different method; this does not mean that this choice of IPMIN is the best one, as may be seen further on.

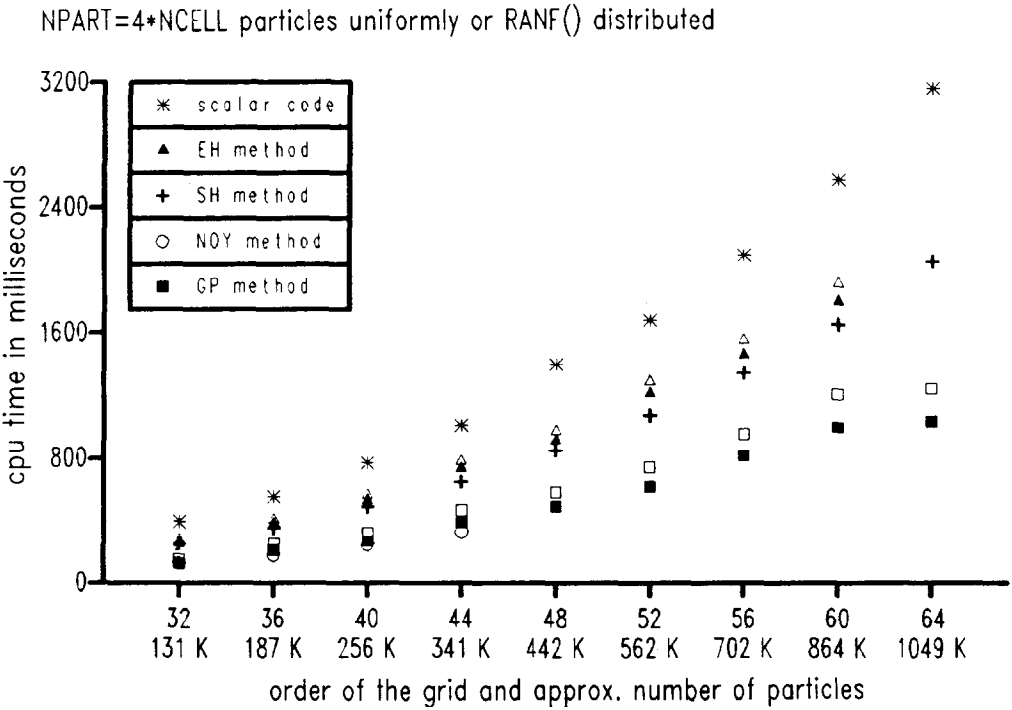

FIG. 4. Execution time with the various methods in the case of cubic grids, number of particles equal to four times the number of cells and with differing data distribution. Full and empty symbols have the same meaning as in Fig. 3. IGNMAX = 6 was used in the EH method. NOY and EH results for grids of order greater than 44 and 60, respectively, are missing, because their storage requirements exceed the memory size of Cray X-MP/48.

with the difference that the number of particles is equal to four times the number of grid cells. This case is significant, as in practical problems the number of particles usually exceed the number of cells. The parameters for the various methods have the same values as the previous case, except IGNMAX. In fact, as the average number of particles per cell is equal to 4, then IGNMAX = 6.

The generation of data with RANF( ) in this case leads to a distribution of data so that by taking the first four particles in each cell we obtain approximately 80% of the total number of data, and the maximum number of elements contained in the same cell varies from 14 to 17.

The shape of the timings in Fig. 4 is identical to that in Fig. 3 for all of the methods, with the exception of the NOY scheme, which is evidently improved. In particular, the fact that the performance of the GP scheme has no decrease testifies to the fact that the overhead represented by the collection of the particles, carried out in DO 300, 400, and 500, is negligible as compared to DO 600 which executes calculation.

Table I reports the amount of additional storage required in some of the examples by the different methods and compares it to the occupation of the scalar code, confirming the observations made before and clearly illustrating the reason for which the NOY and EH methods are missing some experimental results (Cray X-MP/48 memory is 8 Mword wide).

