MPI Quiz

What is MPI?

- A the Message-Passing Interface
- B the Miami Police Investigators
- C the Minimal Polynomial Instantiation
- D the Millipede Podiatry Institution
- E a way of doing distributed-memory parallel programming

SUBMIT ANSWER

To compile and run an MPI program requires

- A special compilers
- B special libraries
- C a special parallel computer
- D a special operating system

SUBMIT ANSWER

After initiating an MPI program with "mpirun -n 4 ./mympiprogram", what does the call to MPI_Init do?

A create the 4 parallel processes

B start program execution

C enable the 4 independent programs subsequently to communicate with each other

D create the 4 parallel threads

SUBMIT ANSWER

If you call MPI_Recv and there is no incoming message, what happens?

- A the Recv fails with an error
- B the Recv reports that there is no incoming message
- C the Recv waits until a message arrives (potentially waiting forever)
- D the Recv times out after some system-specified delay (e.g. a few minutes)

SUBMIT ANSWER

If you call MPI synchronous send (MPI_Ssend) and there is no receive posted

A. the message disappears

B. the send fails

C. the send waits until a receive is posted (potentially waiting forever)

D. the message is stored and delivered later on (if possible)

E. the send times out after some system-specified delay (e.g. a few minutes)

SUBMIT ANSWER

If you call MPI asynchronous send (MPI_Bsend - buffered send) and there is no receive posted

- A the message disappears
- B the send fails
- C the send waits until a receive is posted (potentially waiting forever)
- D the message is stored and delivered later on (if possible)
- E the send times out after some system-specified delay (e.g. a few minutes)
- F the sending process continues execution regardless of whether the message is received

SUBMIT ANSWER

If you call a standard send (MPI_Send) and there is no matching receive, which of the following are possible outcomes?

- A the message disappears
- B the send fails
- C the send waits until a receive is posted (potentially waiting forever)
- D the message is stored and delivered later on (if possible)
- E the send times out after some system-specified delay (e.g. a few minutes)
- F the program continues execution regardless of whether the message is received

SUBMIT ANSWER

The MPI receive routine has a parameter "count" - what does this mean?

- A the size of the incoming message (in bytes)
- B the size of the incoming message (in items, e.g. integers)
- C the size you have reserved for storing the message (in bytes)
- D the size you have reserved for storing the message (in items, e.g. integers)

SUBMIT ANSWER

What happens if the incoming message is larger than "count" ?

A the receive fails with an error

B the receive reports zero data received

C the message writes beyond the end of the available storage

D only the first "count" items are received

SUBMIT ANSWER

What happens if the incoming message (of size "n") is smaller than "count"

A. the receive fails with an error

B. the receive reports zero data received

C. the first "n" items are received

D. the first "n" items are received and the rest of the storage is zeroed

SUBMIT ANSWER

How is the actual size of the incoming message reported?

- A the value of "count" in the receive is updated
- B MPI cannot tell you
- C it is stored in the Status parameter
- D via the associated tag

SUBMIT ANSWER

Consider the following (pseudo) code - remember that Ssend means Synchronous Send. What happens at runtime?

Process A

MPI_Ssend(sendmsg1, B, tag=1)
MPI_Ssend(sendmsg2, B, tag=2)

Process B

MPI_Recv(recvmsg2, A, tag=2)
MPI_Recv(recvmsg1, A, tag=1)

A The code is guaranteed to deadlock

B The code might deadlock

C recvmsg1 = sendmsg1 and recvmsg2 = sendmsg2

D recvmsg1 = sendmsg2 and recvmsg2 = sendmsg1

E both receives complete but their contents are undefined

SUBMIT ANSWER

Consider the following (pseudo) code - remember that Isend is a non-blocking / immediate send which means that the program always continues execution to the next line. What happens at runtime?

It is most useful to consider the case where Process A is running ahead of B, i.e. the sends are all posted in advance of the receives.

(if you prefer you can consider using Bsend - i.e. buffered / asynchronous send - as the answer will be the same).

Process A

```
MPI_Isend(sendmsg1, B, tag=1)
MPI_Isend(sendmsg2, B, tag=1)
```

Process B

```
MPI_Recv(recvmsg1, A, tag=1)
MPI_Recv(recvmsg2, A, tag=1)
```

- A The code is guaranteed to deadlock
- B The code might deadlock
- C recvmsg1 = sendmsg1 and recvmsg2 = sendmsg2
- D recvmsg1 = sendmsg2 and recvmsg2 = sendmsg1
- E both receives complete but their contents are undefined

Consider the following (pseudo) code - remember that Isend is a non-blocking / immediate send. What happens at runtime?

```
Process A
--------------

MPI_Isend(sendmsg1, B, tag=1)
MPI_Isend(sendmsg2, B, tag=2)

Process B
--------------

MPI_Recv(recvmsg2, A, tag=2)
MPI_Recv(recvmsg1, A, tag=1)
```

- A The code is guaranteed to deadlock
- B The code might deadlock
- C recvmsg1 = sendmsg1 and recvmsg2 = sendmsg2
- D recvmsg1 = sendmsg2 and recvmsg2 = sendmsg1
- E both receives complete but their contents are undefined

SUBMIT ANSWER

Consider the following (pseudo) code - remember that Isend is a non-blocking / immediate send. What happens at runtime?

