Lecture 13a – Basic Communication in MPI

- **Bounce Program to find bandwidth and latency of a computer**
  1. Start an even number of instances of the executable
  2. Pair them up (next and previous in a linear array)
  3. Send/receive a series of messages of increasing size and record the time that it takes to send/receive each message
  4. Repeat a number of times and average results
  5. Report *bandwidth* and *latency*

Definitions of Latency and Bandwidth

- ***Latency*** **is the amount of wall clock time that it takes for a message of zero length to be sent from one processor to another**
  – Latency penalized programs that send/receive a large number of messages
- ***Bandwidth*** **is the number of bytes/sec that can be sent from one processor to another**
  – Low bandwidth penalizes programs that send/receive a large amount of information
- **Ideal network has low latency and high bandwidth and is VERY expensive**

Bounce Program in Fortran 90/95

```
c
c-------------------------------------------------------------------
c     this program times blocking send/receives, and reports the
c     latency and bandwidth of the communication system.  it is
c     designed to run on an even number of nodes.
c
c     AA220 / CS238
c     Juan J. Alonso, October 2001
c
c-------------------------------------------------------------------
c
      program bounce
      parameter (maxcount=1000000)
      parameter (nrepeats=50)
      parameter (nsizes=7)
c
      implicit real*8 (a-h,o-z)
      include "mpif.h"
c
      dimension sbuf(maxcount), rbuf(maxcount)
      dimension length(nsizes)
      integer   status(mpi_status_size)
```


Bounce Program in Fortran 90/95

```
c
c-------------------------------------
c     define an array of message lengths
c-------------------------------------
c
      length(1) = 0
      length(2) = 1
      length(2) = 10
      length(3) = 100
      length(4) = 1000
      length(5) = 10000
      length(6) = 100000
      length(7) = 1000000
c
c-------------------------------------
c     initialize the send buffer to zero
c-------------------------------------
c
      do n=1,maxcount
      sbuf(n) = 0.0d0
      rbuf(n) = 0.0d0
      end do
c
c------------------------------------
c     set up the parallel environment
c------------------------------------
c
```

Bounce Program in Fortran 90/95

```
      call mpi_init(ierr)
      call mpi_comm_size(mpi_comm_world,nnodes,ierr)
      call mpi_comm_rank(mpi_comm_world,nodeid,ierr)
c
      if (nodeid.eq.0) write(*,*)'number of processors =',nnodes
c
c     signal error if an odd number of nodes is specified
c
      if (mod(nnodes,2) .ne. 0) then
         if (nodeid .eq. 0) then
            write(6,*) ' you must specify an even number of nodes.'
         end if
         call mpi_finalize(ierr)
      end if
c
c---------------------------------------------------------
c     send or receive messages, and time it.
c     even nodes send, odd nodes receive, then the reverse
c---------------------------------------------------------
c
      do ns=1, nsizes
         time1 = mpi_wtime()
         do nr=1, nrepeats
```


Bounce Program in Fortran 90/95

```
c-----------------------------------------------
c           send in one direction i->i+1
c-----------------------------------------------
c
         if (mod(nodeid,2) .eq. 0) then
         call mpi_send(sbuf, length(ns), mpi_real8, nodeid+1, 1,
     .                 mpi_comm_world, ierr)
         else
         call mpi_recv(rbuf, length(ns), mpi_real8, nodeid-1, 1,
     .                 mpi_comm_world, status, ierr)
         end if
c
c---------------------------------------------------
c           send in the reverse direction i+1->i
c---------------------------------------------------
c
         if (mod(nodeid,2) .eq. 1) then
         call mpi_send(sbuf, length(ns), mpi_real8, nodeid-1, 1,
     .                 mpi_comm_world, ierr)
         else
         call mpi_recv(rbuf, length(ns), mpi_real8, nodeid+1, 1,
     .                 mpi_comm_world, status, ierr)
         end if
         end do
         time2 = mpi_wtime()
```


