UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

HELM3: A Hierarchical Element Method in Three Dimensions

Christopher R. Anderson

July 1990

CAM Report 90-16

Department of Mathematics
University of California, Los Angeles
Los Angeles, CA. 90024-1555

HELM3 : A HIERARCHICAL ELEMENT METHOD IN THREE DIMENSIONS

CHRISTOPHER R. ANDERSON*

* Department of Mathematics, UCLA, Los Angeles, California, 90024
Research Supported by ONR Contracts #N00014-86-K-0691, #N00014-86-K-0727 and NSF Grant DM586-57663

This report contains the listings for HELM3, a hierarchical element method for computing the potential induced by a collection of point or 'blob' charges in three dimensions. For a general description of the method, one should consult "An Implementation of the Fast Multipole Method Without Multipoles". This report also contains a sample test program. Unlike the two dimensional code, this version is not optimized, nor is automatic level selection included. These features will be added soon and the report updated.

```
      program h3test
c
      implicit real*8 (a-h,o-z)
c
c     Test program for  Helm3 - a three dimensional
c     heirarchical element method.
c
      real*8 xl(500),yl(500),zl(500),str(500)
      real*8 xtst(500),ytst(500),ztst(500),pot(500),potst(500)
      integer*4 ilink(500)
c
      real*8 rand
      open(13,file='diag')
c
c     initialize test particles
c
c
      write(*,10)
10    format(1x,' Enter Number of Particles : ')
      read(*,20) npart
      write(*,11)
11    format(1x,' Enter Order of Integration Formula : ')
      read(*,20) korder
      write(*,*)' Enter Level : '
      read(*,*) mxlev
      write(*,*) 'Enter x width : '
      read(*,*) xwid
      write(*,*) 'Enter y width : '
      read(*,*) ywid
      write(*,*) 'Enter z width : '
      read(*,*) zwid
20    format(i5)
21    format(f10.5)
c
c
c     generate the test vortices in the box
c
      delta=0.05d00
c
      xmin=0.0d00
      ymin=0.0d00
      zmin=0.0d00
      xmax=1.0d00
      ymax=1.0d00
      zmax=1.0d00
c
      do 100 i=1,npart
        xl(i)=rand(0.0)*(xwid)  + (xmax + xmin)/2.0
        yl(i)=rand(0.0)*(ywid)  + (ymax + ymin)/2.0
        zl(i)=rand(0.0)*(zwid)  + (zmax + zmin)/2.0
        str(i)=rand(0.0d00) - 0.5d00
100   continue
c
c     make the strength average the same
c
      avg=0.0d00
```



```
      do 110 i=1,npart
        avg=avg + str(i)
110   continue
c
c
      do 120 i=1,npart
      str(i)=(str(i) - avg/dble(npart)) + 1.0d00/dble(npart)
120   continue
c
c     set up test points
c
      ntst=npart
      do 130 i=1,ntst
         xtst(i)=xl(i)
         ytst(i)=yl(i)
         ztst(i)=zl(i)
130   continue
c
c     call the fast method
c
      call eltime(rvala)
      call helm3(xl,yl,zl,str,ilink,npart,xtst,ytst,ztst,
     *          ntst,pot,delta,korder,mxlev)
      call eltime(rvalb)
      write(*,*) ' Fast Time : ',rvalb-rvala
c
c     error checking first compute the interactions the slow way
c
c
      do 800 j=1,ntst
       xpos=xtst(j)
       ypos=ytst(j)
       zpos=ztst(j)
       potst(j)=0.0d00
       do 800 i=1,npart
       xprt=xl(i)
       yprt=yl(i)
       zprt=zl(i)
       strprt=str(i)
       potst(j)=potst(j)+dirpot(xpos,ypos,zpos,xprt,yprt,zprt,
     *                          strprt,delta)
800   continue
c
c
      ermax=0.0d00
      do 1000 j=1,ntst
      err=dabs(potst(j) - pot(j))
      ermax=dmax1(err,ermax)
1000  continue
      write(*,900) ermax
900   format(1x,' Error  : ',e15.6)
      pause
      stop
      end
c
```

```
      double precision function rand(r)
      implicit real*8(a-h,o-z)
      integer*4 ix0,ix1,ia0,ia1,ialma0,ic,iy0,iy1
      save ix0,ix1
      data ia1, ia0, ialma0 /1536, 1029, 507/
      data ic /1731/
      data ix1, ix0 /0, 0/
c
      if (r.lt.0.) go to 10
      if (r.gt.0.) go to 20
      iy0 = ia0*ix0
      iy1 = ia1*ix1 + ialma0*(ix0-ix1) + iy0
      iy0 = iy0 + ic
      ix0 = mod (iy0, 2048)
      iy1 = iy1 + (iy0-ix0)/2048
      ix1 = mod (iy1, 2048)
c
 10   rand = dble(ix1*2048 + ix0)
      rand = rand / 4194304.d00
      return
c
 20   ix1 = dmod(r,1.0d00)*4194304.0d00 + 0.50d00
      ix0 = mod (ix1, 2048)
      ix1 = (ix1-ix0)/2048
      go to 10
c
      end
c
```



```
      subroutine eltime(rval)
      implicit real*8 (a-h,o-z)
c
c     This subroutine gets elapsed time in seconds from
c     the start of program execution, and returns it as
c     a real variable.
c
      real*4 tarray(2)
c
c     Unix timing Call
c
      call etime(tarray)
      rval=dble(tarray(1))
      return
c
      end
c
```

```
      subroutine helm3(xl,yl,zl,str,ilink,npart,xtst,
     *  ytst,ztst,ntst,pot,delta,korder,mxlev)
      implicit real*8(a-h,o-z)
c
      real*8 xl(1),yl(1),zl(1),str(1)
      integer*4 ilink(1),npart,iterms
      real*8 xtst(1),ytst(1),ztst(1),pot(1)
c
c     helm3 - a Hierarchial Element Method in Three Dimensions
c
c     This version is unoptimized and does not do automatic
c     level selection.
c
c     May 19, 1990
c
c     xl,yl,zl = particle location vectors
c
c     str   = particle strength vector
c
c     ilink = link list vector the size of number of particles
c
c     xtst, ytst, ztst  = the evaluation point vectors
c
c     pot       = the potential at location (xtst,ytst)
c
c     mplpar.par is the parameter file for the method. In it the
c     data structure parameters are stored.
c
c
      include 'threed.par'
c
c     ipart = array whose (i,j,k) value is the address of the first particle
c             in the box (i,j,k)
c
c     ilink = vector whose kth element gives the next address of the particle
c             in the box at which particle k is located in. (-9 if there are
c             no more particles in the box.
c
c     ipole = nested array pointing to pole values. At any given level
c             ipole(i,j,k) points to the starting address where the pole value
c             located in the vector polval.
c
c     ipow =  nested array pointing to power values. At any given level
c             ipow(i,j,k) points to the starting address where the pole values
c             located in the vector powval.
c
c     iterms = number of fourier modes we want in the expansion of the Poisson
c              kernel. This is the same as the value of  p' in the original
c              Greengard - Rohklin implementation of the multipole method.
c
      integer*4 ipart(ngsiza)
      integer*4 ipole(ngsizb)
      integer*4 ipow(ngsizb)
c
      real*8 polval(npolsz)
```



```
      real*8 powval(npowsz)
c
c
      real*8 v(3,intsiz),wt(intsiz)
c
c     Determine limits of the computational domain and find maximum
c     level in each direction
c
      call boxsiz(xl,yl,zl,npart,xtst,ytst,ztst,ntst,
     *  mxlev,mxlevx,mxlevy,mxlevz,xmin,xmax,ymin,ymax,zmin,zmax)
c
c
      mfine=2**mxlevx
      nfine=2**mxlevy
      kfine=2**mxlevz
      write(*,*) mxlevx
      write(*,*) mxlevy
      write(*,*) mxlevz
      write(*,*) xmin,xmax
      write(*,*) ymin,ymax
      write(*,*) zmin,zmax
c
c     Get the nodes of the integration coefficients
c     and their weights. Iterms is returned with the number of points
c     in the integration formula. nexp is the recommended number
c     of terms in the Poisson kernel to keep when using this
c     order of integration. beta is the radius for the outer ring
c     expansions - typically large for low order integration formulas.
