```
%Buneman, O., TRISTAN: The 3-D E-M Particle Code, in {\it Computer Space Plasma
%    Physics: Simulation Techniques and Software}, ed. H. Matsumoto and Y.
%    Omura, P.67, 1993.

      common /partls/ x(8192),y(8192),z(8192),u(8192),v(8192),w(8192)
      common /fields/ ex1(8192),ey1(8192),ez1(8192),
     &bx1(8192),by1(8192),bz1(8192)
      dimension smooth(30),mooths(30)
      equivalence (smooth(28),rs),(smooth(29),ps),(smooth(30),qs),
     &(mooths(28),ix),(mooths(29),iy),(mooths(30),iz)
      real me,mi
C Fields can be treated as single-indexed or triple-indexed:
      dimension ex(21,19,20),ey(21,19,20),ez(21,19,20),
     &bx(21,19,20),by(21,19,20),bz(21,19,20)
      equivalence (ex1(1),ex(1,1,1)), (ey1(1),ey(1,1,1)), (ez1(1),ez(1,
     &1,1)), (bx1(1),bx(1,1,1)), (by1(1),by(1,1,1)), (bz1(1),bz(1,1,1)),
     &(ez1(8001),c),(ez1(8002),q),(ez1(8003),smooth(1)),
     &(ez1(8033),mooths(1))
      dimension rho(21,19,20)                                    !for div
      equivalence (x(97),rho(1,1,1))                             !-check
C For CRAY-s, the first two field dimensions should be ODD, as here:
      mx=21
      my=19
      mz=20
C Strides for single-indexed field arrays:
      ix=1
      iy=mx
      iz=iy*my
      lot=iz*mz
C Miscellaneuos constants:
      qe=-1.                                                  *8.for test
      qi=1.                                                   *8.for test
      me=1.                                                  e10for test
      mi=32.                                                 e10for test
      qme=qe/me
      qmi=qi/mi
      c=.5
C Our finite difference equations imply delta_t = delta_x =
C delta_y = delta_z = 1. So c must satisfy the Courant condition.
C The bx,by and bz arrays are really records of c*Bx, c*By,
C c*Bz: this makes for e <-----> b symmetry in Maxwell's equations.
C In the particle arrays the ions are at the bottom, the electrons
C are stacked against the top, the total number not exceeding maxptl:
      maxptl=8192
C     ions=3700
C     lecs=4300
C These are sample values for the number of ions, "ions", and the number
C of electrons, "lecs". The initialiser (see below) may change them.
C The code treats unpaired electrons as having been initially
C dissociated from infinitely heavy ions which remain in situ.
C In general the  total information for particles will be somehwat
C larger than that for fields. The limit "8192" per component is
C used here for convenience of fitting the data into a segmented
C PC memory.
C For use in the boundary field calculation:
      rs=(1.-c)/(1.+c)
      tsq=.1666667
      ps=r*(1.-tsq)
      qs=.5*r*(1.+tsq)
C Data for smoothing: the currents fed into Maxwell's equations
C are smoothed by convolving with the sequence  .25, .5, .25 in
C each dimension. Generate the 27 weights ("smooth") and index
C displacements ("mooths"):
      n=1
      do 1 nz=-1,1
      do 1 ny=-1,1
      do 1 nx=-1,1
      smooth(n)=.015625*(2-nx*nx)*(2-ny*ny)*(2-nz*nz)
      mooths(n)=ix*nx+iy*ny+iz*nz
```



```
  1   n=n+1
C Initialise the particles: Place electrons in same locations as ions
C for zero initial net charge density. Keep particles 2 units away from
C the boundaries of the field domain. Here we populate the interior of
C a sphere (radius 7.75 and centered in the field domain)  uniformily
C with ion-electron pairs:
      rsq=60.
