EXERCISE: Simple MPI programs.

Comment: ==========
==========
======

Comment: Created by AbiWord, a free, Open Source wordprocessor.

Comment: For more information visit http://www.abisource.com.

Comment: ==========
==========
======

DESCRIPTION:

In this exercise you will become acquainted with 2 simple MPI programs that demonstrate 2 simple but different modes of communication (broadcasting and point-to-point) between processes. You will install and understand the programs and execute them as multiple-processes on your desktop. In the final step you will enable the machine of a neighbor such that you can execute the program on 2 machines simultaneously over the network.

Tasks:

1, Download and install code

2, Compile code and link to MPI library

3, Run the code on the local desktop and understand how it works.

4, Configure a second machine and run the code on both your desktop and the second machine at the same time..

EXERCISE GUIDE for free Libraries

1.1 Go to /scratch directory on nfs server by mounting it as an

NFS partition

>mount 10.34.37.61:/scratch /mnt/cdrom

and get lab_mpi.tar.

Un-mount the NFS partition

>umout /mnt/cdrom

1.2 make a directory **lab_mpi** on your machine and unpack tar file

tar –xzvf lab_mpi.tar

2,1 Enter directory and compile the codes: pi.f, matvec.f

Note: if you compile with the script mpif77 you will be defaulting to the gnu compiler.

>mpif77 pi.f –o pi.x

To change fortran compiler (see example wrappers in lab_mi/bin) you need to change the LAMHF77 environment variable

Ex.

LAMHF77=ifort

Export LAMHF77

Also the way lam is installed is problematic for ifort and pgi compilers

(check to make sure your MPI library is installed if not then retrieve the

lam-6.5..9-1.i386 rpm from the NFS server /scratch /lab1/SL303//RPM and perform an rpm install

>rpm –i lam-6.5.9-1.i386.rpm)

3.1 Boot lam, run code (on however many processes you want) halt lam

>lamboot

>mpirun –np (pick an integer number) pi.x

(play with codes till you are sick of them)

>lamhalt

DO the same with both codes till you are comfortable with how they work what they do etc.

(note the following does not work on the current set up due to a VERY strange ssh configuration try this on your own)

4.1 Team up with your neighbor and install an identical user account on their machine

(same user id, path etc).

4.2 Set lam environment variables LAMRSH=ssh in /etc/profile.d/lam.sh

(see class notes on both machines).

4.3 Prepare the user account. A, Enable public/private ssh keys on both machines to perform login without a password.

In order to use the queue system you need to be able to log into one machine from each other machine without giving a password. This is required for mpi.

This is how you achieve this with OpenSSH (the ssh software installed):
Go to your private '.ssh' directory:
>cd ~/.ssh
If the directory does not exist yet, just create it with:
>mkdir ~/.ssh ; chmod 0700 ~/.ssh
Generate a private/public identity key pair with:
>ssh-keygen -d
Hit return (2 times) when asked for a password.
This gives you 2 new files 'id_dsa' and 'id_dsa.pub'.
Take the latter file and copy it in the ~/.ssh directory under the name authorized_keys2:
>cp id_dsa.pub authorized_keys2

Try it out and log into your own machine with:
>ssh machine_name

B, Place the executable in the same place on both machines!…

4.4 Edit a host file in the lab_mpi directory:

ex

machine1 cpu=1

machine2 cpu=1

where machine is the machine name or IP number of both the machines you wish to use.

Type:

>lamboot –v hostfile

(should boot lam environment on BOTH machines)

4.5 Run your code as before with the mpirun command and do a top on both machines to make sure the code is running on both platforms.

4.6 Halt lam on both machines

>lamhalt –v hostfile

4.6 Do a wall time (with the date command) and a cpu time (with the time command) measurement for the code running on the desktop and over the network.

****

CODES: Comments attempt to explain in Excessive detail what they are doing.

