JOURNAL OF COMPUTATIONAL PHYSICS 85, 284–301 (1989)

Particle Code Optimization on Vector Computers

A. HÉRON AND J. C. ADAM

Centre de Physique Théorique, Ecole Polytechnique,
LP du CNRS n° 14, 91128 Palaiseau Cedex, France

Received May 31, 1988; revised December 8, 1988

Big particle codes routinely employed for plasma physics simulation use large amounts of computer time. Their optimization is therefore a must. In this paper we discuss proper organization of the particle pusher in order to take full advantage of the vectorization facilities offered by modern super computers. We show that this optimization is computer dependent and introduce a new method of vectorization that we compare with the two previously published ones. © 1989 Academic Press, Inc.

1. INTRODUCTION

Particle in cell codes are well-known tools for the study of kinetic phenomena in plasma physics [1, 2]. The object of this paper is the optimization, on vector computers, of the particle pusher itself. For a given algorithm this obviously involves vectorization of the code, but also an appropriate organization of the data to effectively utilize the computer architecture. To emphasize this point, at each step of the discussion we shall compare results obtained on two different computers: the Cray-2 and the VP200 of Fujitsu. The purpose is not to compare computer performances, but rather to show that the optimum can be very computer dependent.

Any particle code involves three steps:

(1) solving fields on a grid,

(2) advancing particles with the fields consistent with particle positions,

(3) depositing charge and current densities associated with each particle to the grid.

We shall limit our discussion to steps (2) and (3). This is sufficient for optimization of an explicit particle code, because the time spent in the field solver is usually negligible compared to the time spent advancing particles and computing charge and current densities. This may no longer be true in an implicit code [3, 4] for which the solution of the field equations implies solving iteratively a complicated linear system, but optimizing steps (2) and (3) would still necessary.

In this paper we introduce a new vectorization technique for step (3) that we compare to two previously published ones [5, 6]. The next section provides a brief description of the architecture of the Cray-2 and VP200.

0021-9991/89 $3.00

2. Architectures of Cray-2 and VP200

The Cray-2 on which timings have been made has four processors and dynamic MOS memory. The cycle time is 4. 1 ns and the size of the real memory available is 256 Mwords or 2 Gigabytes. It also has 8 vector registers of 64 words and a very fast local memory of 16 Kwords or 128 Kilobytes per processor. The add and multiply units can work in parallel but there is no chaining either between functional units or between memory and functional units. The memory has 256 pseudo banks, that are organized into four quadrants, connected to each CPU every four cycles.

Because of the importance of scatter/gather operations for a particle pusher we have measured the average time of these operations using three different vectors of indices. The length of the vector is 10^6 elements.

First, as a reference, let us indicate that for sequential access to the data, the average time necessary to execute the fortran instruction (the compiler used in the following is CFT77 1.3)

$$X(I) = EX(I),$$

is 10 ns/element. This instruction involves a read and a write to memory.

Table I shows the average time to execute the gather instruction, $X(I) = EX(ID(I))$, and the scatter instruction, $EX(ID(I)) = X(I)$. Case number 1 corresponds to maximum memory conflict; case number 2 is a sequential access, so that it provides a measure of he overhead introduced by the scatter/gather operation (keep in mind that it involves at least the load from memory of the extra quantity ID(I). Finally, in case 3, ID(I) is loaded using a uniform random number generator so that it provides a good estimate of random access to memory as needed in a particle pusher.

An important feature of the Cray-2 that appeared during these measurements is that it behaves well even short vectors (typically for a length larger than 8), so that the vector length is not a very important factor for optimization on this computer.

The VP200 is a single processor machine. The cycle time of the vector functional units is 7 ns and the real memory available on the one that was used is 128 Megabytes which corresponds to 32 Mwords in single precision or 16 Mwords in double precision. There are two data paths between the memory and functional units and each of the functional units is a double pipe line. This means that each

TABLE I

CRAY2	Index	Gather	Scatter
1	ID(I) = const	273 ns	270 ns
2	ID(I) = I	29 ns	15 ns
3	ID(I) = RANF( )	46 ns	36 ns

TABLE II

VP200	Index	Gather		Scatter	
		Single precision	Double precision	Single precision	Double precision
1	ID(I) = const	103 ns	104 ns	340 ns	286 ns
2	ID(I) = I	54 ns	10 ns	172 ns	9.7 ns
3	ID(I) = RANF( )	13 ns	15 ns	25 ns	24 ns

of them is able to deliver two results per cycle. Contrary to the Cray-2 chaining is possible between memory and functional units or between functional units themselves. These units can also operate in parallel. The vector register has a size of 64 Kbytes that can be dynamically configured as 256 registers of 32 elements of 8 bytes up to 8 registers of 1024 elements of 8 bytes. The memory is organized in 256 banks.

