UCLA
COMPUTATIONAL AND APPLIED MATHEMATICS

HELM2: A Hierarchical Element Method in Two Dimensions

Christopher R. Anderson

July 1990

CAM Report 90-15

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

HELM2 : A HIERARCHICAL ELEMENT METHOD IN TWO 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 HELM2, a hierarchical element method for computing the potential induced by a collection of point or 'blob' charges, or the velocity field induced by a collection of point or 'blob' vortices. 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 as well as the program which is used to compute the timing constants.

The first step in getting the code running consists of compiling the program helm2 along with the test code h2test. The test code should be run and the accuracy of the fast method checked. (This is done by specifying the appropriate parameter for h2test.) A sufficient number of particles should be used so that the fast method is actually employed for the computation. The fact that the fast method was actually used can be determined by considering the output in the file "runout", and verifying that a level greater than one was selected for the computation.

Once an accuracy check has been accomplished then one needs to obtain the correct timing constants. The timing constants are set in the subroutine getcon - if the machine one uses happens to be listed there, then one should use the appropriate constant assignment. If the machine you are using is not listed there, then the timing program h2tmng.f should be compiled and run. This program generates two files - htime.con and htime.dat. The file htime.con should be copied into the subroutine getcon and used to set the timing constants in that routine. The file htime.dat provides data about the timing run. In order to run h2tmng.f one needs to specify a system CPU timing routine. This is done by modifying the routine eltime. The timing routine needed is one which computes the elapsed CPU time since the start of a program. With the timing constants set, one should then recompile helm2 and verify it's accuracy.

If one doesn't want to bother with the timing constants the program should still run acceptably. If the machine used happens to have relative timings similar to a SUN Sparstation (the default timings) then the correct level will be selected, but the estimate of the running time will be off by a constant factor. If the machine used has very different relative timings, then the program will still run, but probably inefficiently.

Notes on program implementation :

1) One component in the program is the calculation of the direct interaction between source and receiver particles in adjacent boxes at the finest level of discretization chosen. In order to enhance computational performance, the source and receiver particles are sorted. Particles in the same finest level box are put in consecutive array locations. This sorting requires the arrays linkr, links, tmps, and tmpr. The sorting is accomplished in the subroutines srcsrt, recsrt, srcput, recput and cmpput.

2) The user specifies a finest level of discretization. It is desirable to have the program pick a level which is less than this finest level specified - one is then certain that

increasing the levels will not reduce the work. If the level actually chosen is the finest level specified, then one might want to consider increasing the specified finest level. One can obtain the information about the level chosen from the file "runout".

3) The evaluation of the kernel for both the inner and outer ring approximations requires the computation of terms of the form

$$\cos(\theta - s) \quad \cos(M(\theta - s)) \quad \text{and} \quad \cos((M+1)(\theta - s))$$

where $\theta = j\,\delta\theta, \quad j = 0, 1, 2, \ldots$. To speed up the computation of the kernel (by avoiding the computation of these trigonometric functions) I use the trigonometric identity

$$\cos(t - s) = \cos(t)\cos(s) - \sin(t)\sin(s).$$

For the terms under consideration we have

$$\cos(\theta - s) = \cos(\theta)\cos(s) - \sin(\theta)\sin(s)$$

$$\cos(M(\theta - s)) = \cos(M\theta)\cos(Ms) - \sin(M\theta)\sin(Ms)$$

$$\cos((M+1)(\theta - s)) = \cos((M+1)\theta)\cos((M+1)s) - \sin((M+1)\theta)\sin((M+1)s).$$

In the program, the values of $\cos(\theta), \sin(\theta), \cos(M\theta), \sin(M\theta)$, and $\cos((M+1)\theta), \sin((M+1)\theta)$ are precomputed for $\theta = j\,\delta\theta, \quad j = 0, 1, 2, \ldots$. The evaluation of the kernel requires one computation of $\cos(s), \sin(s)$ etc. and the product of these values with the precomputed constants.

4) The computation for the storage of the nested pointer arrays is as follows. Assume that $n \geq m$ where n and m are the levels chosen in each direction. We have that the amount of elements of the nested grid is

$$S = (2^m 2^n + 2^{m-1}2^{n-1} + \cdots + 2^{m-(m-1)}2^{n-(m-1)}) + (2^{n-m} + 2^{(n-m)-1} + \cdots + 2)$$

