```fortran
c-----------------------------------------------------------------
c     Data structure for FAS multi-grid program
c-----------------------------------------------------------------

c-----------------------------------------------------------------
c     Main storage array; all arrays used in MG solve are
c     "allocated" from 'q'.  The integer 'qptr' maintains
c     the index into 'q' representing the "high-water mark"
c     of allocation. See routine 'gsinit()'.
c
c     Make sure that this array is "sensibly" dimensioned
c     for the machine you are running on.  On SGI machines
c     you can type 'hinv' to find out how much physical
c     memory is available; the limit for regular Fortran
c     storage will less than or equal to physical memory
c     and sometimes substantially less.  On einstein
c     the limit *is* the amount of physical memory,
c     but you should avoid running with that much.
c
c     Also note that 2**20 = 1M, so that 2**20 real*8
c     variables is 8M of storage.
c-----------------------------------------------------------------
      integer       qsize
      parameter   ( qsize = 2**20 )

      real*8        q(qsize)
      common      / comr8 /      q

      integer       qptr
      common      / comi4 /      qptr

      integer       lmax
      parameter   ( lmax = 12 )
```

```fortran
c-------------------------------------------------------------
c     Pointers for various grid functions.   Once these
c     arrays have been initialized (see routine 'gsinit'),
c     q(u(l)), for example, is the start of storage for
c     the level-l 'unknown' grid function.  This
c     implementation uses temporary arrays ('t1', 't2', 't3')
c     heavily in order to  keep the code relatively
c     straightforward.  A production code (particularly a
c     3-d code) should be more economical of storage.
c-------------------------------------------------------------
      integer       u(lmax),     rhs(lmax),     tau(lmax),
     &              t1(lmax),    t2(lmax),      t3(lmax)

      common      / comi4 /
     &              u,           rhs,           tau,
     &              t1,          t2,            t3

c-------------------------------------------------------------
c     Mesh spacings and grid sizes.
c-------------------------------------------------------------
      real*8        h(lmax)
      common     /  comr8 /      h

      integer       nx(lmax),    ny(lmax)
      common      / comi4 /
     &              nx,          ny

c-------------------------------------------------------------
c     Coarse grid convergence criterion and iteration limit
c-------------------------------------------------------------
      real*8        epsi1
      parameter   ( epsi1  = 1.0d-10 )

      integer       maxsweep1
      parameter   ( maxsweep1 = 20 )
```

```
c-----------------------------------------------------------
c    'mgfas2d' and supporting routines constitute a complete
c    FAS multi-grid solver for
c
c       u_xx + u_yy + G(u) = rhs(x,y)       (1)
c
c     on the unit square with homogenous Dirichlet boundary
c     conditions.  Literature (particularly Brandt) should be
c     consulted for treatment of Neumann and mixed boundary
c     conditions.
c
c     Currently program solves (1) with
c
c        G(u) = sigma * u*u
c
c     and 'rhs' specified so that
c
c        u = sin(Pi lx x) sin(Pi ly y)
c
c     for some integers, 'lx' and 'ly' defined in
c     'mgprob.inc'.
c-----------------------------------------------------------
      program      mgfas2d

      implicit     none
c-----------------------------------------------------------
c     Functions called
c-----------------------------------------------------------
      integer      iargc,      i4arg
      real*8       r8arg,      dmnrm2
c-----------------------------------------------------------
c     Command-line arguments
c-----------------------------------------------------------
      integer      level,      nvcycle,    preswp,    pstswp
c-----------------------------------------------------------
c     'mg.inc' contains the main 'data structure' used
c     in this implementation.
c-----------------------------------------------------------
      include     'mg.inc'
```

```
c-----------------------------------------------------------
c     'mgprob.inc' contains problem-specific parameters
c     including command-line arguments 'sigma' and
c     'gft_trace'.
c-----------------------------------------------------------
      include     'mgprob.inc'
c-----------------------------------------------------------
c     Locals.
c-----------------------------------------------------------
      integer      l,          cycle
c-----------------------------------------------------------
c     Argument parsing.
