Introduction to MPI

Johnny Chang
NAS Division
NASA Ames Research Center

2012 Summer Short Course for Earth System Modeling and Supercomputing

Preliminaries

- What is MPI?
  - MPI -- Message Passing Interface is a library of functions (C/C++) or routines (Fortran) used by programmers to exploit parallelism in their codes to take advantage of multi-processor computers
- When to use MPI?
  - When it takes too long to solve a problem with a (serial) code running on a single processor, one needs to resort to using multiple processors to run the code in parallel
- Why MPI?
  - It has been the *de facto* standard since June, 1994, and its acceptance has grown with the rise of clusters in supercomputing
  - Codes written in MPI are portable to a wide variety of computers (contrast this to OpenMP, another widely-accepted approach to parallelism, which can run only on shared memory computers)

“Hello World” in Fortran

```
program hello_mpi
include "mpif.h”

call mpi_init(ierr)
call mpi_comm_rank(mpi_comm_world, myid, ierr)
call mpi_comm_size(mpi_comm_world, nprocs, ierr)

print *, 'Rank', myid, ’of', nprocs, 'has started.’

call mpi_finalize(ierr)
end
```

Example output:

```
Rank      0 of      12 has started.
Rank      1 of      12 has started.
Rank      3 of      12 has started.
Rank      4 of      12 has started.
Rank      2 of      12 has started.
Rank      5 of      12 has started.
Rank      8 of      12 has started.
Rank      9 of      12 has started.
Rank      7 of      12 has started.
Rank      6 of      12 has started.
Rank     10 of      12 has started.
Rank     11 of      12 has started.
```

Basic elements of all MPI programs

- include “mpif.h” or use mpi
  - MPI include file containing macro definitions and parameters
- call MPI_init(ierr)
  - MPI initialization to start all processes
- call MPI_comm_rank(MPI_COMM_WORLD, myrank, ierr)
  - Retrieves the rank, myrank, of the process. It’s an integer: 0, 1, 2, …, nprocs - 1
  - Each process has a unique value of myrank and it’s what distinguishes one process from another
  - MPI_COMM_WORLD is the default, pre-defined MPI communicator
  - A communicator is a collection of processes that can send messages to each other.
- call MPI_comm_size(MPI_COMM_WORLD, nprocs, ierr)
  - nprocs is the number of processes running this program
  - Typically used for work partitioning among the nprocs processes
- call MPI_finalize(ierr)
  - Terminates all MPI processing

“Hello World” in C

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
  int numprocs, rank;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Process %d out of %d is alive.\n", rank, numprocs);

  MPI_Finalize();
}
```

Retrieve programs from Pleiades and environmental setup

- login to any one of the Bridge nodes via ssh
- cp /tmp/SummerSchool.tar .
- tar xvf SummerSchool.tar
- add the following to your .cshrc file

```
set path = ($path /u/scicon/tools/bin .)
set prompt="`hostname -s`:`pwd`> ”
module load comp-intel/2012.0.032 mpi-sgi/mpt.2.06a67
alias ls '/bin/ls -CF'
alias cd 'cd \!* ; set prompt="`hostname -s`:`pwd`> "'
alias qstat 'qstat -W shares’
```

- or, to save typing:

cat /tmp/cshrc_mods >> ~/.cshrc

pi program (part 1 of 3)

```
program main
include "mpif.h"
double precision PI25DT
double precision starttime, endtime
parameter (PI25DT = 3.141592653589793238462643d0)
double precision mypi, pi, h, sum, x, f, a
integer n, myid, numprocs, i, ierr

f(a)= 4.d0 / (1.d0 + a*a)
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)
if (myid .eq. 0) then
  print *, 'Enter number of intervals:'
  read *, n
endif
```

pi program (part 2 of 3)