The second part of the sum is just

$$\sum_{j=1}^{n-m} 2^j$$

while for the first part of the sum

$$= 2^{n-m}(2^{m+n-(n-m)} + 2^{m+n-(n-m)-2} + \cdots + 2^4 + 2^2)$$

$$= 2^{n-m}(2^{2m} + 2^{2m-2} + \cdots + 2^4 + 2^2)$$

$$= 2^{n-m}(4^m + 4^{m-1} + \cdots + 4) = 2^{n-m}(\sum_{j=1}^{m} 4^j)$$

Therefore,

$$S = 2^{n-m}(\sum_{j=1}^{m} 4^j) + \sum_{j=1}^{n-m} 2^j$$

or

$$S = \frac{4}{3}(4^m - 1) + 2(2^{n-m} - 1)$$

```
      subroutine helm2(xsrc,ysrc,str,nsrc,xrec,yrec,
     *  nrec,iwflag,pot,velx,vely,delta,iterms,icor,mxlev,tol,
     *  linkr,links,tmpr,tmps,ngsiza,ngsizb,nrealw,nintw,
     *  realwk,intwk,iprint)
c
      implicit real*8(a-h,o-z)
c-----------------------------------------------------------------------
c
c     Anderson's Hierarchical ELeMent method in 2 dimensions
c
c
c     March 31,1990
c-----------------------------------------------------------------------
c     This code uses  : DOUBLE PRECISION real variables
c                     : INTEGER*4 integers
c-----------------------------------------------------------------------
c     OUTPUT
c-----------------------------------------------------------------------
c
c     pot         = the potential at the evaluation points
c                   (should be of dimension >= nrec)
c
c  velx,vely      = the velocity at the evaluation points
c                   (should be of dimension >= nrec)
c
c     iterms      = the actual number of terms used in the expansion
c                   (equal to "p" in the original G. - R. method)
c-----------------------------------------------------------------------
c
c     OUTPUT ERROR FILE = UNIT 12 = errout
c
c     RUN TIME INFO     = UNIT 9  = runout (only if iprint .ne. 0)
c-----------------------------------------------------------------------
c
c
c     INPUT PARAMETERS
c
c-----------------------------------------------------------------------
c
c     xsrc,ysrc  = the source particle location vectors
c                  (should be of dimension >= nsrc)
c
c     str        = the source particle strength vector
c                  (should be of dimension >= nsrc)
c
c     nsrc       = number of source particles
c
c     xrec, yrec = the evaluation point vectors
c                  (should be of dimension >= nrec)
c
c     nrec       = the number of evaluation points
c
c     iwflag     = work flag =  0 : just potentials are calculated
c                            =  1 : just velocities are calculated
c                            =  any other value - both are calculated
c
```



```
c     tol        = tolerance desired for the computation
c
c     mxlev      = maximum number of levels scanned for hierarchical
c                  computation
c
c     icor       = correction distance (in box units at finest level)
c                  (set equal to 1 usually)
c
c    iprint      = flag for whether or not timing estimates get printed
c                  (set to 0 for no printing, non-zero for printing)
c
c-----------------------------------------------------------------------
c     INPUT STORAGE AND STORAGE PARAMETERS
c-----------------------------------------------------------------------
c
c  links          = integer*4 array used for sorting sources
c                   (should be of dimension >= nsrc)
c
c  linkr          = integer*4 array used for sorting receivers
c                   (should be of dimension >= nrec)
c
c   tmps          = real*8 array for sorting sources
c                   (should be of dimension >= nsrc)
c
c   tmpr          = real*8 array for sorting receivers
c                   (should be of dimension >= nrec)
c-----------------------------------------------------------------------
c     DATA STRUCTURE ARRAYS
c-----------------------------------------------------------------------
c
c      intwk  = integer work array - it should be dimensioned
c               of size nintw (as described below) in the calling
c               program.
c
c      realwk = real work array - it should be dimensioned
c               of size nrealw (as described below) in the calling
c               program.
c
c      ngsiza = number of points in pointer array at finest level
c
c      ngsizb = number of points in nested pointer array
c
c      nintw  = total integer work space allocated
c
c      nrealw = total real work space allocated
c
c-----------------------------------------------------------------------
c
c    HOW TO SELECT THE VALUES :
c
c       If levx = refinement level chosen in the x direction
c          levy = refinement level chosen in the y direction
c
c       ngsiza  needed is equal to 2^(levx) * 2^(levy)
c
c       CHOOSE   ngsiza > 2^(levx) * 2^(levy)
```

```
c
c        SET      ngsizb > (4/3)*ngsiza + (1/3)*2^(level+1)
c
c        where level=max(levx,levy)
c
c        SET      nintw > 4*ngsiza + 2*ngsizb
c
c        The worst case amount of storage for the ring approximation
c        values is 4*iterms*ngsizb - so
c
c        SET     nrealw on the order of 4*iterms*ngsizb
c
c        IF INSUFFICIENT SPACE IS ALLOCATED THE PROGRAM WILL GENERATE
c        AN ERROR MESSAGE AND THEN STOP. THE ERROR MESSAGE WILL
c        INDICATE THE STORAGE NEEDED.
c
c-----------------------------------------------------------------------
c
c       The Following Arrays are used in the program and the space
c       for these arrays is obtained from either realwk or intwk.
c
c       ipole = NESTED array pointing to outer ring approximation
c               values. At any given level ipole(i,j) points to the
c               starting address where the outer ring approximation
c               values are located in the vector polval.
c               Required size : ngsizb
c
c       ipow  = NESTED array pointing to inner ring approximation
c               values. At any given level ipow(i,j) points to the
c               starting address where the inner ring approximation
c               values are located in the vector powval.
c               Required size : ngsizb
c
c     polval =  vector in which the values needed for the
c               outer ring approximations are stored.
c               Required size : ngsizb*2*iterms (worst case)
c
c     powval =  vector in which the values of the inner ring approximations
c               are stored.
c               Required size : ngsizb*2*iterms (worst case)
c
c      isrc  =  the value in isrc(i,j) points to the first address in
c               the link list associated with the source particles in the
c               (i,j)th box.
c               The dimension should be larger than that needed for the
c               finest refinement.
c               Required size : ngsiza
c
c    isrcsz  =  the value in isrcsz is the number of source
c               particles in the (i,j)th box.
c               The dimension should be larger than that needed for the
c               finest refinement.
c               Required size : ngsiza
c
c      irec  =  the value in irec(i,j) points to the first address in
c               the link list associated with the source particles in the
```



```
c               (i,j)th box.
c               The dimension should be larger than that needed for the
c               finest refinement.
c               Required size : ngsiza
c
c    irecsz  =  the value in irecsz is the number of source
c               particles in the (i,j)th box.
c               The dimension should be larger than that needed for the
c               finest refinement.
c               Required size : ngsiza
c
c       particle and evaluation point vectors
c
        real*8 xsrc(1),ysrc(1),str(1)
        real*8 xrec(1),yrec(1)
        real*8 tmpr(1),tmps(1)
        real*8 pot(1),velx(1),vely(1)
c
        integer*4 linkr(1),links(1)
        integer*4 iwflag
c
c       work arrays
c
        real*8 realwk(1)
        integer*4 intwk(1)
c
c       local arrays, for timings and pre-computed constants
c
        real*8 totime(20)
        real*8 v(101,2),cn(101,3),sn(101,3)
c
c-----------------------------------------------------------------------
c
c       OPEN STATEMENTS
c-----------------------------------------------------------------------
c
        open(12,file='errout')
        if(iprint.eq. 1) then
         open(9,file='runout')
        endif
c
c-----------------------------------------------------------------------
c
c       Set up pointers into data structure arrays
c
        ipt1=1
        ipt2=ipt1 + ngsiza
        ipt3=ipt2 + ngsiza
        ipt4=ipt3 + ngsiza
        ipt5=ipt4 + ngsiza
        ipt6=ipt5 + ngsizb
        ilim=(ipt6 + ngsizb) - 1
        if(ilim .gt. nintw) then
        write(12,*)' Error : Insufficient Storage Space : '
        write(12,*)' Integer Work Array (intwk) Size : ',nintw
        write(12,*)' Needed Allocation (value of nintw)  : ',ilim
```

```
      write(*,*) 'Fatal Error : See errout '
      stop
      endif
c
      jpt1=1
      jpt2=jpt1 + (nrealw/2)
      npolsz=(nrealw/2)
      npowsz=(nrealw/2)
c
c     Relation between Work Arrays and
c     Arrays used in the Program
c
c     isrc(ngsiza)    = intwrk(ipt1:ipt2-1)
c     isrcsz(ngsiza)  = intwrk(ipt2:ipt3-1)
c     irec(ngsiza)    = intwrk(ipt3:ipt4-1)
c     irecsz(ngsiza)  = intwrk(ipt4:ipt5-1)
c     ipole(ngsizb)   = intwrk(ipt5:ipt6-1)
c     ipow(ngsizb)    = intwrk(ipt6:nintw)
c
c
c     powval(npolsz)  = realwk(jpt1:jpt2-1)
c     polval(npowsz)  = realwk(jpt2:nrealw)
c
c-----------------------------------------------------------------
c     BEGINNING OF PROGRAM
c-----------------------------------------------------------------
c
c     Estimate the number of terms needed to achieve the
c     approximate error of tol.
c
      iterms=0
      if(icor.eq. 1) then
      denom=dsqrt(2.0d00)/3.0d00
      zterms=dlog(tol)/dlog(denom)
      iterms=Int(zterms)
      endif
      if(icor.eq. 2) then
      denom=dsqrt(2.0d00)/5.0d00
      zterms=dlog(tol)/dlog(denom)
      iterms=Int(zterms)
      endif
      if(iprint.eq.1) then
      write(9,*) 'Number of terms : ',iterms
      write(9,*) '  '
      endif
c
c     Determine the limits of the computational domain  - i.e.
c     put a box around the domain to use in determining the
c     refinement. Also find the refinment possible in each direction
c     if the region is not a rectangle.
c
      call boxsiz(xsrc,ysrc,nsrc,xrec,yrec,nrec,
     *            mxlev,mxlevx,mxlevy,xmin,xmax,ymin,ymax)

c
      if(iprint.eq. 1) then
```



```
      write(9,*) 'Domain Limits Selected : '
      write(9,*) '    '
      write(9,5) xmin,xmax,ymin,ymax
5     format(1x,' xmin : ',f8.2,' xmax : ',f8.2,' ymin : ',
     *                                f8.2,' ymax : ',f8.2,//)
      write(9,*) '      '
      write(9,*) 'Max. Levels Selected For Timing Estimates : '
      write(9,*) '      '
      write(9,6) mxlevx,mxlevy
6     format(1x,' Max. X Level : ',i3,' Max. Y Level : ',i3,//)
      write(9,*) '  '
      write(9,*) 'Number of Sources   : ',nsrc
      write(9,*) 'Number of Recievers : ',nrec
      write(9,*) '  '
      endif
c
c     Perform timing estimates of the work on each level
c
      call esttim(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,iwflag,delta,
     * intwk(ipt5),intwk(ipt6),ngsizb,iterms,mxlevx,mxlevy,
     * xmin,ymin,xmax,ymax,icor,totime,iprint)
c
c     Select the optimal level
c
      idepth=max0(mxlevx,mxlevy)
c
c     To skip automatic level selection, uncomment
c     the following goto. This is usually only done in
c     order to check the timing estimates.
c
c
c     Goto 1999
c
      ibeg=1
      ttt=totime(ibeg)
c
c     scan the levels - use a level if it takes less work