c
c     level:     Finest level solution requested.  Full MG
c                process solves sequence of problems
c                l = 1 ... level (n_l = 2^l+1 pts per side)
c     sigma:     Controls nonlinearity in PDE (1)
c     nvcycle:   # of vcycles to apply to each problem
c     preswp:    # of relaxation sweeps before a CGC
c     pstswp:    # of relaxation sweeps after a CGC
c     gft_trace: If present, enables RNPL-style .hdf
c                output of solution estimate 'u' after
c                each relaxation sweep.
c-----------------------------------------------------------
      if( iargc() .lt. 2 ) go to 900
      level     = i4arg(1,-1)
      if( level .le. 0 ) go to 900
      if( level .gt. lmax ) then
         write(0,*) 'mgfas2d: Specified level is too high'
         write(0,*) 'mgfas2d: Current maximum level: ', lmax
         stop
      end if
      sigma     = r8arg(2,0.0d0)

      nvcycle   = i4arg(3,1)
      preswp    = i4arg(4,1)
      pstswp    = i4arg(5,1)
      gft_trace = iargc() .gt. 5

      if( gft_trace ) write(0,*) 'gft tracing enabled'
```

```
c-----------------------------------------------------------
c     Initialize data structure.  All grid function arrays
c     are allocated from global array 'q' (see 'mg.inc')
c-----------------------------------------------------------
      call gsinit(level)

c-----------------------------------------------------------
c     Solve series of problems from l = 1 through
c     l = level.
c-----------------------------------------------------------
      do l = 1 , level
         if( l .eq. 1 ) then
c-----------------------------------------------------------
c           Initialize level 1 solution (coarsest) with 0.
c-----------------------------------------------------------
            call dmls  ( q(u(l)), 0.0d0, nx(l), ny(l) )
         else
c-----------------------------------------------------------
c           Initialize level l soln via linear interpolation
c           of level l-1 soln.
c-----------------------------------------------------------
            call dmlint ( q(u(l-1)), q(u(l)),
     &                    nx(l-1), ny(l-1) )
         end if
c-----------------------------------------------------------
c        Initialize 'rhs' (rhs = f).
c-----------------------------------------------------------
         call clcrhs2   ( q(rhs(l)), h(l), nx(l), ny(l) )
c-----------------------------------------------------------
c        "Solve" level l problem by applying cycle
c        (preswp,pstswp) V-cycles.
c-----------------------------------------------------------
         do cycle = 1 , nvcycle
            call vcycle(l,cycle,preswp,pstswp)
         end do
      end do
```

```
c-----------------------------------------------------------
c     Compute norm of error on finest gird and output to
c     standard error.
c-----------------------------------------------------------
      l = level
      call clcu2 ( q(t1(l)), h(l), nx(l), ny(l) )
      call dmms  ( q(t1(l)), q(u(l)), q(t1(l)),
     &             nx(l), ny(l) )

      write(0,1000) dmnrm2( q(t1(l)), nx(l), ny(l) )
1000  format(' || Error ||: ',1p,e12.4)

c-----------------------------------------------------------
c     Close RNPL-style .hdf files if option enabled.
c-----------------------------------------------------------
      if( gft_trace ) then
         call gft_close_all()
      end if

c-----------------------------------------------------------
c     Normal exit.
c-----------------------------------------------------------
      stop

c-----------------------------------------------------------
c     Usage exit.
c-----------------------------------------------------------
 900  continue
         write(0,*) 'usage: mgfas2d <level> <sigma> '//
     &              '[<nvcycle> <preswp> <pstswp> <gft_trace>]'
      stop

      end
```

```
c-----------------------------------------------------------
c     Performs one FAS V-cycle of level 'l' problem, using
c     'preswp' pre-CGC and 'pstswp' pst-CGC sweeps.
c     Supply cycle=N for Nth consecutive invocation of this
c     routine on the same level.