c
      call intval(korder,iterms,v,wt,nexp,beta)
c
c
      idepth=max0(mxlevx,mxlevy)
      idepth=max0(idepth,mxlevz)
c
c
      if(idepth.eq.1) then
      if(iwflag.ne.1) then
      call potex(xl,yl,zl,str,npart,xtst,ytst,ztst,
     *                  ntst,pot,delta)
      endif
      return
c
      endif
c
c
      call dwnpas(xl,yl,zl,str,npart,ilink,xmin,ymin,zmin,
     *  xmax,ymax,zmax,mxlevx,mxlevy,mxlevz,mfine,nfine,kfine,
     *  ipart(1),ipole(1),ngsizb,polval,npolsz,iterms,v,wt,nexp,beta)
c
      call uppas(xl,yl,zl,str,npart,ilink,xmin,ymin,zmin,
     *  xmax,ymax,zmax,mxlevx,mxlevy,mxlevz,mfine,nfine,kfine,
     *  ipart(1),ipole(1),ngsizb,polval,xtst,ytst,ztst,ntst,ipow,ngsizb,
     *  powval,npowsz,iterms,v,wt,nexp,beta)
c
c     Evaluate the potential from the local interactions and the power series
```

```
c
c
      call potevl(xl,yl,zl,str,ipart,ilink,mfine,nfine,kfine,
     *  xmin,ymin,zmin,xmax,ymax,zmax,xtst,ytst,ztst,ntst,
     *  ipow(l),powval,iterms,pot,delta,v,wt,nexp)
c
c     done
c
      return
      end
c
```



```
      subroutine boxsiz(xl,yl,zl,npart,xtst,ytst,ztst,ntst,
     *  mxlev,mxlevx,mxlevy,mxlevz,xmin,xmax,ymin,ymax,zmin,zmax)
      implicit real*8 (a-h,o-z)
c
c     This routine figures out the dimensions of the computational box
c     as well as determing the levels to be used in each direction
c
      real*8 xl(1),yl(1),zl(1),xtst(1),ytst(1),ztst(1)
c
      xmin=xl(1)
      ymin=yl(1)
      zmin=zl(1)
      xmax=xl(1)
      ymax=yl(1)
      zmax=zl(1)
      do 100 i=2,npart
        xmin=dmin1(xmin,xl(i))
        xmax=dmax1(xmax,xl(i))
        ymin=dmin1(ymin,yl(i))
        ymax=dmax1(ymax,yl(i))
        zmin=dmin1(zmin,zl(i))
        zmax=dmax1(zmax,zl(i))
100   continue
      do 200 i=1,ntst
        xmin=dmin1(xmin,xtst(i))
        xmax=dmax1(xmax,xtst(i))
        ymin=dmin1(ymin,ytst(i))
        ymax=dmax1(ymax,ytst(i))
        zmin=dmin1(zmin,ztst(i))
        zmax=dmax1(zmax,ztst(i))
200   continue
c
c     add a little padding
c
      dx=xmax-xmin
      dy=ymax-ymin
      dz=zmax-zmin
      xmin=xmin - (.001d00)*dx
      xmax=xmax + (.001d00)*dx
      ymin=ymin - (.001d00)*dy
      ymax=ymax + (.001d00)*dy
      zmin=zmin - (.001d00)*dz
      zmax=zmax + (.001d00)*dz
      dx=xmax-xmin
      dy=ymax-ymin
      dz=zmax-zmin
      if((dx.eq.0.0d00).and.(dy.eq.0.0d00).and.(dz.eq.0.0d00)) then
        write(12,250)
250     format(1x,' Computational Box of Zero Size ')
        stop
      endif
c
c     now scale the sides - the largest side remains invariant, while the
c     smaller sides are increased in order that the resulting region has
c     an aspect ration which is a power of 2
```

```
c
      if((dx.ge.dy).and.(dx.ge.dz)) then
      if(dy.ge.dz) then
       call rscale(dy,dx,ymin,ymax,mxlev,mxlevy,mxlevx)
       call rscale(dz,dy,zmin,zmax,mxlevy,mxlevz,mxdumy)
      else
       call rscale(dz,dx,zmin,zmax,mxlev,mxlevz,mxlevx)
       call rscale(dy,dz,ymin,ymax,mxlevz,mxlevy,mxdumy)
      endif
c
c
      elseif((dz.ge.dy).and.(dz.ge.dx)) then
      if(dy.ge.dx) then
       call rscale(dy,dz,ymin,ymax,mxlev,mxlevy,mxlevz)
       call rscale(dx,dy,xmin,xmax,mxlevy,mxlevx,mxdumy)
      else
       call rscale(dx,dz,xmin,xmax,mxlev,mxlevx,mxlevz)
       call rscale(dy,dx,ymin,ymax,mxlevx,mxlevy,mxdumy)
      endif
c
      elseif ((dy.ge.dx).and.(dy.ge.dz)) then
      if(dx.ge.dz) then
       call rscale(dx,dy,xmin,xmax,mxlev,mxlevx,mxlevy)
       call rscale(dz,dx,zmin,zmax,mxlevx,mxlevz,mxdumy)
      else
       call rscale(dz,dy,zmin,zmax,mxlev,mxlevz,mxlevy)
       call rscale(dx,dz,xmin,xmax,mxlevz,mxlevx,mxdumy)
      endif
c
      endif
c
      return
      end
c
```



```
      subroutine rscale(da,db,astrt,aend,mxlev,mxleva,mxlevb)
      implicit real*8 (a-h,o-z)
c
c     Given two sides of length alen and blen (alen <= blen ) this
c     routine scales the smaller side (side a) so that the resulting
c     sides are in a ratio which is a power of two. This routine also
c     figures out the maximum level of refinement for each side.
c
      ratio=da/db
      if(ratio .eq. 0.0d00) then
        mxlevb=mxlev
        mxleva=0
        ifine=2**mxlevb
        danew=db/dble(ifine)
      else
      levoff=Int(dlog(ratio)/dlog(0.5d00))
      mxlevb=mxlev
      mxleva=mxlevb-levoff
      if(mxleva.lt.0) then
       mxleva=0
       ifine=2**mxlevb
       danew=db/dble(ifine)
      else
       danew=((1.0d00/2.0d00)**levoff)*db
      endif
      endif
      diff=(danew-da)
      astrt=astrt-(diff/2.0d00)
      aend=aend+(diff/2.0d00)
      da=(aend-astrt)
      return
      end
c
```

```
      subroutine potevl(xl,yl,zl,str,ipart,ilink,mfine,nfine,kfine,
     *  xmin,ymin,zmin,xmax,ymax,zmax,
     *  xtst,ytst,ztst,ntst,ipow,powval,iterms,pot,delta,
     *  v,wt,nexp)
c
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the potential at the test particles
c     from the power expansion values in powval and the local particles
c     which are pointed to by ilink and ipart.