c               and the mesh size is greater than delta.
c
      do 15 k=2,idepth
       if(ttt.gt.totime(k)) then
         mx=mxlevx-(idepth-k)
         my=mxlevy-(idepth-k)
         if(mx .lt. 0 ) mx=0
         if(my .lt. 0)  my=0
         mfine=2**mx
         h=(xmax-xmin)/dble(mfine)
         if(delta .le. 1.5d00*h) then
          ibeg=k
          ttt=totime(k)
         else
         write(12,*) 'Delta too large for Optimal Refinement : '
         write(12,*) 'Delta : ',delta, 'Box Limit : ',1.5*h
         endif
       endif
15    continue
```

```
c
      mxlevx=mxlevx-(idepth-ibeg)
      mxlevy=mxlevy-(idepth-ibeg)
      if(mxlevx .lt. 0 ) mxlevx=0
      if(mxlevy .lt. 0)  mxlevy=0
      idepth=max0(mxlevx,mxlevy)
c
c
      if(iprint .ne.0) then
      write(9,* ) 'Chosen Level of Refinement : '
      write(9,*) '           '
      write(9,*) ' X Level : ',mxlevx
      write(9,*) ' Y Level : ',mxlevy
      endif
c
1999  continue
c
      mfine=2**mxlevx
      nfine=2**mxlevy
c
c
c     If idepth = 1 - do the direct interaction
c
      if(idepth.eq.1) then
c
      if(iwflag.ne.1) then
      call potex(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,pot,delta)
      endif
c
      if(iwflag.ne.0) then
      call veloex(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,velx,
     *                  vely,delta)
      endif
c
      return
c
      endif
c
c     Sort the source and receiver particles :
c
c     The sort consists of combining into contiguous array
c     locations particles which are in the same box at
c     the finest level of discretization
c
      call srcsrt(xsrc,ysrc,str,tmps,nsrc,links,intwk(iptl),
     *            intwk(ipt2),mfine,nfine,xmin,ymin,xmax,ymax)
c
      call recsrt(xrec,yrec,tmpr,nrec,linkr,intwk(ipt3),
     *            intwk(ipt4),mfine,nfine,xmin,ymin,xmax,ymax)
c
c     compute the pre-computed constants for use in the kernel
c
      call makcon(iterms,v,cn,sn)
c
c
      call dwnpas(xsrc,ysrc,str,nsrc,xmin,ymin,xmax,ymax,
```



```
     *  mxlevx,mxlevy,mfine,nfine,intwk(iptl),intwk(ipt2),
     *  intwk(ipt5),ngsizb,realwk(jptl),npolsz,iterms,v,cn,sn)
c
c
      call uppas(xsrc,ysrc,str,nsrc,xmin,ymin,xmax,ymax,
     *  mxlevx,mxlevy,mfine,nfine,intwk(ipt5),ngsizb,realwk(jptl),
     *  npolsz,iterms,xrec,yrec,nrec,intwk(ipt6),realwk(jpt2),
     *  npowsz,icor,intwk(ipt3),intwk(ipt4),v,cn,sn)
c
c     Evaluate the potential from the local interactions
c     and the power series
      if(iwflag.ne.1) then
      call potevl(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     *  pot,intwk(ipt6),realwk(jpt2),iterms,tol,
     *  xmin,xmax,ymin,ymax,icor,delta,mfine,nfine,
     *  intwk(iptl),intwk(ipt2),intwk(ipt3),intwk(ipt4),
     *  v,cn,sn)
c
      endif
c
c
c     Evaluate the velocity from the local
c     interactions and the inner ring approximations
c
      if(iwflag.ne.0) then
      call velevl(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     *  velx,vely,intwk(ipt6),realwk(jpt2),iterms,tol,
     *  xmin,xmax,ymin,ymax,icor,delta,mfine,nfine,
     *  intwk(iptl),intwk(ipt2),intwk(ipt3),intwk(ipt4),
     *  v,cn,sn)
      endif
c
c
c     put the particles back in their original order
c
c
      call srcput(xsrc,ysrc,str,tmps,nsrc,links)
      call recput(xrec,yrec,tmpr,nrec,linkr)
c
c     put the computed quantities back in their original order
c
      if(iwflag.ne.1) then
       call cmpput(pot,tmpr,nrec,linkr)
      endif
c
      if(iwflag.ne.0) then
       call cmpput(velx,tmpr,nrec,linkr)
       call cmpput(vely,tmpr,nrec,linkr)
      endif
c
c     done
c
      return
      end
c
```

```
      subroutine getcon(tcon,scale)
      implicit real*8(a-h,o-z)
      real*8 tcon(10,2)
c
c     In this subroutine the timing constants are
c     initialized
c
c     Assignment Initialization - you should  uncomment
c     the  correct block to get the appropriate constants.
c
c     If you use the wrong constants - the program will still
c     function properly as long as the relative timings for
c     the operations on the machine in use are similar to that
c     for one of the timings given below.
c
c
c      SUN Sparcstation :
c
       scale=  0.100000D+04
       tcon(1,1)=  0.158189E-01
       tcon(1,2)=  0.202309E-03
       tcon(2,1)=  0.198674E-01
       tcon(2,2)=  0.324907E-02
       tcon(3,1)=  0.197324E-01
       tcon(3,2)=  0.313568E-02
       tcon(4,1)=  0.196765E-01
       tcon(4,2)=  0.276008E-02
       tcon(5,1)=  0.518148E-01
       tcon(5,2)=  0.0
       tcon(6,1)=  0.452239E-01
       tcon(6,2)=  0.455900E-02
       tcon(7,1)=  0.235182E-01
       tcon(7,2)=  0.415860E-02
       tcon(8,1)=  0.539492E-01
       tcon(8,2)=  0.538150E-02
c
c     On An Alliant FX-80 with a single CE
c
      goto 100
      scale=  0.100000D+04
      tcon(1,1)=  0.215534E-01
      tcon(1,2)=  0.153142E-02
      tcon(2,1)=  0.210423E-01
      tcon(2,2)=  0.148380E-02
      tcon(3,1)=  0.202719E-01
      tcon(3,2)=  0.171566E-02
      tcon(4,1)=  0.212157E-01
      tcon(4,2)=  0.115326E-02
      tcon(5,1)=  0.230024E-01
      tcon(5,2)=  0.
      tcon(6,1)=  0.253528E-01
      tcon(6,2)=  0.239412E-03
      tcon(7,1)=  0.208908E-01
      tcon(7,2)=  0.765751E-02
      tcon(8,1)=  0.662837E-01
```



```
      tcon(8,2)=  0.690708E-02
c
100   continue
c
      return
      end
c
```

```
      subroutine boxsiz(xl,yl,npart,xtst,ytst,ntst,
     *  mxlev,mxlevx,mxlevy,xmin,xmax,ymin,ymax)
      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),xtst(1),ytst(1)
c
      xmin=xl(1)
      ymin=yl(1)
      xmax=xl(1)
      ymax=yl(1)
      do 100 i=2,npart
        xmin=dminl(xmin,xl(i))
        xmax=dmaxl(xmax,xl(i))
        ymin=dminl(ymin,yl(i))
        ymax=dmaxl(ymax,yl(i))
100   continue
      do 200 i=1,ntst
        xmin=dminl(xmin,xtst(i))
        xmax=dmaxl(xmax,xtst(i))
        ymin=dminl(ymin,ytst(i))
        ymax=dmaxl(ymax,ytst(i))
200   continue
c
c     add a little padding
c
      dx=xmax-xmin
      dy=ymax-ymin
      xmin=xmin - (.001d00)*dx
      xmax=xmax + (.001d00)*dx
      ymin=ymin - (.001d00)*dy
      ymax=ymax + (.001d00)*dy
      dx=xmax-xmin
      dy=ymax-ymin
      if((dx.eq.0.0d00).and.(dy.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 ratio which is a power of 2
c
      if(dx.ge.dy) then
      call rscale(dy,dx,ymin,ymax,mxlev,mxlevy,mxlevx)
      else
      call rscale(dx,dy,xmin,xmax,mxlev,mxlevx,mxlevy)
      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 da and db (da <= db ) 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 makcon(iterms,v,cn,sn)
      implicit real*8 (a-h,o-z)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c     This subroutine precomputes the integration points
c     for the Poisson kernel (in v), and certain constants
c     used in the evaluation of the integral of Poisson Kernel.
c
      pi=3.14159265359d00
      pi2=2.0d00*pi
      itot=2*iterms + 1
      dtheta=pi2/dble(itot)
      nn=(itot-1)/2
c
c     Integration Points
c
      do 10 i=1,itot
       v(i,1)=dcos(dble(i-1)*dtheta)
       v(i,2)=dsin(dble(i-1)*dtheta)
10    continue
c
c     Constants for the Modified Poisson Kernel evaluation
c
      do 20 i=1,itot
       cn(i,1)=dcos(dble(i-1)*dtheta)
       cn(i,2)=dcos(dble(i-1)*dble(nn)*dtheta)
       cn(i,3)=dcos(dble(i-1)*dble(nn+1)*dtheta)
       sn(i,1)=dsin(dble(i-1)*dtheta)
       sn(i,2)=dsin(dble(i-1)*dble(nn)*dtheta)
       sn(i,3)=dsin(dble(i-1)*dble(nn+1)*dtheta)
20    continue
      return
      end
c
```



```
      subroutine dwnpas(xl,yl,str,npart,xmin,ymin,xmax,ymax,
     *  mxlevx,mxlevy,mfine,nfine,isrc,isrcsz,ipole,ngsizb,
     *  polval,npolsz,iterms,v,cn,sn)
c
      implicit real*8(a-h,o-z)
c
c     On the basis of the particles located at (xl,yl) this subroutine
c     computes the outer ring approximations at all levels of a nested
c     set of boxes.
c
      real*8 xl(1),yl(1),str(1)
      integer*4 isrc(mfine,nfine),isrcsz(mfine,nfine)
      integer*4 ipole(1)
      real*8 polval(1)
c
c
c     nested grid information arrays (maximum depth of 20)
c
      integer*4 npanx(20),npany(20),nptr(20)
      real*8 hpanx(20),hpany(20)
c
c     integration and constant arrays
c
      real*8 v(101,2),cn(101,3),sn(101,3)
c
      idepth=max0(mxlevx,mxlevy)
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,mxstor,npanx,npany,nptr)
c
      if(mxstor .gt. ngsizb) then
       write(12,*) 'Error : Insufficient Integer Wrk. Space : '
       write(12,*) 'Storage Needed (value of ngsizb) : ',mxstor
       write(12,*) 'Storage Allocated (value of ngsizb) :',ngsizb
       write(*,*) 'Fatal Error : See errout '
      stop
      endif
c
c     Initialze the ring approx. pointer arrays - set all entries
c     to -9 to indicate the absence of a ring approximation.
c
c
      do 130 level=1,idepth
        nx=npanx(level)
        ny=npany(level)
        nstrt=nptr(level)
        call niner(nx,ny,ipole(nstrt))
130   continue
c
c     get mesh sizes figured out
c
c
      hxstrt=(xmax-xmin)/dble(mfine)
```

```
      hystrt=(ymax-ymin)/dble(nfine)
      hpanx(idepth)=hxstrt
      hpany(idepth)=hystrt
      do 140 i=1,idepth-1
       level=idepth-i
       ix=min0(i,mxlevx)
       iy=min0(i,mxlevy)
       hpanx(level)=hxstrt*(2**ix)
       hpany(level)=hystrt*(2**iy)
140   continue
c
c     Allocate space and designate locations for outer ring values
c     (2*iterms+2)  values for each representation are needed.
c
c     The first 2*iterms +1 values are the potential values on
c     the ring about the center.
c     The 2*iterms + 2 value is the strength of the log term.
c
c     iptr = pointer to next available address location in the array polval
c
c     Finest level - use information from isrcsz
c
      if(idepth.gt.1) then
       level=idepth
       nstrt=nptr(level)
       iptr=1
       isize=(2*iterms)+2
       call fstalc(mfine,nfine,ipole(nstrt),isrcsz,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)
       nstrta=nptr(levela)
       levelb=levela+1
       nxb=npanx(levelb)
       nyb=npany(levelb)
       nstrtb=nptr(levelb)
       call alloc(nxa,nya,ipole(nstrta),nxb,nyb,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 outer ring approximation 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.npolsz) then
       write(12,*) 'Error Insufficient Real Wrk. Storage :'
       write(12,*) 'Storage Allocated  : ',2*npolsz
       write(12,*) 'Storage Necessary (value of nrealw) : ',2*nvals
       write(*,*) 'Fatal Error : See errout '
      stop
      endif
c
c     Zero Coefficients of Storage Array
c
      do 400 i=1,nvals
        polval(i)=0.0d00
400   continue
c
c     Form outer ring approx. on the finest level
c
      level=idepth
      nx=npanx(level)
      ny=npany(level)
      nstrt=nptr(level)
      hx=hpanx(level)
      hy=hpany(level)
      call fstpol(nx,ny,ipole(nstrt),mfine,nfine,isrc,isrcsz,
     * xmin,ymin,hx,hy,npart,xl,yl,str,polval,iterms,v)
c
c     Form outer ring approx. on the coarser levels
c
      do 430 i=1,idepth-2
       levela=idepth-i
       nxa=npanx(levela)
       nya=npany(levela)
       nstrta=nptr(levela)
       hxa=hpanx(levela)
       hya=hpany(levela)
       levelb=levela+1
       nxb=npanx(levelb)
       nyb=npany(levelb)
       hxb=hpanx(levelb)
       hyb=hpany(levelb)
       nstrtb=nptr(levelb)
       call mrgpol(nxa,nya,ipole(nstrta),nxb,nyb,ipole(nstrtb),
     *      xmin,ymin,hxa,hya,hxb,hyb,polval,iterms,v,cn,sn)
430   continue
c
c     Downward Pass is Complete
c
c
      return
      end
c
```