C Place ions within sphere on cubic grid:
CC    ions=0
CC    nzmax=sqrt(rsq)
CC    do 2 nz=-nzmax,nzmax
CC    nymax=sqrt(rsq-nz*nz)
CC    do 2 ny=-nymax,nymax
CC    nxmax=sqrt(rsq-nz*nz-ny*ny)
CC    do 2 nx=-nxmax,nxmax
CC    ions=ions+1
CC    x(ions)=11.+nx
CC    y(ions)=10.+ny
CC2   z(ions)=10.5+nz
C Place ions within sphere on interstitial grid (cube centers):
CC    nzmax=-1+2*ifix(.5+sqrt(rsq))
CC    do 3 nz=-nzmax,nzmax,2
CC    nymax=-1+2*ifix(.5+sqrt(rsq-.25*nz*nz))
CC    do 3 ny=-nymax,nymax,2
CC    nxmax=-1+2*ifix(.5+sqrt(rsq-.25*nz*nz-.25*ny*ny))
CC    do 3 nx=-nxmax,nxmax,2
CC    ions=ions+1
CC    x(ions)=11.+.5*nx
CC    y(ions)=10.+.5*ny
CC3   z(ions)=10.5+.5*nz
CC    open(6,file='con')
CC    write(6,97)ions
CC97  format(i6)
C As another example, fill one cell uniformily with 64 pairs:
            ions=0
            do 80 k=1,4
            do 80 j=1,4
            do 80 i=1,4
            ions=ions+1
            x(ions)=11.375+.25*i
            y(ions)=10.375+.25*j
  80        z(ions)=11.375+.25*k
C Put electrons in the same places:
      lecs=ions
      do 4 n=1,lecs
      x(n+maxptl-lecs)=x(n)
      y(n+maxptl-lecs)=y(n)
  4   z(n+maxptl-lecs)=z(n)
C Since "ions" is less than 4096 in these cases, there would be room
C for more particles. For instance, several hundred unpaired electrons
C could be added at the bottom of the electron array.
C Initialise velocities: these should not exceed  c  in magnitude!
C For thermal distributions, add four random numbers for each
C component and scale. In this test we use c for each component (too much!):
      do 85 n=1,ions
      u(n)=-0.5
      v(n)=-0.5
      w(n)=-0.5
      u(maxptl-lecs+n)=0.5
      v(maxptl-lecs+n)=0.5
  85  w(maxptl-lecs+n)=0.5
C Initialise the fields, typically to uniform components, such as
C just a uniform magnetic field parallel to the z-axis. Here we create
C linearly varying fields, the E-field curl-free as if static and both
C E and B divergence-free:
      do 5 k=1,mz
      do 5 j=1,my
      do 5 i=1,mx
      ex(i,j,k)= .001*(i-10.5)-.005*(j-10.0)-.004*(k-10.5)
```

```
      ey(i,j,k)=-.002*(j- 9.5)+.006*(k-10.5)+.010*(i-11.0)
      ez(i,j,k)= .001*(k-10.0)-.004*(i-11.0)-.003*(j-10.0)
      bx(i,j,k)= .003*(i-11.0)+.004*(j- 9.5)+.005*(k-10.0)
      by(i,j,k)=-.008*(j-10.0)-.010*(k-10.0)-.006*(i-10.5)
  5   bz(i,j,k)= .005*(k-10.5)+.003*(i-10.5)+.004*(j- 9.5)
C Initial fields, both electric and magnetic, must be divergence-free.
C Part of the Earth's magnetic field would be ok. If the Earth is included
C in the field domain, its magnetic dipole field is readily established
C by maintaining a steady ring current in the Earth's core.
C Optional random fields using a random number generator which assumes
C two-byte integer (modulo 2**16) arithmetic:
CC      lucky=12345
CC      scale=1./65536.
CC      do 90 k=1,mz
CC      do 90 j=1,my
CC      do 90 i=1,mx
CC      lucky=lucky*261
CC      ex(i,j,k)=lucky*scale
CC      lucky=lucky*261
CC      ey(i,j,k)=lucky*scale
CC      lucky=lucky*261
CC      ez(i,j,k)=lucky*scale
CC      lucky=lucky*261
CC      bx(i,j,k)=lucky*scale
CC      lucky=lucky*261
CC      by(i,j,k)=lucky*scale
CC      lucky=lucky*261
CC90    bz(i,j,k)=lucky*scale
C     Begin time stepping
      last=                   1                                    ! test
      nstep=1
C Before moving particles, the magnetic field is Maxwell-advanced
C by half a timestep:
  6   do 7 i=1,mx-1
      do 7 j=1,my-1
      do 7 k=1,mz-1
      bx(i,j,k)=bx(i,j,k) + (.5*c) *
     &         (ey(i,j,k+1)-ey(i,j,k)-ez(i,j+1,k)+ez(i,j,k))
      by(i,j,k)=by(i,j,k) + (.5*c) *
     &         (ez(i+1,j,k)-ez(i,j,k)-ex(i,j,k+1)+ex(i,j,k))
  7   bz(i,j,k)=bz(i,j,k) + (.5*c) *
     &         (ex(i,j+1,k)-ex(i,j,k)-ey(i+1,j,k)+ey(i,j,k))
C Now move ions:
      call mover(1,ions,qmi)
C and electrons:
      call mover(maxptl-lecs+1,maxptl,qme)
C The Maxwell-advance of the fields begins with another half-step
C advance of the magnetic field since for Maxwell's equations the
C B - information and the E - information must be staggered in time.