c-----------------------------------------------------------
      subroutine vcycle(l,cycle,preswp,pstswp)

         implicit       none

         include       'mg.inc'

         integer        l,         cycle,    preswp,   pstswp

         real*8         dmnrm2,    relaxc2

         integer        m,         sweep
         real*8         rresnrm

c-----------------------------------------------------------
c        Work from fine level to coarse level, performing
c        pre-CGC sweeps and intitiating coarse grid
c        corrections.  Skipped if on coarsest level, l = 1.
c-----------------------------------------------------------
         do m = l , 2 , -1
c-----------------------------------------------------------
c           Pre-CGC relaxation sweeps.
c-----------------------------------------------------------
            if( cycle .eq. 1  .or.  m .ne. l ) then
               do sweep = 1 , preswp
                  rresnrm = relaxc2(q(u(m)),q(rhs(m)),
     &                              nx(m),ny(m),h(m))
                  call gft_out(m)
                  call trace(m,'Pre-CGC',rresnrm)
               end do
            end if
```

```
c-----------------------------------------------------------
c           Compute truncation error estimate.
c
c               2h       2h 2h h      2h h  h
c           tau    :=  L  I  u  -  I  L  u
c               h         h        h
c
c           As pointed out in 'mg.inc', this implementation
c           uses three temporary arrays for clarity (:-));
c           production implementation should economize.
c-----------------------------------------------------------

c-----------------------------------------------------------
c             2h         2h 2h h
c           t2     :=  L  I  u
c                          h
c-----------------------------------------------------------
            call dmrshw ( q(u(m)), q(t1(m-1)),
     &                    nx(m-1), ny(m-1) )
            call lop    ( q(t2(m-1)), q(t1(m-1)),
     &                    nx(m-1), ny(m-1), h(m-1) )

c-----------------------------------------------------------
c             2h         2h h  h
c           t3     :=  I  L  u
c                       h
c-----------------------------------------------------------
            call lop    ( q(t1(m)), q(u(m)),
     &                    nx(m), ny(m), h(m) )
            call dmrshw ( q(t1(m)), q(t3(m-1)),
     &                    nx(m-1),ny(m-1) )

c-----------------------------------------------------------
c               2h        2h        2h
c           tau    :=  t2    -  t3
c               h
c-----------------------------------------------------------
            call dmms   ( q(t2(m-1)), q(t3(m-1)), q(tau(m-1)),
     &                    nx(m-1), ny(m-1) )
```

```
c-----------------------------------------------------------
c          Define right hand side and initialize unknown
c          for coarse grid problem.
c
c              2h       2h    h       2h
c          rhs     :=  I   rhs   + tau
c                       h             h
c
c           2h         2h  h
c          u       := I   u
c                      h
c-----------------------------------------------------------
           call dmrshw ( q(rhs(m)), q(rhs(m-1)),
     &                   nx(m-1),ny(m-1))
           call dmma   ( q(rhs(m-1)), q(tau(m-1)), q(rhs(m-1)),
     &                   nx(m-1), ny(m-1) )

           call dmrshw ( q(u(m)), q(u(m-1)),
     &                   nx(m-1), ny(m-1) )

c-----------------------------------------------------------
c          Continue loop to perform coarse-grid
c          correction.
c-----------------------------------------------------------
        end do
```

```
c-----------------------------------------------------------
c       Solve coarse-grid problem using relaxation.
c       Note use of 'do while()'
c-----------------------------------------------------------
        rresnrm = 1.0d0 + epsi1
        sweep = 1
        do while( rresnrm .gt. epsi1  .and.