c
      real*8 xl(1),yl(1),zl(1),str(1)
      real*8 xtst(1),ytst(1),ztst(1),pot(1)
c
      integer*4 ipow(mfine,nfine,kfine),ipart(mfine,nfine,kfine)
      Integer*4 ilink(1)
      real*8 powval(1)
      real*8 v(3,iterms),wt(iterms)
c
c
      hx=(xmax-xmin)/dble(mfine)
      hy=(ymax-ymin)/dble(nfine)
      hz=(zmax-zmin)/dble(kfine)
      hmax=dmax1(hx,hy)
      hmax=dmax1(hmax,hz)
c
      do 1000 kkk=1,ntst
      pot(kkk)=0.0d00
      xpos=xtst(kkk)
      ypos=ytst(kkk)
      zpos=ztst(kkk)
      val=0.0d00
c
c     evaluate power expansion for the test point
c
      i=Int((xpos - xmin)/hx) + 1
      j=Int((ypos - ymin)/hy) + 1
      k=Int((zpos - zmin)/hz) + 1
      if(ipow(i,j,k) .lt. 0) then
       write(*,10)
10     format(1x,' Error in Pot. Evaluation : ')
       write(*,11) kkk
11     format(1x,' No power expansion for test particle : ',i5)
      endif
      iptr=ipow(i,j,k)
      xpow=(dble(i-1)*hx + (hx/2.0d00)) + xmin
      ypow=(dble(j-1)*hy + (hy/2.0d00)) + ymin
      zpow=(dble(k-1)*hz + (hz/2.0d00)) + zmin
      rpow=hmax/2.0d00
      val = powevl(xpos,ypos,zpos,iterms,powval(iptr),xpow,ypow,
     *  zpow,rpow,v,wt,nexp)
c
c     evaluate the potential due to well separated neighbors
c
```



```
      ibot=i-2
      jbot=j-2
      kbot=k-2
      itop=i+2
      jtop=j+2
      ktop=k+2
      ibot=max(ibot,1)
      jbot=max(jbot,1)
      kbot=max(kbot,1)
      itop=min(itop,mfine)
      jtop=min(jtop,nfine)
      ktop=min(ktop,kfine)
      do 100 L=ibot,itop
      do 100 M=jbot,jtop
      do 100 N=kbot,ktop
       if(ipart(L,M,N) .gt. 0) then
       nadd=ipart(L,M,N)
50     continue
       xprt=xl(nadd)
       yprt=yl(nadd)
       zprt=zl(nadd)
       strprt=str(nadd)
       val = val + dirpot(xpos,ypos,zpos,xprt,yprt,zprt,
     *                    strprt,delta)
       nadd=ilink(nadd)
       if(nadd .gt. 0 ) goto 50
      endif
100   continue
      pot(kkk)=val
1000  continue
      return
      end
c
```

```
      subroutine uppas(xl,yl,zl,str,npart,ilink,xmin,ymin,zmin,
     *  xmax,ymax,zmax,mxlevx,mxlevy,mxlevz,mfine,nfine,kfine,
     *  ipart,ipole,lennes,polval,xtst,ytst,ztst,ntst,ipow,lennes,
     *  powval,npowsz,iterms,v,wt,nexp,beta)
c
      implicit real*8(a-h,o-z)
c
c     Upward Pass - Poles known at all levels, now compute
c                   the local expansions
c
c
c
      real*8 xl(1),yl(1),zl(1),str(1)
      real*8 xtst(1),ytst(1),ztst(1)
c
      integer*4 ipart(mfine,nfine),ilink(1),ipole(1)
      integer*4 ipow(1)
      real*8 polval(1)
      real*8 powval(1)
      real*8 v(3,iterms),wt(iterms)
c
c
c     nested grid information arrays (maximum depth of 20)
c
      integer*4 npanx(20),npany(20),npanz(20),nptr(20)
      real*8 hpanx(20),hpany(20),hpanz(20)
c
c
      idepth=max0(mxlevx,mxlevy)
      idepth=max0(mxlevz,idepth)
c
c     set up the indicies for the power expansion pointers
c
      call  setptr(mxlevx,mxlevy,mxlevz,mxstor,npanx,npany,npanz,nptr)
      if(mxstor .gt. lennes) then
       write(*,120)
120    format(1x,'Error : Insufficient Space Allocated to IPOW ',/)
       write(*,121) mxstor,lennes
121    format(1x,i5,'  Needed  : ',i5,'  Allocated ')
       pause
       stop
      endif
c
c     Initialze the power value pointer arrays - set all entries
c     to -9 to indicate the dabsence of a multipole.
c
c
      do 130 level=1,idepth
        nx=npanx(level)
        ny=npany(level)
        nz=npanz(level)
        nstrt=nptr(level)
        call niner(nx,ny,nz,ipow(nstrt))
130   continue
c
```



```
c     set up mesh sizes
c
      hxstrt=(xmax-xmin)/dble(mfine)
      hystrt=(ymax-ymin)/dble(nfine)
      hzstrt=(zmax-zmin)/dble(kfine)
      hpanx(idepth)=hxstrt
      hpany(idepth)=hystrt
      hpanz(idepth)=hzstrt
      do 140 i=1,idepth-1
       level=idepth-i
       ix=min0(i,mxlevx)
       iy=min0(i,mxlevy)
       iz=min0(i,mxlevz)
       hpanx(level)=hxstrt*(2**ix)
       hpany(level)=hystrt*(2**iy)
       hpanz(level)=hzstrt*(2**iz)
140   continue
c
c
c     Allocate space and designate locations for pow values
c     iterms values for each representation are needed.
c
c     iptr - pointer to next available address location in the array powval
c
c     Finest Level - Use information from ipart
c
      level=idepth
      nx=npanx(level)
      ny=npany(level)
      nz=npanz(level)
      nstrt=nptr(level)
      iptr=1
      isize=iterms
      call powal(nx,ny,nz,ipow(nstrt),xmin,ymin,zmin,
     *  hxstrt,hystrt,hzstrt,xtst,ytst,ztst,ntst,iptr,isize)
c
c     Allocate on succesively coarser Levels
c
c     levela= level where we need allocation
c     levelb= previous level
c
c
      do 300 i=1,idepth-2
       levela=idepth-i
       nxa=npanx(levela)
       nya=npany(levela)
       nza=npanz(levela)
       nstrta=nptr(levela)
       levelb=levela+1
       nxb=npanx(levelb)
       nyb=npany(levelb)
       nzb=npanz(levelb)
       nstrtb=nptr(levelb)
       call alloc(nxa,nya,nza,ipow(nstrta),nxb,nyb,nzb,
     *  ipow(nstrtb),iptr,isize)
300   continue
```

```
c
c       Each non-negetive entry of ipow points to the address of
c       the array powval where the power values
c       for that grid node are going to be stored.
c
c       Check to see if there is enough space allocated
c
        nvals=iptr-1
        if(nvals.gt.npowsz) then
         write(*,320)
320      format(1x,' Not Enough Coefficient Storage (Power) ')
         write(*,321) nvals,npowsz
321      format(1x,' Needed : ',i5,' Allocated : ',i5)
         pause
         stop
        endif
c
c       Now form all of the required power expansions
c
c       Zero Coefficients of Multiple Storage Array
c
        do 400 i=1,nvals
          powval(i)=0.0d00
400     continue
c
c       Form power expansions for all the other levels
c
c
c
        do 410 level=2,idepth
        nxa=npanx(level)
        nya=npany(level)
        nza=npanz(level)
        nstrta=nptr(level)
        nxb=npanx(level-1)
        nyb=npany(level-1)
        nzb=npanz(level-1)
        nstrtb=nptr(level-1)
        hxa=hpanx(level)
        hya=hpany(level)
        hza=hpanz(level)
        hxb=hpanx(level-1)
        hyb=hpany(level-1)
        hzb=hpanz(level-1)
        call valpow(nxa,nya,nza,ipow(nstrta),ipole(nstrta),nxb,
     *  nyb,nzb,ipow(nstrtb),xmin,ymin,zmin,hxa,hya,hza,hxb,hyb,
     *  hzb,powval,polval,iterms,v,wt,nexp,beta)
410     continue
c
c
        return
        end
c
```



```
        subroutine  valpow(nxa,nya,nza,ipowa,ipole,nxb,nyb,nzb,
     *  ipowb,xmin,ymin,zmin,hxa,hya,hza,hxb,hyb,hzb,pow,pol,
     *  iterms,v,wt,nexp,beta)
        implicit real*8 (a-h,o-z)
c
c       This subroutine computes the power expansions from the parent
c       power expansions and the pole expansions which very well separted
c       contained under the parents well separted neighbors.
c
c       a= level where we need the expansions
c
c       b= previous level (coarser) at which the expansions have already been
c          obtained.
