Question 4 - Basic MPI commands

MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv, MPI_Finalize, MPI_Reduce, MPI_Wtime, MPI_Wtick, MPI_Barrier

MPI_INIT
Initialize the MPI Execution environment. This routine must be called before any other MPI routine is called; however, an exception is when MPI_Initialized is used due to the fact that MPI_Initialized may be called after the initialization of MPI Libraries to check if MPI has been initialized.

```
C syntax
#include <mpi.h>
int MPI_Init(int *argc,char ***argv);
```

```
C++ syntax
#include mpi.h
void MPI::Init(int& argc, char**& argv);
#include mpi.h
void MPI::Init();
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_INIT(INTEGER IERROR)
```

MPI_COMM_SIZE
Returns the size of the group associated with a communicator. This function indicates the number of processes involved in a communicator. this function returns the size of the local group. To return the size of the remote group, use the MPI_Comm_remote_size function.

```
C syntax
#include <mpi.h>
int MPI_Comm_size(MPI_Comm comm,int *size);
```

```
C++ syntax
#include mpi.h
int MPI::Comm::Get_size() const;
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_COMM_SIZE (INTEGER COMM,INTEGER SIZE,INTEGER IERROR)
```

MPI_COMM_RANK
Returns the rank of the local task in the group associated with a communicator. You can use this subroutine with MPI_COMM_SIZE to determine the amount of concurrency available for a specific job.

```
C syntax
#include <mpi.h>
int MPI_Comm_rank(MPI_Comm comm,int *rank);
```

```
C++ syntax
#include mpi.h
int MPI::Comm::Get_rank() const;
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_COMM_RANK(INTEGER COMM,INTEGER RANK,INTEGER IERROR)
```

MPI_SEND
Performs a blocking standard mode send operation. This subroutine is a blocking standard mode send operation. MPI_SEND causes count elements of type datatype to be sent from buf to the task specified by dest. dest is a task rank that can be any value from 0 to (n-1), where n is the number of tasks in comm.

```
C syntax
#include <mpi.h>
int MPI_Send(void* buf,int count,MPI_Datatype datatype,int dest,
int tag,MPI_Comm comm);
```

```
C++ syntax
#include mpi.h
void MPI::Comm::Send(const void* buf, int count, const MPI::
Datatype& datatype,int dest, int tag) const;
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_SEND(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER DEST,
 INTEGER TAG,INTEGER COMM,INTEGER IERROR)
```

MPI_RECV
Performs a blocking receive operation. MPI_RECV is a blocking receives operation. The receive buffer is storage containing room for count consecutive elements of the type specified by datatype, starting at address buf. The message received must be less than or equal to the length of the receive buffer. If all incoming messages do not fit without truncation, an overflow error occurs. If a message arrives that is shorter than the receive buffer, then only those locations corresponding to the actual message are changed.

```
C syntax
#include <mpi.h>
int MPI_Recv(void* buf,int count,MPI_Datatype datatype, int
source,int tag,
MPI_Comm comm,MPI_Status *status);
```

```
C++ syntax
#include mpi.h
void MPI::Comm::Recv(void* buf, int count, const MPI::Datatype&
datatype, int source, int tag) const;
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_RECV(CHOICE BUF,INTEGER COUNT,INTEGER DATATYPE,INTEGER SOURCE,
      INTEGER TAG,INTEGER COMM,INTEGER STATUS(MPI_STATUS_SIZE),
INTEGER IERROR)
```

MPI_REDUCE
Applies a reduction operation to the vector sendbuf over the set of tasks specified by comm and places the result in recvbuf on root. The input buffer and the output buffer have the same number of elements with the same type. The arguments sendbuf, count, and datatype define the send or input buffer. The arguments recvbuf, count and datatype define the output buffer. MPI_REDUCE is called by all group members using the same arguments for count, datatype, op, and root.

```
C syntax
#include <mpi.h>
int MPI_Reduce(void* sendbuf,void* recvbuf,int count,MPI_Datatype
datatype,MPI_Op op,int root,MPI_Comm comm);
```

```
C++ syntax
#include mpi.h
void MPI::Comm::Reduce(const void* sendbuf, void* recvbuf, int
count,const
MPI::Datatype& datatype, const MPI::Op& op, int root) const;
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_REDUCE(CHOICE SENDBUF,CHOICE RECVBUF,INTEGER COUNT, INTEGER
DATATYPE,
INTEGER OP,INTEGER ROOT,INTEGER COMM, INTEGER IERROR)
```

MPI_FINALIZE
Terminates all MPI processing.

```
C syntax
#include <mpi.h>
int MPI_Finalize(void);
```

```
C++ syntax
#include mpi.h
void MPI::Finalize();
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_FINALIZE(INTEGER IERROR)
```

MPI_WTIME
Measure the number of seconds that elapse from the time the execution is initialized until the program terminates.

```
C syntax
#include <mpi.h>
int MPI_Wtime(void);
```

```
C++ syntax
#include mpi.h
void MPI::Wtime();
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_Wtime(INTEGER IERROR)
MPI_INIT
Initialize the MPI Execution environment. This routine must be
called before any other MPI routine is called; however, an
exception is when MPI_Initialized is used due to the fact that
MPI_Initialized may be called after the initialization of MPI
Libraries to check if MPI has been initialized.
```

MPI_WTICK
Find the accuracy of the timer.

```
C syntax
#include <mpi.h>
int MPI_Wtick(void);
```

```
C++ syntax
#include mpi.h
void MPI::Wtick();
```

```
FORTRAN syntax
include 'mpif.h' or use mpi
MPI_Wtick(INTEGER IERROR)
MPI_INIT
Initialize the MPI Execution environment. This routine must be
called before any other MPI routine is called; however, an
exception is when MPI_Initialized is used due to the fact that
MPI_Initialized may be called after the initialization of MPI
Libraries to check if MPI has been initialized.
```

MPI_BARRIER
Perform a barrier synchronization. None of the processes can pass this barrier until all of them have reach it.

```
C syntax
#include <mpi.h>
int MPI_Barrier(void);
```

```
C++ syntax
#include mpi.h
void MPI::Barrier();
```

```
FORTRAN syntaxa
include 'mpif.h' or use mpi
MPI_Barrier(INTEGER IERROR)
MPI_INIT
Initialize the MPI Execution environment. This routine must be
called before any other MPI routine is called; however, an
exception is when MPI_Initialized is used due to the fact that
MPI_Initialized may be called after the initialization of MPI
Libraries to check if MPI has been initialized.
```