As before let us first indicate that for sequential access to the data the time to execute the instruction X(I) = EX(I) is $t = 4.9$ ns/element in both single or double precision.

Table II shows the same measures of performance as for the Cray-2 in the same conditions. The degradation of the timings for single precision gather or scatter operations is due to the memory organization. Contrary to the Cray-2, the VP200 is much more efficient on long vectors than on short ones. In all what follows single precision which is accurate enough for a particle pusher, is used. The preceding measurement done with a vector of 10^6 elements are asymptotic transfer rates that cannot be obtained with vector length much lower than 10^3 elements. For this computer, the vector length is an important factor of optimization.

3. Optimization of the Particle Pusher

Solving the equations of motion for particles implies as a first step interpolating the fields from the grid to the positions of the particles. Because these positions are random, it involves an indirection corresponding to a gather operation. Once the fields seen by the particle have been obtained the vectorization of the equations of particle motion is trivial.

Because of the absence of adequate hardware, the vectorization of the gather operation was not possible on the Cray-1. This means that in order to be able to vectorize the equations of motion, the interpolation of the fields was done in a seperate loop and the results stored in an auxiliary array.

This is no longer true on modern vector computers because gather and scatter are allowed by the hardware and also because most modern compilers are able to

recognize the corresponding structures. This means that the preceding decomposition of the particle loop is no longer necessary. On the contrary, an important feature of most "super computers" is a relatively slow access to memory as compared to the speed of the functional units. Optimization implies suppression of unnecessary transfers to and from memory. Thus the creation of auxiliary arrays must be avoided. Scalar temporary quantities are much more efficient because they are extended to vector temporary quantities by the compiler and not saved in memory.

As an example, for a 1D electrostatic code using appropriate normalization, the advance of particles can be reduced to the following loop

```
      DO 1 N = 1,NPART
      I = INT(X(N))
      XINT = X(N) - FLOAT(I)
      VX(N) = VX(N) - EX(I) - XINT*(EX(I + 1) - EX(I))
      X(N) = X(N) + VX(N)
    1 CONTINUE
```

where NPART is the number of particles, X and VX are respectively the position and velocity of the particle, EX is the electrostatic field defined on a grid.

We have shown in the preceding section, that on both computers considered here, timing of the gather operations depends on the structure of the vector of indirection. To illustrate this point we shall indicate now the timings obtained for the previous loop for two kinds of initial loading of the particles. Both cases correspond to a homogeneous system with one hundred particles per cell.

First, particles were loaded randomly, i.e., their initial positions were obtained using a uniform random number generator providing numbers in the range [0,NCEL], where NCEL is the total number of cells. This yielded the following times per particle per time step:

Cray-2	131 ns,
VP200	33 ns.

Next, particles were loaded regularly, as for a "quiet start" with DX = FLOAT(NCEL)/FLOAT(NPART). Due to memory access conflicts the timings became

Cray-2	500 ns,
VP200	190 ns.

This shows the importance of the initial loading on the performance of a particle code. This is especially true for the ions that move very slowly, but it is also valid for electrons for which despite their small mass, a degradation of a factor of about two can persist for a long time.

4. Vectorization of the Charge and Current Density Deposition

In the absence of scatter/gather hardware operators the optimum structure of a 2D electro magnetic particle code follows. It yield partial vectorization and is given here as a reference before "full optimization."

In this approach of vectorization the deposition loop is split into two loops. The first computes all the weights and indices that are necessary. The next is a scalar accumulation of the weights previously computed.

In the subsequent parts of the paper the variable NVECT designate the length of auxiliary vectors used for the computation of the interpolation weights, as well as the corresponding cell indices. If one had enough memory NVECT could be set equal to NPART, but this is usually not the case even on a Cray-2. The size of NVECT results from a trade-off between memory size constraint and speed up obtained by increasing the vector length.