Bounce Program in Fortran 90/95

```
c---------------------------------------------------
c           timings and report results
c---------------------------------------------------
c
         if (nodeid .eq. 0) then
         write(6,fmt='(a,i9,a,10x,a,f10.4,a)') 'msglen =',8*length(ns),
     &   ' bytes,','elapsed time =',0.5d3*(time2-time1)/nrepeats,' msec'
         call flush(6)
         end if
         if (ns .eq. 1) then
            tlatency = 0.5d6*(time2-time1)/nrepeats
         end if
         if (ns .eq. nsizes) then
            bw = 8.*length(ns)/(0.5d6*(time2-time1)/nrepeats)
         end if
      end do
c
c---------------------------------------------------------
c     report apporximate numbers for bandwidth and latency
c---------------------------------------------------------
c
```


Bounce Program in Fortran 90/95

```
c
c---------------------------------------------------------
c     report apporximate numbers for bandwidth and latency
c---------------------------------------------------------
c
      if (nodeid .eq. 0) then
         write(6,fmt='(a,f6.1,a)') 'latency =',tlatency,' microseconds'
         write(6,*) 'bandwidth =',bw,' mbytes/sec'
c        write(6,fmt='(a,f8.4,a)') 'bandwidth =',bw,' mbytes/sec'
         write(6,fmt='(a)') '(approximate values for mp_send/mp_recv)'
      end if
c
      call mpi_finalize(ierr)
      end
```

Bounce with Scali Cards on Previous Generation Matrx Beowulf Cluster (1.6 GHz Athlons)

```
With scali cards on matrx:
mpimon bounce_pgf -- n01 1 n02 1 n03 1 n04 1 n05 1 n06 1 n07 1 n08 1
  Number of processors = 8
msglen =        0 bytes,          elapsed time =    0.0499 msec
msglen =       80 bytes,          elapsed time =    0.3579 msec
msglen =      800 bytes,          elapsed time =    0.0173 msec
msglen =     8000 bytes,          elapsed time =    0.0620 msec
msglen =    80000 bytes,          elapsed time =    2.5253 msec
msglen =   800000 bytes,          elapsed time =   12.2922 msec
msglen =  8000000 bytes,          elapsed time =   90.0449 msec
latency =  49.9 microseconds
 bandwidth =    88.84453955776175       MBytes/sec
(approximate values for mp_send/mp_recv)
```


Bounce with 100BT Ethernet Switch on Previous Generation Matrx Beowulf Cluster

```
With 100BT Switched Ethernet on Beowulf cluster:
n8 3% /usr/local/mpich-1.2.0/bin/mpirun -np 8 ./bounce_eth
 Number of processors = 8
msglen =        0 bytes,          elapsed time =    0.0942 msec
msglen =       80 bytes,          elapsed time =    0.0991 msec
msglen =      800 bytes,          elapsed time =    0.2459 msec
msglen =     8000 bytes,          elapsed time =    0.9316 msec
msglen =    80000 bytes,          elapsed time =    7.1869 msec
msglen =   800000 bytes,          elapsed time =   71.6712 msec
msglen =  8000000 bytes,          elapsed time =  712.2905 msec
latency =  94.2 microseconds
 bandwidth =    11.23137338146257       MBytes/sec
(approximate values for mp_send/mp_recv)
```


Bounce on Previous Generation Multiprocessor Sun

```
With SUNWhpc MPI implementation:
junior:~/bounce /opt/SUNWhpc/bin/mprun -np 8 ./bounce
  Number of processors = 8
 msglen =        0 bytes,          elapsed time =    0.0066 msec
 msglen =       80 bytes,          elapsed time =    0.0102 msec
 msglen =      800 bytes,          elapsed time =    0.0606 msec
 msglen =     8000 bytes,          elapsed time =    0.0781 msec
 msglen =    80000 bytes,          elapsed time =    0.5065 msec
 msglen =   800000 bytes,          elapsed time =    4.7291 msec
 msglen =  8000000 bytes,          elapsed time =   46.0369 msec
 latency =   6.6 microseconds
  bandwidth = 173.77383012979993  MBytes/sec
 (approximate values for mp_bsend/mp_brecv)
```


