c 11.8 Brief Review About "MPI-parallel" FORTRAN'2003 Programming
c --

c Message Passing Interface (MPI) parallel application codes can be
c implemented in either FORTRAN, or C++ language, under UNIX, LINUX,
c or WINDOWS environments. The syntax for "parallel" MPI/FORTRAN-90 are
c essentially identical to the ones used in "serial" FORTRAN'90/FORTRAN'2003,
c with few exceptions for "specific parallel computational purposes".

c Regardless the computer language adopted by the users (such as C, or C++,
c or FORTRAN-77, or FORTRAN-90 (with features like Dynamic Storage Allocations,
c such as ALLOCATE and DEALLOCATE statements)/FORTRAN'2003 (with very powerful
c STREAM I/O, which can write/read on/from any location of the file, and without
c requiring the FIXED record-length for I/O), or BASIC etc...), one only needs
c to be familiar with the syntax for "IF" statements, "DO" loop, "DIMENSION"
c statements (for handling 1-D, and/or 2-D, and/or 3-D integer/real arrays),
c input/output, and usage of "subroutines".

c The following listed MPI/FORTRAN-90 demonstrated code can be conveniently
c used to understand the "syntax" for writing any general application codes.

```
c=====================================================================! 000
c23456789012345678901234567890123456789012345678901234567890123456789012 ! 001
c Purposes:  Reviewing some basic FORTRAN_90 syntax, and MPI_FORTRAN     ! 002
c Author(s): Prof. Duc Thai NGUYEN (757-683-3761; DNguyen@odu.edu)       ! 003
c Date:      January 1, 2010                                             ! 004
c Stored At: cd ~/cee/*odu*class*/teach_fortran90_mpi.f                  ! 005
c                                                                        ! 006
      implicit real*8(a-h,o-z)                                           ! 007
      include 'mpif.h'                                                   ! 008
      character*80 title                                                 ! 009
      parameter (num=10)                                                 ! 010
      parameter (master=0)                                               ! 011
      parameter (from_master=1)                                          ! 012
      parameter (from_worker=2)                                          ! 013
      dimension a(num),b(num)                                            ! 014
      allocatable:: ia(:),a11(:,:),a22(:,:)                              ! 015
c-----------------------------------------------------------------------  ! 016
      call MPI_INIT(ierr)                                                ! 017
      call MPI_COMM_RANK(MPI_COMM_WORLD, me, ierr)                       ! 018
      call MPI_COMM_SIZE(MPI_COMM_WORLD, np, ierr)                       ! 019
       if (me .eq. 0) then                                               ! 019.1
```

```
      write(6,*) '                              '                         ! 019.2
      write(6,*) '========================================'             ! 019.3
      write(6,*) 'Prof. Duc T. Nguyen; January 1, 2010'                 ! 019.4
      write(6,*) '========================================'             ! 019.5
      write(6,*) '                              '                         ! 019.6
     endif                                                              ! 019.7
c-----------------------------------------------------------------       ! 020
c     call MPI_BARRIER(MPI_COMM_WORLD, ierr)                            ! 021
c     call MPI_SEND(num,1,MPI_INTEGER,i_destination,1,MPI_COMM_WORLD,   !
022
c    $ierr)                                                             ! 023
c     call MPI_RECV(num,1,MPI_INTEGER,master,mtype,MPI_COMM_WORLD,      !
024
c    $status,ierr)                                                      ! 025
c-----------------------------------------------------------------       ! 026
      idum=0                                                            ! 027
      sum=0.d0                                                          ! 028
      do 1 i=1,num,1                                                    ! 029
       a(i)=drand(idum)                                                 ! 030
       sum=sum+a(i)                                                     ! 031
       if (i .le. 10) then                                              ! 032
       write(6,*) 'i,a(i) = ',i,a(i)                                    ! 033
       elseif (i .gt. 10) then                                          ! 034
       write(6,*) 'skip printing too many random numbers !'             ! 035
       endif                                                            ! 036
 1    continue                                                          ! 037
c                                                                       ! 038
       open (unit=7, file='K.INFO', status='old', form='formatted')     ! 039
c      open (unit=6, file='out1', status='old', form='formatted')       ! 040
       read(7,115) title                                                ! 041
 115   format(a60)                                                      ! 042
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
c      read(7,28) intnum,floatnum,expnum,characstring
c28    format(i5, f10.2, e10.3, a15)
c  25   -127.85 +4.267+02 Bill Gate
c234567890123456789012345678901234567890
c+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
       write(6,115) title                                               ! 043
c                                                                       ! 044
       memory_need=2*num                                                ! 045
       allocate ( ia(memory_need), a11(memory_need,memory_need),        ! 046
     $            a22(num,num) )                                        ! 047
      do 2 i=1,memory_need,1                                            ! 048
       ia(i)=i                                                          ! 049
 2    continue                                                          ! 050
       deallocate(a11,a22)                                              ! 051
```

```
     call dummy1(num,memory_need,a,sum_real)                                  ! 052
     write(6,*) ' sum_real= ', sum_real                                       ! 053
c-----------------------------------------------------------------------      ! 054
      num_workers=np-1                                                        ! 055
      bigest_local=0.d0                                                       ! 056
c......each processor (master and workers) will:                              ! 057
c......generate its own portions of random (real) numbers                     ! 058
c......then, it will find its own local maximum number                        ! 059
     do 11 i=me+1, num, np                                                    ! 060
      b(i)=drand(idum)                                                        ! 061
      if ( b(i) .gt. bigest_local ) bigest_local=b(i)                         ! 062
       write(6,*) 'processor id# ',me, 'i,b(i) = ',i,b(i)                     ! 062.1