c
        integer*4 ipowa(nxa,nya,nza),ipole(nxa,nya,nza)
        integer*4 ipowb(nxb,nyb,nzb)
        real*8 pol(1),pow(1)
        real*8 v(3,iterms),wt(iterms)
c
        hmaxa=dmax1(hxa,hya)
        hmaxa=dmax1(hza,hmaxa)
        hmaxb=dmax1(hxb,hyb)
        hmaxb=dmax1(hzb,hmaxb)
c
c
c     Loop over coarser level power expansions. If there is a coarse level
c     expansion, then one of the children must need an expansion so compute it.
c
        do 1000 i=1,nxa
        do 1000 j=1,nya
        do 1000 k=1,nza
        if(ipowa(i,j,k) .gt. 0) then
        ind=int((float(i)-.5)/2.0) + 1
        jnd=int((float(j)-.5)/2.0) + 1
        knd=int((float(k)-.5)/2.0) + 1
          if(ipowb(ind,jnd,knd) .gt. 0 ) then
c
c         obtain values from parent
c
          xpar=(dble(ind-1)*hxb + (hxb/2.0d00)) + xmin
          ypar=(dble(jnd-1)*hyb + (hyb/2.0d00)) + ymin
          zpar=(dble(knd-1)*hzb + (hzb/2.0d00)) + zmin
          rpar=hmaxb/2.0d00
          ippar=ipowb(ind,jnd,knd)
          xchld=(dble(i-1)*hxa + (hxa/2.0d00)) + xmin
          ychld=(dble(j-1)*hya + (hya/2.0d00)) + ymin
          zchld=(dble(k-1)*hza + (hza/2.0d00)) + zmin
          rchld=hmaxa/2.0d00
          ipchld=ipowa(i,j,k)
          call powpow(xpar,ypar,zpar,rpar,pow(ippar),
     *     xchld,ychld,zchld,rchld,pow(ipchld),iterms,v,wt,nexp)
          endif
c
c         obtain values from nearby poles (but not neighbors)
c
```

```
          indbot=ind-2
          jndbot=jnd-2
          kndbot=knd-2
          ibot=(indbot*2 -1)
          jbot=(jndbot*2 -1)
          kbot=(kndbot*2 -1)
          itop=ibot+9
          jtop=jbot+9
          ktop=kbot+9
          ibot=max(ibot,1)
          jbot=max(jbot,1)
          kbot=max(kbot,1)
          itop=min(itop,nxa)
          jtop=min(jtop,nya)
          ktop=min(ktop,nza)
c
          do 200 L=ibot,itop
          do 200 M=jbot,jtop
          do 200 N=kbot,ktop
          Ld=abs(L-i)
          Md=abs(M-j)
          Nd=abs(N-k)
          if((Ld.ge.3).or.(Md.ge.3).or.(Nd.ge.3)) then
          if(ipole(L,M,N) .gt. 0) then
          xpole=(dble(L-1)*hxa + xmin) + (hxa/2.0d00)
          ypole=(dble(M-1)*hya + ymin) + (hya/2.0d00)
          zpole=(dble(N-1)*hza + zmin) + (hza/2.0d00)
          rpole=beta*hmaxa
          xchld=(dble(i-1)*hxa + (hxa/2.0d00)) + xmin
          ychld=(dble(j-1)*hya + (hya/2.0d00)) + ymin
          zchld=(dble(k-1)*hza + (hza/2.0d00)) + zmin
          rchld=hmaxa/2.0d00
          ipchld=ipowa(i,j,k)
          iplptr=ipole(L,M,N)
          call polpow(xpole,ypole,zpole,rpole,pol(iplptr),
     *    xchld,ychld,zchld,rchld,pow(ipchld),iterms,v,wt,nexp)
          endif
          endif
200       continue
        endif
1000    continue
        return
        end
c
```



```
        subroutine powpow(xpar,ypar,zpar,rpar,powb,
     *    xchld,ychld,zchld,rchld,powa,iterms,v,wt,nexp)
        implicit real*8 (a-h,o-z)
c
c       In this subroutine the power values located at the parent center, (xpar,
c       with radius rpar is evaluated at the points of a ring about a child with
c       center (xchild,ychild) with radius rchild
c
c       child  = a
c
c       parent = b
c
        real*8 powa(1),powb(1)
        real*8 v(3,iterms),wt(iterms)
c
        do 100 m=1,iterms
         xpos=rchld*v(1,m) + xchld
         ypos=rchld*v(2,m) + ychld
         zpos=rchld*v(3,m) + zchld
         powa(m)=powa(m) + powevl(xpos,ypos,zpos,iterms,powb,
     *                    xpar,ypar,zpar,rpar,v,wt,nexp)
100     continue
        return
        end
c
```

```
      subroutine polpow(xpole,ypole,zpole,rpole,pol,
     *          xpow,ypow,zpow,rpow,pow,iterms,v,wt,nexp)
      implicit real*8 (a-h,o-z)
c
c     In this subroutine the pole located at (xpole,ypole) with radius rpole
c     is evaluated at the points of a ring about (rpow,ypow) with radius rpow.
c
      real*8 pol(1),pow(1)
      real*8 v(3,iterms),wt(iterms)
c
      do 100 j=1,iterms
       xpos=rpow*v(1,j) + xpow
       ypos=rpow*v(2,j) + ypow
       zpos=rpow*v(3,j) + zpow
       pow(j)=pow(j) + polevl(xpos,ypos,zpos,iterms,pol,
     *                  xpole,ypole,zpole,rpole,v,wt,nexp)
100   continue
      return
      end
c
```



```
      function powevl(xpos,ypos,zpos,iterms,pow,xpow,ypow,zpow,
     *                  rpow,v,wt,nexp)
      implicit real*8(a-h,o-z)
      real*8 pow(1)
      real*8 v(3,iterms),wt(iterms)
c
c     This procedure evalutes the sphere expansion from
c     the iterms boundary values in pow.
c
c
      pi=3.14159265359d00
      xt=xpos-xpow
      yt=ypos-ypow
      zt=zpos-zpow
      rtrns=dsqrt(xt**2 + yt**2 + zt**2)
      if(rtrns.lt. 0.00000001d00) then
       xt=0.0d00
       yt=0.0d00
       zt=0.0d00
       else
       xt=xt/rtrns
       yt=yt/rtrns
       zt=zt/rtrns
      endif
      alpha=rtrns/rpow
c
c     evaluate the integral of the reduced Poisson kernel.
c
      val=0.0d00
      do 200 j=1,iterms
      xxx=(xt*v(1,j) + yt*v(2,j) + zt*v(3,j))
      val=val + powker(nexp,alpha,xxx)*pow(j)*wt(j)
200   continue
      powevl=val/(4.0d00*pi)
c
      return
      end
c
```

```
      function powker(n,r,x)
      implicit real*8 (a-h,o-z)
c
c     This function computes the reduced Poisson kernel of n terms
c     with radius r and x=cos(theta) for a harmonic function defined inside
c     a sphere. ('powker' = power series kernel)
c
      pnm2=1.0d00
      pnm1=x
c
c     first two terms
c
      sum=pnm2
      sum=sum + 3.0d00*r*pnm1
c
c     remaining terms
c
      do 100 i=2,n
      di=dble(i)
      pn=((2.d0*di - 1.d0)*x*pnm1 - (di-1.d0)*pnm2)/di
      sum=sum + dble(2*i +1)*(r**i)*pn
      pnm2=pnm1
      pnm1=pn
100   continue
      powker=sum
      return
      end
c
```



```
      subroutine powal(nx,ny,nz,ipow,xmin,ymin,zmin,
     *  hxstrt,hystrt,hzstrt,xtst,ytst,ztst,ntst,iptr,isize)
c
      implicit real*8(a-h,o-z)
c
c     This subroutine allocates space on the lowest level for the local
c     local expansions.  The rule is "If there is a test particle in the
c     box, then there should be a local expansion associated  with that box"
c
      integer*4 ipow(nx,ny,nz)
      real*8 xtst(1),ytst(1),ztst(1)
c
c     find out which boxes have particle in them
c
      do 10 i=1,ntst
       ind=Int((xtst(i) - xmin)/hxstrt ) + 1
       jnd=Int((ytst(i) - ymin)/hystrt ) + 1
       knd=Int((ztst(i) - zmin)/hzstrt ) + 1
       ipow(ind,jnd,knd)=1
10    continue
c
c     now allocate space
c
      do 20 i=1,nx
      do 20 j=1,ny
      do 20 k=1,nz
      if(ipow(i,j,k) .gt. 0) then
      ipow(i,j,k)=iptr
      iptr=iptr + isize
      endif
20    continue
      return
      end
c
```

```
      subroutine dwnpas(xl,yl,zl,str,npart,ilink,xmin,ymin,
     *  zmin,xmax,ymax,zmax,mxlevx,mxlevy,mxlevz,mfine,nfine,
     *  kfine,ipart,ipole,lennes,polval,npolsz,iterms,v,wt,nexp,beta)
c
      implicit real*8(a-h,o-z)
c
c     On the basis of the particles located at (xl,yl,zl) this subroutine
c     computes the pole ring approximations at all levels of a nested
c     set of boxes.