Bounce with Myrinet on Beowulf Cluster

```
With Myrinet on Beowulf cluster:
n8 3% mpiexec bounce_myr
 Number of processors = 8
msglen =        0 bytes,          elapsed time =    0.0117 msec
msglen =       80 bytes,          elapsed time =    0.0149 msec
msglen =      800 bytes,          elapsed time =    0.0488 msec
msglen =     8000 bytes,          elapsed time =    0.2376 msec
msglen =    80000 bytes,          elapsed time =    1.2652 msec
msglen =   800000 bytes,          elapsed time =   12.3286 msec
msglen =  8000000 bytes,          elapsed time =  122.5315 msec
latency =  11.7 microseconds
 bandwidth =    65.28934971611689       MBytes/sec
(approximate values for mp_bsend/mp_brecv)
```

Bounce on Previous Generation Davistron Beowulf Cluster using PGI and OpenMPI 2.1 GHz Athlons)

```
With 1Gb Ethernet on davistron:
/share/apps/openmpi-pgi/bin/mpirun bounce_pgi
  Number of processors = 6
msglen =       0 bytes,          elapsed time =    0.0021 msec
msglen =      80 bytes,          elapsed time =    0.0023 msec
msglen =     800 bytes,          elapsed time =    0.0028 msec
msglen =    8000 bytes,          elapsed time =    0.0178 msec
msglen =   80000 bytes,          elapsed time =    0.0877 msec
msglen =  800000 bytes,          elapsed time =    1.1344 msec
msglen = 8000000 bytes,          elapsed time =   10.6623 msec
latency =   2.1 microseconds
 bandwidth =    750.3042635355571       MBytes/sec
```


Bounce on Previous Generation Davistron Beowulf Cluster using Gfortran and OpenMPI (2.1 GHz Athlons)

```
With 1Gb Ethernet on davistron:
/opt/openmpi/bin/mpirun bounce_gnu
  Number of processors = 6
msglen =       0 bytes,          elapsed time =    0.0013 msec
msglen =      80 bytes,          elapsed time =    0.0017 msec
msglen =     800 bytes,          elapsed time =    0.0024 msec
msglen =    8000 bytes,          elapsed time =    0.0120 msec
msglen =   80000 bytes,          elapsed time =    0.0846 msec
msglen =  800000 bytes,          elapsed time =    1.1014 msec
msglen = 8000000 bytes,          elapsed time =   10.3057 msec
latency =   1.3 microseconds
 bandwidth =   776.269240779406       MBytes/sec
```


Bounce on Vortex Beowulf Cluster using PGI and OpenMPI (3.2 GHz Athlons)

```
With 1Gb Ethernet on vortex:
/share/apps/openmpi-1.4.3-pgi-10.9/bin/mpirun bounce_pgi
  Number of processors = 8
msglen =       0 bytes,          elapsed time =    0.0024 msec
msglen =      80 bytes,          elapsed time =    0.0028 msec
msglen =     800 bytes,          elapsed time =    0.0037 msec
msglen =    8000 bytes,          elapsed time =    0.0218 msec
msglen =   80000 bytes,          elapsed time =    0.1253 msec
msglen =  800000 bytes,          elapsed time =    1.8298 msec
msglen = 8000000 bytes,          elapsed time =   18.1576 msec
latency =   2.4 microseconds
 bandwidth =    440.5856717617291       MBytes/sec
```


Bounce on Vortex Beowulf Cluster using Gfortran and OpenMPI (3.2 GHz Athlons)

```
With 1Gb Ethernet on davistron:
/opt/openmpi/bin/mpirun bounce_gnu
  Number of processors = 8
msglen =       0 bytes,          elapsed time =    0.0020 msec
msglen =      80 bytes,          elapsed time =    0.0027 msec
msglen =     800 bytes,          elapsed time =    0.0028 msec
msglen =    8000 bytes,          elapsed time =    0.0179 msec
msglen =   80000 bytes,          elapsed time =    0.1195 msec
msglen =  800000 bytes,          elapsed time =    1.8690 msec
msglen = 8000000 bytes,          elapsed time =   18.7854 msec
latency =   2.0 microseconds
 bandwidth =   425.86171031648576       MBytes/sec
```

