```
c---------------------------------------------------------------------
c---------------------------------------------------------------------
c
c  header.h
c
c---------------------------------------------------------------------
c---------------------------------------------------------------------

      implicit none

c---------------------------------------------------------------------
c The following include file is generated automatically by the
c "setparams" utility. It defines
c      ncells:        the square root of the number of processors
c      problem_size:  12, 64, 102, 162 (for class T, A, B, C)
c      dt_default:    default time step for this problem size if no
c                     config file
c      niter_default: default number of iterations for this problem size
c---------------------------------------------------------------------

      include 'npbparams.h'

      integer           aa, bb, cc, BLOCK_SIZE
      parameter (aa=1, bb=2, cc=3, BLOCK_SIZE=5)

      logical           active
      integer           no_cells, grid_points(3)
      double precision  elapsed_time
      common /global/   elapsed_time, no_cells, grid_points, active

      double precision  tx1, tx2, tx3, ty1, ty2, ty3, tz1, tz2, tz3,
     >                  dx1, dx2, dx3, dx4, dx5, dy1, dy2, dy3, dy4,
     >                  dy5, dz1, dz2, dz3, dz4, dz5, dssp, dt,
     >                  ce(5,13), dxmax, dymax, dzmax, xxcon1, xxcon2,
     >                  xxcon3, xxcon4, xxcon5, dx1tx1, dx2tx1, dx3tx1,
     >                  dx4tx1, dx5tx1, yycon1, yycon2, yycon3, yycon4,
     >                  yycon5, dy1ty1, dy2ty1, dy3ty1, dy4ty1, dy5ty1,
     >                  zzcon1, zzcon2, zzcon3, zzcon4, zzcon5, dz1tz1,
```

```
>                  dz2tz1, dz3tz1, dz4tz1, dz5tz1, dnxm1, dnym1,
>                  dnzm1, c1c2, c1c5, c3c4, c1345, conz1, c1, c2,
>                  c3, c4, c5, c4dssp, c5dssp, dtdssp, dttx1, bt,
>                  dttx2, dtty1, dtty2, dttz1, dttz2, c2dttx1,
>                  c2dtty1, c2dttz1, comz1, comz4, comz5, comz6,
>                  c3c4tx3, c3c4ty3, c3c4tz3, c2iv, con43, con16

 common /constants/ tx1, tx2, tx3, ty1, ty2, ty3, tz1, tz2, tz3,
>                  dx1, dx2, dx3, dx4, dx5, dy1, dy2, dy3, dy4,
>                  dy5, dz1, dz2, dz3, dz4, dz5, dssp, dt,
>                  ce, dxmax, dymax, dzmax, xxcon1, xxcon2,
>                  xxcon3, xxcon4, xxcon5, dx1tx1, dx2tx1, dx3tx1,
>                  dx4tx1, dx5tx1, yycon1, yycon2, yycon3, yycon4,
>                  yycon5, dy1ty1, dy2ty1, dy3ty1, dy4ty1, dy5ty1,
>                  zzcon1, zzcon2, zzcon3, zzcon4, zzcon5, dz1tz1,
>                  dz2tz1, dz3tz1, dz4tz1, dz5tz1, dnxm1, dnym1,
>                  dnzm1, c1c2, c1c5, c3c4, c1345, conz1, c1, c2,
>                  c3, c4, c5, c4dssp, c5dssp, dtdssp, dttx1, bt,
>                  dttx2, dtty1, dtty2, dttz1, dttz2, c2dttx1,
>                  c2dtty1, c2dttz1, comz1, comz4, comz5, comz6,
>                  c3c4tx3, c3c4ty3, c3c4tz3, c2iv, con43, con16

 integer           EAST, WEST, NORTH, SOUTH,
>                  BOTTOM, TOP

 parameter (EAST=2000, WEST=3000,      NORTH=4000, SOUTH=5000,
>             BOTTOM=6000, TOP=7000)

 integer cell_coord (3,ncells), cell_low (3,ncells),
>        cell_high  (3,ncells), cell_size(3,ncells),
>        predecessor(3),           slice    (3,ncells),
>        grid_size  (3),           successor(3)          ,
>        start      (3,ncells), end      (3,ncells)
 common /partition/ cell_coord, cell_low, cell_high, cell_size,
>                   grid_size, successor, predecessor, slice,
>                   start, end

 integer IMAX, JMAX, KMAX, MAX_CELL_DIM, MAX_SIZE, BUF_SIZE
```

```
 parameter (MAX_CELL_DIM = (problem_size/ncells)+1)

 parameter (IMAX=MAX_CELL_DIM,JMAX=MAX_CELL_DIM,KMAX=MAX_CELL_DIM)

 parameter (BUF_SIZE=MAX_CELL_DIM*MAX_CELL_DIM*(ncells-1)*60+1)

 double precision