Table II reports a good indicator of the performance of different methods which can be obtained from the results of testing carried out: the average CPU time required for the processing of one particle. Table II clearly shows that the method proposed appears as the best, with a speedup ranging from 2 to 3 on the scalar code.

TABLE I

Memory Requirements of the Different Methods, in Three of the Cases Illustrated in Figs. 3 and 4

Problem size		Basic + Additional storage requirement (words)				
Grid order	Particles	Scalar	NOY	EH	SH	GP
32	32768	134241	2299968 + *1713* %	418570 + *312* %	2304 + *2* %	65537 + *49* %
48	110592	449425	7529536 + *1675* %	1383562 + *308* %	2304 + *1* %	221185 + *49* %
64	262144	1061057	17576000 + *1656* %	3245578 + *306* %	2304 + *0* %	524289 + *49* %
32	131072	429153	2299968 + *534* %	647950 + *151* %	2304 + *1* %	163841 + *38* %
48	442368	1444753	7529536 + *521* %	2157710 + *149* %	2304 + *0* %	552961 + *38* %
64	1048576	3420353	17576000 + *514* %	5080590 + *148* %	2304 + *0* %	1310721 + *38* %

Note. In the scalar code column the number of words occupied by vectors X, Y, Z, and D is reported; the successive columns report the occupation of words relative to auxilary arrays alone in the different methods, and thus these values must be added to that relative to the scalar code, in order to obtain a general estimate of the memory required. The percentage value provides an indication as to how much the use of one scheme overheads the occupation of memory of the scalar code.

TABLE II

Average CPU Time Required for the Deposition of the Mass of a Particle on the Surrounding Grid Points (in Microseconds)

Method	Microseconds	Notes
Scalar	3	Constant
NOY	2.1	particles = cells
	1	particles = 4 × cells
EH	2.1 ÷ 2.4	Depending on distribution uniformity
SH	1.9	
GP	1 ÷ 1.4	Depending on distribution uniformity

The NOY method obtains good performance when there are many more particles than cells, but when this is not the case, the performance is quite poor and does not justify the extra storage requirements.

The EH scheme matches considerable amount of memory needed with performances which, compared with those of other methods, do not justify its use.

The performance of the SH method is, instead, interesting; it is better than the scalar code by approximately a 1.6 factor, and it has the advantage of being constant, at least in the range of tests considered.

The last test, presented in Fig. 5, make it possible to evaluate the strength and the flexibility of the method proposed when data distribution becomes critical. Locating the particles at the center of a fixed number of cells, the non-uniformity of the distribution is varied in a controlled manner. The grid considered is $32 \times 32 \times 32$ cells wide, with 40,000 particles uniformly occupying a portion of it ranging from 10 to 100%.

In particular, IPMIN = NCELL/12, which means that even when the particles occupy only 10% of the grid the GP method processes the data in vector mode and does not consider all of them as remainders (in the 10% case there are 12 groups of 3276 particles, with a remainder of 688).

The NOY and SH methods and scalar code are independent of the type of data distribution, so they are executed in the same time: 73 ms for NOY, 78.5 for SH, 120.7 for the scalar code.

The EH scheme, instead, is restricted by the IGNMAX value, which from a certain point of view forces us to choose between performance and space in memory. The scalar grouping of the particles is a relevant overhead, as demonstrated by the time of 154.5 ms in the 10% case. From 70% on, the particles are contained in two groups only, so all of the particles are processed vectorially in a time of 84.3 ms.