In order to minimize the number of arihmetical operations and especially to avoid the presence of divisions which are very costly on any computer, in all that follows, X is expressed in units of DX while Y is expressed in units of DY, yielding a grid cell area equal to one.

```
C
C    SCALAR METHOD FOR ACCUMULATION OF CHARGE AND CURRENT DENSITIES
C
C        NDIMX:NUMBER OF CELLS IN X DIRECTION

         DO 1 IN=1,NPART,NVECT
         N=IN-1
         DO 2 K=1,NVECT
         N=N+1
C
C    INDICES FOR CHARGE DENSITY
C
         I=INT(X(N))
         XINT=X(N)-FLOAT(I)
         J=INT(Y(N))
         YINT=Y(N)-FLOAT(J)
         ID(K)=I+J*NDIMX
C
C    WEIGHTS FOR CHARGE DENSITY
C
         SR1(K)=(1.-XINT)*(1.-YINT)
         SR4(K)=XINT*YINT
         SR2(K)=XINT-SR4(K)
         SR3(K)=YINT-SR4(K)
C
C    INDICES FOR CURRENT DENSITY
C
         XAUX=X(N)-0.5*VX(N)
         I=INT(XAUX)
         XINT=XAUX-FLOAT(I)
         YAUX=Y(N)-0.5*VY(N)
```

```
          J=INT(YAUX)
          YINT=YAUX-FLOAT(J)
          JD(K)=I+J*NDIMX
C
C     WEIGHTS FOR CURRENT DENSITY
C
          S1=(1.-XINT)*(1.-YINT)
          S4=XINT*YINT
          S2=XINT-S4
          S3=YINT-S4
C
          SJX1(K)=S1*VX(N)
          SJX2(K)=S2*VX(N)
          SJX3(K)=S3*VX(N)
          SJX4(K)=S4*VX(N)
C
          SJY1(K)=S1*VY(N)
          SJY2(K)=S2*VY(N)
          SJY3(K)=S3*VY(N)
          SJY4(K)=S4*VY(N)
C
      2   CONTINUE
C
C      SCALAR ACCUMULATION OF CHARGE AND CURRENT DENSITIES
C
          DO 3 K=1,NVECT
C
          RHO(ID(K))          =RHO(ID(K))+SR1(K)
          RHO(ID(K)+1)        =RHO(ID(K)+1)+SR2(K)
          RHO(ID(K)+NDIMX)    =RHO(ID(K)+NDIMX)+SR3(K)
          RHO(ID(K)+NDIMX+1)=RHO(ID(K)+NDIMX+1)+SR4(K)
C
          RJX(JD(K))=RJX(JD(K))+SJX1(K)
          RJY(JD(K))=RJY(JD(K))+SJY1(K)
          JD2=JD(K)+1
          RJX(JD2)=RJX(JD2)+SJX2(K)
          RJY(JD2)=RJY(JD2)+SJY2(K)
          JD3=JD(K)+NDIMX
          RJX(JD3)=RJX(JD3)+SJX3(K)
          RJY(JD3)=RJY(JD3)+SJY3(K)
          JD4=JD3+1
          RJX(JD4)=RJX(JD4)+SJX4(K)
          RJY(JD4)=RJY(JD4)+SJY4(K)
C
      3   CONTINUE
      1   CONTINUE
```

Using this structure the following times per particle per time step were obtained:

Cray-2 3.25μs,

VP200: 2.72μs.

Contrary to the evolution of particles, because two particles belonging to the same vector can contribute to the same location in memory, the availability of the necessary hardware to vectorize scatter/gather operations is not sufficient to fully vectorize charge and current deposition. Full vectorization implies suppression of these dependencies. This suppression must not induce a large overhead nor may it use too much memory. Two methods to remove the dependencies have been proposed in the literature.