       write(6,*) 'processor id# ',me, 'bigest_local = ', bigest_local        ! 062.2
 11   continue                                                                ! 063
c                                                                             ! 064
c...... each worker will send its own local maximum to the master             ! 065
     if (me .gt. 0) then                                                      ! 066
      mtype=from_worker                                                       ! 067
      call MPI_SEND(bigest_local,1,MPI_DOUBLE_PRECISION,master,mtype   ! 068
    $,MPI_COMM_WORLD,ierr)                                                    ! 069
      write(6,*) 'sent by worker # ',me, ' bigest_local= ',bigest_local       ! 069.1
c...... the master processor will receive local maximum                       ! 070
c...... (from each worker)                                                    ! 071
c...... and then, comparing amongst all local max to find/print               ! 072
c...... global max                                                            ! 073
     elseif (me .eq. 0) then                                                   ! 074
     bigest_global=bigest_local                                               ! 075
     mtype=from_worker                                                        ! 076
      write(6,*) 'processor id # ',me, ' bigest_local= ',bigest_local         ! 076.1
     do 60 i=1,num_workers,1                                                  ! 077
      isource=i                                                               ! 078
      call MPI_RECV(bigest_local,1,MPI_DOUBLE_PRECISION,isource,mtype, ! 079
    $MPI_COMM_WORLD,status,ierr)                                               ! 080
      if (bigest_local .gt. bigest_global) bigest_global=bigest_local         ! 081
 60   continue                                                                ! 082
      write(6,*) 'amongst local max, the global max is ',bigest_global        ! 083
c-----------------------------------------------------------------------------
c
c......the following subroutine demonstrates how to use powerful/             ! 083.1
c......standard STREAM I/O features available in FORTRAN'2003                 ! 083.2
c
      call learnstreamio                                                      ! 083.3
c-----------------------------------------------------------------------------
     endif                                                                    ! 084
c                                                                             ! 085
     write(6,*) 'processor id# ',me, 'out of ',np, ' is alive'                ! 086
```

```
      call MPI_FINALIZE(ierr)                                           ! 087
c---------------------------------------------------------------------  ! 088
      stop                                                              ! 089
      end                                                               ! 090
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                                         ! 091
       subroutine learnstreamio                                         ! 091.01
c
c......purpose: learn/demonstrate how to use powerful "stream i/o"       ! 091.02
c              in fortran-2003                                          ! 091.03
c......stored at:  cd ~/cee/*odu*class*/*learn*stream*.f
c......notes:   to compile this program, just type  f90 learn*stream*.f
c              to execute this program, just type  ./a.out >&! out1 &
c
       IMPLICIT REAL*8(A-H,O-Z)
       DIMENSION A(1000,1000), IAA(1000000), ibb(1000000)

       OPEN(UNIT=92, FILE="aaa.demo", STATUS="OLD", ACCESS="STREAM")    !
091.04

       write(6,*) ' Duc T. Nguyen, January 1, 2010'                     ! 091.05

c
c......generate 2-d real array a(-,-), and 1-d integer array iaa(-)
c
       do 1 i=1,1000
       iaa(i)=i+1
        do 2 j=1,1000
        a(i,j)=i+j
 2      continue
 1     continue
c
c......using the default FORM="UNFORMATTED" stream i/o WRITE, available
c......(and standardize) in FORTRAN'2003
c
       write(92) ((a(i,j),j=1,1000),i=1,2)                            ! 091.06
       inquire (unit=92,POS=mypos)                                    ! 091.07
       write(6,*) 'mypos after writing 2000 double pre numbers = ',mypos! 091.08
       write(92) (iaa(i),i=1,500)                                     ! 091.09
       inquire (unit=92,POS=mypos)                                    ! 091.10
       write(6,*) 'mypos after writing 500 integer numbers = ',mypos  ! 091.11
       write(6,*) 'iaa(40-60) = 41,42,...,61 = ',(iaa(i),i=40,60)
c
c......using the default FORM="UNFORMATTED" stream i/o READ, available
c......(and standardize) in FORTRAN'2003
c
```

```
      read(92,pos=16001) (ibb(i),i=1,20)                              ! 091.12
      write(6,*) 'ibb(1-20) = 2, 3, 4,...,21 = ',(ibb(i),i=1,20)      ! 091.13
c......each integer = 4 bytes; and each real/double precision = 8 bytes
c......thus, calculated position POS = 16161
      read(92,pos=16161) (ibb(i),i=41,60)                             ! 091.14
      write(6,*) 'ibb(41-60) = 42,43,...,61 = ',(ibb(i),i=41,60)      ! 091.15
c
      return
      end
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c  OUTPUTS of the above testing program (using FORTRAN'2003 STREAM I/O) are
shown below:
c Duc T. Nguyen, January 1, 2010
cmypos after writing 2000 double pre numbers =  16001
cmypos after writing 500 integer numbers =  18001
ciaa(40-60) = 41,42,...,61 =  41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
c58 59 60 61
cibb(1-20) = 2, 3, 4,...,21 =  2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
c21
cibb(41-60) = 42,43,...,61 =  42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58
c59 60 61
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
     subroutine dummy1(num,memory_need,a,sum_real)                     ! 092
     implicit real*8(a-h,o-z)                                          ! 093
     dimension a(*)                                                    ! 094
     sum_real=0.d0                                                     ! 095
     do 1 i=1,num,1                                                    ! 096
      sum_real=sum_real+a(i)                                           ! 097
 1    continue                                                         ! 098
     return                                                            ! 099
     end                                                               ! 100
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                                  ! 101

c  Lines #001-006:
c  In FORTRAN, if a character "c" is typed in column1, then the line will be
c  treated like a "comment" statement.
c
c  Line #007:
c  In FORTRAN, all "executable" statements should be typed between column
c  numbers 7 through 72. Any "real" array should be named with the first
c  character as a, b, c, ..., h, and o, p, q, ..., z. Any "integer" array
c  should be named with the first character as i,j,k,l,m,n.
c  This statement implies that each real number will need 8 bytes to store
```