The method here proposed (GP) has a storage requirement independent of the uniformity of particle distribution and thus provides a flexible response to the decay of the vectorizability of the physical system. The better performance of GP as compared to SH in the 10% case should be noted, despite the fact that DO 600 vectors are an average of 6.4 elements long (while SH vectorizes on dimensions of

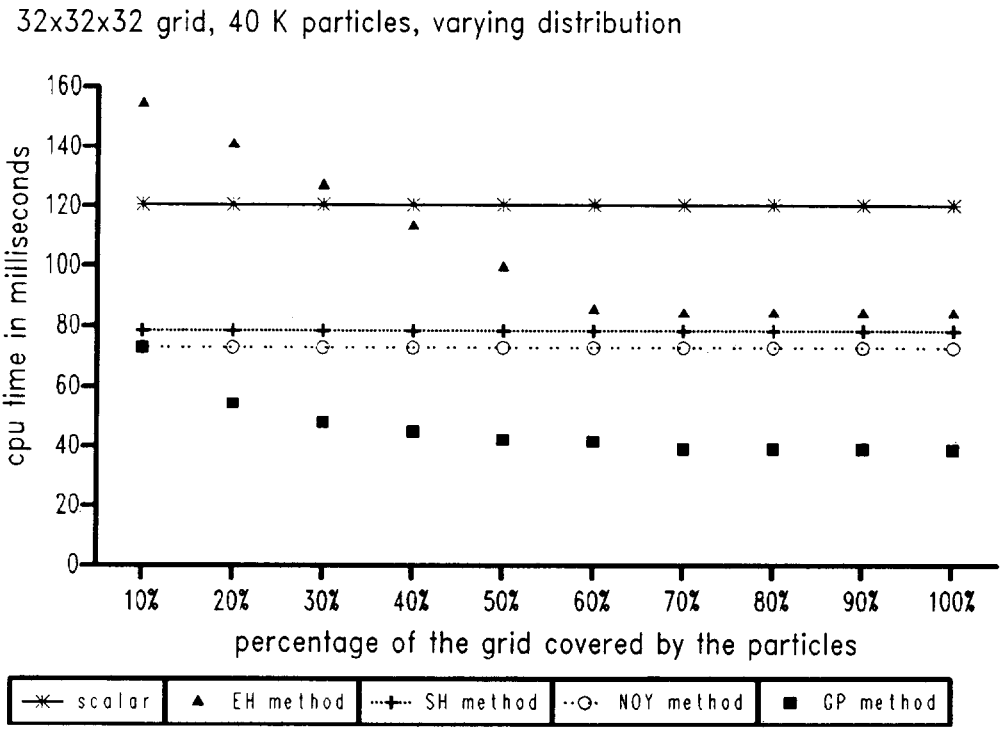

FIG. 5. Performance of different methods with accentuated variations of the uniformity degree of particle distribution on the grid, by locating all the particles in a fixed percentage of the space covered by the grid. The parameters are fixed at: LVEC = 64, IGNMAX = 2, NBLK = 128, IPMIN = NCELL/12.

128 and 8). This implies that a value of IPMIN less than NCELL/10 is preferable in the example considered.

In the example shown in Fig. 5 there are no relevant differences between the GP version with the remainders in scalar mode and that with the remainders processed with the SH scheme (the maximum differences are on the order of 0.8 ms, representing at most a 2% calculation time). If IPMIN = NCELL/8 had been used, the case relative to 10% would have been processed entirely as a remainder by the GP method, for the following corresponding times: 131.1 ms with the remainders in scalar mode and 87.4 with the remainder in SH. Here it is possible to appreciate the convenience of processing the remainders with SH instead of the scalar code, and the very small overhead due to the execution of the first iteration by the GP method before processing all of the particles as remainders.

There is a further observation to be made. Let us suppose that several quantities have to be calculated on the grid: the EH and NOY methods would require a larger extra storage, with no benefits for their performance: the memory required by GP

method would remain unchanged as would the performance; the SH method, instead, could operate on longer vectors, and thus its performance could increase.

Thus, opportunely choosing IPMIN, the algorithm proposed could make use of the vectorization possibilities of the physical system, passing control over to SH when the average length of the vectors on which it operates is small enough to prefer execution with SH. Such a method is free of trouble related to memory requirement, completely vectorized, and suited to achieve the most in terms of performance on a vast range of simulations.