(a) *Nishiguchi, Orii, and Yabe Method*

In their method [5], particles of a vector contribue to different auxiliary arrays where charge and current are accumulated. At the end of the accumulation step a summation of these different arrays is made, in order to obtain the total charge and current densities. As an example we have written below the new structure of the structure of the deposition loop. NGROUP is the number of particles that are used simultaneously in order to vectorize charge and current deposition. It determines the vector length in this method.

```
C
C   METHOD OF  NISHIGUCHI ET AL
C

      DO 1 IN=1,NPART,NGROUP
      N=IN-1
CDIR$ IVDEP
*VOCL LOOP,NOVREC
      DO 2 K=1,NGROUP
      N=N+1
C
C   INDICES FOR CHARGE DENSITY
C
      I=INT(X(N))
      XINT=X(N)-FLOAT(I)
      J=INT(Y(N))
      YINT=Y(N)-FLOAT(J)
      ID=I+J*NDIMX
C
C   WEIGHTS FOR CHARGE DENSITY
C
      SR1=(1.-XINT)*(1.-YINT)
      SR4=XINT*YINT
      SR2=XINT-SR4
      SR3=YINT-SR4
C
C  ACCUMULATION FOR CHARGE DENSITY
C
      RAUX(ID,K)          =RAUX(ID,K)+SR1
      RAUX(ID+1,K)        =RAUX(ID+1,K)+SR2
      RAUX(ID+NDIMX,K)    =RAUX(ID+NDIMX,K)+SR3
      RAUX(ID+NDIMX+1,K)=RAUX(ID+NDIMX+1,K)+SR4
```

```
C
C    INDICES FOR CURRENT DENSITY
C
          XAUX=X(N)-0.5*VX(N)
          I=INT(XAUX)
          XINT=XAUX-FLOAT(I)
          YAUX=Y(N)-0.5*VY(N)
          J=INT(YAUX)
          YINT=YAUX-FLOAT(J)
          JD=I+J*NDIMX
C
C    WEIGHTS AND ACCUMULATION FOR CURRENT DENSITY
C
          S1=(1.-XINT)*(1.-YINT)
          S4=XINT*YINT
          S2=XINT-S4
          S3=YINT-S4
C
          RAUJX(JD,K)=RAUJX(JD,K)+S1*VX(N)
          RAUJY(JD,K)=RAUJY(JD,K)+S1*VY(N)
          JD2=JD+1
          RAUJX(JD2,K)=RAUJX(JD2,K)+S2*VX(N)
          RAUJY(JD2,K)=RAUJY(JD2,K)+S2*VY(N)
          JD3=JD+NDIMX
          RAUJX(JD3,K)=RAUJX(JD3,K)+S3*VX(N)
          RAUJY(JD3,K)=RAUJY(JD3,K)+S3*VY(N)
          JD4=JD3+1
          RAUJX(JD4,K)=RAUJX(JD4,K)+S4*VX(N)
          RAUJY(JD4,K)=RAUJY(JD4,K)+S4*VY(N)
C
      2   CONTINUE
      1   CONTINUE
C
C
          DO 3 K=1,NGROUP
          DO 3 I=0,NCEL
          RHO(I)=RHO(I)+RAUX(I,K)
          RJX(I)=RJX(I)+RAUJX(I,K)
          RJY(I)=RJY(I)+RAUJY(I,K)
      3   CONTINUE
```

The CDIR$ IVDEP and *VOCL LOOP, NOREC are compiler directives for the Cray-2 and VP200, respectively, specifying the absence of recurrence dependancies in the loop.

One of the factors of speedup of this method is that the inclusion of the charge accumulation into "loop 2" suppresses the necessity to store the vector of indices and the corresponding weights, which in turn minimizes memory accesses.

Some overhead is induced by the supplementary additions that are needed at the end of the accumulation step. In the following, F is the number of physical quantities that must be accumulated. It is one for an electrostatic code and up to four, for an electromagnetic code dealing with three components of the current density.

TABLE III

	Original method	Modified method
VP200	1.22μs	1.48μs
CRAY2	1.63μs	1.80μs

Using the already introduced notations, the total number of extra additions is:

$$N = \text{NGROUP} \times \text{NCEL} \times F;$$

the size of the required auxiliary storage is:

$$M = \text{NGROUP} \times \text{NCEL} \times F.$$

Vectorization is efficient only if the vector length NGROUP is large enough. If NGROUP has to be larger than 32, the storage requirement of the method can become a problem in 2D and 3D. It can be reduced by using the same set of auxiliary arrays successively for the F physical quantities. This involves recomputing the interpolation weights and introducing an additional overhead of the order of 20%. These two variations of the same algorithm correspond to the "original method" and "modified method" in the tables.

The results shown in Table III were obtained for the same model as previously (F = 3) and for NGROUP = 16. Table IV shows the influence of the length of NGROUP for the modified version of the algorithm. A larger value of NGROUP implies extra additions that can only be compensated by a speed up of the vector operation. The figures given for the VP200 clearly show that this is indeed true. On the contrary, on the Cray-2 the optimum is reached for NGROUP $\approx$ 16 which is a relatively short vector.

In all the following we have chosen to refer to the timings obtained using this modified method.

(b) *Horowitz Method*

The second method to remove dependancies is due to Horowitz [6]. It is based on the idea that to obtain correct results, two particles of a given vector must not contribue to the same cell. This requires sorting of the particles and the paper by Horowitz is essentially a description of an optimal enumeration method for a particle code.

TABLE IV

NGROUP	8	16	32	64
VP200	2.66 μs	1.48 μs	0.895 μs	0.74 μs
CRAY2	2.5 μs	1.8 μs	1.8 μs	2.6 μs

The sort involves splitting particles into groups, each group corresponding to a vector for charge and current density deposition. An important point is that the area weighting scheme used in standard particle codes does not guarantee that because two particles are in different cells, their contributions to the neighbouring cells are independent. This means that the different contributions of a given particle must be accumulated in different arrays. The corresponding algorithm can be written schematically in the following form (The reader is referred to Horowitz's paper for details of the sort).