     &            sweep .le. maxsweep1 )
           rresnrm = relaxc2( q(u(1)), q(rhs(1)),
     &                        nx(1), ny(1), h(1) )
           sweep = sweep + 1
           call gft_out(1)
           call trace(1,'Coarse grid',rresnrm)
        end do
        if( rresnrm .gt. epsi1 ) then
           write(0,*) '*** WARNING vcycle: Coarse grid '//
     &                'relaxation failed.'
        end if
```

```
c-----------------------------------------------------------
c       Now work from coarse to fine level, updating
c       unknowns and performing post-CGC relaxtion
c       sweeps.  Skipped if l = 1.
c-----------------------------------------------------------
        do m = 2 , l
c-----------------------------------------------------------
c          Linearly interpolate correction and update
c          level m unknown.
c
c           h       h    h    2h     2h  h
c          u    := u  + I  ( u   -  I   u  )
c                        2h          h
c-----------------------------------------------------------
           call dmrshw ( q(u(m)), q(t1(m)),
     &                   nx(m-1), ny(m-1) )
           call dmms   ( q(u(m-1)), q(t1(m)), q(t1(m)),
     &                   nx(m-1), ny(m-1) )
           call dmlint ( q(t1(m)), q(t2(m)),
     &                   nx(m-1),ny(m-1) )
           call dmma   ( q(u(m)), q(t2(m)), q(u(m)),
     &                   nx(m),ny(m) )
c-----------------------------------------------------------
c          Post-CGC relaxation sweeps.
c-----------------------------------------------------------
           do sweep = 1 , pstswp
              rresnrm = relaxc2 ( q(u(m)), q(rhs(m)),
     &                            nx(m),ny(m),h(m) )
              call gft_out(m)
              call trace(m,'Post-CGC',rresnrm)
           end do
         end do

c-----------------------------------------------------------
c        End of the V-cycle.
c-----------------------------------------------------------
        write(0,*)

        return
      end
```

```
c===========================================================
c     Performs red-black GS relaxation sweep of 2d model
c     problem and returns norm of running residuals.
c===========================================================
      double precision function relaxc2(u,rhs,nx,ny,h)
         implicit       none
         include       'mgprob.inc'

         integer        nx,               ny
         real*8         u(nx,ny),         rhs(nx,ny)
         real*8         h
         real*8         hm2,      m4hm2,    rresl
         integer        i,        j,
     &                  isw,      jsw,      pass

         relaxc2 =  0.0d0
         hm2     =  1.0d0 / (h * h)
         m4hm2   = -4.0d0 * hm2

         jsw = 1
          do pass = 1 , 2
            isw = jsw
            do j = 2 , ny - 1
               do i = isw + 1 , nx - 1, 2
                  rresl = hm2 * (u(i+1,j) + u(i-1,j) +
     &                         u(i,j+1) + u(i,j-1) -
     &                         4.0d0 * u(i,j)) +
     &                  sigma * u(i,j)**2 -
     &                  rhs(i,j)
                  u(i,j) = u(i,j) - rresl  /
     &                      (m4hm2 + 2.0d0 * sigma * u(i,j))
                  relaxc2 = relaxc2 + rresl * rresl
               end do
               isw = 3 - isw
            end do
            jsw = 3 - jsw
         end do
         relaxc2 = sqrt(relaxc2 / (nx * ny))
         return
      end
```

```fortran
c===========================================================
c    Computes
c               h  h
c      lopu := L  u
c===========================================================
      subroutine lop(lopu,u,nx,ny,h)

         implicit       none

         include       'mgprob.inc'

         integer        nx,       ny
         real*8         lopu(nx,ny),   u(nx,ny)
         real*8         h

         integer        i,       j
         real*8         hm2

         hm2 = 1.0d0 / (h * h)

         do j = 2 , ny - 1
            do i = 2 , nx - 1
               lopu(i,j) = hm2 * (u(i+1,j) + u(i-1,j) +
     &                           u(i,j+1) + u(i,j-1) -
     &                           4.0d0*u(i,j)) +
     &                           sigma * u(i,j)**2
            end do
         end do
         call dmlbs(lopu,0.0d0,nx,ny)

         return
      end
```

```
c===========================================================
c     2--D half-weighted 2:1 restriction with injection
c     of border values.