c
      real*8 xl(1),yl(1),zl(1),str(1)
      real*8 v(3,iterms),wt(iterms)
      integer*4 ipart(mfine,nfine,kfine),ilink(1),ipole(lennes)
      real*8 polval(npolsz)
c
c
c     nested grid information arrays (maximum depth of 20)
c
      integer*4 npanx(20),npany(20),npanz(20),nptr(20)
      real*8 hpanx(20),hpany(20),hpanz(20)
c
c
c
      idepth=max0(mxlevx,mxlevy)
      idepth=max0(mxlevz,idepth)
c
c     Initialize the link list arrays to keep track of the boxes.
c     Value -9 indicates the  end of a  link or the  dabsence of
c     any particles in a box.
c
      do 100 i=1,npart
        ilink(i)=-9
100   continue
      do 110 i=1,mfine
      do 110 j=1,nfine
      do 110 k=1,kfine
        ipart(i,j,k)=-9
110   continue
c
c     set up the indicies for the nested grids
c     and check to see if there is enough space
c
      call  setptr(mxlevx,mxlevy,mxlevz,mxstor,npanx,npany,
     *                                    npanz,nptr)
      if(mxstor .gt. lennes) then
       write(*,120)
120    format(1x,'Error : Insufficient Space Allocated to IPOLE ',/)
       write(*,121) mxstor,lennes
121    format(1x,i5,'  Needed  : ',i5,'  Allocated ')
       pause
       stop
      endif
c
c     Initialze the multipole coefficient  pointer arrays - set all entries
c     to -9 to indicate the absence of a multipole.
```



```
c
c
      do 130 level=1,idepth
        nx=npanx(level)
        ny=npany(level)
        nz=npanz(level)
        nstrt=nptr(level)
        call niner(nx,ny,nz,ipole(nstrt))
130   continue
c
c     Get mesh sizes figured out
c
      hxstrt=(xmax-xmin)/dble(mfine)
      hystrt=(ymax-ymin)/dble(nfine)
      hzstrt=(zmax-zmin)/dble(kfine)
      hpanx(idepth)=hxstrt
      hpany(idepth)=hystrt
      hpanz(idepth)=hzstrt
      do 140 i=1,idepth-1
       level=idepth-i
       ix=min0(i,mxlevx)
       iy=min0(i,mxlevy)
       iz=min0(i,mxlevz)
       hpanx(level)=hxstrt*(2**ix)
       hpany(level)=hystrt*(2**iy)
       hpanz(level)=hzstrt*(2**iz)
140   continue
c
c     Create the link list for the particles :
c
c
      do 250 i=1,npart
      xpos=xl(i)
      ypos=yl(i)
      zpos=zl(i)
      ind=Int((xpos-xmin)/hxstrt) + 1
      jnd=Int((ypos-ymin)/hystrt) + 1
      knd=Int((zpos-zmin)/hzstrt) + 1
      if(ipart(ind,jnd,knd) .lt. 0) then
        ipart(ind,jnd,knd)=i
      else
        iptr=ipart(ind,jnd,knd)
        ipart(ind,jnd,knd)=i
        ilink(i)=iptr
      endif
c
250   continue
c
c     If there is a non-neg. element in the array ipart, then that
c     elements value (say iadd) is the address of the first particle
c     in that box. The value of ilink(iadd) is the address of the
c     next particle in the box, etc.
c
c
c     Allocate space and designate locations for pole values
c     iterms  values for each representation are needed.
```

```
c
c       iptr = pointer to next available address location in the array polval
c
c       Finest Level - Use information from ipart
c
        if(idepth.gt.1) then
        level=idepth
        nx=npanx(level)
        ny=npany(level)
        nz=npanz(level)
        nstrt=nptr(level)
        iptr=1
        isize=iterms
        call polalc(nx,ny,nz,ipole(nstrt),mfine,nfine,kfine,
     *   ipart,iptr,isize)
        endif
c
c       Allocate on succesively coarser Levels
c
c       levela= level where we need allocation
c
c       levelb= previous level
c
        do 300 i=1,idepth-2
         levela=idepth-i
         nxa=npanx(levela)
         nya=npany(levela)
         nza=npanz(levela)
         nstrta=nptr(levela)
         levelb=levela+1
         nxb=npanx(levelb)
         nyb=npany(levelb)
         nzb=npanz(levelb)
         nstrtb=nptr(levelb)
         call alloc(nxa,nya,nza,ipole(nstrta),nxb,nyb,nzb,
     *   ipole(nstrtb),iptr,isize)
300     continue
c
c       Each non-negetive entry of ipole points to the address of
c       the array polval where the multipole coefficients
c       for that grid node are going to be stored.