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



```
      subroutine setptr(mxlevx,mxlevy,mxstor,npanx,npany,nptr)
      integer*4 npanx(20),npany(20),nptr(20)
      integer*4 mxlevx,mxlevy,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     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     nptr(level)  => index of first component of grid at level level.
c
      idepth=max0(mxlevx,mxlevy)
      level=idepth
      levx=mxlevx
      levy=mxlevy
      nptr(level)=1
      npanx(level)=2**levx
      npany(level)=2**levy
      do 10 i=2,idepth
        level=level-1
        levx=levx-1
        levy=levy-1
        nptr(level)=nptr(level+1) + npanx(level+1)*npany(level+1)
        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
10    continue
      level=1
      mxstor=nptr(level) + npanx(level)*npany(level)
      return
      end
c
```

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



```
      subroutine fstalc(mfine,nfine,iua,iub,iptr,isize)
c
c     This subroutine determines at which points of the
c     finest level grid there will be outer ring approximations
c     and inner ring approximations.
c
c     The rule used in the construction is "If there are any
c     particles in the box, then an approximation will be required.''
c
c     iua  array = finest level grid pointer array
c
c     iub array = array which indicates the presence of particles in
c                 box.
c
c     iptr    = pointer to location where the storage of the ring values
c               starts.
c
      integer*4 iua(mfine,nfine),iub(mfine,nfine)
c
      do 10 j=1,nfine
      do 10 i=1,mfine
       if(iub(i,j).gt. 0) then
         iua(i,j)=iptr
         iptr=iptr+(isize)
       endif
10    continue
      return
      end
c
```

```
      subroutine fstpol(mfine,nfine,ipole,nxb,nyb,isrc,isrcsz,
     *  xmin,ymin,hx,hy,nsrc,xsrc,ysrc,str,polval,iterms,v)
      implicit real*8(a-h,o-z)
c
c     This subroutine computes potential values on rings whose
c     centers are the finest level boxes. This potential is that
c     which is induced by source the particles in those boxes.
c
c     beta = radius (in units of mesh lengths) of ring approximations.
c
      integer*4 ipole(mfine,nfine)
      integer*4 isrc(mfine,nfine)
      integer*4 isrcsz(mfine,nfine)
c
      real*8 xsrc(1),ysrc(1),str(1)
      real*8 polval(1)
c
      real*8 v(101,2)
      real*8 vposx(101),vposy(101)
c
c     Loop over all boxes.
c
      beta=2.0d00
      hmax=dmaxl(hx,hy)
      rad=beta*hmax
      drad=dlog(rad)
      itot=(2*iterms)+1
      pi=3.14159265359d00
      pi2=pi*2.0d00
c
      do 100 i=1,mfine
      do 100 j=1,nfine
      if(ipole(i,j).eq.0) goto 100
       xpole=(dble(i-1)*hx + xmin) + (hx/2.0d00)
       ypole=(dble(j-1)*hy + ymin) + (hy/2.0d00)
c
       do 10 k=1,itot
        vposx(k)=rad*v(k,1) + xpole
        vposy(k)=rad*v(k,2) + ypole
10     continue
c
c     Get the addresses and limits for the loop over the particles
c     in the boxes.
c
       isrcnm=isrcsz(i,j)
       isbeg=isrc(i,j)
       istrt=ipole(i,j)
       iend=istrt + itot
c
c      accumulate strengths
c
       do 15 m=1,isrcnm
        isind=isbeg + (m-1)
        polval(iend)=polval(iend) + str(isind)
```



```
15     continue
c
c      In the formation of the outer ring approximations we
c      compute values on the ring which are the difference
c      between the values a given source particle induces and the
c      value of a particle centered at the origin with the
c      same strength. This calculation
c      must be done carefully to avoid problems with round off - i.e.
c      in the following loop one should do the subtraction of the
c      term str(ll)*drad/pi2 inside the loop and not bring it outside.
c
c      For more general type distributions (other than point particles)
c      the .5*log(rr) term would be replaced with the other potential
c      function.
c
c
       do 25 k=1,itot
       do 25 l=1,isrcnm
       kk=istrt + (k-1)
       ll=isbeg + (l-1)
       rr=(xsrc(ll)-vposx(k))**2 + (ysrc(ll)-vposy(k))**2
       polval(kk)=polval(kk)+str(ll)*(0.5d0*dlog(rr)-drad)/pi2
25     continue
100    continue
       return
       end
c
```

```
      subroutine mrgpol(nxa,nya,iua,nxb,nyb,iub,xmin,ymin,
     *             hxa,hya,hxb,hyb,polval,iterms,v,cn,sn)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the potential values for an outer
c     ring approximation whose center is an 'a' level box.
c     The potential is that induced by the outer ring approx.s assoc.
c     with 'b' level boxes contained within it. (It's children.)
c
c     a level = level at which the ring approximation values are comp.
c     b level = children
c
      integer*4 iua(nxa,nya),iub(nxb,nyb)
      real*8 polval(1)
c
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c     Loop over all children.  Give the values of the children
c     to the parent
c
c


      beta=2.0d00
      hmaxa=dmax1(hxa,hya)
      rada=beta*hmaxa
      hmaxb=dmax1(hxb,hyb)
      radb=beta*hmaxb
c
      do 100 ind=1,nxb
      do 100 jnd=1,nyb
      if(iub(ind,jnd).gt.0) then
        i=Int((float(ind)-.5)/2.0) + 1
        j=Int((float(jnd)-.5)/2.0) + 1
        xapole=(dble(i-1)*hxa + xmin) + hxa/2.0d00
        yapole=(dble(j-1)*hya + ymin) + hya/2.0d00
        istrta=iua(i,j)
        xbpole=(dble(ind-1)*hxb + xmin) + hxb/2.0d00
        ybpole=(dble(jnd-1)*hyb + ymin) + hyb/2.0d00
        istrtb=iub(ind,jnd)
        call polpol(polval(istrta),xapole,yapole,rada,
     *    polval(istrtb),xbpole,ybpole,radb,iterms,v,cn,sn)
      endif
100   continue
      return
      end
c
```



```
      function polevl(xpos,ypos,itot,pol,xpole,ypole,rpole,cn,sn)
      implicit real*8(a-h,o-z)
      real*8 pol(1)
      real*8 ak(101)
      real*8 cn(101,3),sn(101,3)
c
c     This procedure evalutes the outer ring approximation from
c     the itot boundary values in pol. The dimension of ak should
c     be more than itot.
c
      val=0.0d00
      pi=3.14159265359d00
      pi2=2.0d00*pi
      a=rpole
      dtheta=pi2/dble(itot)
      nn=(itot-1)/2
c
      xtrns=xpos-xpole
      ytrns=ypos-ypole
      r=dsqrt(xtrns**2 + ytrns**2)
      thet0 =datan2(ytrns,xtrns)
c
      alpha=a/r
      a2=alpha**2
      an=alpha**nn
      anp1=an*alpha
      anp2=anp1*alpha
      c1=dcos(thet0)
      c2=dcos(dble(nn)*thet0)
      c3=dcos(dble(nn+1)*thet0)
      s1=-dsin(thet0)
      s2=-dsin(dble(nn)*thet0)
      s3=-dsin(dble(nn+1)*thet0)
c
c     In the calculation of the kernel  - I avoid computing cosines
c     by using the fact that
c
c     cos(t - s)=cos(t)*cos(-s) - sin(t)*sin(-s)
c
c     This makes the code unreadable, but it runs much faster.
c
      eps=0.0001d00
      if(dabs(alpha-1.0d00) .gt. eps) then
c
      do 250 j=1,itot
      val=val + (pol(j)*dtheta*
     *  (1.0d0 - a2 + 2.d0*
     *  (anp2*(cn(j,2)*c2-sn(j,2)*s2)- anp1*(cn(j,3)*c3-sn(j,3)*s3)))
     *  /(1.d00 - 2.d00*alpha*(cn(j,1)*c1 -sn(j,1)*s1) + a2))
250   continue
c
c
      polevl=(dlog(r)*pol(itot+1) + val)/(pi2)
      else
c
```

```
c       evaluate the kernel in long form
c
        ak(1)=alpha
        do 325 i=2,nn
         ak(i)=alpha*ak(i-1)
325     continue
        do 400 j=1,itot
          theta=dble(j-1)*dtheta
          zker=0.0d00
          do 350 i=1,nn
          zk=dble(i)
          zker=zker+ ak(i)*dcos(zk*(theta-thet0))
350       continue
          val=val + zker*pol(j)*dtheta
400     continue
        polevl=dlog(r)*pol(itot+1)/pi2 + val/pi
        endif
        return
        end
c
```



```
        subroutine polpol(pola,xapole,yapole,rada,
     *        polb,xbpole,ybpole,radb,iterms,v,cn,sn)
        implicit real*8(a-h,o-z)
        real*8 pola(1),polb(1)
        real*8 v(101,2),cn(101,3),sn(101,3)
c
c       Outer Ring Approximation to Outer Ring Approximation Shift
c
c       This subroutine evalutes the potential of
c       an outer ring approximation  (pola) at the node
c       points of another outer ring approximation. These
c       values are stored in (polb)
c
c       increment log term
c
        pi=3.14159265359d00
        pi2=2.0d00*pi
        itot=2*iterms + 1
        pola(itot+1)=pola(itot+1) + polb(itot+1)
        cons=polb(itot+1)*(dlog(rada)/pi2)
c
c       now add all the other values
c
        do 100 i=1,itot
        xpos=rada*v(i,1) + xapole
        ypos=rada*v(i,2) + yapole
        pola(i)=pola(i) +
     *  (polevl(xpos,ypos,itot,polb,xbpole,ybpole,radb,cn,sn) - cons)
100     continue
c
c       -> new values in polb are orthogonal to a constant
c
        return
        end
c
```

```
      subroutine uppas(xl,yl,str,npart,xmin,ymin,xmax,ymax,
     *  mxlevx,mxlevy,mfine,nfine,ipole,ngsizb,polval,npolsz,
     *  iterms,xtst,ytst,ntst,ipow,powval,npowsz,icor,
     *  irec,irecsz,v,cn,sn)
c
      implicit real*8(a-h,o-z)
c
c     Upward Pass - outer approx. known at all levels, so compute
c                   the inner ring approximations.
c
c
c
      real*8 xl(1),yl(1),str(1)
      real*8 xtst(1),ytst(1)
c
      integer*4 ipole(1)
      integer*4 ipow(1)
      integer*4 irec(1),irecsz(1)
      real*8 polval(1)
      real*8 powval(1)
c
c
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c
c     nested grid information arrays (maximum depth of 20)
c
      integer*4 npanx(20),npany(20),nptr(20)
      real*8 hpanx(20),hpany(20)
c
c
c
      idepth=max0(mxlevx,mxlevy)
c
c     set up the indicies for the inner ring approx. pointers
c
      call  setptr(mxlevx,mxlevy,mxstor,npanx,npany,nptr)
      if(mxstor .gt. ngsizb) then
       write(12,*) 'Error : Insufficient Integer Wrk. Space : '
       write(12,*) 'Storage Needed : ',mxstor
       write(12,*) 'Storage Allocated (value of ngsizb) :',ngsizb
       write(*,*) 'Fatal Error : See errout '
       stop
      endif
c
c     Initialze the inner ring pointer arrays - set all entries
c     to -9 to indicate the absence of an inner ring approximation.
c
c
      do 130 level=1,idepth
        nx=npanx(level)
        ny=npany(level)
        nstrt=nptr(level)
        call niner(nx,ny,ipow(nstrt))
130   continue
```



```
c
c     set up mesh sizes
c
      hxstrt=(xmax-xmin)/dble(mfine)
      hystrt=(ymax-ymin)/dble(nfine)
      hpanx(idepth)=hxstrt
      hpany(idepth)=hystrt
      do 140 i=1,idepth-1
       level=idepth-i
       ix=min0(i,mxlevx)
       iy=min0(i,mxlevy)
       hpanx(level)=hxstrt*(2**ix)
       hpany(level)=hystrt*(2**iy)
140   continue
c
c
c     Allocate space and designate locations for inner ring values.
c     (2*iterms+1)  values for each representation are needed.
c
c     iptr = pointer to next available address location
c            in the array powval.
c
      level=idepth
      nstrt=nptr(level)
      iptr=1
      isize=2*iterms + 1
      call fstalc(mfine,nfine,ipow(nstrt),irecsz,iptr,isize)
c
c     Allocate on succesively coarser Levels
c
c     levela= level where we need allocation
c
c     levelb= previous level
c
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)
       nstrta=nptr(levela)
       levelb=levela+1
       nxb=npanx(levelb)
       nyb=npany(levelb)
       nstrtb=nptr(levelb)
       call alloc(nxa,nya,ipow(nstrta),nxb,nyb,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(12,*) 'Error Insufficient Real Wrk. Storage :'
         write(12,*) 'Storage Allocated : ',2*npowsz
         write(12,*) 'Storage Necessary (value of nrealw) : ',2*nvals
         write(*,*) 'Fatal Error : See errout '
        stop
        endif
c
c       Now form all of the required inner ring approximations
c
c       Zero Storage Array
c
        do 400 i=1,nvals
          powval(i)=0.0d00
400     continue
c
c       Form inner ring approx. for all the other levels
c
        do 410 level=2,idepth
        nxa=npanx(level)
        nya=npany(level)
        hxa=hpanx(level)
        hya=hpany(level)
        nstrta=nptr(level)
        nxb=npanx(level-1)
        nyb=npany(level-1)
        hxb=hpanx(level-1)
        hyb=hpany(level-1)
        nstrtb=nptr(level-1)
        call mrgpow(nxa,nya,ipow(nstrta),ipole(nstrta),nxb,nyb,
     *  ipow(nstrtb),xmin,ymin,hxa,hya,hxb,hyb,powval,
     *  polval,iterms,v,cn,sn,icor)
410     continue
c
c       Done - Inner ring approximations available at all levels.
c
        return
        end
c
```



```
        subroutine  mrgpow(nxa,nya,ipowa,ipole,nxb,nyb,ipowb,
     *  xmin,ymin,hxa,hya,hxb,hyb,pow,pol,iterms,v,cn,sn,icor)
        implicit real*8 (a-h,o-z)
c
c       This subroutine computes the inner ring approx. from the parent
c       inner ring approx. and the outer ring approx. which are well
c       separted but contained under the parents neighbors.
c
c       a= level where we need the inner ring approximation
c       b= previous level (coarser) at which the ring approx.
c          have already been obtained.
c
        integer*4 ipowa(nxa,nya),ipole(nxa,nya),ipowb(nxb,nyb)
        real*8 pol(1),pow(1)
        real*8 v(101,2),cn(101,3),sn(101,3)
c
c
c
c
        beta=2.0d00
        hmaxa=dmax1(hxa,hya)
        hmaxb=dmax1(hxb,hyb)
c
c     Loop over current level inner ring approximations.
c
        do 1000 i=1,nxa
        do 1000 j=1,nya
        if(ipowa(i,j) .gt. 0) then
        ind=Int((float(i)-.5)/2.0) + 1
        jnd=Int((float(j)-.5)/2.0) + 1
          if(ipowb(ind,jnd) .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
          rpar=hmaxb/2.0d00
          ippar=ipowb(ind,jnd)
          xchld=(dble(i-1)*hxa + (hxa/2.0d00)) + xmin
          ychld=(dble(j-1)*hya + (hya/2.0d00)) + ymin