Review: Send/Receive Types

- **Standard: similar to Blocking except receive will not allow processor to continue only until its buffer can be reused.**
- **Blocking: receive will not allow processor to continue until it has received its message. Receive acts as a Barrier to that processor.**
- **Synchronous: send (or receive) does not start until a matching receive (or send) is posted indicating it is ready. Send acts as "blocking" until matching receive occurs. In this case, send acts as a Barrier for those processors.**
- **Buffered: either a system or user-defined buffer is made available for send/receive so that communication can proceed.**

Review: Buffered/Non-buffered Communications

- **No-buffering (phone calls)**
  - Proc 0 initiates the send request and rings Proc 1. It waits until Proc 1 is ready to receive. The transmission starts.
  - Synchronous communication – completed only when the message was received by the receiving proc
- **Buffering (beeper)**
  - The message to be sent (by Proc 0) is copied to a system-controlled block of memory (buffer)
  - Proc 0 can continue executing the rest of its program
  - When Proc 1 is ready to receive the message, the system copies the buffered message to Proc 1
  - Asynchronous communication – may be completed even though the receiving proc has not received the message

Review: Buffered Communication

- **Buffering requires system resources, e.g. memory, and can be slower if the receiving proc is ready at the time of requesting the send**
- **Application buffer: address space that holds the data**
- **System buffer: system space for storing messages. In buffered communication, data in application buffer is copied to/from system buffer**
- **MPI allows communication in buffered mode:**
  MPI_Bsend, MPI_Ibsend
- **User allocates the buffer by:**
  MPI_Buffer_attach(buffer, buffer_size)
- **Free the buffer by MPI_Buffer_detach**

Review: Blocking / Non-blocking Communication

- **Blocking Communication (old McDonald's)**
  - The receiving proc has to wait if the message is not ready
  - Different from synchronous communication
    - Standard blocking occurs until buffer can be reused
    - Synchronous mode blocking occurs until receive has occurred
  - Proc 0 may have already buffered the message to system and Proc 1 is ready, but the interconnection network is busy
- **Non-blocking Communication (In & Out)**
  - Proc 1 checks with the system if the message has arrived yet. If not, it continues doing other stuff. Otherwise, get the message from the system.
- **Useful when computation and communication can be performed at the same time**
- **MPI allows both non-blocking send and receive**

MPI_Isend and MPI_Irecv

- **In non-blocking send, program identifies an area in memory to serve as a send buffer. Processing continues immediately without waiting for message to be copied out from the application buffer**
- **The program *should not* modify the application buffer until the non-blocking send has completed**
- **Non-blocking communication can be combined with non-buffering: MPI_Issend, or buffering: MPI_Ibsend**
- **Use MPI_Wait or MPI_Test to determine if the non-blocking send or receive has completed**

Non-Blocking Send Syntax

MPI_ISEND(buf, count, datatype, dest, tag, comm, request)
[IN buf] initial address of send buffer (choice)
[IN count] number of elements in send buffer (integer)
[IN datatype] datatype of each send buffer element (handle)
[IN dest] rank of destination (integer)
[IN tag] message tag (integer)
[IN comm] communicator (handle)
[OUT request] communication request (handle)

C:

```
int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag,MPI_Comm comm, MPI_Request *request)
```

Fortran 90/95:

```
MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST,
          IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
```


Non-Blocking Receive Syntax

MPI_IRECV (buf, count, datatype, source, tag, comm, request)
[OUT buf] initial address of receive buffer (choice)
[IN count] number of elements in receive buffer (integer)
[IN datatype] datatype of each receive buffer element (handle)
[IN source] rank of source (integer)
[IN tag] message tag (integer)
[IN comm] communicator (handle)
[OUT request] communication request (handle)

C:

```
int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm comm, MPI_Request *request)
```

Fortran 90/95:

```
MPI_IRECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM,
          REQUEST, IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST,
   IERROR
```