CONCLUSIONS

The method proposed in this paper allows for complete and efficient vectorization on Cray X-MP (with CIGS hardware) of the interpolation on the grid phase in PIC codes, with a low storage requirement, which in practice may be placed in work areas needed in other phases of the simulation.

The timings obtained in a tridimensional astrophysical simulation with the variation of different parameters have shown an overall performance which is better than that obtained with other methods already in existence. The speedup over the scalar code ranges from 2 to 3. Despite the fact that the method proposed achieves maximum efficiency when the particles are distributed uniformly on the grid, it revealed considerable flexibility even in situations where the distribution is not uniform.

A further possibility of achieving a high performance without having, in practice, any need of extra storage, is that of executing the method proposed until the nature of the problem allows us to operate with significant vectors and to process the rest of the elements with the scheme proposed by Schwarzmeier and Hewitt [3], which is independent of the distribution of particles on the grid. The switch from one method to another depends on the number of array elements for which update within the deposition loop must be carried out.

The method proposed can be multitasked using the ordering of the grid cells introduced to allow vectorization, but this may be dealt with in the future.

ACKNOWLEDGMENTS

I thank Professor Antonio Messina, who provided the concrete example on which this study has been developed and whose competence I appreciated in many useful discussions. I also thank Professor Remo Rossi, director of CINECA, for his encouragement and interest, and the reviewers for useful suggestions.

Note Added in Proof. One of the referees has brought to my attention a recently published paper by Yoshihiko Abe [7], of which I was not aware. The algorithm proposed by Abe is based on the following consideration about the NOY method: if LVEC $\ll$ NGRID, then the probability of overlapping preventing the vectorization of the deposition loop is small, but it is not zero. Abe's method consists in

forcing the vectorization of the deposition loop (with length LVEC $\ll$ NGRID), verifying how many overlappings have occurred and correcting the consequent errors by repeating the previous steps for the particles whose contribution have been lost, and so on. This method can reach performances similar to the NOY scheme, if the frequency of corrections is sufficiently low, with a limited work storage requirement (NGRID + 2 * LVEC words).

Compared with my algorithm, Abe's method presents the disadvantage of repeating the calculation for the particles involved in the errors due to overlappings, and, if on the one hand it does not require any initialization loop to collect the particles into groups, on the other hand the loop needed to determine the particles involved in errors of overlapping represents an overhead. I believe that Abe's method could be preferred only when it is sure that the frequency of corrections needed is very low, and particularly on machines different from Cray. In fact this method is probably well suited for other kind of vector computers (it was developed on a Facom VP-200), as on a Cray the value of LVEC cannot exceed 64: with a value greater than the vector register length the method of detection of the overlapping particles would become unsafe.

REFERENCES

1. A. NISHIGUCHI, S. ORII, AND T. YABE, *J. Comput. Phys.* **61**, 519 (1985).
2. E. J. HOROWITZ, *J. Comput. Phys.* **68**, 56 (1987).
3. J. L. SCHWARZMEIER AND T. G. HEWITT, in *Proceedings, 12th Conf. on the Numerical Simulation of Plasmas, San Fransisco, CA, USA, 1987* (Lawrence Livermore Nat. Laboratories, Livermore, CA).
4. P. LONDRILLO AND A. MESSINA, *Monthly Not. Roy. Astronom. Soc.*, in press (1990).
5. R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles* (Mc Graw–Hill, New York, 1981).
6. J. M. CENTRELLA AND A. L. MELOTT, in *Numerical Astrophysics*, edited by J. M. Centrella *et al.* (Jones and Bartlett, Boston, 1985), p. 334.
7. Y. ABE, in *Supercomputing 88: Volume II, Science and Applications*, edited by J. L. Martin and S. F. Lundstrom (Comput. Soc. Press, Washington, DC, 1989), p. 72.