c===========================================================
      subroutine dmrshw(uf,uc,nxc,nyc)

         implicit       none

         integer        nxc,               nyc
         real*8         uf(2 * nxc - 1, 2 * nyc - 1),
     &                  uc(nxc,nyc)

         integer        ic,       jc,       if,       jf

         do jc = 1 , nyc
            jf = 2 * jc - 1
            do ic = 1 , nyc
               if = 2 * ic - 1
               uc(ic,jc) = 0.5d0 * (uf(if,jf) + 0.25d0 *
     &                     (uf(if+1,jf) + uf(if-1,jf) +
     &                      uf(if,jf+1) + uf(if,jf-1)))
            end do
         end do
         do ic = 1 , nxc
            uc(ic,1  ) = uf(2 * ic - 1,1  )
            uc(ic,nyc) = uf(2 * ic - 1,2 * nyc - 1)
         end do
         do jc = 1 , nyc
            uc(1  ,jc) = uf(1,  2 * jc - 1)
            uc(nxc,jc) = uf(2 * nxc - 1,2 * jc - 1)
         end do

         return

      end
```

```
c===========================================================
c     2 : 1 linear interpolation of coarse 2--function 'uc'
c     to fine function 'uf' (prolongation operator).
c===========================================================
      subroutine dmlint(uc,uf,nxc,nyc)

         implicit       none

         integer        nxc,      nyc
         real*8         uc(nxc,nyc),
     &                  uf(2 * nxc - 1,2 * nyc - 1)

         integer        nxf,      nyf
         integer        ic,       jc,       if,       jf

         nxf = 2 * nxc - 1
         nyf = 2 * nyc - 1

         do jc = 1 , nyc
            jf = 2 * jc - 1
            do ic = 1 , nxc
               uf(2 * ic - 1,jf) = uc(ic,jc)
            end do
         end do
         do jf = 1 , nyf , 2
            do if = 2 , nxf - 1 , 2
               uf(if,jf) = 0.5d0 * (uf(if+1,jf) + uf(if-1,jf))
            end do
         end do
         do jf = 2 , nyf - 1 , 2
            do if = 1 , nxf
               uf(if,jf) = 0.5d0 * (uf(if,jf+1) + uf(if,jf-1))
            end do
         end do

         return

      end
```

```
c=============================================================
c     Calculates exact solution:
c
c        u(x,y) = sin(l_x pi x) sin(l_y pi y)
c=============================================================
      subroutine clcu2(u,h,nx,ny)

         implicit       none

         include       'mgprob.inc'

         integer        nx,       ny
         real*8         u(nx,ny)

         real*8         h

         integer        i,        j

         do  j = 1 , ny
            do  i = 1 , nx
                  u(i,j) = sin(pi * lx * (i - 1) * h) *
     &                     sin(pi * ly * (j - 1) * h)
            end do
         end do
         call dmlbs(u,0.0d0,nx,ny)

         return

      end
```

```
c===========================================================
c     Calculates right hand side corresponding to above
c     exact solution.
c===========================================================
      subroutine clcrhs2(rhs,h,nx,ny)

         implicit       none

         include       'mgprob.inc'

         integer        nx,       ny
         real*8         rhs(nx,ny)

         real*8         h

         real*8         uij
         integer        i,        j

         do  j = 1 , ny
            do  i = 1 , nx
               uij =
     &                    sin(pi * lx * (i - 1) * h) *
     &                    sin(pi * ly * (j - 1) * h)
               rhs(i,j) = (-pi**2 * (lx**2 + ly**2) +
     &                    sigma * uij) * uij
            end do
         end do
         call dmlbs(rhs,0.0d0,nx,ny)

         return

      end
```

```
c===========================================================
c     Initializes grid structure.
c===========================================================
      subroutine gsinit(lev)
         implicit       none
         include       'mg.inc'
         integer        qalloc
         integer        lev,       l,        size
c-----------------------------------------------------------
c        For each level.
c-----------------------------------------------------------
         do l = 1 , lev
c-----------------------------------------------------------
c           Define mesh dimensions and spacing.