          rchld=hmaxa/2.0d00
          ipchld=ipowa(i,j)
          call powpow(xpar,ypar,rpar,pow(ippar),
     *              xchld,ychld,rchld,pow(ipchld),iterms,v,cn,sn)
          endif
c
c         obtain values from nearby outer ring approx. (but not neighbors)
c
          indbot=ind-icor
          jndbot=jnd-icor
          ibot=(indbot*2 -1)
          jbot=(jndbot*2 -1)
          itop=ibot+(4*icor) + 1
          jtop=jbot+(4*icor) + 1
```

```
          ibot=max(ibot,1)
          jbot=max(jbot,1)
          itop=min(itop,nxa)
          jtop=min(jtop,nya)
c
          do 200 k=ibot,itop
          do 200 m=jbot,jtop
          kd=abs(k-i)
          md=abs(m-j)
          if((kd.ge.(icor+1)).or.(md.ge.(icor+1))) then
          if(ipole(k,m) .gt. 0) then
          xpole=(dble(k-1)*hxa + xmin) + (hxa/2.0d00)
          ypole=(dble(m-1)*hya + ymin) + (hya/2.0d00)
          rpole=beta*hmaxa
          xchld=(dble(i-1)*hxa + (hxa/2.0d00)) + xmin
          ychld=(dble(j-1)*hya+ (hya/2.0d00)) + ymin
          rchld=hmaxa/2.0d00
          ipchld=ipowa(i,j)
          iplptr=ipole(k,m)
          call polpow(xpole,ypole,rpole,pol(iplptr),
     *              xchld,ychld,rchld,pow(ipchld),iterms,v,cn,sn)
          endif
          endif
200       continue
        endif
1000    continue
        return
        end
c
```



```
        subroutine powpow(xpar,ypar,rpar,powb,
     *              xchld,ychld,rchld,powa,iterms,v,cn,sn)
        implicit real*8 (a-h,o-z)
c
c       Inner ring approx. to Inner ring approx. shift.
c
c       In this subroutine the inner ring approx. located at the
c       parent center, (xpar,ypar)
c       with radius rpar is evaluated at the points of a ring about
c       for 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(101,2),cn(101,3),sn(101,3)
        npow=2*iterms + 1
c
        do 100 i=1,npow
         xpos=rchld*v(i,1) + xchld
         ypos=rchld*v(i,2) + ychld
         powa(i)=powa(i) + powevl(xpos,ypos,npow,powb,
     *                    xpar,ypar,rpar,cn,sn)
100     continue
        return
        end
c
```

```
      subroutine polpow(xpole,ypole,rpole,pol,
     *                  xpow,ypow,rpow,pow,iterms,v,cn,sn)
      implicit real*8 (a-h,o-z)
c
c     Inner ring approx. to Outer ring approx. shift.
c
c     In this subroutine the inner ring approx.
c     at (xpole,ypole) with radius rpole
c     is evaluated at the points of a ring about
c     (rpow,ypow) with radius rpow.
c
      real*8 pol(1),pow(1)
      real*8 v(101,2),cn(101,3),sn(101,3)
      npol=2*iterms + 1
      npow=2*iterms + 1
c
      do 100 i=1,npow
       xpos=rpow*v(i,1) + xpow
       ypos=rpow*v(i,2) + ypow
       pow(i)=pow(i) + polevl(xpos,ypos,npol,pol,
     *                  xpole,ypole,rpole,cn,sn)
100   continue
      return
      end
c
```



```
      double precision function powevl(xpos,ypos,itot,pow,xpow,
     *                                 ypow,rpow,cn,sn)
      implicit real*8 (a-h,o-z)
      real*8 pow(1)
      real*8 ak(101)
      real*8 cn(101,3),sn(101,3)
c
c     This procedure evalutes the inner ring approx.  from
c     the itot boundary values in pow. It uses the first nn terms
c     of the Poisson Kernel. If the point is too close to the
c     ring on which the points are stored, then the n-terms of the
c     Poisson kernel are evaluated in long form.
c
      val=0.0d00
      pi=3.14159265359d00
      pi2=2.0d00*pi
      a=rpow
      dtheta=pi2/dble(itot)
      nn=(itot-1)/2
c
      xtrns=xpos-xpow
      ytrns=ypos-ypow
      r=dsqrt(xtrns**2 + ytrns**2)
      thet0=0.0d00
      if(r .gt. 1.0d-12) thet0 =datan2(ytrns,xtrns)
c
      alpha=r/a
      a2=alpha**2
      an=alpha**nn
      anp1=an*alpha
      anp2=anp1*alpha
      c1=dcos(thet0)
      c2=dcos(dble(nn)*thet0)
      c3=dcos(dble(nn+1)*thet0)
      s1=-dsin(thet0)
      s2=-dsin(dble(nn)*thet0)
      s3=-dsin(dble(nn+1)*thet0)
c
      eps=0.0001d00
      if(dabs(alpha-1.0d00).gt. eps) then
      do 250 j=1,itot
      val=val + (pow(j)*dtheta*
     *  (1.0d0 - a2 + 2.d0*
     *  (anp2*(cn(j,2)*c2-sn(j,2)*s2)- anp1*(cn(j,3)*c3-sn(j,3)*s3)))
     *  /(1.d00 - 2.d00*alpha*(cn(j,1)*c1 -sn(j,1)*s1) + a2))
250   continue
c
      powevl=val/(2.0d00*pi)
      else
c
c     evaluate the kernel in long form
c
      ak(1)=alpha
      do 325 i=2,nn
       ak(i)=ak(i-1)*alpha
```

```
325     continue
        do 400 j=1,itot
          theta=dble(j-1)*dtheta
          zker=0.5d00
          do 350 i=1,nn
          zk=dble(i)
          zker=zker+ ak(i)*dcos(zk*(theta-thet0))
350       continue
          val=val + zker*pow(j)*dtheta
400     continue
        powevl=val/pi
        endif
        return
        end
c
```



```
        subroutine powdrv(xpos,ypos,itot,pow,xpow,
     *                    ypow,rpow,cn,sn,powdx,powdy)
        implicit real*8 (a-h,o-z)
        real*8 pow(1)
        real*8 ak(101)
        real*8 cn(101,3),sn(101,3)
c
c       This procedure evalutes the derivatives of an inner approx. from
c       the itot boundary values in pow. It uses the first nn terms
c       of the Poisson Kernel. If the point is too close to the
c       ring on which the points are stored, then the n-terms of the
c       Poisson kernel are evaluated in long form.
c
        powdx=0.0d00
        powdy=0.0d00
        powdr=0.0d00
        powdt=0.0d00
        pi=3.14159265359d00
        pi2=2.0d00*pi
c
        a=rpow
        dtheta=pi2/dble(itot)
        nn=(itot-1)/2
        xtrns=xpos-xpow
        ytrns=ypos-ypow
        r=dsqrt(xtrns**2 + ytrns**2)
        thet0=0.0d00
        if(r .gt. 1.0d-12) thet0 =datan2(ytrns,xtrns)
c
        alpha=r/a
        a2=alpha**2
        an=alpha**nn
        anp1=an*alpha
        anp2=anp1*alpha
        c1=dcos(thet0)
        c2=dcos(dble(nn)*thet0)
        c3=dcos(dble(nn+1)*thet0)
        s1=-dsin(thet0)
        s2=-dsin(dble(nn)*thet0)
        s3=-dsin(dble(nn+1)*thet0)
c
c
c
        eps=0.00001d00
        if(dabs(alpha-1.0) .gt. eps) then
        do 250 j=1,itot
          bot=1.d0-2.d0*alpha*(cn(j,1)*c1 - sn(j,1)*s1) + a2
          top=1.d0 - a2 + 2.0d0*
     *    (anp2*(cn(j,2)*c2-sn(j,2)*s2)-anp1*(cn(j,3)*c3-sn(j,3)*s3))
c
c         d/dr derivative
c
          dtop= -(2.d0/a)*alpha
     *    + (2.d0/a)*(dble(nn+2)*anp1*(cn(j,2)*c2 - sn(j,2)*s2))
     *    - (2.d0/a)*(dble(nn+1)*an*(cn(j,3)*c3 - sn(j,3)*s3))
```

```
          dbot=-(2.d0/a)*(cn(j,1)*c1 - sn(j,1)*s1) + (2.d0/a)*alpha
          powdr=powdr+
     *    ((dtop*bot - top*dbot)/(bot**2))*dtheta*pow(j)
c
c         1/r*(d/dtheta) derivative
c
          dtop= (2.d0/a)*anpl*dble(nn)*(sn(j,2)*c2 + cn(j,2)*s2)
     *         -(2.d0/a)*an*dble(nn+1)*(sn(j,3)*c3 + cn(j,3)*s3)
          dbot=-(2.d0/a)*(sn(j,1)*c1 + cn(j,1)*s1)
          powdt=powdt+
     *    ((dtop*bot - top*dbot)/(bot**2))*dtheta*pow(j)
c
c
c
250     continue
c
c       form x and y components from polar components - note 1/r factor
c       in the d theta derivative has already been accounted for.
c
        powdx=dcos(thet0)*powdr - dsin(thet0)*powdt
        powdy=dsin(thet0)*powdr + dcos(thet0)*powdt
        powdx=powdx/(2.d0*pi)
        powdy=powdy/(2.d0*pi)
c
c
        else
c
c       evaluate the kernel in long form
c
        ak(1)=1.0d00
        do 325 i=2,nn
         ak(i)=alpha*ak(i-1)
325     continue
c
        do 400 j=1,itot
          theta=dble(j-1)*dtheta
          zkerdr=0.0d00
          zkerdt=0.0d00
          do 350 i=1,nn
          zk=dble(i)
          zkerdr=zkerdr+(1.d0/a)*ak(i)*zk*dcos(zk*(theta-thet0))
          zkerdt=zkerdt+(1.d0/a)*ak(i)*zk*dsin(zk*(theta-thet0))
350       continue
          powdr=powdr + zkerdr*pow(j)*dtheta
          powdt=powdt + zkerdt*pow(j)*dtheta
400     continue
c
c
        powdx=dcos(thet0)*powdr - dsin(thet0)*powdt
        powdy=dsin(thet0)*powdr + dcos(thet0)*powdt
        powdx=powdx/pi
        powdy=powdy/pi
c
        endif
        return
        end
```



```
c
```

```
      subroutine velevl(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     *  velx,vely,ipow,powval,iterms,tol,xmin,xmax,ymin,ymax,icor,
     *  delta,mfine,nfine,isrc,isrcsz,irec,irecsz,v,cn,sn)
c
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the velocity at the test particle locations
c     from the inner ring approx. values in powval and the local particles
c     which are pointed to by ilink and ipart.
c
c
      real*8 xsrc(nsrc),ysrc(nsrc),str(nsrc)
      real*8 xrec(nsrc),yrec(nsrc),velx(nsrc),vely(nsrc)
c
      integer*4 ipow(mfine,nfine)
c
      integer*4 isrc(mfine,nfine),isrcsz(mfine,nfine)
      integer*4 irec(mfine,nfine),irecsz(mfine,nfine)
c
      real*8 powval(1)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c
c
      pi=3.14159265359d00
      pi2=2.0d00*pi
      hx=(xmax-xmin)/dble(mfine)
      hy=(ymax-ymin)/dble(nfine)
      hmax=dmax1(hx,hy)
c
c     Loop over all the boxes on the finest level
c
      do 100 j=1,nfine
      do 100 i=1,mfine
      irecnm=irecsz(i,j)
      if(irecnm .eq. 0) goto 100
      kbeg=irec(i,j)
      do 10 kkk=1,irecnm
      knd=kbeg + (kkk-1)
c
c     evaluate velocity from power expansions
c
      iptr=ipow(i,j)
      xpow=(dble(i-1)*hx + (hx/2.0d00)) + xmin
      ypow=(dble(j-1)*hy + (hy/2.0d00)) + ymin
      rpow=hmax/2.0d00
      npow=2*iterms + 1
c
c     get derivatives
c
      xa=xrec(knd)
      ya=yrec(knd)
      call powdrv(xa,ya,npow,powval(iptr),xpow,ypow,rpow,
     *            cn,sn,powdx,powdy)
c     y velocity
```



```
c
      vely(knd)=vely(knd) + powdx
c
c     x velocity
c
      velx(knd)=velx(knd) - powdy
c
10    continue
c
c     now do local direct interactions
c
      ibot=i-icor
      itop=i+icor
      jbot=j-icor
      jtop=j+icor
      ibot=max(ibot,1)
      itop=min(itop,mfine)
      jbot=max(jbot,1)
      jtop=min(jtop,nfine)
c
      do 20 jjj=jbot,jtop
      do 20 iii=ibot,itop
       isrcnm=isrcsz(iii,jjj)
       if(isrcnm.eq.0) goto 20
       jbeg=isrc(iii,jjj)
       call veloex(xsrc(jbeg),ysrc(jbeg),str(jbeg),isrcnm,
     *   xrec(kbeg),yrec(kbeg),irecnm,velx(kbeg),vely(kbeg),delta)
20    continue
100   continue
c
      return
      end
c
```

```
      subroutine potevl(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     *  pot,ipow,powval,iterms,tol,xmin,xmax,ymin,ymax,icor,
     *  delta,mfine,nfine,isrc,isrcsz,irec,irecsz,v,cn,sn)
c
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the potential at the test particle locations
c     from the inner ring approx. values in powval and the local particles
c     which are pointed to by ilink and ipart.
c
c
c
      real*8 xsrc(nsrc),ysrc(nsrc),str(nsrc)
      real*8 xrec(nrec),yrec(nrec),pot(nrec)
c
      integer*4 ipow(mfine,nfine)
      integer*4 isrc(mfine,nfine),isrcsz(mfine,nfine)
      integer*4 irec(mfine,nfine),irecsz(mfine,nfine)
c
c
      real*8 powval(1)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c
      pi=3.14159265359d00
      pi2=2.0d00*pi
      hx=(xmax-xmin)/dble(mfine)
      hy=(ymax-ymin)/dble(nfine)
      hmax=dmax1(hx,hy)
c
c     Loop over all the boxes on the finest level
c
      do 100 j=1,nfine
      do 100 i=1,mfine
      irecnm=irecsz(i,j)
      if(irecnm .eq. 0) goto 100
      kbeg=irec(i,j)
c
c     evaluate potential from local power expansion
c
      do 10 kkk=1,irecnm
      knd=kbeg + (kkk-1)
      iptr=ipow(i,j)
      xpow=(dble(i-1)*hx + (hx/2.0d00)) + xmin
      ypow=(dble(j-1)*hy + (hy/2.0d00)) + ymin
      rpow=hmax/2.0d00
      npow=2*iterms + 1
      val = powevl(xrec(knd),yrec(knd),npow,powval(iptr),
     *             xpow,ypow,rpow,cn,sn)
      pot(knd)=pot(knd) + val
10    continue
c
c     now do local direct interactions
c
      ibot=i-icor
```



```
      itop=i+icor
      jbot=j-icor
      jtop=j+icor
      ibot=max(ibot,1)
      itop=min(itop,mfine)
      jbot=max(jbot,1)
      jtop=min(jtop,nfine)
c
      do 20 jjj=jbot,jtop
      do 20 iii=ibot,itop
       isrcnm=isrcsz(iii,jjj)
       if(isrcnm.eq.0) goto 20
       jbeg=isrc(iii,jjj)
       call potex(xsrc(jbeg),ysrc(jbeg),str(jbeg),isrcnm,
     *            xrec(kbeg),yrec(kbeg),irecnm,pot(kbeg),delta)
20    continue
c
c
100   continue
      return
      end
c
```

```
      subroutine esttim(xl,yl,str,npart,xtst,ytst,ntst,iwflag,delta,
     *     ipole,ipow,ngsizb,iterms,mxlevx,mxlevy,
     *     xmin,ymin,xmax,ymax,icor,totime,iprint)
      implicit real*8(a-h,o-z)
c
c     This subroutines purpose is to determine the time it takes to
c     carry out the hierarchical element
c     routine for each level up to the maximum
c     levels prescribed.
c
      real*8 xl(1),yl(1),str(1)
      real*8 xtst(1),ytst(1)
c
      real*8 dwrk(20),uwrk(20),uinc(20)
c
      integer*4 npanx(20),npany(20),nptr(20)
      integer*4 ipole(1),ipow(1)
      integer*4 iwflag
c
c
      real*8 totime(20)
      real*8 tcon(10,2)
c
c
c     Obtain the timing constants
c
      call getcon(tcon,scale)
c
c
c     Set up the nested grid parameters for the finest mesh.
c
      mfine=2**mxlevx
      nfine=2**mxlevy
      h=(xmax-xmin)/dble(mfine)
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,mxstor,npanx,npany,nptr)
      if(mxstor .gt. ngsizb) then
       write(12,*) 'Error : Insufficient Integer Wrk. Space : '
       write(12,*) 'Storage Needed : ',mxstor
       write(12,*) 'Storage Allocated (value of ngsizb) :',ngsizb
       write(*,*) 'Fatal Error : See errout '
      stop
      endif
c
c     First determine the number of particles in
c     each box at each level
c
