TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

MPI Programming

IHPCSS

Nonblocking Communications

July 7-12, 2019

PRESENTED BY:

John Cazes

cazes@tacc.utexas.edu

Lecture and Lab slides available at: tinyurl.com/tacc-2019-ihpcss

Outline

Advantages of Message Passing
Background on MPI
Basic Information
Point to Point Communication
Nonblocking Communication
Wildcards
Probing
Collective Communications
"V" Operations
Derived Datatypes
Communicators

Nonblocking Communication

Nonblocking send: send call returns immediately, send actually occurs later

Nonblocking receive: receive call returns immediately. When received data is needed, call a wait subroutine

Nonblocking communication used to overlap communication with computation.

Can help prevent deadlock

Nonblocking Send with MPI_Isend

C

```
MPI_Request request
ierr = MPI_Isend(&buffer, count, datatype,
                 dest, tag, comm, &request)
```

Fortran

```
Integer :: request
call MPI_Isend(buffer, count, datatype,
               dest, tag, comm, request, ierr)
```

`request` is a new output parameter

Don't change data until communication is complete

Nonblocking Receive w/ MPI_Irecv

C

```
MPI_Request request;
ierr = MPI_Irecv(&buffer, count, datatype,
                 source, tag, comm, &request)
```

Fortran

```
Integer :: request
call MPI_Irecv(buffer, count, datatype,
               source, tag, comm, request, ierr)
```

`request` is a new output parameter

Don't change data until communication is complete

MPI_Wait Used to Complete Communication

Request from Isend or Irecv is input

- The completion of a send operation indicates that the sender is now free to update the data in the send buffer
- The completion of a receive operation indicates that the receive buffer contains the received message

`MPI_Wait` blocks until message specified by "request" completes

MPI_Wait Usage

C

```
MPI_Request request;
MPI_Status status;
ierr = MPI_Wait(&request, &status);
```

Fortran

```
Integer :: request
Integer :: status(MPI_STATUS_SIZE)
call MPI_Wait(request, status, ierr)
```

`MPI_Wait` blocks until message specified by "request" completes

MPI_Test

Similar to **`MPI_Wait`**, but does not block

Value of flags signifies whether a message has been delivered

C

```
int flag;
ierr= MPI_Test(&request,&flag,&status);
```

Fortran

```
Logical :: flag
Call MPI_Test(request,flag,status,ierr)
```

Non-Blocking Send Example

```
Logical :: flag
...
call MPI_Isend (buffer,count,datatype,dest, &
                tag,comm,request,ierr)
call MPI_Test (request,flag,status,ierr)

while (.not. flag) do
   Do other work ...
  call MPI_Test (request,flag,status,ierr)
end do
call MPI_Wait(request, status, ierr)
```

Nonblocking Send and Receive

Write a parallel program to send and receive data using MPI_Isend and MPI_Irecv

- Initialize MPI
- Have processor 0 send an integer to processor 1
- Have processor 1 receive an integer from processor 0
- Both processors check on message completion
- Quit MPI

Example of MPI_Isend/MPI_Irecv

```
#include <stdio.h>
#include "mpi.h"
#include <math.h>
/*************************************************
This is a simple asynchronous send/receive program
*************************************************/
int main(argc,argv)
int argc;
char *argv[];
{
    int mytask, numtasks;
    int tag,source,destination,count;
    int buffer;
    MPI_Status status;
    MPI_Request request;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&mytask);

   tag=1234;  source=0;  destination=1;  count=1;
     request=MPI_REQUEST_NULL;
```

```
  if ( mytask == source ) {
    buffer=5678;
    ierr =
    MPI_Isend(&buffer,count,MPI_INT,destination,tag,MPI_COMM_WORLD,&request);
  }
  else if (mytask == destination) {
    ierr =
    MPI_Irecv(&buffer,count,MPI_INT,source,tag,MPI_COMM_WORLD,&request);
  }
  ierr = MPI_Wait(&request, &status);
  if (mytask == source) {
    printf("processor %3d   sent    %6d\n", mytask, buffer);
  }
  else if (mytask == destination) {
    printf("processor %3d   got     %6d\n", mytask, buffer);
  }
  ierr = MPI_Finalize();
}
```

Example of MPI_Isend/MPI_Irecv

```
program Main
use mpi

    Integer :: mytask, ntasks
    Integer :: tag,source,destination,count
    Integer :: buffer
    Integer :: status(MPI_STATUS_SIZE)
    Logical :: request

    call MPI_Init(ierr)
    call MPI_Comm_size(MPI_COMM_WORLD,ntasks,ierr)
    call MPI_Comm_rank(MPI_COMM_WORLD,mytask,ierr)
```

```
    tag=1234; source=0; destination=1;count=1;
    request=MPI_REQUEST_NULL
    if (mytask == source) then
          buffer=5678
          call MPI_Isend(buffer,count,MPI_INT,destination,tag, &
                    MPI_COMM_WORLD,request,ierr)
    else if (mytask == destination) then
          call MPI_Irecv(buffer,count,MPI_INT,source,tag, &
                     MPI_COMM_WORLD,request,ierr)
    end if
    call MPI_Wait(request,status,ierr);
    if (mytask == source) then
          print*, “processor “, mytask, “ sent “, buffer
    else if (mytask == destination)then
          print*, “processor “, mytask, “ got “, buffer
    end if
    call MPI_Finalize(ierr)
End Program Main
```