C In space, their information is also staggered. Here we show the
C locations where field components are recorded:
C
C
C
C               z
C               ^
C               |
C               |
C               |
C               *---------------Ey--------------*
C             /|                                |
C            / |                                |
C           /  |                                |
C         Ex   |                                |
C        /     |                                |
C       /      |                                |
C      /       |                                |
C     *        |                                |
C     |        |                                |
```



```
C     |        |                                |
C     |        |                                |
C     |       Ez             Bx                Ez
C     |        |                                |
C     |        |                                |
C     |        |                                |
C     |   By   |                                |
C     |        |                                |
C     |        |                                |
C     |        |                                |
C     |        |                                |
C    Ez        |                                |
C     |        |                                |
C     |        |                                |
C     |        |                                |
C     |        *---------------Ey----------------*-----> y
C     |      /                                 /
C     |     /                                  /
C     |    /                                   /
C     |   Ex             Bz                   Ex
C     |  /                                     /
C     | /                                      /
C     |/                                       /
C     *---------------Ey----------------*
C    /
C   /
C  /
C x
C
C
C     Ex(i,j,k) = value of Ex at x=i+.5, y=j,    z=k
C     Ey(i,j,k) = value of Ey at x=i,    y=j+.5, z=k
C     Ez(i,j,k) = value of Ez at x=i,    y=j,    z=k+.5
C
C     Bx(i,j,k) = value of Bx at x=i,    y=j+.5, z=k+.5
C     By(i,j,k) = value of By at x=i+.5, y=j,    z=k+.5
C     Bz(i,j,k) = value of Bz at x=i+.5, y=j+.5, z=k
C
C     Maxwell's laws are implemented (to central difference
C     accuracy) in the form:
C
C     Change of flux of B through a cell face
C         = - circulation of E around that face
C
C     Change of flux of E through a cell face of the offset grid
C         = circulation of B around that face - current through it
C
C Second half-advance of magnetic field:
      do 8 i=1,mx-1
      do 8 j=1,my-1
      do 8 k=1,mz-1
      bx(i,j,k)=bx(i,j,k) + (.5*c) *
     &         (ey(i,j,k+1)-ey(i,j,k)-ez(i,j+1,k)+ez(i,j,k))
      by(i,j,k)=by(i,j,k) + (.5*c) *
     &         (ez(i+1,j,k)-ez(i,j,k)-ex(i,j,k+1)+ex(i,j,k))
  8   bz(i,j,k)=bz(i,j,k) + (.5*c) *
     &         (ex(i,j+1,k)-ex(i,j,k)-ey(i+1,j,k)+ey(i,j,k))
C Front,right and top layers of B must be obtained from a special
C boundary routine based on Lindman's method:
      call surface(bx,by,bz,ex,ey,ez,ix,iy,iz,mx,my,mz,1)
      call surface(by,bz,bx,ey,ez,ex,iy,iz,ix,my,mz,mx,1)
      call surface(bz,bx,by,ez,ex,ey,iz,ix,iy,mz,mx,my,1)
      call edge(bx,ix,iy,iz,mx,my,mz,1)
      call edge(by,iy,iz,ix,my,mz,mx,1)
      call edge(bz,iz,ix,iy,mz,mx,my,1)
C Full advance of the electric field:
      do 9 i=2,mx
      do 9 j=2,my
      do 9 k=2,mz
      ex(i,j,k)=ex(i,j,k) + c *
```

```
     &          (by(i,j,k-1)-by(i,j,k)-bz(i,j-1,k)+bz(i,j,k))
      ey(i,j,k)=ey(i,j,k) + c *
     &          (bz(i-1,j,k)-bz(i,j,k)-bx(i,j,k-1)+bx(i,j,k))
  9   ez(i,j,k)=ez(i,j,k) + c *
     &          (bx(i,j-1,k)-bx(i,j,k)-by(i-1,j,k)+by(i,j,k))
C Boundary values of the E - field must be provided at rear, left
C and bottom faces of the field domain:
      call surface(ex,ey,ez,bx,by,bz,-ix,-iy,-iz,mx,my,mz,lot)
      call surface(ey,ez,ex,by,bz,bx,-iy,-iz,-ix,my,mz,mx,lot)
      call surface(ez,ex,ey,bz,bx,by,-iz,-ix,-iy,mz,mx,my,lot)
      call edge(ex,-ix,-iy,-iz,mx,my,mz,lot)
      call edge(ey,-iy,-iz,-ix,my,mz,mx,lot)
      call edge(ez,-iz,-ix,-iy,mz,mx,my,lot)
C The currents due to the movement of each charge q are applied to the
C E-M fields as decrements of E-flux through cell faces. The movement
C of particles which themselves cross cell boundaries has to be split
C into several separate moves, each only within one cell. Each of
C these moves contributes to flux across twelve faces.