c
c       Check to see if there is enough space allocated
c
        nvals=iptr-1
        if(nvals.gt.npolsz) then
         write(*,320)
320      format(1x,' Not Enough Coefficient Storage (Poles) ')
         write(*,321) nvals,npolsz
321      format(1x,' Needed : ',i5,' Allocated : ',i5)
         pause
         stop
        endif
c
c       Zero Coefficients of Multiple Storage Array
c
```



```
        do 400 i=1,nvals
          polval(i)=0.0d00
400     continue
c
c       Form multipole expansions on the finest level
c
c
        level=idepth
        nx=npanx(level)
        ny=npany(level)
        nz=npanz(level)
        nstrt=nptr(level)
        hx=hpanx(level)
        hy=hpany(level)
        hz=hpanz(level)
        call fstpol(nx,ny,nz,ipole(nstrt),mfine,nfine,
     *   kfine,ipart,ilink,xmin,ymin,zmin,hx,hy,hz,npart,
     *   xl,yl,zl,str,polval,iterms,v,beta)
c
c       Form multipole expansion on the coarser levels
c
c
        do 430 i=1,idepth-2
         levela=idepth-i
         nxa=npanx(levela)
         nya=npany(levela)
         nza=npanz(levela)
         nstrta=nptr(levela)
         levelb=levela+1
         nxb=npanx(levelb)
         nyb=npany(levelb)
         nzb=npanz(levelb)
         nstrtb=nptr(levelb)
         hxa=hpanx(levela)
         hya=hpany(levela)
         hza=hpanz(levela)
         hxb=hpanx(levelb)
         hyb=hpany(levelb)
         hzb=hpanz(levelb)
         call valpol(nxa,nya,nza,ipole(nstrta),nxb,nyb,nzb,
     *   ipole(nstrtb),xmin,ymin,zmin,hxa,hya,hza,hxb,hyb,hzb,
     *   polval,iterms,v,wt,nexp,beta)
430     continue
c
c       Downward Pass is Complete
c

        return
        end
c
```

```
      subroutine valpol(nxa,nya,nza,iua,nxb,nyb,nzb,iub,
     *  xmin,ymin,zmin,hxa,hya,hza,hxb,hyb,hzb,
     *  polval,iterms,v,wt,nexp,beta)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the pole values about the
c     centers of each box induced by the pole values of its children
c     a level = level at which the expansions are needed
c
c     b level = children
c
      integer*4 iua(nxa,nya,nza),iub(nxb,nyb,nzb)
      real*8 polval(1)
      real*8 v(3,iterms),wt(iterms)
c
c     Loop over all children.  Give the values of the children
c     to the parent
c
c
      hmaxa=dmax1(hxa,hya)
      hmaxa=dmax1(hmaxa,hza)
      rada=beta*hmaxa
      hmaxb=dmax1(hxb,hyb)
      hmaxb=dmax1(hmaxb,hzb)
      radb=beta*hmaxb
c
      do 100 ind=1,nxb
      do 100 jnd=1,nyb
      do 100 knd=1,nzb
      if(iub(ind,jnd,knd).gt.0) then
        i=Int((float(ind)-.5)/2.0) + 1
        j=Int((float(jnd)-.5)/2.0) + 1
        k=Int((float(knd)-.5)/2.0) + 1
        xapole=(dble(i-1)*hxa + xmin) + hxa/2.0d00
        yapole=(dble(j-1)*hya + ymin) + hya/2.0d00
        zapole=(dble(k-1)*hza + zmin) + hza/2.0d00
        istrta=iua(i,j,k)
        xbpole=(dble(ind-1)*hxb + xmin) + hxb/2.0d00
        ybpole=(dble(jnd-1)*hyb + ymin) + hyb/2.0d00
        zbpole=(dble(knd-1)*hzb + zmin) + hzb/2.0d00
        istrtb=iub(ind,jnd,knd)
        call polpol(polval(istrta),xapole,yapole,zapole,rada,
     *    polval(istrtb),xbpole,ybpole,zbpole,radb,
     *   iterms,v,wt,nexp)
      endif
100   continue
      return
      end
c
```



```
      subroutine polpol(pola,xapole,yapole,zapole,rada,
     *      polb,xbpole,ybpole,zbpole,radb,iterms,v,wt,nexp)
      implicit real*8(a-h,o-z)
      real*8 pola(1),polb(1)
      real*8 v(3,iterms),wt(iterms)
c
c     this subroutine evalutes the pole from the values in pole b
c
c     and adds them to the values in pole a
c
c     now add all the other values
c
      do 100 m=1,iterms
       xp=rada*v(1,m) + xapole
       yp=rada*v(2,m) + yapole
       zp=rada*v(3,m) + zapole
       pola(m)=pola(m) + polevl(xp,yp,zp,iterms,polb,
     *                 xbpole,ybpole,zbpole,radb,v,wt,nexp)
100   continue
      return
      end
c
```

```
      function polevl(xpos,ypos,zpos,iterms,pol,xpole,ypole,zpole,
     *                rpole,v,wt,nexp)
      implicit real*8(a-h,o-z)
      real*8 pol(1)
      real*8 v(3,iterms),wt(iterms)
c
c     This procedure evalutes the outer ring expansion from
c     the iterms boundary values in pol.
c
c
      pi=3.14159265359d00
      xt=xpos-xpole
      yt=ypos-ypole
      zt=zpos-zpole
      rtrns=dsqrt(xt**2 + yt**2 + zt**2)
      alpha=rtrns/rpole
c
c     evaluate the integral of the reduced Poisson kernel.
c
      val=0.0d00
      do 200 j=1,iterms
      xxx=(xt*v(1,j) + yt*v(2,j) + zt*v(3,j))/rtrns
      val=val + pkern(nexp,alpha,xxx)*pol(j)*wt(j)
200   continue
      polevl=val/(4.0d00*pi)
c
      return
      end
c
```



```
      function pkern(n,r,x)
      implicit real*8 (a-h,o-z)
c
c     This function computes the reduced Poisson kernel of n terms
c     with radius r and x=cos(theta)
c
      rr=(1.0/r)
c
      pnm2=1.0d00
      pnm1=x
c
c     first two terms
c
      sum=rr*pnm2
      sum=sum + 3.0d00*(rr**2)*pnm1
c
c     remaining terms
c
      do 100 i=2,n
      di=dble(i)
      pn=((2.d0*di - 1.d0)*x*pnm1 - (di-1.d0)*pnm2)/di
      sum=sum + dble(2*i +1)*(rr**(i+1))*pn
      pnm2=pnm1
      pnm1=pn
100   continue
      pkern=sum
      return
      end
c
```

```
      subroutine fstpol(nxa,nya,nza,ipole,nxb,nyb,nzb,
     *  ipart,ilink,xmin,ymin,zmin,hx,hy,hz,npart,xl,yl,zl,str,
     *  polval,iterms,v,beta)
c
      implicit real*8(a-h,o-z)
c
c     This subroutine computes the pole values about the
c     centers of the finest level boxes induced by the particles
c     in those boxes.
c
      integer*4 ipole(nxa,nya,nza),ipart(nxb,nyb,nzb)
      integer*4 ilink(1)
      real*8 xl(1),yl(1),zl(1),str(1)
      real*8 polval(1)
      real*8 v(3,iterms)
c
c     Loop over all boxes. If values are needed for a
c     pole reprensentation then compute them
c
      hmax=dmaxl(hx,hy)
      hmax=dmaxl(hmax,hz)
      rad=beta*hmax
c
c
      do 100 i=1,nxa
      do 100 j=1,nya
      do 100 k=1,nza
      if(ipole(i,j,k).gt.0) then
       xpole=(dble(i-1)*hx + xmin) + (hx/2.0d00)
       ypole=(dble(j-1)*hy + ymin) + (hy/2.0d00)
       zpole=(dble(k-1)*hz + zmin) + (hz/2.0d00)
c
c      evaluate values due to first charge
c
       istrt=ipole(i,j,k)
c
       ind=ipart(i,j,k)
       xp=xl(ind)
       yp=yl(ind)
       zp=zl(ind)
c
       do 25 m=1,iterms
        cx=rad*v(1,m) + xpole
        cy=rad*v(2,m) + ypole
        cz=rad*v(3,m) + zpole
        r=dsqrt((xp-cx)**2 + (yp-cy)**2 + (zp-cz)**2)
        jjj=istrt+(m-1)
        polval(jjj)=polval(jjj)+ str(ind)/r
25     continue
c
c      evaluate values due to remaining charges
c
50     continue
       ind=ilink(ind)
       if(ind.lt.0) goto 75
```



```
       xp=xl(ind)
       yp=yl(ind)
       zp=zl(ind)
c
       do 35 m=1,iterms
        cx=rad*v(1,m) + xpole
        cy=rad*v(2,m) + ypole
        cz=rad*v(3,m) + zpole
        r=dsqrt((xp-cx)**2 + (yp-cy)**2 + (zp-cz)**2)
        jjj=istrt+(m-1)
        polval(jjj)=polval(jjj)+ str(ind)/r
35     continue
       goto 50
75    continue
      endif
100   continue
      return
      end
c
```

```
      subroutine niner(nx,ny,nz,iu)
c
c     This subroutine initializes the pointer arrays
c     to -999.
c
      integer*4 iu(nx,ny,nz)
      do 10 i=1,nx
      do 10 j=1,ny
      do 10 k=1,nz
       iu(i,j,k)=-9
10    continue
      return
      end
c
```



```
      subroutine setptr(mxlevx,mxlevy,mxlevz,mxstor,npanx,
     *                  npany,npanz,nptr)
      integer*4 npanx(20),npany(20),npanz(20),nptr(20)
      integer*4 mxlevx,mxlevy,mxlevz,mxstor
c
c     This subroutine sets up the pointer structure for the
c     nested grids. Maximum number of grids is 20.
c
c     mxlevx => 2**mxlevx panels on the finest grid in the i direction
c
c     mxlevy => 2**mxlevy panels on the finest grid in the j direction
c
c     mxlevz +> 2**mxlevz panels on the finest grid in the k direction
c
c     mxstor => the number of points taken up by the nested grid
c
c     npanx(level) => i dimension of grid at level level.
c
c     npany(level) => j dimension of grid at level level.
c
c     npanz(level) => k dimension of grid at level level.
c
c     nptr(level)  => index of first component of grid at level level.