>   us      (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   vs      (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   ws      (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   qs      (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   ainv    (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   rho_i   (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   speed   (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   square  (-1:IMAX,  -1:JMAX,  -1:KMAX,       ncells),
>   forcing ( 0:IMAX-1, 0:JMAX-1, 0:KMAX-1,  5, ncells),
>   u       (5,    -2:IMAX+1,  -2:JMAX+1,  -2:KMAX+1, ncells),
>   rhs     (5,    -1:IMAX-1,  -1:JMAX-1,  -1:KMAX-1, ncells),
>   lhs     (5,5,3,-1:IMAX-1,  -1:JMAX-1,  -1:KMAX-1, ncells),
>   backsub_info (0:MAX_CELL_DIM, 0:MAX_CELL_DIM, 5,  ncells),
>   in_buffer(BUF_SIZE), out_buffer(BUF_SIZE)
 common /fields/  u, us, vs, ws, qs, ainv, rho_i, speed, square,
>                 rhs, forcing, lhs, in_buffer, out_buffer,
>                 backsub_info

 double precision cv(-2:MAX_CELL_DIM+1),   rhon(-2:MAX_CELL_DIM+1),
>                 rhos(-2:MAX_CELL_DIM+1), rhoq(-2:MAX_CELL_DIM+1),
>                 cuf(-2:MAX_CELL_DIM+1),  q(-2:MAX_CELL_DIM+1),
>                 ue(-2:MAX_CELL_DIM+1,5), buf(-2:MAX_CELL_DIM+1,5)
 common /work_1d/ cv, rhon, rhos, rhoq, cuf, q, ue, buf

 integer  west_size, east_size, bottom_size, top_size,
>         north_size, south_size, start_send_west,
>         start_send_east, start_send_south, start_send_north,
>         start_send_bottom, start_send_top, start_recv_west,
>         start_recv_east, start_recv_south, start_recv_north,
>         start_recv_bottom, start_recv_top
```

```
 common /box/ west_size, east_size, bottom_size,
>             top_size, north_size, south_size,
>             start_send_west, start_send_east, start_send_south,
>             start_send_north, start_send_bottom, start_send_top,
>             start_recv_west, start_recv_east, start_recv_south,
>             start_recv_north, start_recv_bottom, start_recv_top

 double precision fjac(5, 5, -2:IMAX+1, -2:JMAX+1, -2:KMAX+1),
>                 njac(5, 5, -2:IMAX+1, -2:JMAX+1, -2:KMAX+1),
>                 tmp1, tmp2, tmp3
 common /work_lhs/ fjac, njac, tmp1, tmp2, tmp3

 double precision  tmp_block(5,5), b_inverse(5,5), tmp_vec(5)
 common /work_solve/ tmp_block, b_inverse, tmp_vec
```

```
!-------------------------------------------------------------------------
-!
!                                                                         !
!         N  A  S     P A R A L L E L     B E N C H M A R K S  1.5        !
!                                                                         !
!                                   B T                                   !
!                     Simple UNIX-like MPI-IO usage
!-------------------------------------------------------------------------
-!
!                                                                         !
!    This benchmark is part of the NAS Parallel Benchmark 1.5 suite.      !
!    It is described in NAS Technical Report 95-xxxx.                     !
!                                                                         !
!    Permission to use, copy, distribute and modify this software         !
!    for any purpose with or without fee is hereby granted.  We           !
!    request, however, that all derived work reference the NAS            !
!    Parallel Benchmarks 1.5. This software is provided "as is"           !
!    without express or implied warranty.                                 !
!                                                                         !
!    More information, including the technical report, original           !
!    specifications, results, and source code, is available at            !
!                                                                         !
!          http://www.nas.nasa.gov/NAS/NPB/                               !