C Ions and electrons are processed in two loops, changing the sign
C of the charge in-between. These loops cannot be vectorised:
C particles get processed on by one. Here is a good place to
C insert the "if" clauses for applying boundary conditions to
C the particles, such as reflection, periodicity, replacement
C by inward moving thermal or streaming particles, etc.
      q=qi
      do 10 n=1,ions
  10  call usplit(x(n),y(n),z(n),u(n),v(n),w(n))
      q=qe
      do 11 n=maxptl-lecs+1,maxptl
  11  call usplit(x(n),y(n),z(n),u(n),v(n),w(n))
C These "split" routines call the deposit routine.
C Countdown:
      nstep=nstep+1
      if (nstep.le.last) go to 6
CC    divergence check:
         do 75 k=2,mz
         do 75 j=2,my
         do 75 i=2,mx
  75     rho(i,j,k)=ex(i,j,k)-ex(i-1,j,k)+ey(i,j,k)-ey(i,j-1,k)
     &          +ez(i,j,k)-ez(i,j,k-1)
         write(*,'(16f5.1)')(((rho(i,j,k),i=8,15),j=7,14),k=8,15)
CC    end of divergence check
CC    write(*,'(6h ions:)')
CC    write(*,'(16f5.2)')(x(n),n=1,ions),(y(n),n=1,ions),(z(n),n=1,ions)
CC    write(*,'(6h lecs:)')
CC    write(*,'(16f5.2)')(x(n),n=maxptl-lecs+1,maxptl),
CC   &                   (y(n),n=maxptl-lecs+1,maxptl),
CC   &                   (z(n),n=maxptl-lecs+1,maxptl)
      stop
      end
C --------
      subroutine surface(bx,by,bz,ex,ey,ez,ix,iy,iz,mx,my,mz,m00)
      dimension bx(1),by(1),bz(1),ex(1),ey(1),ez(1)
      common /fields/ ex0(8192),ey0(8192),ez0(8000),c,q0,smooth(27),
     &r,p,q,dummy(160),bx0(8192),by0(8192),bz0(8192)
      m0=m00+iz*(mz-1)
      assign 5 to next
  6   m=m0
      do 2 j=1,my-1
      n=m
      do 1 i=1,mx-1
      bz(n)=bz(n)+.5*c*(ex(n+iy)-ex(n)-ey(n+ix)+ey(n))
  1   n=n+ix
  2   m=m+iy
      go to next (5,7)
  7   return
  5   m=m0+ix+iy
      do 4 j=2,my-1
      n=m
```



```
C  Directive specifically for the CRAY cft77 compiler:
cdir$ ivdep
      do 3 i=2,mx-1
      bx(n)=bx(n-iz)+r*(bx(n)-bx(n-iz))+p*(bz(n)-bz(n-ix))-q*(ez(n+iy)-
     &ez(n))-(q-c)*(ez(n+iy-iz)-ez(n-iz))-c*(ey(n)-ey(n-iz))
      by(n)=by(n-iz)+r*(by(n)-by(n-iz))+p*(bz(n)-bz(n-iy))+q*(ez(n+ix)-