****

```
c*************************************************************
c*                                                           *
C*                                                           *
c*    pi.f                                                   *
c*                                                           *
c*     a program to perfrom the integral(4/1-x**2) from      *
c*     0 to 1=pi to calculate the value of pi                *
c*     The program is designed to use the MPI protocal       *
c*     to perform this task in parallel                      *
c*************************************************************
c
      program main
```

```
c     include the MPI header file
c     needed to define all the constants
c     and parameters for the mpi implementation
      include 'mpif.h'
c     Pi to more digits than you may ever need
      double precision PI25DT
      parameter (PI25DT=3.14159265358979323846264643d0)
c
c     variables in the code
      double precision mypi, pi, h, sumf, x, f, a
      integer n, myid, numprocs, i, ierr
c     n       -number of rectangless in which to divide the curve
c     mypi    -the area under the curve calcualted by a process
c     numprocs-number of processors from mpirun -np numprocs
C               command
c     i       -loop counter on a given node
c     ierr    -MPI error code
c
c     function to integrate
      f(a)=4.0d0/(1.0d0+a*a)
c
c     First MPI call must always be to mpi_init to set up
c     the "MPI environment"
c
      call MPI_INIT(ierr)
c
c     if all goes right it will reutrn value of the variable
c     (MPI_SUCCESS=0) -see mpif.h file for the value
c
      if(ierr.ne.MPI_SUCCESS) goto 30
c     set up the MPI communicator
c
c     all processors initalized in the MPI environments
c     are defined within the MPI communicator by
c     sending it the value MPI_COMM_WORLD
c     MPI_COMM_WORLD=0 is defined in the mpif.h file
c
      call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)
c     should return the variable numprocs
c
c     The processors
c     are given a rank within the communicator-ie an ID number
c      so each cpu has a different value of myid
      call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
c
c     on cpu 0 read the value of n
c
   10 if(myid.eq.0)then
       print *,'Enter number of intervals, (0 quit)'
       read(*,*) n
      endif
c
c     broadcast n to all the processes
c     read as broadcasting to all cpus in the communicator:
c     variable,number of items,datatype,myid=0,all processors in
c     communicator, error code
```

```
      call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
c
c     check for quit signal on each cpu if n=0 is recieved all
c     all copies of the program stop
      if(n.eq.0) goto 30
c
c     Each compute node calculates  the numprocs'th piece of
c     the interval
c
c     first set the sum counter to zero
c
      sumf = 0.0d0
c
c     then perform a loop on each node each of which sums
c     up it's part of the area under the curve.
c
      h=1.0d0/dfloat(n)
c
c     loop on the processor myid+1
c
      do i=myid+1,n,numprocs
       x= h * (dfloat(i)-0.5d0)
       sumf = sumf + f(x)
      enddo
c
c     each node calcs its piece of the pi
c
      mypi= h * sumf
c
c     collect all the partial sums and turns them into a global sum
c     this command reduces all the mypi s to pi
c     read as:
c     source address, result address, one item of data from each cpu,
c     date type double precission, into a sum, assembled on processor
c     form all cpus in the communicator, error message.
c
      call MPI_REDUCE(mypi,pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
     *   MPI_COMM_WORLD,ierr)
c
c     print the answer on node 0
c
      if(myid.eq.0)then
       print *,'pi estimate= ',pi,' Error= ',(pi - PI25DT)
      endif
c
c     go back to 10 and get another n start all over again.
c
      goto 10
c
c     stop this stupid program
c
   30 call MPI_FINALIZE(ierr)
      stop
      end

c*************************************************************
```

```
c*                                                                     *
C*                                                                     *
c*    matvec.f                                                         *
c*                                                                     *
c*    a program for matrix vector multiplication                       *
c*    in parallel to demonstrait MPI send and RECIEVE                  *
c*    to illustrate point-to-point communication                       *
c*                                                                     *
c*********************************************************************
      program main
c
c     include the MPI header file
c     needed to define all the constants
c     and parameters for the mpi implementation
c
      include 'mpif.h'
c
c
      integer MAX_ROWS, MAX_COLS, rows, cols
      parameter (MAX_ROWS = 1000, MAX_COLS = 1000)
c
      double precision a(MAX_ROWS,MAX_COLS), b(MAX_COLS), c(MAX_ROWS)
      double precision d(MAX_ROWS)
      double precision buffer(MAX_COLS), ans
C
c     program calculated c= a*b
c     where  a is a matrix
c            b and c are vectors
c
      integer myid, master, numprocs, ierr, status(MPI_STATUS_SIZE)
      integer i,j, numsent, sender
      integer anstype, row
c
c     First MPI call must always be to mpi_init to set up