!                                                                         !
!    Send benchmark results to        npb-results@nas.nasa.gov            !
!    Send comments or suggestions to  npb@nas.nasa.gov                    !
!    Send requests for these codes to npb-request@nas.nasa.gov            !
!    Send bug reports to              npb-bugs@nas.nasa.gov               !
!                                                                         !
!         NAS Parallel Benchmarks Group                                   !
!         NASA Ames Research Center                                       !
!         Mail Stop: T27A-1                                               !
!         Moffett Field, CA   94035-1000                                  !
!                                                                         !
!         E-mail:  npb@nas.nasa.gov                                       !
!         Fax:     (415) 604-3957                                         !
!                                                                         !
!-------------------------------------------------------------------------
```

```
-!


c-------------------------------------------------------------------------
       program MPBT
c-------------------------------------------------------------------------

       include   'header.h'
       include   'mpinpb.h'

cBTIO begin
       include   'mpiof.h'
cBTIO end

       integer i, niter, step, c, error, fstatus

cBTIO begin
       integer fp, count, ierr, ix, iio, jio, kio, cio, idump
       integer mstatus(MPI_STATUS_SIZE)
       integer*8 iseek
cBTIO end

       double precision mflops

       double precision t, tmax

       call setup_mpi

cBTIO begin
       call mpio_init(ierr)

       iseek=0
       call MPIO_Open(comm_solve,
     $  'ufs:/scratch1/fineberg/out.mpiio.simple',
     $          MPIO_WRONLY+MPIO_CREATE,
     $          iseek, MPI_DOUBLE_PRECISION, MPI_DOUBLE_PRECISION,
     $          MPIO_OFFSET_RELATIVE, 0, fp, ierr)
       idump=0
```

```
       print *, 'mpio starting, ierr=', ierr
cBTIO end


c---------------------------------------------------------------------
c      Root node reads input file (if it exists) else takes
c      defaults from parameters
c---------------------------------------------------------------------
       if (node .eq. root) then

          write(*, 1000)
          write(*, 1004)
          open (unit=2,file='inputbt.data',status='old', iostat=fstatus)
c
          if (fstatus .eq. 0) then
            write(*,*) ' Reading from input file data.inp'
            read (2,*) niter
            read (2,*) dt
            read (2,*) grid_points(1), grid_points(2), grid_points(3)
            close(2)
          else
            write(*,*) ' No input file. Using compiled defaults '
            niter = niter_default
            dt    = dt_default
            grid_points(1) = problem_size
            grid_points(2) = problem_size
            grid_points(3) = problem_size
          endif

          write(*, 1001) grid_points(1), grid_points(2), grid_points(3)
          write(*, 1002) niter, dt
          write(*, 1003) no_nodes

 1000     format(' NAS Parallel I/O Benchmarks -- BTIO Benchmark ')
 1004     format(' Simple MPI-IO Implementation ')
 1001     format(' Size: ', i3, 'x', i3, 'x', i3)
 1002     format(' Iterations: ', i3, '    dt: ', F10.6)
```

```
 1003      format(' Number of processes: ', i5)

      endif

      call mpi_bcast(niter, 1, integer_type,
    >                 root, comm_setup, error)

      call mpi_bcast(dt, 1, dp_type,
    >                 root, comm_setup, error)

      call mpi_bcast(grid_points(1), 3, integer_type,
    >                 root, comm_setup, error)


      call make_set

      do  c = 1, ncells
         if ( (cell_size(1,c) .gt. IMAX) .or.
    >         (cell_size(2,c) .gt. JMAX) .or.