     &ez(n))+(q-c)*(ez(n+ix-iz)-ez(n-iz))+c*(ex(n)-ex(n-iz))
  3   n=n+ix
  4   m=m+iy
      assign 7 to next
      go to 6
      end
C   ------------
      subroutine edge(bx,ix,iy,iz,mx,my,mz,m00)
      dimension bx(1)
      lx=ix*(mx-1)
      ly=iy*(my-1)
      lz=iz*(mz-1)
      n=m00+iy+lz
cdir$ ivdep
      do 1 j=2,my-1
      bx(n)=bx(n+ix)+bx(n-iz)-bx(n+ix-iz)
      bx(n+lx)=bx(n+lx-ix)+bx(n+lx-iz)-bx(n+lx-ix-iz)
  1   n=n+iy
      n=m00+ly+iz
cdir$ ivdep
      do 2 k=2,mz-1
      bx(n)=bx(n+ix)+bx(n-iy)-bx(n+ix-iy)
      bx(n+lx)=bx(n+lx-ix)+bx(n+lx-iy)-bx(n+lx-ix-iy)
  2   n=n+iz
      n=m00+ly+lz
cdir$ ivdep
      do 3 i=1,mx
      bx(n)=bx(n-iy)+bx(n-iz)-bx(n-iy-iz)
      bx(n-ly)=bx(n-ly+iy)+bx(n-ly-iz)-bx(n-ly+iy-iz)
      bx(n-lz)=bx(n-lz-iy)+bx(n-lz+iz)-bx(n-lz-iy+iz)
  3   n=n+ix
      return
      end
C--------
      subroutine mover(n1,n2,qm)
      common /partls/ x(8192),y(8192),z(8192),u(8192),v(8192),w(8192)
      common /fields/ ex(8192),ey(8192),ez(8192),
     &bx(8192),by(8192),bz(8192)
      dimension mooths(30)
      equivalence (ez(8001),c),(ez(8033),mooths(1)),(mooths(28),ix),
     &(mooths(29),iy),(mooths(30),iz)
      do 1 n=n1,n2
C Cell index & displacement in cell:
      i=x(n)
      dx=x(n)-i
      j=y(n)
      dy=y(n)-j
      k=z(n)
      dz=z(n)-k
      l=i+iy*(j-1)+iz*(k-1)
C (Field components are treated as single-indexed in this subroutine)
C Field interpolations are tri-linear (linear in x times linear in y
C times linear in z). This amounts to the 3-D generalisation of "area
C weighting". A modification of the simple linear interpolation formula
C         f(i+dx) = f(i) + dx * (f(i+1)-f(i))
C is needed since fields are recorded at half-integer locations in certain
C dimensions: see comments and illustration with the Maxwell part of this
C code. One then has first to interpolate from "midpoints" to "gridpoints"
C by averaging neighbors. Then one proceeds with normal interpolation.
C Combining these two steps leads to:
C   f at location i+dx  = half of f(i)+f(i-1) + dx*(f(i+1)-f(i-1))
C where now f(i) means f at location i+1/2. The halving is absorbed
C in the final scaling.