      idepth=max0(mxlevx,mxlevy)
      nx=npanx(idepth)
      ny=npany(idepth)
      nstrt=nptr(idepth)
c
```



```
      call fstnum(xl,yl,npart,nx,ny,ipole(nstrt),h,xmin,ymin)
      call fstnum(xtst,ytst,ntst,nx,ny,ipow(nstrt),h,xmin,ymin)
c
c
      do 200 i=1,idepth-2
       levela=idepth-i
       nxa=npanx(levela)
       nya=npany(levela)
       nstrta=nptr(levela)
       levelb=levela + 1
       nxb=npanx(levelb)
       nyb=npany(levelb)
       nstrtb=nptr(levelb)
       call nxtnum(nxa,nya,ipole(nstrta),nxb,nyb,ipole(nstrtb))
       call nxtnum(nxa,nya,ipow(nstrta),nxb,nyb,ipow(nstrtb))
200   continue
c
c
c     First level treated differently - put -9's into entries
c     (We don't need inner or outer ring approx. at the first level.)
c
      nxa=npanx(1)
      nya=npany(1)
      nstrta=nptr(1)
      call niner(nxa,nya,ipole(nstrta))
      call niner(nxa,nya,ipow(nstrta))
c
c     Each of the non-zero entries of ipole and
c     ipow at a given level indicate how
c     many particles there are in that box
c
c     Estimate the time in the downward pass
c
      do 205 i=1,idepth
       dwrk(i)=0.0d00
205   continue
c
      if(idepth .gt. 1) then
       level=idepth
       nx=npanx(level)
       ny=npany(level)
       nstrt=nptr(level)
       call fstdwn(nx,ny,ipole(nstrt),iterms,dwrk(level),
     *             tcon,scale)
      endif
c
c     now estimate the work on all the coarser levels -
c
      do 300 i=1,idepth-2
       levela=idepth-i
       nxa=npanx(levela)
       nya=npany(levela)
       nstrta=nptr(levela)
       levelb=levela + 1
       nxb=npanx(levelb)
       nyb=npany(levelb)
```

```
         nstrtb=nptr(levelb)
         dtmp=0.0d00
         call nxtdwn(nxa,nya,ipole(nstrta),nxb,nyb,ipole(nstrtb),
     *          iterms,dtmp,dwrk(levela),tcon,scale)
c
c        add current level work to previous level work
c
         do 250 kk=levela+1,idepth
           dwrk(kk)=dwrk(kk) + dtmp
250      continue
300     continue
c
c       dwrk(i) now contains the work estimate for forming the
c       outer ring approximations down to
c       level (i) - (i.e. the entries are cumulative)
c
c       Now estimate the amount of work in the construction and
c       evaluation of the inner ring approximations.
c       At every level we have two times,
c       the amount of time it takes to go to the next step
c       or the time it takes to use that level.
c
        do 400 i=1,idepth
         uwrk(i)=0.0d00
         uinc(i)=0.0d00
400     continue
c
        level=1
c
c       amount of work at this coarsest
c       level is just the fast direct interaction
c
        uinc(1)=0.0d00
        uwrk(1)=
     *  (dble(ntst)*dble(npart)*(tcon(5,1)+tcon(6,1)))/scale+
     *  tcon(5,2) + tcon(6,2)
        if(iwflag.eq.0)uwrk(1)=
     *  (dble(ntst)*dble(npart)*tcon(5,1))/scale +
     *  tcon(5,2)
        if(iwflag.eq.1)uwrk(1)=
     *  (dble(ntst)*dble(npart)*tcon(6,1))/scale +
     *          tcon(6,2)
        utmp=0.0d00
c
c       Determine the work at all finer levels
c
        do 500 kk=2,idepth
         levela=kk
         nxa=npanx(levela)
         nya=npany(levela)
         nstrta=nptr(levela)
         levelb=levela - 1
         nxb=npanx(levelb)
         nyb=npany(levelb)
         nstrtb=nptr(levelb)
         call nxtup(nxa,nya,ipow(nstrta),ipole(nstrta),nxb,nyb,
```



```
     *   ipow(nstrtb),iterms,icor,iwflag,utmp,uwrk(levela),
     *   tcon,scale)
         uinc(kk)=utmp-uinc(kk-1)
500     continue
c
c
        if(iprint .ne. 0) then
        write(9,*) '    '
        write(9,*) ' Timing Estimates : '
        write(9,*) '    '
        endif
        do 600 i=1,idepth
        tot=dwrk(i) + uwrk(i)
        totime(i)=tot
        if(iprint.ne.0) then
        write(9,550) i,dwrk(i),uwrk(i),tot
550     format(1x,i1,' Down Tm.: ',f8.1,'     Up Tm. : ',
     *                  f8.1,'      Tot : ',f8.1)
        endif
600     continue
        if(iprint.ne.0) then
        write(9,*) '       '
        write(9,*) '       '
        endif
        return
        end
c
```

```
      subroutine nxtup(nxa,nya,ipowa,ipol,nxb,nyb,ipowb,
     *             iterms,icor,iwflag,utmp,uwrk,tcon,scale)
      implicit real*8 (a-h,o-z)
c
c     call so that this runs on a parallel machine
c
cvd$r noconcur
c
c     In this subroutine I compute the work to travel up in levels
c     Here utmp is incremented by the amount of work necessary to get
c     to the next step, while uwrk is the amount of work
c     which is necessary
c     if one stops the method at this level
c
c      utmp contains all the work from the previous levels
c     so increment utmp by the amount of work of forming
c     inner ring approx.s from parent as well as getting
c     data from nearby outer ring approx.
c
c     uwrk = the local utmp + whatever amount of work there is
c            in local direct intertion
c
      integer*4 ipowa(nxa,nya),ipol(nxa,nya)
      integer*4 ipowb(nxb,nyb)
      integer*4 iwflag
c
      real*8 tcon(10,2)
c
c
c     set the current work level to the accumulated work level
c
      uwrk=utmp
c
      itot=2*iterms+1
      do 1000 i=1,nxa
      do 1000 j=1,nya
      if(ipowa(i,j) .gt. 0) then
      ind=Int((float(i)-.5)/2.0) + 1
      jnd=Int((float(j)-.5)/2.0) + 1
c
        if(ipowb(ind,jnd) .gt. 0 ) then
c
c       obtain values from parent
c
        uwrk=uwrk + (dble(itot)*dble(itot)*tcon(4,1))/scale
     *            + tcon(4,2)
        utmp=utmp + (dble(itot)*dble(itot)*tcon(4,1))/scale
     *            + tcon(4,2)
c
        endif
c
c       obtain values from nearby outer ring approx. (but not neighbors)
c
        indbot=ind-icor
        jndbot=jnd-icor
```



```
        ibot=(indbot*2 -1)
        jbot=(jndbot*2 -1)
        itop=ibot+(4*icor) + 1
        jtop=jbot+(4*icor) + 1
        ibot=max0(ibot,1)
        jbot=max0(jbot,1)
        itop=min0(itop,nxa)
        jtop=min0(jtop,nya)
c
        do 200 k=ibot,itop
        do 200 m=jbot,jtop
        kd=abs(k-i)
        md=abs(m-j)
        if((kd.ge.(icor+1)).or.(md.ge.(icor+1))) then
        if(ipol(k,m) .gt. 0) then
        uwrk=uwrk + (dble(itot)*dble(itot)*tcon(3,1))/scale
     *            + tcon(3,2)
        utmp=utmp + (dble(itot)*dble(itot)*tcon(3,1))/scale
     *            + tcon(3,2)
        endif
        endif
200     continue
      endif
1000  continue
c
c     If we stop at this level - add local interactions
c     plus one evalution of
c     the inner ring approximations.
c
      factl=tcon(5,1)+tcon(6,1)
      factla=tcon(5,2) + tcon(6,2)
      if(iwflag.eq.0) then
        factl=tcon(5,1)
        factla=tcon(5,2)
      endif
      if(iwflag.eq.1) then
        factl=tcon(6,1)
        factla=tcon(6,2)
      endif
c
      fact2= tcon(7,1) + tcon(8,1)
      fact2a=tcon(7,2) + tcon(8,2)
      if(iwflag.eq.0) then
        fact2=tcon(7,1)
        fact2a=tcon(7,2)
      endif
      if(iwflag.eq.1) then
        fact2=tcon(8,1)
        fact2a=tcon(8,2)
      endif
c
      do 2000 i=1,nxa
      do 2000 j=1,nya
      if(ipowa(i,j) .gt. 0) then
      ibot=i-icor
      jbot=j-icor
```

```
        itop=i+icor
        jtop=j+icor
        ibot=max(ibot,1)
        jbot=max(jbot,1)
        itop=min(itop,nxa)
        jtop=min(jtop,nya)
        do 1750 L=ibot,itop
        do 1750 M=jbot,jtop
         if(ipol(L,M) .gt. 0) then
         uwrk=uwrk + (dble(ipol(L,M))*dble(ipowa(i,j)*factl)/scale)
     *             + factla
         endif
1750    continue
c
c       cost of inner ring approx. evaluation -
c       the number depending on whether
c       one wants the potentials or the velocities or both.
c


        uwrk=uwrk + (dble(itot)*fact2*dble(ipowa(i,j))/scale)
     *            + fact2a
        endif
2000    continue
        return
        end
c
```



```
        subroutine nxtdwn(nxa,nya,iua,nxb,nyb,iub,iterms,
     *                    dwrka,dwrkb,tcon,scale)
        implicit real*8 (a-h,o-z)
c
c       This subroutine computes the time to obtain the parent outer
c       ring approx. from the children (dwrka)
c       (dwrka) or the time it takes to create outer ring approx.
c       from all the particles at that particular level (dwrkb)
c
c       a = coarser level
c
c       b = finer level
c
c
        integer*4 iua(nxa,nya),iub(nxb,nyb)
        real*8 tcon(10,2)
c
c       outer ring - outer ring
c
        itot=2*iterms + 1
        do 100 i=1,nxb
        do 100 j=1,nyb
         if(iub(i,j) .gt. 0 ) then
         dwrka=dwrka + (dble((itot*itot))*tcon(2,1))/scale
     *               + tcon(2,2)
         endif
100     continue
c
c      particle - outer ring
c
        do 200 i=1,nxa
        do 200 j=1,nya
         if(iua(i,j) .gt. 0) then
          dwrkb=dwrkb + ((dble(iua(i,j))*dble(itot)*tcon(1,1))/scale)
     *                + tcon(1,2)
         endif
200     continue
        return
        end
c
```

```
      subroutine fstdwn(nx,ny,iua,iterms,dwrk,tcon,scale)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the time
c     to construct the outer ring approx.s on the finest level
c
      integer*4 iua(nx,ny)
      real*8 tcon(10,2)
c
c
      itot=2*iterms + 1
       do 100 i=1,nx
       do 100 j=1,ny
       if(iua(i,j) .gt. 0) then
        dwrk=dwrk + ((dble(iua(i,j))*dble(itot)*tcon(1,1))/scale)
     *            + tcon(1,2)
       endif
100    continue
       return
       end
c
```



```
      subroutine nxtnum(nxa,nya,iua,nxb,nyb,iub)
c
c     This subroutine determines the number of particles in the iua boxes
c     from the number of particles in the iub boxes
c
c     a = level needing numbers = coarser level
c
c     b = previous level which contains the numbers.
c
      integer*4 iua(nxa,nya),iub(nxb,nyb)
c
      do 100 i=1,nxa
      do 100 j=1,nya
        iua(i,j)=0
100   continue
c
c     loop over finer ones and then increment the coarser values
c
      do 200 ind=1,nxb
      do 200 jnd=1,nyb
        i=Int((float(ind) - .5)/2.0) + 1
        j=Int((float(jnd) - .5)/2.0) + 1
        iua(i,j)=iua(i,j) + iub(ind,jnd)
200   continue
      return
      end
c
```

```
      subroutine fstnum(xp,yp,npoint,nx,ny,iu,h,xmin,ymin)
c
      implicit real*8(a-h,o-z)
c
c     This subroutines purpose is to put in each (i,j)th entry the
c     number of particles (xp(*),yp(*)) which are in the (i,j)th box.
c
      integer*4 iu(nx,ny)
      real*8 xp(1),yp(1)
c
      do 100 i=1,nx
      do 100 j=1,ny
       iu(i,j)=0
100   continue
c
      do 200 kk=1,npoint
       i=Int((xp(kk)-xmin)/h) + 1
       j=Int((yp(kk)-ymin)/h) + 1
       iu(i,j)=iu(i,j) + 1
200   continue
      return
      end
c
```



```
      subroutine srcsrt(xsrc,ysrc,str,tmp,nsrc,links,isrc,
     *                  isrcsz,mfine,nfine,xmin,ymin,xmax,ymax)
      implicit real*8 (a-h,o-z)
      dimension xsrc(nsrc),ysrc(nsrc),str(nsrc),tmp(nsrc)
      dimension links(nsrc)
      dimension isrc(mfine,nfine),isrcsz(mfine,nfine)
c
c     The purpose of this subroutine is to sort the source
c     particles in the same box into contiguous memory locations
c
c
      hx=(xmax-xmin)/dble(mfine)
      hy=(ymax-ymin)/dble(nfine)
c
      do 100 i=1,nsrc
        links(i)=-9
100   continue
c
      do 110 j=1,nfine
      do 110 i=1,mfine
        isrc(i,j)=-9
        isrcsz(i,j)=0
110   continue
c
c     Link list the particles in a box and
c     form a population count
c
      do 300 i=1,nsrc
        ind=Int((xsrc(i) - xmin)/hx) + 1
        jnd=Int((ysrc(i) - ymin)/hy) + 1
        if(isrc(ind,jnd) .gt. 0) then
          iptr=isrc(ind,jnd)
          isrc(ind,jnd)=i
          links(i)=iptr
          isrcsz(ind,jnd)=isrcsz(ind,jnd) + 1
        else
          isrc(ind,jnd)=i
          isrcsz(ind,jnd)=isrcsz(ind,jnd) + 1
        endif
300   continue
c
c     Sort the Particles by mucking with the pointers
c
      iptr=0
      do 400 jnd=1,nfine
      do 400 ind=1,mfine
        if(isrc(ind,jnd).gt.0) then
         iptr=iptr+1
         icurr=isrc(ind,jnd)
         isrc(ind,jnd)=iptr
         inext=links(icurr)
         links(icurr)=iptr
350      if(inext .gt. 0) then
            iptr=iptr+1
            icurr=inext
```

```
            inext=links(icurr)
            links(icurr)=iptr
          goto 350
          endif
        endif
400     continue
c
c       Now rearrange the vector elements
c
        do 500 i=1,nsrc
          tmp(links(i))=xsrc(i)
500     continue
        do 510 i=1,nsrc
          xsrc(i)=tmp(i)
510     continue
c
c
        do 600 i=1,nsrc
          tmp(links(i))=ysrc(i)
600     continue
        do 610  i=1,nsrc
          ysrc(i)=tmp(i)
610     continue
c
c
        do 700 i=1,nsrc
          tmp(links(i))=str(i)
700     continue
        do 710  i=1,nsrc
          str(i)=tmp(i)
710     continue
c
c
        return
        end
c
```



```
        subroutine srcput(xsrc,ysrc,str,tmp,nsrc,links)
        implicit real*8 (a-h,o-z)
        dimension xsrc(nsrc),ysrc(nsrc),str(nsrc),tmp(nsrc)
        dimension links(nsrc)
c
c       The purpose of this subroutine is to un-sort the source
c       particles from contiguous memory locations to their
c       original locations
c
        do 100 i=1,nsrc
          tmp(i)=xsrc(links(i))
100     continue
        do 110 i=1,nsrc
           xsrc(i)=tmp(i)
110     continue
c
c
        do 200 i=1,nsrc
          tmp(i)=ysrc(links(i))
200     continue
        do 210 i=1,nsrc
          ysrc(i)=tmp(i)
210     continue
c
c
        do 300 i=1,nsrc
          tmp(i)=str(links(i))
300     continue
        do 310 i=1,nsrc
          str(i)=tmp(i)
310     continue
c
c
        return