Process A

MPI_Isend(sendmsg1, B, tag=1)
MPI_Isend(sendmsg2, B, tag=2)

Process B

MPI_Recv(recvmsg1, A, tag=MPI_ANY_TAG)
MPI_Recv(recvmsg2, A, tag=MPI_ANY_TAG)

- A The code is guaranteed to deadlock
- B The code might deadlock
- C recvmsg1 = sendmsg1 and recvmsg2 = sendmsg2
- D recvmsg1 = sendmsg2 and recvmsg2 = sendmsg1
- E both receives complete but their contents are undefined

SUBMIT ANSWER

Consider the following (pseudo) code - remember that Isend is a non-blocking / immediate send. What happens at runtime?

Note that the code is written so that the time ordering in which the MPI functions are called is guaranteed to be: Send on A; Send on B; Recv on C.

Process A

```
MPI_Isend(sendmsgA, C)
MPI_Barrier()
```

Process B

```
MPI_Barrier()
MPI_Isend(sendmsgB, C)
```

Process C

```
MPI_Barrier()
MPI_Recv(recvmsgA, source=MPI_ANY_SOURCE)
MPI_Recv(recvmsgB, source=MPI_ANY_SOURCE)
```

A The code is guaranteed to deadlock

B The code might deadlock

MPI_Isend(sendmsgA, C)
MPI_Barrier()

Process B

MPI_Barrier()
MPI_Isend(sendmsgB, C)

Process C

MPI_Barrier()
MPI_Recv(recvmsgA, source=MPI_ANY_SOURCE)
MPI_Recv(recvmsgB, source=MPI_ANY_SOURCE)

- A The code is guaranteed to deadlock
- B The code might deadlock
- C recvmsgA = sendmsgA and recvmsgB = sendmsgB
- D recvmsgA = sendmsgB and recvmsgB = sendmsgA
- E both receives complete but their contents are undefined

SUBMIT ANSWER

Consider the following (pseudo) code - remember that Isend is a non-blocking / immediate send.

Which of the following are possible outcomes where we send 10 integers and receive 10 real numbers?

Process A

MPI_Send(B, sendmsg1, 10, MPI_INT)

Process B

MPI_Recv(A, recvmsg1, 10, MPI_FLOAT)

A The call is erroneous so MPI reports an error

B The integers are converted to floats and stored in recvmsg1

C The message is not delivered as the send and receive do not match, and the program continues

D The message is delivered but the contents of recvmsg1 are potentially garbage

E The send does not match the receive, so the receive keeps waiting for a message of type MPI_INT

SUBMIT ANSWER

Some MPI collective calls specify both a send type and a receive type, e.g. MPI_Scatter(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, ...).

However, the vast majority of times you see this call used in practice we have sendtype = recvtype (and also sendcount=recvcount).

Why does MPI make you specify both types?

- A So it can check at runtime that you haven't made a silly mistake
- B So it can do type conversion (e.g integer -> float) if required
- C The types and counts can be different provided that at least one of them is an MPI derived type
- D The types and counts can be different provided that the two buffers are the same length in bytes

SUBMIT ANSWER

What is the output of this MPI code on 8 processes, i.e. on running ranks 0, 1, 2, 3, 4, 5, 6 and 7?

```
if (rank % 2 == 0) // Even processes
{
 MPI_Allreduce(&rank, &evensum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
 if (rank == 0) printf("evensum = %d\n", evensum);
}
else // Odd processes
{
 MPI_Allreduce(&rank, &oddsum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
 if (rank == 1) printf("oddsum = %d\n", oddsum);
}
```

- A evensum = 16, oddsum = 12
- B evensum = 28, oddsum = 28
- C evensum = 12, oddsum = 16
- D evensum = 6, oddsum = 22

SUBMIT ANSWER

You receive an MPI program from a colleague and see that it has a large number of calls to MPI_Barrier(). Which of these are plausible explanations (assuming the program uses relatively standard two-sided MPI functionality and doesn't push the boundaries of the standard)

- A The barriers are required to ensure consistent timing of various parallel operations, but have no impact on program correctness
- B The barriers are required for program correctness as it uses lots of non-blocking operations
- C The barriers are unnecessary and can safely be removed if the program is otherwise correct
- D The barriers are required to ensure that subsequent collective operations can be called safely

SUBMIT ANSWER

Was this tutorial useful?

- T True
- F False

SUBMIT ANSWER

The End

archer safe epcc THE UNIVERSITY OF EDINBURGH