c (in double precision). Similarly, a statement:
c implicit real*4(a-h,o-z) implies that each real number will need 4 bytes
c to store (in single precision).

c Line #008:
c This include statement "MUST" always be followed the implicit statement
c for any MPI/FORTRAN application code

c Line #009 (also see lines # 041-043):
c This statement is necessary only if the user want to read (or write)
c a title heading, with upto 80 characters (also see lines # 041-043)

c Lines # 010-013:
c Numerical values of certain variables can be defined/given/assigned by
c the parameter statements.

c Line # 014:
c Maximum dimension (or size) for certain arrays are defined by the
c "dimension" statement. Note that the value of "num" must already be
c earlier defined (through the parameter statements)

c Line # 015 (also see lines # 045-047):
c This is one of the "very useful" features in FORTRAN-90, for which
c the users can declare some arrays for "dynamic storage allocation" purposes.
c The actual, exact "dimension" for these arrays do NOT have to be declared
c in the begining (such as arrays defined on line # 014). These "exact"
c "dimension" needed can be declared "later on", whenever the user knows
c exactly how much memory storage one needs for these arrays (also see
c lines # 045-047)

c Lines # 017-019:
c These 3 "special" MPI/FORTRAN statements "MUST" be defined in any MPI
c application codes (and should be inserted right after dimension statements).
c The variable "np" on line # 019 represents (Number of Processors". Thus,
c if 3 processors are used, then np will be assigned the value 3 by the system.
c The variable "me" on line # 018 will have the values (assined by the computer
c system) 0,1,2, ..., np-1. This variable "me" will play a CRUCIAL role in
c any MPI application codes.
c
c It should be emphasized here that all processor ID # = 0,1,2, ..., np
c will execute the same application code. However, depending on the algorithms,
c the user will have direct control of deciding "WHICH processor ID" will
c execute on "WHAT portions of the code" etc..., through the usage of variable
c "me" (also refer to lines # 060-063)