c-----------------------------------------------------------
            if( l .eq. 1 ) then
               nx(1)  = 3
               ny(1)  = 3
               h(1)   = 0.5d0
            else
               h(l)   = 0.5d0 * h(l-1)
               nx(l)  = 2 * nx(l-1) - 1
               ny(l)  = 2 * ny(l-1) - 1
            end if
c-----------------------------------------------------------
c           "Allocate" storage for level-l unknown 'u',
c           right-hand-side 'rhs', relative local truncation
c           error estimate 'tau', and three temporary
c           arrays 't1', 't2' and 't3'.
c-----------------------------------------------------------
            size    = nx(l) * ny(l)
            u(l)    = qalloc(size)
            rhs(l)  = qalloc(size)
            tau(l)  = qalloc(size)
            t1(l)   = qalloc(size)
            t2(l)   = qalloc(size)
            t3(l)   = qalloc(size)
         end do
         return
      end
```

```
c===========================================================
c     "Memory allocation" routine.
c
c     'qalloc' returns the current "pointer" (index) into
c     the main storage array 'q' and increments the pointer
c     by size, effectively "allocating" a block of that
c     size.  If memory is exhausted, the routine prints
c     an error message and EXITS.
c===========================================================

      integer function qalloc(size)
         implicit       none

         include       'mg.inc'

         integer        size

         logical        first
         data           first / .true. /

         save

         if( first ) then
            qptr  = 1
            first = .false.
         end if
         qalloc = qptr
         if( qptr + size .gt. qsize ) then
            write(0,1000) size, qsize
1000        format(' qalloc: Could not allocate block of ',
     &             'size ',i8,'  Total size: ',i8)
            stop
         end if
         qptr = qptr + size

         return
      end
```

```
c===========================================================
c     Loads "border" of 2-D array with scalar.
c===========================================================
      subroutine dmlbs(a,s,d1,d2)

         implicit      none

         integer       d1,          d2
         real*8        a(d1,d2)

         real*8        s

         integer       i1,          i2

         do i2 = 1 , d2
            a(1, i2) = s
            a(d1,i2) = s
         end do

         do i1 = 1 , d1
            a(i1,1)  = s
            a(i1,d2) = s
         end do

         return

      end
```

```
c=============================================================
c     Load 2-D array with scalar.
c=============================================================
      subroutine dmls(a,s,d1,d2)

         implicit       none

         integer        d1,       d2
         real*8         a(d1,d2)
         real*8         s

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a(i,j) = s
            end do
         end do

         return

      end
```

```fortran
c===========================================================
c     Matrix-matrix add.
c===========================================================
      subroutine dmma(a1,a2,a3,d1,d2)

         implicit       none

         integer        d1,         d2
         real*8         a1(d1,d2),  a2(d1,d2), a3(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a3(i,j) = a1(i,j) + a2(i,j)
            end do
         end do

         return

      end
```

```
c=============================================================
c     Matrix-matrix subtract.
c=============================================================
      subroutine dmms(a1,a2,a3,d1,d2)

         implicit       none

         integer        d1,         d2
         real*8         a1(d1,d2),  a2(d1,d2), a3(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a3(i,j) = a1(i,j) - a2(i,j)
            end do
         end do

         return

      end
```

```
c=============================================================
c     Matrix-matrix copy.
c=============================================================
      subroutine dmcopy(a1,a2,d1,d2)

         implicit       none

         integer        d1,        d2
         real*8         a1(d1,d2), a2(d1,d2)

         integer        i,        j

         do j = 1 , d2
            do i = 1 , d1
               a2(i,j) = a1(i,j)
            end do
         end do

         return

      end
```

```
c=============================================================
c     Matrix l2-norm.
c=============================================================
      double precision function dmnrm2(a,d1,d2)

         implicit       none

         integer        d1,       d2
         real*8         a(d1,d2)

         integer        i,        j

         dmnrm2 = 0.0d0

         do j = 1 , d2
            do i = 1 , d1
               dmnrm2 = dmnrm2 + a(i,j) * a(i,j)
            end do
         end do
         dmnrm2 = sqrt(dmnrm2 / (d1 * d2))

         return

      end
```

```
c===========================================================
c     Residual tracing routine.   Generates cheesy "2-d"
c     display.