c     the "MPI environment"
c
      call MPI_INIT(ierr)
c
c     if all goes right it will reutrn value of the variable
c     (MPI_SUCCESS=0) -see mpif.h file for the value
c
c     set up the MPI communicator
c
c     all processors initalized in the MPI environments
c     are defined within the MPI communicator by
c     sending it the value MPI_COMM_WORLD
c     MPI_COMM_WORLD=0 is defined in the mpif.h file
c
      call MPI_COMM_SIZE(MPI_COMM_WORLD,numprocs,ierr)
c     should return the variable numprocs
c
c     The processors
c     are given a rank within the communicator-ie an ID number
c     so each cpu has a different value of myid
      call MPI_COMM_RANK(MPI_COMM_WORLD,myid,ierr)
c
```

```
c     assign a master node
c
      master = 0
c     assign matric size
c
      rows = 100
      cols = 100
c     if statement sets up different scenarios for master and slaves
c
      if ( myid .eq. master ) then
       print *,'This program is calculates c(i)=sum_j a(i,j)*b(j)'
       print *,'In parallel where process',master,' is the master'
       print *,'With',numprocs-1,' slaves'
       print *
c
c      master initializes and then dispatches
c      initialize a and b (arbitrary)
c
c      compute vector d just to compare with the MPI version
c      first set to zero
      do i = 1,rows
       d(i) = 0.0d0
      enddo
      do j = 1,cols
        b(j) = 1.0d0
        do i = 1,rows
         a(i,j) = dfloat(i)
        enddo
       enddo
       print *,'Vector'
       do j = 1,cols
        print *,'b(',j,')=',b(j)
       enddo
       print *
       print *,'matrix'
       do i = 1,rows
        print *,'a(',i,',j)=',(a(i,j),j=1,cols)
c
c       whilst I am at it I will compute my check
c
        do j=1,cols
       d(i)=d(i)+a(i,j)*b(j)
        enddo
       enddo
       print *
       print *,'Sending data to slaves'
       print *
       numsent = 0
c
c     send b to all slave process by broadcast
c
        call MPI_BCAST(b, cols, MPI_DOUBLE_PRECISION, master,
     * MPI_COMM_WORLD, ierr)
c
c    send a row to each slave process; tag with row number
c    by using MPI_send command
```

```
c
       do i = 1,min(numprocs-1,rows)
        do j = 1,cols
         buffer(j) = a(i,j)
        enddo
c
c       we send variable buffer, with dimension cols, data typoe double,
c       to process I, with tag I, command applicable to
c       all processes in the communicator
c
        call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, i,
     *  i, MPI_COMM_WORLD, ierr)
        numsent = numsent+1
        print *,'sent'
      print *,'row ',i,' sent to process ',i
       enddo
c
       do  i =1,rows
c
c       we recieve variable ans, with dimension 1, data typoe double,
c       from process  MPI_ANY_SOURCE,
c       with tag MPI_ANY_TAG, command applicable to
c       all processes in the communicator
c
        call MPI_RECV(ans, 1, MPI_DOUBLE_PRECISION,
     *   MPI_ANY_SOURCE, MPI_ANY_TAG,MPI_COMM_WORLD, status, ierr)
        sender = status(MPI_SOURCE)
        anstype = status(MPI_TAG)
c
        print *,'recieved'
      print *,'c(',anstype,') as recieved from ',sender,'=',ans
c
c       row is tag value of i
c
        c(anstype) = ans
c
c       IF thee are more rows to work on
c       then send them out if not tell the slvae
c       there is no more work
c
        if(numsent .lt. rows) then
c
c       send another row
c
         do j = 1,cols
          buffer(j) =a(numsent+1,j)
         enddo
         call MPI_SEND(buffer, cols,MPI_DOUBLE_PRECISION,
     *    sender, numsent+1, MPI_COMM_WORLD, ierr)
         numsent= numsent+1
         print *,'sent'
       print *,'process # ',sender,' row #',numsent
        else
c
c     Tell slave that there is no more work by sending an empty message
c
```

```
         call MPI_SEND(MPI_BOTTOM, 0, MPI_DOUBLE_PRECISION,
     *   sender, 0,MPI_COMM_WORLD, ierr)
         print *,'process ',sender,' has been told to stopped working'
       endif
        enddo
       else
c
c    slaves receives b
c
       call MPI_BCAST(b,cols, MPI_DOUBLE_PRECISION, master,
     *  MPI_COMM_WORLD, ierr)
c
c    done message received?
c     skip if more processes than work
c
       if (rank .gt. rows) goto 200
c
c     IF not then recieve 'buffer' from the master node and
c     get to work!
c
  90   call MPI_RECV(buffer, cols, MPI_DOUBLE_PRECISION, master,
     *  MPI_ANY_TAG,MPI_COMM_WORLD, status, ierr)
c
c     IF the no more work signal is recieved then quit
c
        if (status(MPI_TAG) .eq. 0) then
         goto 200
c
c      if not do some work
c
        else
         row = status(MPI_TAG)
c
c       reset anser to zero on each node
c
         ans = 0.0
c
c       perform loop sum_j a(i,j)*b(j)
c
         do j = 1,cols
        ans =ans+buffer(j)*b(j)
       enddo
c
c       send answer to the master
c
       call MPI_SEND(ans, 1,MPI_DOUBLE_PRECISION, master,
     *  row, MPI_COMM_WORLD, ierr)
c
c       go get another 'buffer' (if there is one left)
c
         goto 90
        endif
  200   continue
       endif
c
c    make sure answer is ok
```

```
c
      if(myid.eq.0)then
       print *,'Resulting Vector'
       do i=1,rows
        print *,'i,    ,MPI result   serial result'
        print *,i,'   ',c(i),'   ',d(i)
        if(c(i).ne.d(i))then
       print *, 'Something is messed up c(i) and d(i) should be the same!'
        endif
       enddo
      endif
c
c     stop this stupid program
c
   30 call MPI_FINALIZE(ierr)
      stop
      end
```