c Lines # 019.1-019.7:

```
c  Only the "master" processor (me=0) will execute this block of statements,
c  which basically print out some output message [any desired output message
c  can be placed inside (open/close) single quotes].

c  Lines # 020-026:
c  There are about 10-20 "special, parallel" MPI constructs that are very
c  commonly used in any application codes. Amongst these MPI statements,
c  however, BARRIER, SEND and RECV are probably the most important ones to
c  be used. Basically, BARRIER statement will make sure that all processors
c  have to arrive at this statement, before they can proceed to execute
c  subsequent statements of the application code. SEND statement will send
c  a message (such as an integer/real variable, or integer/real arrays) from
c  one processor to another (specified) processor. Important argument lists
c  are explained as following:
c  1-st Argument = name of a variable (or array)
c  2-nd Argument = the "dimension" associated with this variable (or array)
c  3-rd Argument = the variable (or array) must be defined as INTEGER, or
c                 REAL (or DOUBLE PRECISION)
c  4-th Argument = send to WHICH processor ??
c  5-th Argument = message type #
c  6-th Argument = user does NOT need to know !
c  7-th Argument = user does NOT need to know !
c
c  RECV statement can be used for RECEIVING a message. Important argument
c  lists are explained as following:
c  1-st Argument = name of a variable (or array)
c  2-nd Argument = the "dimension" associated with this variable (or array)
c  3-rd Argument = the variable (or array) must be defined as INTEGER, or
c                 REAL (or DOUBLE PRECISION)
c  4-th Argument = receive from WHICH processor ??
c  5-th Argument = message type #
c  6-th Argument = user does NOT need to know !
c  7-th Argument = user does NOT need to know !
c  8-th Argument = user does NOT need to know !
c
c  The user does NOT need to know about the 2 argument lists of the MPI
c  BARRIER statement.

c  Lines # 027-037:
c  The purpose of this block of FORTRAN statements are:
c  to show the "syntax" of "do" loop (see line # 029), the integer index "i"
c  will have the values from 1 through num (=10), with the increment of 1.
c  Lines # 027, and # 030 show how to use "built-in" library function to
c  generate a real random number (between 0.00 and 1.00).
c  to show the "syntax" of "IF" statement (see lines # 032, # 034, and # 036)
c  to show the "syntax" of writing/printing some intermediate output variables.
```

```
c  Lines # 038-044:
c  Input (read), and output (write) data files can be used through the "open"
c  statements on line # 039 and line # 040, respectively.

c  Lines # 045-050:
c  At this moment, the user knows "exactly" how much memory space that he/she
c  needs to allocate (or assign) to INTEGER array ia(-), REAL arrays a11(-,-),
c  and a22(-,-). Thus, request to allocate memory space was done on line # 046-
c  # 047.

c  Line # 051:
c  Assuming that at this stage the user does NOT need the arrays a11(-,-), and
c  a22(-,-) any more, hence he/she can request to DELETE all memory spaces
c  allocated to these 2 arrays, through the DEALLOCATE statement.

c  Lines # 052-054:
c  A subroutine dummy1 is called by the main program, in order to perform a
c  certain task. In this example, the first 3 argument lists are "INPUT"
c  to this subroutine, and the 4-th argument list ( = sum_real) provide the
c  "OUTPUT" from this subroutine.

c  Line # 055:
c  Since in this example np = Number of Processors = 3, hence processor ID#0
c  will be the "master" processor, and processor ID# 1, #2 are considered
c  as "worker" processors.

c  Lines # 056-063:
c  Each processor will generate its own random numbers, and also compute/print
c  its own (local) maximum number (amongst its own random numbers). The most
c  important statement for this block is shown on line # 060 (please pay
c  attention to variable "me").
c  For the "master" processor (such as me=0), it will generate random numbers
c  coresponding to the do-loop integer index i = 1, 4, 7, and 10 (the increment
c  for index i is np = 3).
c  For the "worker" processor (such as me=1), it will generate random numbers
c  coresponding to the do-loop integer index i = 2, 5 and 8.
c  For the "worker" processor (such as me=2), it will generate random numbers
c  coresponding to the do-loop integer index i = 3, 6 and 9.
c  Also, all 3 processors (such as the "master" processor me=0, and "slave"
c  processors me=1, 2) will compute its own local maximum value (stored in
c  variable name bigest_local)
c
c  Lines 064-069.1:
c  Upon completion its task, each "slave" worker will send its own local maximum
c  to the "master" processor.
```

```
c
c  Lines 070-085
c  The "master" will receive all "slaves'" local maximum values, and it will
c  compare all these local maximum (including the "master's" own local maximum),
c  in order to identify , and print the global maximum (stored in variable name
c  bigest_global).
c
c  Line 086
c  All (master and slave) processors will print out a message before exiting.
c
c  Lines 087-091
c  This MPI_FINALIZE(ierr) "must" be placed before the program stops
c
c  Lines 092-101
c  This subroutine just computes some dummy works, such as calculating
c  the summation of a given 1-D real array
```