    >         (cell_size(3,c) .gt. KMAX) ) then
            print *,node, c, (cell_size(i,c),i=1,3)
            print *,'Problem size too big'
            stop
         endif
      end do

      call set_constants

      call initialize

      call lhsinit

      call exact_rhs

      call compute_buffer_size(5)

c---------------------------------------------------------------------
c      do one time step to touch all code, and reinitialize
```

```
c-----------------------------------------------------------------------
       call adi
       call initialize

c-----------------------------------------------------------------------
c      Synchronize before placing time stamp
c-----------------------------------------------------------------------
       call mpi_barrier(comm_setup, error)

       t = mpi_wtime()

       do  step = 1, niter

          if (node .eq. root) then
             if (mod(step, 1) .eq. 0) print *, 'Time step ', step
          endif

          call adi

cBTIO begin
          if (mod(step, 5).eq.0) then
             do cio=1,no_cells
                do kio=0, cell_size(3,cio)-1
                   do jio=0, cell_size(2,cio)-1

                      iseek=5*(cell_low(1,cio) +
     $                   PROBLEM_SIZE*((cell_low(2,cio)+jio) +
     $                   PROBLEM_SIZE*((cell_low(3,cio)+kio) +
     $                   PROBLEM_SIZE*idump))

                      count=5*cell_size(1,cio)

                      call MPIO_Write(fp, iseek,
     $                  u(1,0,jio,kio,cio),
     $                  MPI_DOUBLE_PRECISION, count,
     $                  mstatus, ierr)
```

```
                enddo
             enddo
          enddo
          idump = idump + 1
       endif
cBTIO end


       end do

       call verify(niter)

       t = mpi_wtime() - t

cBTIO begin
       call MPIO_Close(fp, ierr)
cBTIO end


       call mpi_reduce(t, tmax, 1,
     >                dp_type, MPI_MAX,
     >                root, comm_setup, error)

       if( node .eq. root ) then
          write (*, 2)
 2        format(//, '  BT Benchmark Completed.')
          write (*, 3) grid_points(1), grid_points(2), grid_points(3)
 3        format('  Size:       ', i3,'x',i3,'x',i3)
          write (*, 4) niter
 4        format('  Iterations: ', i4)

          write (*, 5) tmax
 5        format('  Time          = ',f12.2,' seconds')

          if( tmax .ne. 0. ) then
c xxxx xxx xxx xxx need megaflop count
             mflops = 0
c xxx xxx xxx xxx
```

```
         else
            mflops = 0.0
         endif

         write (*,6) mflops, no_nodes
 6       format('  MFLOP/s total = ',f12.2,' on ',i6,' processors')

         write (*,7) mflops/float( no_nodes )
 7       format('  MFLOP/s/Node  = ',f12.2)

        write (*,121) cs1
 121    format(/, '  Compile options:', /, '    F77     = ', A)

        write (*,122) cs2
 122    format('    LOAD    = ', A)

        write (*,123) cs3
 123    format('    MPI_LIB = ', A)

        write (*,124) cs4
 124    format('    MPI_INC = ', A)

        write (*,125) cs5
 125    format('    FFLAGS  = ', A)

        write (*,126) cs6
 126    format('    LFLAGS  = ', A, //)

       endif

cBTIO begin
       call MPIO_Finalize(ierr)
cBTIO end

       call mpi_finalize(error)

       stop
       end
```

```
!--------------------------------------------------------------------------
-!
!                                                                          !
!           N  A  S     P A R A L L E L     B E N C H M A R K S  1.5       !
!                                                                          !
!                                   B T                                    !
!                            Full MPI-IO Usage
!--------------------------------------------------------------------------
-!
!                                                                          !
!    This benchmark is part of the NAS Parallel Benchmark 1.5 suite.       !
!    It is described in NAS Technical Report 95-xxxx.                      !
!                                                                          !
!    Permission to use, copy, distribute and modify this software          !
!    for any purpose with or without fee is hereby granted.  We            !
!    request, however, that all derived work reference the NAS             !
!    Parallel Benchmarks 1.5. This software is provided "as is"            !
!    without express or implied warranty.                                  !
!                                                                          !
!    More information, including the technical report, original            !
!    specifications, results, and source code, is available at             !
!                                                                          !
!          http://www.nas.nasa.gov/NAS/NPB/                                !
!                                                                          !
!    Send benchmark results to         npb-results@nas.nasa.gov            !
!    Send comments or suggestions to  npb@nas.nasa.gov                     !
!    Send requests for these codes to npb-request@nas.nasa.gov             !
!    Send bug reports to               npb-bugs@nas.nasa.gov               !
!                                                                          !
!          NAS Parallel Benchmarks Group                                   !
!          NASA Ames Research Center                                       !
!          Mail Stop: T27A-1                                               !
!          Moffett Field, CA   94035-1000                                  !
!                                                                          !