Step 1. Execute Horowitz's sort.

At the end of this step the following quantities have been obtained (in Horowitz's notation)

IGNMAX: is total number of groups of independent particles

IPOINT: is a pointer giving the address of the beginning of each group

IP: is an array of indices pointing at each particle of the group.

Step 2 is the deposition step by itself which becomes:

```
C
C   METHOD OF  HOROWITZ  -   DEPOSITION STEP
C

      DO 1 IGN=1,IGNMAX
CDIR$ IVDEP
*VOCL LOOP,NOVREC
      DO 2 K=IPOINT(IGN),IPOINT(IGN+1)-1
C
      N=IP(K)
C
C   INDICES FOR CHARGE DENSITY
C
      I=INT(X(N))
      XINT=X(N)-FLOAT(I)
      J=INT(Y(N))
      YINT=Y(N)-FLOAT(J)
      ID=I+J*NDIMX
C
C   WEIGHTS FOR CHARGE DENSITY
C
      SR1=(1.-XINT)*(1.-YINT)
      SR4=XINT*YINT
      SR2=XINT-SR4
      SR3=YINT-SR4
C
C  ACCUMULATION FOR CHARGE DENSITY
C
      RAUX(ID,1)          =RAUX(ID,1)+SR1
      RAUX(ID+1,2)        =RAUX(ID+1,2)+SR2
      RAUX(ID+NDIMX,3)    =RAUX(ID+NDIMX,3)+SR3
      RAUX(ID+NDIMX+1,4)=RAUX(ID+NDIMX+1,4)+SR4
```

```
C
    2   CONTINUE
    1   CONTINUE
C
C       HOROWITZ'S SORT FOR CURRENT DENSITY
C
        DO 11 IGN=1,IGNMAX
  CDIR$ IVDEP
  *VOCL LOOP,NOVREC
        DO 12 K=IPOINT(IGN),IPOINT(IGN+1)-1
C
        N=IP(K)
C
C   INDICES FOR CURRENT DENSITY
C
        XAUX=X(N)-0.5*VX(N)
        I=INT(XAUX)
        XINT=XAUX-FLOAT(I)
        YAUX=Y(N)-0.5*VY(N)
        J=INT(YAUX)
        YINT=YAUX-FLOAT(J)
        JD=I+J*NDIMX
C
C   WEIGHTS AND ACCUMULATION FOR CURRENT DENSITY
C
        S1=(1.-XINT)*(1.-YINT)
        S4=XINT*YINT
        S2=XINT-S4
        S3=YINT-S4
C
        RAUJX(JD,1)=RAUJX(JD,1)+S1*VX(N)
        RAUJY(JD,1)=RAUJY(JD,1)+S1*VY(N)
        JD2=JD+1
        RAUJX(JD2,2)=RAUJX(JD2,2)+S2*VX(N)
        RAUJY(JD2,2)=RAUJY(JD2,2)+S2*VY(N)
        JD3=JD+NDIMX
        RAUJX(JD3,3)=RAUJX(JD3,3)+S3*VX(N)
        RAUJY(JD3,3)=RAUJY(JD3,3)+S3*VY(N)
        JD4=JD3+1
        RAUJX(JD4,4)=RAUJX(JD4,4)+S4*VX(N)
        RAUJY(JD4,4)=RAUJY(JD4,4)+S4*VY(N)
C
   12   CONTINUE
   11   CONTINUE
C
C
        DO 3 I=0,NCEL
        RHO(I)=RAUX(I,1)+RAUX(I,2)+RAUX(I,3)+RAUX(I,4)
        RJX(I)=RAUJX(I,1)+RAUJX(I,2)+RAUJX(I,3)+RAUJX(I,4)
        RJY(I)=RAUJY(I,1)+RAUJY(I,2)+RAUJY(I,3)+RAUJY(I,4)
    3   CONTINUE
```

The main advantage of this method is that it can yield to vectors which can be significantly larger than in the first method.

TABLE V

VP200	Original method	Modified method
Charge and Current deposition	0.72 μs	0.77 μs
Total time including sorting	2.86 μs	2.91 μs

In this case, the overhead induced by vectorization is essentially related to the sorting of particles. The extra additions are much less important than in the preceding method. Using the previous notation, the additional storage required is approximatively given by

$$\text{MAX}(2*\text{NPART} + \text{NCEL}, \text{NPART} + 2^{\text{D}} \times F \times \text{NCEL}).$$

Here again, if the term $2^{\text{D}} \times F \times \text{NCEL}$ is dominant in terms of storage, the extra storage required can be reduced at very little cost by recomputing the weights associated with each physical quantity. As previously, the two variations of this algorithm correspond to the "original method" and the "modified method." Let us emphasize that for an electromagnetic code, ρ and J, are usually not defined at the same position and the same time. This implies that it is necessary to sort the particles twice, which increases the overhead.