```

```
C   E-component interpolations:
      f=ex(l)+ex(l-ix)+dx*(ex(l+ix)-ex(l-ix))
      f=f+dy*(ex(l+iy)+ex(l-ix+iy)+dx*(ex(l+ix+iy)-ex(l-ix+iy))-f)
      g=ex(l+iz)+ex(l-ix+iz)+dx*(ex(l+ix+iz)-ex(l-ix+iz))
      g=g+dy*
     & (ex(l+iy+iz)+ex(l-ix+iy+iz)+dx*(ex(l+ix+iy+iz)-ex(l-ix+iy+iz))-g)
      ex0=(f+dz*(g-f))*(.25*qm)
C   ------------
      f=ey(l)+ey(l-iy)+dy*(ey(l+iy)-ey(l-iy))
      f=f+dz*(ey(l+iz)+ey(l-iy+iz)+dy*(ey(l+iy+iz)-ey(l-iy+iz))-f)
      g=ey(l+ix)+ey(l-iy+ix)+dy*(ey(l+iy+ix)-ey(l-iy+ix))
      g=g+dz*
     & (ey(l+iz+ix)+ey(l-iy+iz+ix)+dy*(ey(l+iy+iz+ix)-ey(l-iy+iz+ix))-g)
      ey0=(f+dx*(g-f))*(.25*qm)
C   ------------
      f=ez(l)+ez(l-iz)+dz*(ez(l+iz)-ez(l-iz))
      f=f+dx*(ez(l+ix)+ez(l-iz+ix)+dz*(ez(l+iz+ix)-ez(l-iz+ix))-f)
      g=ez(l+iy)+ez(l-iz+iy)+dz*(ez(l+iz+iy)-ez(l-iz+iy))
      g=g+dx*
     & (ez(l+ix+iy)+ez(l-iz+ix+iy)+dz*(ez(l+iz+ix+iy)-ez(l-iz+ix+iy))-g)
      ez0=(f+dy*(g-f))*(.25*qm)
C   ---------
C   B-component interpolations:
      f=bx(l-iy)+bx(l-iy-iz)+dz*(bx(l-iy+iz)-bx(l-iy-iz))
      f=bx(l)+bx(l-iz)+dz*(bx(l+iz)-bx(l-iz))+f+dy*
     & (bx(l+iy)+bx(l+iy-iz)+dz*(bx(l+iy+iz)-bx(l+iy-iz))-f)
      g=bx(l+ix-iy)+bx(l+ix-iy-iz)+dz*(bx(l+ix-iy+iz)-bx(l+ix-iy-iz))
      g=bx(l+ix)+bx(l+ix-iz)+dz*(bx(l+ix+iz)-bx(l+ix-iz))+g+dy*
     & (bx(l+ix+iy)+bx(l+ix+iy-iz)+dz*(bx(l+ix+iy+iz)-bx(l+ix+iy-iz))-g)
      bx0=(f+dx*(g-f))*(.125*qm/c)
C   ------------
      f=by(l-iz)+by(l-iz-ix)+dx*(by(l-iz+ix)-by(l-iz-ix))
      f=by(l)+by(l-ix)+dx*(by(l+ix)-by(l-ix))+f+dz*
     & (by(l+iz)+by(l+iz-ix)+dx*(by(l+iz+ix)-by(l+iz-ix))-f)
      g=by(l+iy-iz)+by(l+iy-iz-ix)+dx*(by(l+iy-iz+ix)-by(l+iy-iz-ix))
      g=by(l+iy)+by(l+iy-ix)+dx*(by(l+iy+ix)-by(l+iy-ix))+g+dz*
     & (by(l+iy+iz)+by(l+iy+iz-ix)+dx*(by(l+iy+iz+ix)-by(l+iy+iz-ix))-g)
      by0=(f+dy*(g-f))*(.125*qm/c)
C   ------------
      f=bz(l-ix)+bz(l-ix-iy)+dy*(bz(l-ix+iy)-bz(l-ix-iy))
      f=bz(l)+bz(l-iy)+dy*(bz(l+iy)-bz(l-iy))+f+dx*
     & (bz(l+ix)+bz(l+ix-iy)+dy*(bz(l+ix+iy)-bz(l+ix-iy))-f)
      g=bz(l+iz-ix)+bz(l+iz-ix-iy)+dy*(bz(l+iz-ix+iy)-bz(l+iz-ix-iy))
      g=bz(l+iz)+bz(l+iz-iy)+dy*(bz(l+iz+iy)-bz(l+iz-iy))+g+dx*
     & (bz(l+iz+ix)+bz(l+iz+ix-iy)+dy*(bz(l+iz+ix+iy)-bz(l+iz+ix-iy))-g)
      bz0=(f+dz*(g-f))*(.125*qm/c)
C   ---------
CC    write(*,'(6f11.8)')ex0,ey0,ez0,bx0,by0,bz0
C   First half electric acceleration:
      u0=u(n)+ex0
      v0=v(n)+ey0
      w0=w(n)+ez0
C   First half magnetic rotation:
      g=2./(1.+bx0*bx0+by0*by0+bz0*bz0)
      u1=(u0+v0*bz0-w0*by0)*g
      v1=(v0+w0*bx0-u0*bz0)*g
      w1=(w0+u0*by0-v0*bx0)*g
C   Second half mag. rot'n & el. acc'n, and position advance:
      u(n)=u0+v1*bz0-w1*by0+ex0
      x(n)=x(n)+u(n)
      v(n)=v0+w1*bx0-u1*bz0+ey0
      y(n)=y(n)+v(n)
      w(n)=w0+u1*by0-v1*bx0+ez0
  1   z(n)=z(n)+w(n)
      return
      end
C --------
      subroutine usplit(x,y,z,u,v,w)
      if(ifix(x).ne.ifix(x-u))go to 1
```



```
      call vsplit(x,y,z,u,v,w)
      return
  1   u1=x-.5*(1+ifix(x)+ifix(x-u))
      v1=v*(u1/u)
      w1=w*(u1/u)
      call vsplit(x,y,z,u1,v1,w1)
      call vsplit(x-u1,y-v1,z-w1,u-u1,v-v1,w-w1)
      return
      end
C --------
      subroutine vsplit(x,y,z,u,v,w)
      if(ifix(y).ne.ifix(y-v))go to 1
      call wsplit(x,y,z,u,v,w)
      return
  1   v1=y-.5*(1+ifix(y)+ifix(y-v))
      u1=u*(v1/v)
      w1=w*(v1/v)
      call wsplit(x,y,z,u1,v1,w1)
      call wsplit(x-u1,y-v1,z-w1,u-u1,v-v1,w-w1)
      return
      end
C --------
      subroutine wsplit(x,y,z,u,v,w)
      if(ifix(z).ne.ifix(z-w))go to 1
      call depsit(x,y,z,u,v,w)
      return
  1   w1=z-.5*(1+ifix(z)+ifix(z-w))
      u1=u*(w1/w)
      v1=v*(w1/w)
      call depsit(x,y,z,u1,v1,w1)
      call depsit(x-u1,y-v1,z-w1,u-u1,v-v1,w-w1)
      return
      end
C --------
      subroutine depsit(x,y,z,u,v,w)
      common /fields/ ex(8192),ey(8192),ez(8192),
     &bx(8192),by(8192),bz(8192)
      dimension sm(30),ms(30)
      equivalence (ez(8002),q),(ez(8003),sm(1)),
     &(ez(8033),ms(1)),(ms(28),ix),(ms(29),iy),(ms(30),iz)
C   cell indices of half-way point:
      i=x-.5*u
      j=y-.5*v
      k=z-.5*w
C   displacements in cell of half-way point:
      dx=x-.5*u-i
      dy=y-.5*v-j
      dz=z-.5*w-k
      i=i+iy*(j-1)+iz*(k-1)
C   current elements:
      qu=q*u
      qv=q*v
      qw=q*w
      delt=.08333333*qu*v*w
C  Directive specifically for the CRAY cft77 compiler:
cdir$ ivdep
C  This will make the compiler use the "gather-scatter" hardware.
      do 1 n=1,27
      ex(ms(n)+i+iy+iz)=ex(ms(n)+i+iy+iz)-sm(n)*(qu*dy*dz+delt)
      ex(ms(n)+i+iz)=ex(ms(n)+i+iz)-sm(n)*(qu*(1.-dy)*dz-delt)
      ex(ms(n)+i+iy)=ex(ms(n)+i+iy)-sm(n)*(qu*dy*(1.-dz)-delt)
      ex(ms(n)+i)=ex(ms(n)+i)-sm(n)*(qu*(1.-dy)*(1.-dz)+delt)
      ey(ms(n)+i+iz+ix)=ey(ms(n)+i+iz+ix)-sm(n)*(qv*dz*dx+delt)
      ey(ms(n)+i+ix)=ey(ms(n)+i+ix)-sm(n)*(qv*(1.-dz)*dx-delt)
      ey(ms(n)+i+iz)=ey(ms(n)+i+iz)-sm(n)*(qv*dz*(1.-dx)-delt)
      ey(ms(n)+i)=ey(ms(n)+i)-sm(n)*(qv*(1.-dz)*(1.-dx)+delt)
      ez(ms(n)+i+ix+iy)=ez(ms(n)+i+ix+iy)-sm(n)*(qw*dx*dy+delt)
      ez(ms(n)+i+iy)=ez(ms(n)+i+iy)-sm(n)*(qw*(1.-dx)*dy-delt)
      ez(ms(n)+i+ix)=ez(ms(n)+i+ix)-sm(n)*(qw*dx*(1.-dy)-delt)
```

```
  1   ez(ms(n)+i)=ez(ms(n)+i)-sm(n)*(qw*(1.-dx)*(1.-dy)+delt)
      return
      end

-------
```