!          E-mail:  npb@nas.nasa.gov                                       !
!          Fax:     (415) 604-3957                                         !
!                                                                          !
!--------------------------------------------------------------------------
```

```
-!


c---------------------------------------------------------------------
       program MPBT
c---------------------------------------------------------------------

       include  'header.h'
       include  'mpinpb.h'

cMPIO begin
       include 'mpiof.h'

       integer cell_disp(ncells+1)

       integer eltext, iosize
       integer element
       integer fp, ierr
       integer combined_btype, combined_ftype
       integer mstatus(MPI_STATUS_SIZE)
       integer sizes(3), starts(3), subsizes(3)
       integer cell_btype(ncells+1), cell_ftype(ncells)
       integer cell_blength(ncells+1)
       integer*8 iseek
cMPIO end

       integer i, niter, step, c, error, fstatus
       double precision mflops

       double precision t, tmax

       call setup_mpi

cMPIO begin
       call MPIO_Init(ierr)
cMPIO end

c---------------------------------------------------------------------
```

```
c       Root node reads input file (if it exists) else takes
c       defaults from parameters
c---------------------------------------------------------------------
        if (node .eq. root) then

           write(*, 1000)
           write(*, 1004)
           open (unit=2,file='inputbt.data',status='old', iostat=fstatus)
c
           if (fstatus .eq. 0) then
             write(*,*) ' Reading from input file data.inp'
             read (2,*) niter
             read (2,*) dt
             read (2,*) grid_points(1), grid_points(2), grid_points(3)
             close(2)
           else
             write(*,*) ' No input file. Using compiled defaults '
             niter = niter_default
             dt    = dt_default
             grid_points(1) = problem_size
             grid_points(2) = problem_size
             grid_points(3) = problem_size
           endif

           write(*, 1001) grid_points(1), grid_points(2), grid_points(3)
           write(*, 1002) niter, dt
           write(*, 1003) no_nodes

 1000      format(' NAS Parallel I/O Benchmarks -- BTIO Benchmark ')
 1004      format(' Full MPI-IO Implementation ')
 1001      format(' Size: ', i3, 'x', i3, 'x', i3)
 1002      format(' Iterations: ', i3, '    dt: ', F10.6)
 1003      format(' Number of processes: ', i5)

        endif

        call mpi_bcast(niter, 1, integer_type,
     >                 root, comm_setup, error)
```

```
      call mpi_bcast(dt, 1, dp_type,
     >               root, comm_setup, error)

      call mpi_bcast(grid_points(1), 3, integer_type,
     >               root, comm_setup, error)


      call make_set

      do  c = 1, ncells
         if ( (cell_size(1,c) .gt. IMAX) .or.
     >        (cell_size(2,c) .gt. JMAX) .or.
     >        (cell_size(3,c) .gt. KMAX) ) then
            print *,node, c, (cell_size(i,c),i=1,3)
            print *,'Problem size too big'
            stop
         endif
      end do

      call set_constants

      call initialize

cMPIO begin
      call MPI_Type_contiguous(5, MPI_DOUBLE_PRECISION,
     $       element, ierr)
      call MPI_Type_commit(element, ierr)
      call MPI_Type_extent(element, eltext, ierr)
c      print *, node,': eltext=', eltext

      do  c = 1, ncells
          sizes(1) = IMAX+4
          sizes(2) = JMAX+4
          sizes(3) = KMAX+4

          subsizes(1) = cell_size(1, c)
          subsizes(2) = cell_size(2, c)
```

```
          subsizes(3) = cell_size(3, c)

          starts(1) = 2
          starts(2) = 2
          starts(3) = 2

          call MPIO_Type_nd_array(3, sizes, subsizes, starts,
     $    1, element, cell_btype(c), ierr)
          cell_blength(c) = 1
          cell_disp(c) = eltext*(IMAX+4)*(JMAX+4)*(KMAX+4)*(c-1)
c        print *, node,': cell_disp(',c,')=',cell_disp(c)
        enddo

        cell_blength(ncells+1) = 1