On both computers, the sort takes about 1 μs per particle. The corresponding timings are shown Tables V and VI.

(c) *A New Approach*

Both preceding methods use extra storage to allow vectorization. This storage is usually available for 2D simulations, but it becomes a problem in 3D. The aim of the new approach that we have developed is to reduce the required auxiliary storage and to generate a more efficient algorithm by also reducing the number of auxiliary operations introduced by vectorization.

The basic idea is that, for a 2D particle code using linear weights of interpolation, the four grid cells to which these weights are assigned are distinct. It is also true that the memory location of each physical quantity is distinct. We combine all these independent contributions into a single vector. This yields a vector of length 12 associated with the three physical quantities ρ, J_x, J_y. Specifically for the particle K, ID(I, K) ($I = 1, 4$) contain the addresses of the four cell numbers corresponding to the charge density ρ and SD(I, K) ($I = 1, 4$) the corresponding weights. Similarly,

TABLE VI

CRAY2	Original method	Modified method
Charge and current deposition	0.99 μs	1.19 μs
Total time including sorting	3.05 μs	3.25 μs

ID(I, K) and SD(I, K) ($I = 5, 8$) contain the addresses and weights corresponding to J_x while ID(I, K) and SD(I, K) ($I = 9, 12$) contain the same quantities for J_y. The general length is $2^D \times F$. It is fair to note that this idea has been presented simultaneously by J. L. Schwarzmeier *et al.* and us at the 12th Conference on the Numerical Simulation of Plasma in 1987.

At this stage we have generated a short vector and it can be shown that the extra work induced by vectorization is:

> NPART auxiliary additions for an electrostatic code used to build the vector of indices, $(2^D \times (F-2)+2) \times$ NPART auxiliary additions for an electromagnetic code.

In order to improve the efficiency of vectorization the vector length must be increased. This is done by combining our approach with the first method. This means that we handle NGROUP particles simultaneously. This yields a vector length equal to NGROUP $\times 2^D \times F$. The advantage is that as we start with a vector length larger than one, the number of auxiliary arrays needed is reduced by a factor 2^D and the number of extra additions is reduced by $2^D \times F$. As before, we shall now illustrate the practical implementation of the method by writing down the corresponding FORTRAN loop. Note however that the arrays involved must be declared in a single "COMMON" statement in order to guarantee the proper memory organization. As NGROUP is a small number of particles, we preserve the efficiency of the computation of the interpolation weights by using auxiliary arrays of length NVECT as in the reference code:

```
C
C  DEFINITION OF CONSTANTS ,NGR AND NGJ
C  NGR,NGJ,POINTERS TO BEGINNING OF PARTICLE GROUP
C
        NCF3=NCEL*3
        NVF12=NVECT*12
        NPF12=NGROUP*12
C
        DO 10 IK=1,NVECT,NGROUP
        K=IK-1
        DO 11 IGROUP=1,NGROUP
        K=K+1
        NGR(K)=1+(IGROUP-1)*NCF3
        NGJ(K)=NGR(K)+NCEL
     11 CONTINUE
     10 CONTINUE
C
C   NEW  METHOD FOR ACCUMULATION OF CHARGE AND CURRENT DENSITIES
C
        COMMON/ARRAY/RHO(0:NDIM),RJX(0:NDIM),RJY(0:NDIM),
       .             TABAUX(NCF3,NGROUP-1)
        DIMENSION ID(12,NVECT),SD(12,NVECT),ID1D(NVF12),SD1D(NVF
        EQUIVALENCE (ID(1,1),ID1D(1)),(SD(1,1),SD1D(1))
        DIMENSION TAB(NCF3,NGROUP),TAB1D(NCF3*NGROUP)
        EQUIVALENCE (TAB(1,1),RHO(0)),(TAB(1,1),TAB1D(1))
```

```
C
      DO 1 IN=1,NPART,NVECT
      N=IN-1
      DO 2 K=1,NVECT
      N=N+1
C
C   INDICES FOR CHARGE DENSITY
C
      I=INT(X(N))
      XINT=X(N)-FLOAT(I)
      J=INT(Y(N))
      YINT=Y(N)-FLOAT(J)
      ID(1,K)=I+J*NDIMX+NGR(K)
      ID(2,K)=ID(1,K)+1
      ID(3,K)=ID(1,K)+NDIMX
      ID(4,K)=ID(3,K)+1
C
C   WEIGHTS FOR CHARGE DENSITY
C
      SD(1,K)=(1.-XINT)*(1.-YINT)
      SD(4,K)=XINT*YINT
      SD(2,K)=XINT-SD(4,K)
      SD(3,K)=YINT-SD(4,K)
C
C   INDICES FOR CURRENT DENSITY
C
      XAUX=X(N)-0.5*VX(N)
      I=INT(XAUX)
      XINT=XAUX-FLOAT(I)
      YAUX=Y(N)-0.5*VY(N)
      J=INT(YAUX)
      YINT=YAUX-FLOAT(J)
      ID(5,K)=I+J*NDIMX+NGJ(K)
      ID(6,K)=ID(5,K)+1
      ID(7,K)=ID(5,K)+NDIMX
      ID(8,K)=ID(7,K)+1
      ID(9,K)=ID(5,K)+NCEL
      ID(10,K)=ID(6,K)+NCEL
      ID(11,K)=ID(7,K)+NCEL
      ID(12,K)=ID(8,K)+NCEL
C
C   WEIGHTS FOR CURRENT DENSITY
C
      S1=(1.-XINT)*(1.-YINT)
      S4=XINT*YINT
      S2=XINT-S4
      S3=YINT-S4
C
      SD(5,K)=S1*VX(N)
      SD(6,K)=S2*VX(N)
      SD(7,K)=S3*VX(N)
      SD(8,K)=S4*VX(N)
      SD(9,K)=S1*VY(N)
      SD(10,K)=S2*VY(N)
```

```
          SD(11,K)=S3*VY(N)
          SD(12,K)=S4*VY(N)
C
      2   CONTINUE
C
C     ACCUMULATION OF CHARGE AND CURRENT DENSITIES
C
          DO 3 K=1,NVF12,NPF12
C
   CDIR$ IVDEP
   *VOCL LOOP,NOVREC
          DO 4 IK=K,K+NPF12-1
          TAB1D(ID1D(IK))=TAB1D(ID1D(IK))+SD1D(IK)
      4   CONTINUE
      3   CONTINUE
C
      1   CONTINUE
C
C
          DO 5 IGROUP=2,NGROUP
          DO 5 I=1,NCEL
          TAB(I,1)=TAB(I,1)+TAB(I,IGROUP)
      5   CONTINUE
```