```
      starttime = MPI_WTIME()

      call MPI_BCAST(n,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      h = 1.d0/n
      sum = 0.d0
      do 20 i=myid+1, n, numprocs
        x = h * (float(i) - 0.5d0)
        sum = sum + f(x)
20    continue
      mypi = h * sum

      call MPI_REDUCE(mypi,pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,
     $     MPI_COMM_WORLD,ierr)

      endtime = MPI_WTIME()
```

pi program (part 3 of 3)

```
 if (myid.eq.0) then
 write (6,*) 'pi is', pi, 'Error is', abs(pi-PI25DT)
 write (6,*) 'time is ', endtime-starttime, ' seconds'
endif

30  call MPI_FINALIZE(ierr)
    stop
    end
```

MPI collectives: MPI_Bcast

MPI_Bcast - Broadcasts a message from the process with a specified rank (called a root) to all other processes of the group

```
<type> BUFFER(*)    <type> = real, integer, …
INTEGER count, datatype, root, comm, ierror

CALL MPI_BCAST(buffer, count, datatype, root, comm, ierror)
```

buffer = starting address of the data to be broadcast

count = number of entries in the buffer

datatype = data type of the buffer (one of the MPI_ datatypes)

root = rank of the process doing the broadcast (usually rank 0)

comm = the communicator (usually MPI_COMM_WORLD)

ierror = return code value for successful completion, which is in MPI_SUCCESS. MPI_SUCCESS is defined in the mpif.h file.

MPI datatypes

MPI datatype	Fortran datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

MPI collectives: MPI_Reduce

MPI_Reduce - Reduces values on all processes to a single value

```
<type> SENDBUF(*), RECVBUF(*)
INTEGER count, datatype, op, root, comm, ierror

CALL MPI_REDUCE(sendbuf, recvbuf, count, datatype, op,
                root, comm, ierror)
```

sendbuf = address of send buffer

recvbuf = address of receive buffer

count = number of elements in the send buffer

datatype = data type of the buffer (one of the MPI_ datatypes)

op = the reduce operation (one of the MPI_ operations)

root = rank of the process that will use the recvbuf (usually rank 0)

MPI reduction operations

Operation Name	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical And
MPI_BAND	Bitwise And
MPI_LOR	Logical Or
MPI_BOR	Bitwist Or
MPI_LXOR	Logical Exclusive Or
MPI_BXOR	Bitwise Exclusive Or
MPI_MAXLOC	Maximum and Location of Maximum
MPI_MINLOC	Minimum and Location of Minimum

MPI_Send

MPI_Send - Performs a basic send operation

```
<type> BUF(*)
INTEGER count, datatype, dest, tag, comm, ierror

CALL MPI_SEND(buf, count, datatype, dest, tag, comm, ierror)
```

<u>buffer</u> = starting address of the data to be sent

<u>count</u> = number of entries in the buffer

<u>datatype</u> = data type of the buffer (one of the MPI_ datatypes)

<u>dest</u> = rank of the destination process (the “receiver”)

<u>tag</u> = a message tag used for identification (like a luggage tag)

<u>comm</u> = the communicator (usually MPI_COMM_WORLD)

MPI_Recv

MPI_Recv - Provides a basic receive operation

```
<type> BUF(*)
INTEGER count, datatype, source, tag, comm,
            status(MPI_STATUS_SIZE), ierror

CALL MPI_RECV(buf, count, datatype, source, tag, comm,
                    status(MPI_STATUS_SIZE), ierror)
```

source = rank of the sending process (the “sender”),
can be MPI_ANY_SOURCE

tag = a message tag used for identification,
can be MPI_ANY_TAG

If MPI_ANY_SOURCE or MPI_ANY_TAG is used, then status(MPI_SOURCE) contains the source, and similarly status(MPI_TAG) will contain the tag.

References

- “Parallel Programming with MPI” by Peter S. Pacheco, Morgan Kaufmann Publishers Inc.
- “A User’s Guide to MPI” by Peter Pacheco, Web document both C and Fortran versions available for download
- many references on the web, google “Intro to MPI”
- manpages available on Pleiades (i.e., type: man mpi_recv)