        cell_btype(ncells+1) = MPI_UB
        cell_disp(ncells+1) =
     $             eltext*(IMAX+4)*(JMAX+4)*(KMAX+4)*ncells

        call MPI_Type_struct(ncells+1, cell_blength, cell_disp,
     $             cell_btype, combined_btype, ierr)
        call MPI_Type_commit(combined_btype, ierr)

      call MPI_Type_extent(combined_btype, iosize, ierr)
      print *, 'btype extent = ', iosize

        do  c = 1, ncells
          sizes(1) = PROBLEM_SIZE
          sizes(2) = PROBLEM_SIZE
          sizes(3) = PROBLEM_SIZE

          subsizes(1) = cell_size(1, c)
          subsizes(2) = cell_size(2, c)
          subsizes(3) = cell_size(3, c)

          starts(1) = cell_low(1,c)
          starts(2) = cell_low(2,c)
          starts(3) = cell_low(3,c)
```

```
c       print *, 'starts = ', cell_low(1,c), cell_low(2,c),
c     $      cell_low(3,c)

            call MPIO_Type_nd_array(3, sizes, subsizes, starts,
     $    1, element, cell_ftype(c), ierr)
            cell_blength(c) = 1
            cell_disp(c) = 0
        enddo

        call MPI_Type_struct(ncells, cell_blength, cell_disp,
     $             cell_ftype, combined_ftype, ierr)
        call MPI_Type_commit(combined_ftype, ierr)
        iseek=0

        call MPIO_Open(comm_solve,
     $         'ufs:/scratch1/fineberg/out.full.mpiio',
     $         MPIO_WRONLY+MPIO_CREATE,
     $         iseek, element, combined_ftype,
     $         MPIO_OFFSET_RELATIVE, 0, fp, ierr)

cMPIO end

       call lhsinit

       call exact_rhs

       call compute_buffer_size(5)

c---------------------------------------------------------------------
c      do one time step to touch all code, and reinitialize
c---------------------------------------------------------------------
       call adi
       call initialize

c---------------------------------------------------------------------
c      Synchronize before placing time stamp
c---------------------------------------------------------------------
       call mpi_barrier(comm_setup, error)
```

```
      t = mpi_wtime()

      do  step = 1, niter

         call adi

cBTIO begin
         if (mod(step, 5) .eq. 0) then
             if (node .eq. root)
     $    print *, 'Writing data set, time step', step

             call MPIO_Write_all(fp, iseek, u,
     $                        combined_btype, 1, mstatus, ierr)
             call MPI_Type_size(combined_btype, iosize, ierr)
             iseek = iseek + iosize/eltext
         endif
cBTIO end


      end do

      call verify(niter)

      t = mpi_wtime() - t

cBTIO begin
      call MPIO_Close(fp, ierr)
cBTIO end

      call mpi_reduce(t, tmax, 1,
     >                dp_type, MPI_MAX,
     >                root, comm_setup, error)

      if( node .eq. root ) then
         write (*, 2)
 2       format(//, `  BT Benchmark Completed.')
         write (*, 3) grid_points(1), grid_points(2), grid_points(3)
```

```
 3        format('  Size:        ', i3,'x',i3,'x',i3)
          write (*, 4) niter
 4        format('  Iterations: ', i4)

          write (*, 5) tmax
 5        format('  Time            = ',f12.2,' seconds')

          if( tmax .ne. 0. ) then
c xxxx xxx xxx xxx need megaflop count
             mflops = 0
c xxx xxx xxx xxx
          else
             mflops = 0.0
          endif

          write (*,6) mflops, no_nodes
 6        format('  MFLOP/s total = ',f12.2,' on ',i6,' processors')

          write (*,7) mflops/float( no_nodes )
 7        format('  MFLOP/s/Node   = ',f12.2)

         write (*,121) cs1
 121     format(/, '  Compile options:', /, '    F77      = ', A)

         write (*,122) cs2
 122     format('    LOAD    = ', A)

         write (*,123) cs3
 123     format('    MPI_LIB = ', A)

         write (*,124) cs4
 124     format('    MPI_INC = ', A)

         write (*,125) cs5
 125     format('    FFLAGS  = ', A)

         write (*,126) cs6
 126     format('    LFLAGS  = ', A, //)
```

```
      endif

cBTIO begin
      call MPIO_Finalize(ierr)
cBTIO end

      call mpi_finalize(error)

      stop
      end
```