c
      idepth=max0(mxlevx,mxlevy)
      idepth=max0(mxlevz,idepth)
      level=idepth
      levx=mxlevx
      levy=mxlevy
      levz=mxlevz
      nptr(level)=1
      npanx(level)=2**levx
      npany(level)=2**levy
      npanz(level)=2**levz
      do 10 i=2,idepth
        level=level-1
        levx=levx-1
        levy=levy-1
        levz=levz-1
        istor=npanx(level+1)*npany(level+1)*npanz(level+1)
        nptr(level)=nptr(level+1) + istor
        if(levx.le.0) then
         npanx(level)=1
        else
        npanx(level)=2**levx
        endif
        if(levy.le.0) then
         npany(level)=1
        else
        npany(level)=2**levy
        endif
        if(levz.le.0) then
         npanz(level)=1
        else
        npanz(level)=2**levz
```

```
          endif
10      continue
        level=1
        istor= npanx(level)*npany(level)*npanz(level)
        mxstor=nptr(level) + istor
        return
        end
c
```



```
        subroutine alloc(nxa,nya,nza,iua,nxb,nyb,nzb,iub,
     *                                   iptr,isize)
c
c       This subroutine determines at which points of the grid
c       there will be a multipole expansion.
c
c       The rule used in the construction is "if a child of any
c       box has a multipole expansion in it, then that box will need
c       a multipole expansion"
c
c       iua array = current level needing allocation
c       iub array = previous level (finer level)
c       iptr    = pointer to location where the coefficient expansion
c                 starts.
        integer*4 iua(nxa,nya,nza),iub(nxb,nyb,nzb)
c
c       loop over children and set flag if an expansion is needed.
c
        do 10 i=1,nxb
        do 10 j=1,nyb
        do 10 k=1,nzb
         if(iub(i,j,k) .gt. 0 ) then
         ind=Int((i-.1)/2) + 1
         jnd=Int((j-.1)/2) + 1
         knd=Int((k-.1)/2) + 1
         iua(ind,jnd,knd)=9
         endif
10      continue
c
c       now allocate storage
c
        do 20 i=1,nxa
        do 20 j=1,nya
        do 20 k=1,nza
         if(iua(i,j,k) .gt. 0) then
         iua(i,j,k)=iptr
         iptr=iptr + isize
         endif
20      continue
        return
        end
c
```

```
      subroutine polalc(nxa,nya,nza,iua,nxb,nyb,nzb,iub,
     *                  iptr,isize)
c
c     This subroutine determines at which points of the
c     finest level grid there will be a multipole expansion
c
c     The rule used in the construction is "If there are any
c     particles in the box, then that box will need
c     a multipole expansion"
c
c     iua array = finest level grid pointer array
c     iub array = array which indicates the presence of particles in
c                 box..
c
c     iptr    = pointer to location where the coefficient expansion
c                 starts.
      integer*4 iua(nxa,nya,nza),iub(nxb,nyb,nzb)
      do 10 i=1,nxa
      do 10 j=1,nya
      do 10 k=1,nza
       iflag=0
       if(iub(i,j,k)     .gt. 0) then
         iua(i,j,k)=iptr
         iptr=iptr+(isize)
       endif
10    continue
      return
      end
c
```



```
      subroutine intval(korder,iterms,v,wt,nexp,beta)
      implicit real*8(a-h,o-z)
      include 'threed.par'
c
c     This routine computes the various integration points and
c     weights for the sphere.
c
c     korder is the order of the formula
c
c     v is the array of integration points
c
c     wt is the wts for the integration points
c
c     iterms is the number of integration points
c
c     nexp is the recommended number of terms retained in the
c     Poisson Kernel - usually equal to iterms/2
c
c     beta is the radius scale factor for the outer expansion -
c     low order formulas have a larger beta to prevent aliasing
c     errors
c
c
c
      dimension v(3,intsiz),wt(intsiz)
      dimension x(3,5)
c
c
c
      pi=3.14159265359d00
c
c
      if(korder .eq. 1) then
c
c     this generates one vector for test purposes
c
      iterms=1
      nexp=1
      v(1,1)=0.0d00
      v(2,1)=0.0d00
      v(3,1)=1.0d00
      return
c
c
      elseif(korder .eq. 5 ) then
c
c
c     korder = 5 Formula U3:5:1 5th order 12 point formula
c                due to Albrecht and Collatz  (pg. 296 Stroud).
c
      iterms=12
      nexp=2
      beta=10.0d00
c
      do 10 j=1,iterms
```

```
      do 10 i=1,3
        v(i,j)=0.0d00
10    continue

      r=dsqrt((5.0d00 + dsqrt(5.0d00))/10.0d00)
      s=dsqrt((5.0d00 - dsqrt(5.0d00))/10.0d00)

c
c     load up the coefficients
c
      zro=0.0d00
      call stuff(v(1,1),  r,  s,  zro)
      call stuff(v(1,2), -r,  s,  zro)
      call stuff(v(1,3),  r, -s,  zro)
      call stuff(v(1,4), -r, -s,  zro)
      call stuff(v(1,5),zro,  r,   s)
      call stuff(v(1,6),zro, -r,   s)
      call stuff(v(1,7),zro,  r,  -s)
      call stuff(v(1,8),zro, -r,  -s)
      call stuff(v(1,9),  s,zro,   r)
      call stuff(v(1,10), -s,zro,  r)
      call stuff(v(1,11),  s,zro, -r)
      call stuff(v(1,12), -s,zro, -r)
c
c
      do 20 i=1,iterms
      wt(i)=(4.0d00*pi)/12.0d00
20    continue
      return
      elseif (korder .eq. 7 ) then
c
c     korder = 7 Formula U3:7:1 7th order 24 point formula
c                due to Mc Laren (pg. 298 Stroud).