        end
c
```

```
      subroutine recsrt(xrec,yrec,tmp,nrec,linkr,irec,
     *                  irecsz,mfine,nfine,xmin,ymin,xmax,ymax)
      implicit real*8 (a-h,o-z)
      dimension xrec(1),yrec(1),tmp(1)
      dimension linkr(1)
      dimension irec(mfine,nfine),irecsz(mfine,nfine)
c
c     The purpose of this subroutine is to sort the receiver
c     particles into contiguous memory locations
c
c
      hx=(xmax-xmin)/dble(mfine)
      hy=(ymax-ymin)/dble(nfine)
c
      do 100 i=1,nrec
        linkr(i)=-9
100   continue
c
      do 110 j=1,nfine
      do 110 i=1,mfine
        irec(i,j)=-9
        irecsz(i,j)=0
110   continue
c
c     Link List And Form a Population Count
c
      do 300 i=1,nrec
        ind=Int((xrec(i) - xmin)/hx) + 1
        jnd=Int((yrec(i) - ymin)/hy) + 1
        if(irec(ind,jnd) .gt. 0) then
          iptr=irec(ind,jnd)
          irec(ind,jnd)=i
          linkr(i)=iptr
          irecsz(ind,jnd)=irecsz(ind,jnd) + 1
        else
          irec(ind,jnd)=i
          irecsz(ind,jnd)=irecsz(ind,jnd) + 1
        endif
300   continue
c
c     Sort the Particles by mucking with the pointers
c
      iptr=0
      do 400 jnd=1,nfine
      do 400 ind=1,mfine
        if(irec(ind,jnd).gt.0) then
         iptr=iptr+1
         icurr=irec(ind,jnd)
         irec(ind,jnd)=iptr
         inext=linkr(icurr)
         linkr(icurr)=iptr
350      if(inext .gt. 0) then
           iptr=iptr+1
           icurr=inext
           inext=linkr(icurr)
```



```
           linkr(icurr)=iptr
           goto 350
         endif
        endif
400   continue
c
c     Now rearrange the vector elements
c
      do 500 i=1,nrec
        tmp(linkr(i))=xrec(i)
500   continue
      do 510 i=1,nrec
        xrec(i)=tmp(i)
510   continue
c
c
      do 600 i=1,nrec
        tmp(linkr(i))=yrec(i)
600   continue
      do 610  i=1,nrec
        yrec(i)=tmp(i)
610   continue
      return
      end
c
```

```
      subroutine recput(xrec,yrec,tmp,nrec,linkr)
      implicit real*8 (a-h,o-z)
      dimension xrec(1),yrec(1),tmp(1)
      dimension linkr(1)
c
c     The purpose of this subroutine is to un-sort the source
c     particles from contiguous memory locations to their
c     original locations
c
      do 100 i=1,nrec
        tmp(i)=xrec(linkr(i))
100   continue
      do 110 i=1,nrec
         xrec(i)=tmp(i)
110   continue
c
c
      do 200 i=1,nrec
        tmp(i)=yrec(linkr(i))
200   continue
      do 210 i=1,nrec
        yrec(i)=tmp(i)
210   continue
c
c
      return
      end
c
```



```
      subroutine cmpput(cmp,tmp,n,link)
      implicit real*8 (a-h,o-z)
      dimension cmp(n),tmp(n)
      dimension link(n)
c
c     The purpose of this subroutine is to un-sort the vector cmp
c     from contiguous memory locations to their
c     original locations
c
      do 100 i=1,n
        tmp(i)=cmp(link(i))
100   continue
      do 110 i=1,n
         cmp(i)=tmp(i)
110   continue
      return
      end
```

```
      program h2test
      implicit real*8 (a-h,o-z)
c
c     This program performs times and tests subroutine
c
c                        helm2
c
c      helm2 = A heirarchical element method based on Poisson's
c              formula.
c
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
c     Standard declarations for variables :
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
c
      real*8 xsrc(5000),ysrc(5000),str(5000)
      real*8 xrec(5000),yrec(5000)
      real*8 tmpr(5000),tmps(5000)
      real*8 pot(5000),velx(5000),vely(5000)
      integer*4 linkr(5000),links(5000)
c
c     Data Structure Parameters and Workspace Allocation
c
      parameter(ngsiza=2000)
      parameter(ngsizb=6000)
      parameter(nintw=20000)
      parameter(nrealw=30000)
      dimension intwk(nintw)
      dimension realwk(nrealw)
c
cxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
c
c
c     test program variables
c
      real*8 potst(5000),velxts(5000),velyts(5000)
c
      external xrand
      real*8 xrand
      character*1 cspeed
c
c     initialize test particles
c
c
c     open(18,file='tinput')
      write(*,*) 'Enter Number of Particles : '
      read(*,*) nsrc
      write(*,*)'Enter Maximum Level of Refinement : '
      read(*,*) mxlev
      write(*,*) ' Enter Correction Radius : '
      read(*,*) icor
      write(*,*)'Enter Tolerance : '
      read(*,*) tol
      write(*,*)'Enter f)ast s)low  or b)oth '
      read(*,10) cspeed
      write(*,*) 'Enter Work Flag 0) pot 1)vel 2) both'
      read(*,*) iwflag
```



```
10    format(a1)
c
c     generate the test vortices in the box
c
      delta=0.01d00
      nrec=nsrc
c
c     construct random source particles
c
      do 100 i=1,nsrc
        xsrc(i)=xrand(0.0d00)*(1.0)
        ysrc(i)=xrand(0.0d00)*(1.00)
        str(i)=xrand(0.0d00) - 0.5d00
100   continue
c
c     set up receiver particles
c
      do 105 i=1,nrec
        xrec(i)=xsrc(i)
        yrec(i)=ysrc(i)
        pot(i)=0.0d00
       velx(i)=0.0d00
       vely(i)=0.0d00
105   continue
c
c     make the strength average the same
c
      avg=0.0d00
      do 110 i=1,nsrc
        avg=avg + str(i)
110   continue
c
c
      do 120 i=1,nsrc
      str(i)=(str(i) - avg/dble(nsrc)) + 1.0d00/dble(nsrc)
120   continue
c
c     call the fast method and time the calculation
c
      if((cspeed.eq.'f') .or. (cspeed.eq.'b')) then
      call eltime(rvala)
      iprint=1
      call helm2(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     * iwflag,pot,velx,vely,delta,iterms,icor,mxlev,
     * tol,linkr,links,tmpr,tmps,
     * ngsiza,ngsizb,nrealw,nintw,realwk,intwk,iprint)
      endif
      call eltime(rvalb)
      write(*,*) 'Multipole Time : ',rvalb-rvala
c
c     error checking - compute the interactions the slow way
c
      if((cspeed.eq.'s') .or. (cspeed.eq.'b')) then
      call eltime(rvala)
      if(iwflag .ne. 1) then
      call potex(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,potst,delta)
```

```
        endif
        if(iwflag .ne. 0) then
        call veloex(xsrc,ysrc,str,nsrc,xrec,yrec,nrec,
     *              velxts,velyts,delta)
        endif
        call eltime(rvalb)
        write(*,*)'Direct Time : ',rvalb-rvala
        endif
c
        if(cspeed .eq. 'b') then
        ermax=0.0d00
        erxmax=0.0d00
        erymax=0.0d00
        do 1000 j=1,nsrc
        if(iwflag.ne.1) then
          err=dabs(potst(j) - pot(j))
          ermax=dmax1(err,ermax)
        endif
        if(iwflag.ne.0) then
          erx=dabs(velxts(j) - velx(j))
          ery=dabs(velyts(j) - vely(j))
          erxmax=dmax1(erx,erxmax)
          erymax=dmax1(ery,erymax)
        endif
1000    continue
        if(iwflag.ne.1) then
        write(*,900) ermax
900     format(1x,' Error Pot : ',e15.6,//)
        endif
        if(iwflag.ne.0) then
        write(*,901) erxmax,erymax
901     format(1x,' Err Velx : ',e15.6,' Err Vely : ',e15.6)
        endif
c
c
        endif
        stop
        end
c
```



```
        double precision function xrand(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.0.0d00) go to 10
        if (r.gt.0.0d00) 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      xrand = dble(ix1*2048 + ix0)
        xrand = xrand / 4194304.d00
        return
c
20      ix1 = dmod(r,1.d00)*4194304.d00 + 0.5d00
        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     Mac II timing Call
c
c     call time(ival)
c
c     rval=dble(ival)
c     return
c
c
c     Unix timing Call
c
c
      call etime(tarray)
      rval=dble(tarray(1))
      return
c
      end
```

```
      program h2tmng
      implicit real*8 (a-h,o-z)
c
c     This program performs timing of the various routines
c     used in  helm2  - a heierarchical element method based
c     on Poisson's formula.
c
c     This program calls subroutines which call the
c     various helm2 routines with dummy parameters.
c
c     This code should be linked to helm2 subroutines
c     used so that accurate timing results.
c
c     The parameters used to call the timing subroutines can
c     be changed. They should be chosen so that the coefficients
c     for the work estimate are reliably estimated. One can get
c     an idea of the reliability of the coefficients by looking
c     at the output file htime.dat which contains an estimate for each
c     number of timing samples taken. If the estimate appears
c     constant then that is a good sign.
c
c     The result of each call is an alpha and beta - to be used
c     in a work estimate of time = (alpha/scale)*x + beta
c     where the variable x is the appropriate count of the
c     number of particles, terms etc.
c
c
      real*8 alpha,beta
      integer*4 repeat(10)
c
      integer*4 prtbeg,prtend,prtjmp
      integer*4 trmbeg,trmend,trmjmp
c
c     htime.dat contains the timing data used to
c     estimate the coefficients - i.e. the timings of the
c     routines for the range of parameters specified.
c
c     htime.con contains the estimated timing constants
c
      open(10,file='htime.dat')
      open(11,file='htime.con')
c
c
c
c
c    a)   1) fstpol - particle to outer ring
c    b)   2) mrgpol - outer ring to outer ring
c    c)   3) polpow - outer ring to inner ring
c    d)   4) powpow - inner ring to inner ring
c    e)   5) potex  - exact potential calculation (n^2 method)
c    f)   6) velex  - exact velocity calculation  (n^2 method)
c    g)   7) powevl - evaluation of potential approximation.
c    h)   8) powdrv - evaluation of the derivatives of the potential
c                     approximation.
c
c
```



```
c     The following vales determine how many instances of the
c     particular operation are to be performed. They should
c     be chosen large enough so that the constants are well
c     determined (which can be seen from an analysis of htime.dat)
c     but small enough so that you don't have to spend excessive
c     amounts of CPU time to obtain them.
c
      repeat(1)=80
      repeat(2)=80
      repeat(3)=80
      repeat(4)=80
      repeat(5)=80
      repeat(6)=80
      repeat(7)=80
      repeat(8)=80
c
c     The following parameters determine the range and step size
c     for the number of particles used in the constant estimation
c
c     prtbeg = beginning number of particles
c     prtend = ending number of particles
c     prtjmp = particle number increment used in loop
      prtbeg=40
      prtend=200
      prtjmp=40
c
c     The following parameters determine the beginning, ending and
c     jump of the number of terms used in the hierarchical element
c     approximations.
c
      trmbeg=8
      trmend=14
      trmjmp=2
c
c     The folling factor is a scale factor for the timing
c     constants so that they are more convienient to
c     deal with.
c
      scale=1000.0d00
c
c
      write(11,5) scale
5     format(1x,'       scale=',d14.6)
c
c
c     Call to time FSTPOL - source particles to outer ring approx.
c
c     ntimes=the number of times the routine is called for
c     a fixed number of terms and particles.
c
      ntimes=repeat(1)
      call tmsega(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *            trmend,trmjmp,alpha,beta)
      write(11,10) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
```

```
      write(11,11) beta

10    format(1x,'      tcon(1,1)=',e14.6)
11    format(1x,'      tcon(1,2)=',e14.6)
c
c     Call to time VALPOL - outer ring to outer ring shift.
c
c
      ntimes=repeat(2)
      call tmsegb(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
      write(11,20) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,21) beta
20    format(1x,'      tcon(2,1)=',e14.6)
21    format(1x,'      tcon(2,2)=',e14.6)
c
c
c
c     Call to time POLPOW - outer ring to inner ring shifting.
c
c
      ntimes=repeat(3)
      call tmsegc(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
      write(11,30) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,31) beta
30    format(1x,'      tcon(3,1)=',e14.6)
31    format(1x,'      tcon(3,2)=',e14.6)
c
c     Call to time POWPOW - inner ring to inner ring shifting.
c
      ntimes=repeat(4)
      call tmsegd(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
      write(11,40) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,41) beta
40    format(1x,'      tcon(4,1)=',e14.6)
41    format(1x,'      tcon(4,2)=',e14.6)
c
c
c
c     Call to time POTEX - n^2 potential evaluation
c
      ntimes=repeat(5)
      call tmsege(ntimes,scale,prtbeg,prtend,prtjmp,alpha,beta)
      write(11,60) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,61) beta
60    format(1x,'      tcon(5,1)=',e14.6)
61    format(1x,'      tcon(5,2)=',e14.6)
c
c     Call to time VELOEX - n^2 velocity evaluation
c
      ntimes=repeat(6)
      call tmsegf(ntimes,scale,prtbeg,prtend,prtjmp,alpha,beta)
      write(11,70) alpha
```



```
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,71) beta
70    format(1x,'      tcon(6,1)=',e14.6)
71    format(1x,'      tcon(6,2)=',e14.6)
c
c     Call to time POWEVL - inner ring potential evaluation
c
      ntimes=repeat(7)
      call tmsegg(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *            trmend,trmjmp,alpha,beta)
      write(11,100) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,101) beta
100   format(1x,'      tcon(7,1)=',e14.6)
101   format(1x,'      tcon(7,2)=',e14.6)
c
c     Call to time POWDRV - inner ring potential derivative calculation
c
      ntimes=repeat(8)
      call tmsegh(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *            trmend,trmjmp,alpha,beta)