c===========================================================
      subroutine trace(l,string,resnrm)
         implicit      none

         character*(*)  string
         integer        l
         real*8         resnrm

         integer        indent
         parameter    ( indent = 2 )

         character*128  spaces
         data           spaces / ' ' /

         write(0,1000)  spaces(1:1 + (l-1) * indent),
     &                  l, string, resnrm
 1000    format(a,'Level ',i2,' ',a', sweep ',
     &          t50,1p,e8.2)

         return

      end
```

```
c===========================================================
c     'gft' output routine.
c
c     This routine uses 'gft_write_bbox': bbox = bounding
c     box which specifies the limits of the grid function's
c     domain.
c
c         bbox := ( xmin, xmax, ymin, ymax )
c
c     On the SGIs, specify the libraries
c
c         -lbbhutil -lmfhdf -ldf -ljpeg -lz
c
c     (in that precise order) at load-time when using
c     'gft_write_bbox' or any other routines from the
c     'bbhutil' library.
c===========================================================
```

```
subroutine gft_out(l)
   implicit      none

   include      'mg.inc'
   include      'mgprob.inc'

   integer       l

   logical       first
   real*8        bbox(4),     time
   integer       shape(2)

   data          first / .true. /

   save

   if( .not. gft_trace ) return

   if( first ) then
      time = 0.0d0
      bbox(1) = 0.0d0
      bbox(2) = 1.0d0
      bbox(3) = 0.0d0
      bbox(4) = 1.0d0
      first = .false.
   else
      time = time + 1.0d0
   end if

   shape(1) = nx(l)
   shape(2) = nx(l)
   call gft_write_bbox('u',time,shape,2,bbox,q(u(l)))

   return

end
```

```
c===========================================================
c     Problem-specific quantities for multi-grid program.
c===========================================================

      real*8            pi
      parameter       ( pi = 3.141 5926 5358 9793 d0 )

      real*8            lx,            ly
      parameter       ( lx = 1.0d0,    ly = 2.0d0 )

      real*8            sigma
      common          / compar /
     &                  sigma

      logical           gft_trace
      common          / compar /
     &                  gft_trace
```

```
############################################################
# Sample build and execution of mgfas2d
############################################################

% ls *.f *.inc Makefile
Makefile      mg.inc         mgfas2d.f      mgfas2dlib.f
mgprob.inc

############################################################
# ALL of the libraries
#
#    -lbbhutil -lmfhdf -ldf -ljpeg -lz
#
# should be specified (in the above order) to use the
# gft_ ... utility routines.
#
# Note that you can safely ignore warnings about one or
# more libraries not being used to resolve any symbols.
############################################################
% make
        make -f Makefile
        f77 -g -n32 -c mgfas2d.f
        f77 -g -n32 -c mgfas2dlib.f
        f77 -g -n32 -L/usr/localn32/lib -n32 mgfas2d.o mgfas2dlib.o \
            -lp329f -lbbhutil -lmfhdf -ldf -ljpeg -lz -o mgfas2d

ld32: WARNING 84: /usr/localn32/lib/libjpeg.a is not used for \
      resolving any symbol.

############################################################
# Usage
############################################################
% mgfas2d
 usage: mgfas2d <level> <sigma> [<nvcycle> <preswp> <pstswp> \
        <gft_trace>]
```