The optimum organization of the vector of indices is computer dependent. On the Cray-2, it has been indicated in Section 2 that the memory was organized into four quadrants to which each CPU has access every four cycles. This means that the indices associated with the four weights of each physical quantities must point to different quadrant of the memory. According to the definition of ID(3, K) as a function of ID(1, K) this implies NDIMX $= 4N + 2$. A further constraint of optimization is that the total dimension of the arrays involve is such that NCEL $= 4N$. This ensures that the indices of the different components of the current density are in consecutive quadrants. If a particle has not changed cell in half a time step the indices of the charge density are also in consecutive quadrants.

Because of the difference in memory structure, the previous organization is not optimum on the VP200 in single precision. The detailed timings of scatter/gather operations that we have provided at the beginning of this paper have shown that referencing two consecutive elements of memory in single precision yields bank conflicts that are catastrophic in terms of performance. A much better organization on the VP200 is: ID(1, K), ID(3, K), ID(5, K), ID(7, K), ID(9, K), ID(11, K), ID(2, K), ID(4, K), ...

Because of the bank organization of the VP200 this scheme is optimal only if NDIMX $= 4N + 2$ and NCEL $= 4N$.

It is left to the reader to discover better combinations. Timings of Table VII illustrate the previous consideration. In each case we have combined particles in group of 5 (NGROUP $= 5$).

The influence of bank conflicts is clearly visible on the VP200. On the Cray-2, the use of the VP200 organization of the vector of indices only yields quadrant conflicts

TABLE VII

	Cray organization	VP organization
VP200	2.18 μs	1.22 μs
CRAY2	1.35 μs	1.5 μs

every two indices and no bank conflicts, which explains why the decrease in performance is not very high.

5. Comparison of the Three Methods

In the preceding sections we have shown the influence of the different free parameters (number of particles in a group, memory organization) on the overall performance of each of the methods. In this section we shall compare their relative performances under the constraint that the auxiliary storage, used by each of them, is approximatively the same. Assuming 10 particles per cell, this yields NGROUP = 16 for the first method and NGROUP = 5 for the third method, when collecting ρ, J_x, J_y or NGROUP = 16 when collecting only ρ. The corresponding timings are shown in Table VIII. The number given in parentheses beside each timing is the vector length. This number is not indicated in the Horowitz method because it depends on the output of the sort and, as it is usually large, it is much less critical. The results referred to as "scalar deposition" correspond to the reference code, i.e., to the structure of the program before "full vectorization" of the particle pusher. Since the basis of the new method corresponds to the case NGROUP = 1, we have also included the corresponding timings under the reference NO X-STOR (no extra storage).