      write(11,110) alpha
      if(beta .lt. 0.0d00) beta=0.0d00
      write(11,111) beta
110   format(1x,'      tcon(8,1)=',e14.6)
111   format(1x,'      tcon(8,2)=',e14.6)
      stop
      end
c
```

```
      subroutine tmsega(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *                  trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This subroutine times fstpol
c
      real*8 resul(2,100)
      integer*4 prtbeg,prtend,prtjmp
      integer*4 trmbeg,trmend,trmjmp
c
      real*8 xsrc(1000),ysrc(1000),str(1000)
c
      real*8 v(101,2)
      real*8 polval(100)
      integer*4 ipole(1)
      integer*4 isrc(1),isrcsz(1)
c
c     set up constants
c
      nxa=1
      nya=1
      nxb=1
      nyb=1
      hx=1.0d00
      hy=1.0d00
      xmin=0.0d00
      ymin=0.0d00
      ipole(1)=1
      isrc(1)=1
      nvals=100
      write(10,950)
950   format(1x,'Timing Data from fstpol')
      write(10,*) '   '
      write(10,951)
951   format(1x,'Particle -> Outer Ring Construction')
      write(10,*) ' '
c
c     Loop over iterms and the number of particles
c
      icnt=0
      n1=(trmend-trmbeg)/trmjmp
      n2=(prtend-prtbeg)/prtjmp
c
      do 1000 K=1,n1+1
      iterms=trmbeg + (K-1)*trmjmp

c
c     compute integration points for the circle
c
      pi=3.14159265359d00
      pi2=2.0d00*pi
      itot=2*iterms +1
      dtheta=pi2/dble(itot)
      do 10 i=1,itot
```



```
       theta=dble(i-1)*dtheta
       v(i,1)=dcos(theta)
       v(i,2)=dsin(theta)
10    continue
c
      do 1000 L=1,n2+1
      icnt=icnt+1
      npart=prtbeg + (L-1)*prtjmp
c
c     compute the test points
c
      do 30 i=1,npart
       xsrc(i)=.5d00
       ysrc(i)=.5d00
       str(i)=0.0d00
30    continue
      isrcsz(1)=npart
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call fstpol(nxa,nya,ipole,nxb,nyb,isrc,isrcsz,
     * xmin,ymin,hx,hy,npart,xsrc,ysrc,str,polval,iterms,v)
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(itot*npart)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
c
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegb(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times the routine mrgpol in my
c     fast multipole code.
c
c
      real*8 resul(2,100)
      integer*4 trmbeg,trmend,trmjmp
c
c
      real*8 v(101,2),cn(101,3),sn(101,3)
c
      real*8 polval(200)
      integer*4 iua(1)
      integer*4 iub(1)
c
c     set up constants
c
      nxa=1
      nya=1
      nxb=1
      nyb=1
      hxa=1.0d00
      hya=1.0d00
      hxb=0.5d00
      hyb=0.5d00
      xmin=0.0d00
      ymin=0.0d00
      iua(1)=100
      iub(1)=1
      write(10,950)
950   format(1x,'Timing Data from mrgpol')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Outer Ring -> Outer Ring ')
      write(19,*)' '
c
c     Loop over iterms
c
      icnt=0
      Nl=(trmend-trmbeg)/trmjmp
c
      do 1000 K=1,Nl+1
      iterms=trmbeg + (K-1)*trmjmp
      itot=2*iterms + 1
      icnt=icnt+1
c
c
c     compute integration points for the circle
c     and precomputed constants
c
      call makcon(iterms,v,cn,sn)
c
```



```
      do 15 i=1,200
       polval(i)=0.0d00
15    continue
c
c     compute the test points
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call mrgpol(nxa,nya,iua,nxb,nyb,iub,
     * xmin,ymin,hxa,hya,hxb,hyb,polval,iterms,v,cn,sn)
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(itot*itot)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegc(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times the routine polpow in my
c     fast multipole code.
c
c
      real*8 resul(2,100)
      integer*4 trmbeg,trmend,trmjmp
c
c
      real*8 pol(100),pow(100)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
c     set up constants
c
      xpole=0.0d00
      ypole=0.0d00
      rpole=0.25d00
      xchld=0.0d00
      ychld=1.0d00
      rchld=0.25d00
      write(10,950)
950   format(1x,'Timing Data from polpow')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Outer Ring -> Inner Ring')
      write(10,*) ' '
c
c     Loop over iterms
c
      icnt=0
      Nl=(trmend-trmbeg)/trmjmp
c
      do 100 i=1,100
      pol(i)=0.0d00
      pow(i)=0.0d00
100   continue
c
      do 1000 K=1,Nl+1
      iterms=trmbeg + (K-1)*trmjmp
      itot=2*iterms + 1
      icnt=icnt+1

c
c     compute integration points for the circle
c     and precomputed constants
c
      call makcon(iterms,v,cn,sn)
c
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call polpow(xpole,ypole,rpole,pol(1),xchld,
```



```
     *  ychld,rchld,pow(1),iterms,v,cn,sn)
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(itot*itot)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegd(ntimes,scale,trmbeg,trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times the routine powpow in my
c     fast multipole code.
c
c
      real*8 resul(2,100)
      real*8 v(101,2),cn(101,3),sn(101,3)
      integer*4 trmbeg,trmend,trmjmp
c
c
      real*8 powa(100),powb(100)
c
c     set up constants
c
      xpole=0.0d00
      ypole=0.0d00
      rpole=0.25d00
      xchld=0.0d00
      ychld=1.0d00
      rchld=0.25d00
      write(10,950)
950   format(1x,'Timing Data from powpow')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Inner Ring-> Inner Ring')
      write(10,*) ' '
c
c     Loop over iterms
c
      icnt=0
      N1=(trmend-trmbeg)/trmjmp
      do 100 i=1,100
      powa(i)=0.0d00
      powb(i)=0.0d00
100   continue
c
      do 1000 K=1,N1+1
      iterms=trmbeg + (K-1)*trmjmp
      itot=2*iterms + 1
      icnt=icnt+1
c
c     compute integration points for the circle
c     and precomputed kernel quantities
c
      call makcon(iterms,v,cn,sn)
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call powpow(xpole,ypole,rpole,powa(1),xchld,
     *  ychld,rchld,powb(1),iterms,v,cn,sn)
500   continue
```



```
      call eltime(rvalb)
      resul(1,icnt)=dble(itot*itot)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsege(ntimes,scale,prtbeg,prtend,prtjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times potex - the direct potential code
c
c
      real*8 resul(2,100)
      integer*4 prtbeg,prtend,prtjmp
c
      real*8 xl(1000),yl(1000),str(1000)
      real*8 xtst(1000),ytst(1000),pot(1000)
c
c     set up constants
c
      delta=.1d00
      write(10,950)
950   format(1x,'Timing Data from potex')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Particle -> Particle Potential Calc. ')
      write(10,*) ' '
c
c
c     Loop over number of receiver and test particles
c
      icnt=0
      n2=(prtend-prtbeg)/prtjmp
c
      do 1000 K=1,n2+1
      npart=prtbeg + (K-1)*prtjmp
      ntst=prtbeg + (K-1)*prtjmp
      icnt=icnt+1
c
c     compute the test points
c
      do 30 i=1,npart
       xl(i)=.25d00
       yl(i)=.25d00
       str(i)=0.0d00
30    continue
c
      do 40 i=1,ntst
       xtst(i)=0.75d00
       ytst(i)=0.75d00
       pot(i)=0.0d00
40    continue
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call potex(xl,yl,str,npart,xtst,ytst,ntst,pot,delta)
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(ntst*npart)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
```



```
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegf(ntimes,scale,prtbeg,prtend,prtjmp,alpha,beta)
c
c
      implicit real*8 (a-h,o-z)
c
c     This program times veloex - the direct velocity code
c
c
      real*8 resul(2,100)
      integer*4 prtbeg,prtend,prtjmp
c
      real*8 xl(4000),yl(4000),str(4000)
      real*8 xtst(4000),ytst(4000),velx(4000),vely(4000)
c
c     set up constants
c
      delta=.1d00
      write(10,950)
950   format(1x,'Timing Data from veloex')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Particle -> Particle Velo. Calc.')
      write(10,*) ' '
c
c     Loop over number of receiver and test particles
c
      icnt=0
      N2=(prtend-prtbeg)/prtjmp
c
      do 1000 K=1,N2+1
      npart=prtbeg + (K-1)*prtjmp
      ntst=prtbeg + (K-1)*prtjmp
      icnt=icnt+1
c
c     compute the test points
c
      do 30 i=1,npart
       xl(i)=.25d00
       yl(i)=.25d00
       str(i)=0.0d00
30    continue
c
      do 40 i=1,ntst
       xtst(i)=0.75d00
       ytst(i)=0.75d00
       velx(i)=0.0d00
       vely(i)=0.0d00
40    continue
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      call veloex(xl,yl,str,npart,xtst,ytst,ntst,velx,
```



```
     *            vely,delta)
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(ntst*npart)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegg(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *                  trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times powevl - evaluation of the local power
c     expansion
c
c
      real*8 resul(2,100)
      integer*4 prtbeg,prtend,prtjmp
      integer*4 trmbeg,trmend,trmjmp
c
      real*8 xtst(1000),ytst(1000)
c
      real*8 powval(100)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
      write(10,950)
950   format(1x,'Timing Data from potevl')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Inner Ring-> Particle Pot. Calc.')
      write(10,*) ' '
c
c     Loop over number of test particles and iterms
c
      nl=(trmend-trmbeg)/trmjmp
      n2=(prtend-prtbeg)/prtjmp
      xpow=0.0d00
      ypow=0.0d00
      rpow=0.5d00
      icnt=0
c
      do 1000 K=1,nl+1
      iterms=trmbeg + (K-1)*trmjmp
      itot=2*iterms + 1
c
c
      do 10 i=1,itot
       powval(i)=0.0d00
10    continue
c
      do 1000 L=1,n2+1
      ntst=prtbeg + (L-1)*prtjmp
      icnt=icnt+1
c
c     compute the test points
c
      do 40 i=1,ntst
       xtst(i)=0.25d00
       ytst(i)=0.25d00
40    continue
c
c     get precomputed kernel constants
```



```
c
      call makcon(iterms,v,cn,sn)
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      do 400 i=1,ntst
       val = powevl(xtst(i),ytst(i),itot,powval(1),
     *                xpow,ypow,rpow,cn,sn)
400   continue
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(ntst*itot)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine tmsegh(ntimes,scale,prtbeg,prtend,prtjmp,trmbeg,
     *                  trmend,trmjmp,alpha,beta)
c
      implicit real*8 (a-h,o-z)
c
c     This program times powdrv - evaluation of the local power
c     expansion derivatives
c
c
      real*8 resul(2,100)
      integer*4 prtbeg,prtend,prtjmp
      integer*4 trmbeg,trmend,trmjmp
c
      real*8 xtst(1000),ytst(1000)
c
      real*8 powval(100)
      real*8 v(101,2),cn(101,3),sn(101,3)
c
      write(10,950)
950   format(1x,'Timing Data from potdrv')
      write(10,*) ' '
      write(10,951)
951   format(1x,'Inner Ring -> Particle Pot. Deriv. Calc. ')
      write(10,*) ' '
c
c     Loop over number of test particles and iterms
c
      n1=(trmend-trmbeg)/trmjmp
      n2=(prtend-prtbeg)/prtjmp
      xpow=0.0d00
      ypow=0.0d00
      rpow=0.5d00
      icnt=0
c
      do 1000 K=1,n1+1
      iterms=trmbeg + (K-1)*trmjmp
      itot=2*iterms + 1
c
c
      do 10 i=1,itot
       powval(i)=0.0d00
10    continue
c
      do 1000 L=1,n2+1
      ntst=prtbeg + (L-1)*prtjmp
      icnt=icnt+1
c
c     compute the test points
c
      do 40 i=1,ntst
       xtst(i)=0.25d00
       ytst(i)=0.25d00
40    continue
c
c     get precomputed kernel constants
```



```
c
      call makcon(iterms,v,cn,sn)
c
      call eltime(rvala)
      do 500 iii=1,ntimes
      do 400 i=1,ntst
      call powdrv(xtst(i),ytst(i),itot,powval(1),
     *                  xpow,ypow,rpow,cn,sn,powdx,powdy)
400   continue
500   continue
      call eltime(rvalb)
      resul(1,icnt)=dble(ntst*itot)/scale
      resul(2,icnt)=(rvalb-rvala)/dble(ntimes)
c
      if(icnt.eq.1) then
      alpha=resul(2,icnt)/resul(1,icnt)
      beta=0.0d00
      endif
c
      if(icnt.gt.1) then
      call fit(icnt,resul,alpha,beta)
      endif
c
      write(10,*) 'alpha = ',alpha,' beta = ',beta
1000  continue
c
      write(10,*) ' '
      write(10,*) ' Function Values '
      write(10,*) ' '
      do 2000 i=1,icnt
      write(10,1500) resul(1,i),resul(2,i)
1500  format(1x,e15.6,2x,e15.6)
2000  continue
      write(10,*) ' '
      return
      end
c
```

```
      subroutine fit(nsamp,resul,alpha,beta)
      implicit real*8 (a-h,o-z)
c
c     This subroutine computes the linear least
c     squares fit of the form y = alpha + beta*x
c     for the data in resul.
c
      real*8 resul(2,100)
      real*8 s(2,2),sinv(2,2)
      real*8 b(2)
      real*8 alpha,beta
c
c     form the normal matrix
c
      sum=0.0d00
      do 10 i=1,nsamp
       sum=sum + resul(1,i)*resul(1,i)
10    continue
      s(1,1)=sum
c
      sum=0.0d00
      do 20 i=1,nsamp
       sum=sum + resul(1,i)
20    continue
      s(1,2)=sum
      s(2,1)=sum
      s(2,2)=dble(nsamp)
c
c     construct right hand side
c
      b(1)=0.0d00
      b(2)=0.0d00
      do 30 i=1,nsamp
       b(1)=b(1) + resul(2,i)*resul(1,i)
       b(2)=b(2) + resul(2,i)
30    continue
c
c     solve the normal equations
c
      det=s(1,1)*s(2,2) - s(1,2)*s(2,1)
      sinv(1,1)=s(2,2)/det
      sinv(2,2)=s(1,1)/det
      sinv(1,2)=-s(1,2)/det
      sinv(2,1)=-s(2,1)/det
c
      alpha=sinv(1,1)*b(1) + sinv(1,2)*b(2)
      beta= sinv(2,1)*b(1) + sinv(2,2)*b(2)
c
      return
      end
c
```



```
      subroutine eltime(rval)
      implicit real*8 (a-h,o-z)
      real*4 tarray(2)
c
c     Mac II timing Call
c
c     call time(ival)
c     rval=dble(ival)
c     return
c
c
c     Unix timing Call
c
c     This subroutine gets the current time in seconds and returns
c     it as a real variable
c
      call etime(tarray)
      rval=dble(tarray(1))
      return
      end
```