```
############################################################
# Level 1 run---difference solution is essentially exact in
# this case, but this is largely "accidental" (combination
# of very coarse spacing, and nature of exact solution).
############################################################
% mgfas2d 1 1.0
 Level  1 Coarse grid, sweep                    2.01E-15

 || Error ||:   8.5082E-17

############################################################
# Level 3 run
############################################################
% mgfas2d 3 1.0
 Level  1 Coarse grid, sweep                    2.01E-15

   Level  2 Pre-CGC, sweep                      2.35E+01
 Level  1 Coarse grid, sweep                    3.42E-02
 Level  1 Coarse grid, sweep                    1.37E-05
 Level  1 Coarse grid, sweep                    2.19E-12
   Level  2 Post-CGC, sweep                     7.08E+00

     Level  3 Pre-CGC, sweep                    1.83E+01
   Level  2 Pre-CGC, sweep                      8.98E-01
 Level  1 Coarse grid, sweep                    3.11E-02
 Level  1 Coarse grid, sweep                    1.13E-05
 Level  1 Coarse grid, sweep                    1.50E-12
   Level  2 Post-CGC, sweep                     2.31E-01
     Level  3 Post-CGC, sweep                   2.55E+00

 || Error ||:   9.2996E-03
```

```
# Level 5 run
% mgfas2d 5 1.0
 Level  1 Coarse grid, sweep                      2.01E-15

   Level  2 Pre-CGC, sweep                        2.35E+01
 Level  1 Coarse grid, sweep                      3.42E-02
 Level  1 Coarse grid, sweep                      1.37E-05
 Level  1 Coarse grid, sweep                      2.19E-12
   Level  2 Post-CGC, sweep                       7.08E+00

     Level  3 Pre-CGC, sweep                      1.83E+01
   Level  2 Pre-CGC, sweep                        8.98E-01
 Level  1 Coarse grid, sweep                      3.11E-02
 Level  1 Coarse grid, sweep                      1.13E-05
 Level  1 Coarse grid, sweep                      1.50E-12
   Level  2 Post-CGC, sweep                       2.31E-01
     Level  3 Post-CGC, sweep                     2.55E+00

       Level  4 Pre-CGC, sweep                    1.89E+01
     Level  3 Pre-CGC, sweep                      8.86E-01
   Level  2 Pre-CGC, sweep                        4.68E-01
 Level  1 Coarse grid, sweep                      6.69E-04
 Level  1 Coarse grid, sweep                      5.24E-09
 Level  1 Coarse grid, sweep                      2.89E-19
   Level  2 Post-CGC, sweep                       1.41E-01
     Level  3 Post-CGC, sweep                     4.82E-01
       Level  4 Post-CGC, sweep                   1.10E+00

         Level  5 Pre-CGC, sweep                  1.96E+01
       Level  4 Pre-CGC, sweep                    3.25E-01
     Level  3 Pre-CGC, sweep                      2.69E-01
   Level  2 Pre-CGC, sweep                        1.45E-01
 Level  1 Coarse grid, sweep                      7.53E-05
 Level  1 Coarse grid, sweep                      6.64E-11
   Level  2 Post-CGC, sweep                       4.36E-02
     Level  3 Post-CGC, sweep                     1.49E-01
       Level  4 Post-CGC, sweep                   2.12E-01
         Level  5 Post-CGC, sweep                 3.39E-01

 || Error ||:   3.5222E-04
```

```
#############################################################
# Level 11 run
#############################################################
% mgfas2d 11 1
 qalloc: Could not allocate block of size   263169  \
         Total size:  1048576

#############################################################
# Define an alias for running the same problem at various
# resolutions and extracting the '|| Error ||' line
# from the output.
#
# Note the use of the 'csh' construct '|&' to
# redirect both standard error and standard output
# through a pipeline.
#############################################################
% alias doit 'mgfas2d \!* 1.0 |& grep Error'

% doit 3
 || Error ||:   9.2996E-03
% doit 4
 || Error ||:   1.6056E-03
% doit 5
 || Error ||:   3.5222E-04
% doit 6
 || Error ||:   8.4800E-05
% doit 7
 || Error ||:   2.1139E-05
% doit 8
 || Error ||:   5.2794E-06
```

```
################################################################
# Compute approximate convergence rate using level 6, 7 and
# 8 computations.
################################################################
% maple

> evalf(8.4800E-05/2.1139E-05);

                                    4.011542646

> evalf(2.1139E-05/5.2794E-06);

                                    4.004053491

> quit;
```