Finally in Table IX we present timings obtained for the 3D case. We have assumed that the memory constraint was more severe than in 2D. In particular to limit extra storage we have assumed that in the Horowitz method particles are

TABLE VIII

	CRAY2		VP200	
	ρ	ρ, J_x, J_y	ρ	ρ, J_x, J_y
Scalar deposition	1.22 μs	3.03 μs	1 μs	2.72 μs
Horowitz	1.41 μs	3.45 μs	1.35 μs	2.91 μs
Nishiguchi *et al.*	0.55 μs (16)	1.80 μs (16)	0.42 μs (16)	1.48 μs (16)
Heron, Adam	0.52 μs (64)	1.35 μs (60)	0.55 μs (64)	1.22 μs (60)
Heron, Adam (NO X-STOR)	1.35 μs (4)	1.90 μs (12)	2.14 μs (4)	2.3 μs (12)

TABLE IX

	CRAY2		VP200	
	ρ	ρ, J_x, J_y, J_z	ρ	ρ, J_x, J_y, J_z
Scalar deposition	3.4 μs	9.4 μs	1.56 μs	6.48 μs
Horowitz	2 μs	5.95 μs	1.36 μs	3.51 μs
Nishiguchi *et al.*	1.4 μs	5.7 μs	1.75 μs	7.12 μs
Heron, Adam	1 μs (64)	2.6 μs (64)	0.89 μs (64)	2.12 μs (64)
Heron, Adam (NO X-STOR)	1.6 μs (8)	3.1 μs (32)	2.2 μs (8)	2.65 μs (32)

sorted by blocks. For all the methods we have limited the auxiliary memory requirement to eight times NCEL (note that for NCEL $= 128^3$ this is already in the range of 16 Mwords). This implies NGROUP $= 8$ for Nishiguchi's method, NGROUP $= 8$ for the electrostatic case and NGROUP $= 2$ for the electromagnetic case in our method.

The comparison of the three preceding methods of vectorization yields the following conclusions: For the Cray 2,

(a) The new method that we have introduced is always more efficient.

(b) The results obtained without extra storage show that, if the vector length is smaller than eight (2D electrostatic code), the vectorization is inefficient and should not be implemented. This case which corresponds to a severe memory constraint seems rather unlikely on the Cray-2.

(c) In 3D where memory constraint is likely to occur, good performance can be obtained without auxiliary storage and without any programming pain.

For the VP200,

(a) Without memory constraint the method by Nishiguchi *et al.* always the fastest (NGROUP ≈ 64).

(b) If one is more realistic in terms of memory constraint, then, our method is always better except for the 2D electrostatic case.

(c) The timings without extra storage also show that vectorization should be avoided if the vector length is smaller than 12.

Finally we feel that the two computers that we have compared provide some more general insight on the behavior of the algorithms for a larger class of computers. If the archictecture of a given computer is such that it handles efficiently short vectors, the conclusions for the Cray-2 hold, otherwise the conclusion for the VP200 are probably correct.

It is also worthwhile to note that, on vector computers, the deposition step of a particle pusher is an expensive step. Depending on the model (electrostatic

or electromagnetic) and the number of dimensions, we have found that after optimization this step costs typically between 50 and 65% of the total particle pusher time.

The computers used were the VP200 of CIRCE and the Cray-2 of CCVR.

REFERENCES

1. R. W. HOCKNEY AND J. W. EASTWOOD, *Computer Simulation Using Particles* (McGraw–Hill, New York, 1981).
2. C. K. BIRDSALL AND A. B. LANGDON, *Plasma Physics via Computer Simulation* (McGraw–Hill, New York, 1985).
3. J. U. BRACKBILL AND D. W. FORSLUND, *J. Comput. Phys.* **46**, 271 (1982).
4. A. B. LANGDON, B. I. COHEN, AND A. FRIEDMAN, *J. Comput. Phys.* **51**, 107 (1983); D. W. HEWETT AND A. B. LANGDON, *J. Comput. Phys.* **72**, 121 (1987).
5. A. NISHIGUCHI, S. ORII, AND T. YABE, *J. Comput. Phys.* **61**, 519 (1985).
6. E. J. HOROWITZ, *J. Comput. Phys.* **68**, 56 (1987).