MPI Data Types

MPI has many different predefined data types

- All your favorite C data types are included

MPI data types can be used in any communications operation

MPI Predefined Data Types in C

MPI datatype	C datatype
MPI_CHAR	char (treated as printable character)
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG (as a synonym)	signed long long int
MPI_SIGNED_CHAR	signed char (treated as integral value)
MPI_UNSIGNED_CHAR	unsigned char (treated as integral value)
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t (defined in <stddef.h>) (treated as printable character)

MPI Predefined Data Types in C

MPI datatype	C datatype
MPI_C_BOOL	_Bool
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPI_C_COMPLEX	float _Complex
MPI_C_FLOAT_COMPLEX	float _Complex
(as a synonym)	
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_BYTE	
MPI_PACKED	

MPI Predefined Data Types in F90

MPI datatype	Fortran datatype
MPI_INTEGER	Integer
MPI_REAL	Real
MPI_DOUBLE_PRECISION	Double Precision
MPI_COMPLEX	Complex
MPI_LOGICAL	Logical
MPI_CHARACTER	Character
MPI_BYTE	Byte
MPI_PACKED	MPI Packed

Status

The status parameter returns additional information for some MPI routines

- additional error status information
- additional information with wildcard parameters
- Item count information

C declaration : a predefined struct

```
MPI_Status status;
```

Fortran declaration : an array is used instead

```
Integer :: Status(MPI_STATUS_SIZE)
```

Accessing Status Information

The tag of a received message

C : `status.MPI_TAG`

Fortran : `status(MPI_TAG)`

The source of a received message

C : `status.MPI_SOURCE`

Fortran : `status(MPI_SOURCE)`

The error code of the MPI call

C : `status.MPI_ERROR`

Fortran : `status(MPI_ERROR)`

Other uses...

Wildcards

Enables programmer to avoid having to specify a tag and/or source.

```
Integer :: MPIstatus(MPI_STATUS_SIZE)
Integer :: buffer(5)
Integer :: error
call MPI_Recv(buffer, 5, MPI_INTEGER, &
                  MPI_ANY_SOURCE, MPI_ANY_TAG, &
                  MPI_COMM_WORLD,MPIstatus,error)
```

MPI_ANY_SOURCE and **MPI_ANY_TAG** are wild cards

MPIstatus array is used to get wildcard values

MPI_Probe

MPI_Probe allows incoming messages to be checked without actually receiving them.

- The user can then decide how to receive the data
- Useful when different action needs to be taken depending on the "who, what, and how much" information of the message

MPI_Probe

C

```
MPI_Probe(source, tag, comm, &status)
```

Fortran

```
MPI_Probe(SOURCE, TAG, COMM, STATUS, IERROR)
```

Parameters

Source: source rank, or **`MPI_ANY_SOURCE`**

Tag: tag value, or **`MPI_ANY_TAG`**

Comm: communicator

Status: status object

MPI_Probe Example - C

```
#include <stdio.h>
#include "mpi.h"
#include <math.h>
/*****************************************************
 * Program shows how to use probe and get_count
 * to find the size of an incoming message
 *****************************************************/
int main(argc,argv)
int argc;
char *argv[];
{

  int mytask, ntasks;
  int i, tag, ierr, icount;

  MPI_Request request;
  MPI_Status status;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&mytask);

/* Print out my rank and this run's PE size */
  printf("mytask = %3d  ntasks = %3d\n", mytask, ntasks);
```

MPI_Probe Example – C

```
tag=123;
  i=0;
  icount=0;
  if (mytask == 0) {
     i=100;
     icount=1;
     ierr = MPI_Send(&i,icount,MPI_INTEGER,1,tag,MPI_COMM_WORLD);
  }
  else if (mytask == 1) {
      ierr = MPI_Probe(0,tag,MPI_COMM_WORLD,&status);
      ierr = MPI_Get_count(&status,MPI_INTEGER,&icount);
      printf("getting count = %3d\n", icount);
      ierr = MPI_Recv(&i,icount,MPI_INT,0,tag,MPI_COMM_WORLD,&status);
      printf("i = %3d\n", i);
  }
  ierr = MPI_Finalize();
```

MPI_Probe Example - F90

```
Program Main
!** Program shows how to use probe and get_count
!** to find the size of an incoming message

    Integer :: mytask, ntasks
    Integer :: status(MPI_STATUS_SIZE)
    Integer :: i,tag,ierr,icount

    call MPI_Init(ierr)
    call MPI_Comm_size(MPI_COMM_WORLD,ntasks,ierr)
    call MPI_Comm_rank(MPI_COMM_WORLD,mytask,ierr)
!** print out my rank and this run's PE size
    print*, “mytask=“, mytask, “ ntasks=“, ntasks
```

MPI_Probe Example – F90

```
    tag=123
    i=0
    icount=0
    if (mytask == 0) then
       i=100
       icount=1
       call MPI_Send(i,icount,MPI_INTEGER,1,mytag,MPI_COMM_WORLD,ierr)
    else if (mytask == 1) then
        call MPI_Probe(0,tag,MPI_COMM_WORLD,status,ierr)
        call MPI_Get_count(status,MPI_INTEGER,icount,ierr)
        print*, "getting count=", icount
        call MPI_Recv(i,icount,MPI_INT,0,tag,MPI_COMM_WORLD, &
                      status,ierr)
        print*, "i=", i
    end if
    call MPI_Finalize(ierr)
End Program Main
```

License