c
      iterms=24
      nexp=3
      beta=7.0d00
c
      r=0.866246818107820d00
      s=0.422518653761111d00
      t=0.266635401516705d00

c
c     load up the coefficients
c
      call stuff(v(1,1) , r, s ,t)
      call stuff(v(1,2) ,-r, t ,s)
      call stuff(v(1,3) , s, t, r)
      call stuff(v(1,4),-s, r, t)
      call stuff(v(1,5), t, r, s)
      call stuff(v(1,6),-t, s, r)
      do 110 j=1,6
       jj=6+j
       call stuff(v(1,jj),v(1,j),-v(2,j),-v(3,j))
110   continue
c
```



```
      do 120 j=1,6
       jj=12+j
       call stuff(v(1,jj),v(1,j),v(3,j),-v(2,j))
120   continue
c
      do 130 j=1,6
       jj=18+j
       call stuff(v(1,jj),v(1,j),-v(3,j),v(2,j))
130   continue
c
      do 150 i=1,24
      wt(i)=(4.0d00*pi)/24.0d00
150   continue
c
c     end of 7th order formula
c
      return
      elseif(korder.eq.9) then
c
c     korder =9 :  U3,9:1 a 32 point 9th order formula
c
      iterms=32
      nexp=4
      beta=5.0d00
c
      r=dsqrt((5.0d00 + dsqrt(5.0d00))/10.0d00)
      s=dsqrt((5.0d00 - dsqrt(5.0d00))/10.0d00)
      up=dsqrt((3.0d00 - dsqrt(5.0d00))/6.0d00)
      vp=dsqrt((3.0d00 + dsqrt(5.0d00))/6.0d00)
      t=1.0d0/dsqrt(3.0d00)
c
      do 152 j=1,iterms
      do 152 i=1,3
        v(i,j)=0.0d00
152   continue
      pi=3.14159265359d00
c
c     load up the coefficients
c
      zro=0.0d00
      call stuff(v(1,1),  r,  s,  zro)
      call stuff(v(1,2), -r,  s,  zro)
      call stuff(v(1,3),  r, -s,  zro)
      call stuff(v(1,4), -r, -s,  zro)
      call stuff(v(1,5),zro,  r,   s)
      call stuff(v(1,6),zro, -r,   s)
      call stuff(v(1,7),zro,  r,  -s)
      call stuff(v(1,8),zro, -r,  -s)
      call stuff(v(1,9),  s,zro,   r)
      call stuff(v(1,10), -s,zro,  r)
      call stuff(v(1,11),  s,zro, -r)
      call stuff(v(1,12), -s,zro, -r)
c
c
      call stuff(v(1,13),  up,  vp,  zro)
      call stuff(v(1,14), -up,  vp,  zro)
```

```
      call stuff(v(1,15),  up, -vp,  zro)
      call stuff(v(1,16), -up, -vp,  zro)
      call stuff(v(1,17),zro,  up,    vp)
      call stuff(v(1,18),zro, -up,    vp)
      call stuff(v(1,19),zro,  up,   -vp)
      call stuff(v(1,20),zro, -up,   -vp)
      call stuff(v(1,21),  vp,zro,    up)
      call stuff(v(1,22), -vp,zro,    up)
      call stuff(v(1,23),  vp,zro,   -up)
      call stuff(v(1,24), -vp,zro,   -up)
c
      call stuff(v(1,25),  t,  t,  t)
      call stuff(v(1,26), -t,  t,  t)
      call stuff(v(1,27),  t, -t,  t)
      call stuff(v(1,28), -t, -t,  t)
      call stuff(v(1,29),  t,  t, -t)
      call stuff(v(1,30), -t,  t, -t)
      call stuff(v(1,31),  t, -t, -t)
      call stuff(v(1,32), -t, -t, -t)
c
c
      do 160 i=1,12
      wt(i)=(4.0d00*pi)*(25.0d00/840.0d00)
160   continue
      do 161 i=13,32
      wt(i)=(4.0d00*pi)*(27.0d00/840.0d00)
161   continue
c
c     end of order 9 formula
c
      return
      elseif(korder .eq. 11) then
c
      iterms=50
      nexp=5
      beta=2.5d00
c
      do 170 j=1,iterms
      do 170 i=1,3
        v(i,j)=0.0d00
170   continue
      pi=3.14159265359d00
c
c     load up the coefficients
c
      x1=1.0d00
      x2=0.0d00
      x3=0.0d00
      jnd=0
      do 180 i=1,2
      x1=-x1
      do 180 j=1,3
      xt=x1
      x1=x3
      x3=x2
      x2=xt
```



```
      jnd=jnd + 1
      call stuff(v(1,jnd),x1,x2,x3)
      wt(jnd)=(4.0d00*pi)*(9216.0d00/725760.0d00)
180   continue
c
      x1=dsqrt(0.5d00)
      x2=x1
      x3=0.0d00
      do 183 i=1,2
      x1=-x1
      do 183 j=1,2
      x2=-x2
      do 183 k=1,3
      xt=x1
      x1=x3
      x3=x2
      x2=xt
      jnd=jnd+1
      call stuff(v(1,jnd),x1,x2,x3)
      wt(jnd)=(4.0d00*pi)*(16384.0d00/725760.0d00)
183   continue
c
      x1=dsqrt(1.0d00/3.0d00)
      x2=x1
      x3=x1
      do 185 i=1,2
      x1=-x1
      do 185 j=1,2
      x2=-x2
      do 185 k=1,2
      x3=-x3
      jnd=jnd+1
      call stuff(v(1,jnd),x1,x2,x3)
      wt(jnd)=(4.0d00*pi)*(15309.0d00/725760.0d00)
185   continue
c
      x1=dsqrt(1.0d00/11.0d00)
      x2=x1
      x3=3.0d00*x1
      do 190 i=1,2
      x1=-x1
      do 190 j=1,2
      x2=-x2
      do 190 k=1,2
      x3=-x3
      do 190 l=1,3
       xt=x1
       x1=x3
       x3=x2
       x2=xt
       jnd=jnd+1
       call stuff(v(1,jnd),x1,x2,x3)
       wt(jnd)=(4.0d00*pi)*(14641.0d00/725760.0d00)
190   continue
c
      return
```

```
c
c       end of 11th order formula
c
c
        elseif(korder .eq. 14 ) then
c
c
c       korder = 14 Formula U3:14:1 14th order 72 point formula
c                 due to Mc Laren (pg. 302 Stroud).
c
        DATA X/-.151108275d0, .155240600d0, .976251323d0,
     *          .315838353d0, .257049387d0, .913330032d0,
     *          .346307112d0, .666277790d0, .660412970d0,
     *         -.101808787d0, .817386065d0, .567022920d0,
     *         -.409228403d0, .501547712d0, .762221757d0/
c
        iterms=72
        nexp=7
        beta=2.0d00
c
        do 200 j=1,72
        do 200 i=1,3
          v(i,j)=0.0d00
200     continue
c
c
        jnd=0
c
        x1=0.525731112d0
        x2=0.850650808d0
        x3=0.0d00
        do 206 j=1,2
        x1=-x1
        do 204 k=1,2
        x2=-x2
        do 202 l=1,3
        xt=x1
        x1=x3
        x3=x2
        x2=xt
        jnd=jnd+1
        call stuff(v(1,jnd),x1,x2,x3)
        wt(jnd)=(4.0*pi)*(125.0d00/10080.0d00)
202     continue
204     continue
206     continue
c
        do 212 n=1,5
        x1=x(1,n)
        x2=x(2,n)
        x3=x(3,n)
        do 210 i=1,3
        xt=x1
        x1=x3
        x3=-x2
        x2=-xt
```



```
        do 208 j=1,3
        xt=x1
        x1=x3
        x3=x2
        x2=xt
        jnd=jnd+1
        call stuff(v(1,jnd),x1,x2,x3)
        wt(jnd)=(4.0*pi)*(143.0d00/10080.0d00)
208     continue
        x2=-x2
        x3=-x3
        jnd=jnd+1
        call stuff(v(1,jnd),x1,x2,x3)
        wt(jnd)=(4.0*pi)*(143.0d00/10080.0d00)
210     continue
212     continue
c
c       end of 14th order formula
c
        return
        else
        write(*,1000) korder
1000    format(1x,i5,' Is not an available Integration Formula ')
        pause
        stop
        endif
        return
        end
c
```

```
      subroutine stuff(v,a,b,c)
      implicit real*8(a-h,o-z)
      real*8 v(3,1)
      v(1,1)=a
      v(2,1)=b
      v(3,1)=c
      return
      end
c
```



```
      function dirpot(xpos,ypos,zpos,xprt,yprt,zprt,strprt,
     *                delta)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the direct potential of a particle at
c     (xprt,yprt,zprt) and strength strprt acting on a particle at (xpos,ypos,
c     The cutoff length delta is included in case the potential is truncated.
c
      rdist=dsqrt((xpos-xprt)**2+(ypos-yprt)**2+(zpos-zprt)**2)
      if(rdist .lt. delta) then
        dirpot=strprt*(1.0d00/delta)
      else
        dirpot=strprt*(1.0d00/rdist)
      endif
      return
      end
c
```

```
      subroutine potex(xl,yl,zl,str,npart,xtst,ytst,ztst,
     *                 ntst,pot,delta)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the direct potential of  particles at
c     (xl,yl,zl) and strength str acting
c     on  particles at (xtst,ytst,ztst).
c
c     The cutoff length delta is included in case the potential is truncated.
c
      real*8 xl(1),yl(1),zl(1),str(1),xtst(1),ytst(1),ztst(1),pot(1)
c
      do 50 j=1,ntst
       xpos=xtst(j)
       ypos=ytst(j)
       zpos=ztst(j)
       pot(j)=0.0d00
       do 50 i=1,npart
       xprt=xl(i)
       yprt=yl(i)
       zprt=zl(i)
       strprt=str(i)
       valp=0.0d00
       rdist=dsqrt((xpos-xprt)**2+(ypos-yprt)**2+(zpos-zprt)**2)
       if(rdist .lt. delta) then
        valp=strprt*(1.0d00/delta)
       else
        valp=strprt*(1.0d00/rdist)
       endif
       pot(j)=pot(j) + valp
50    continue
      return
      end
c
```