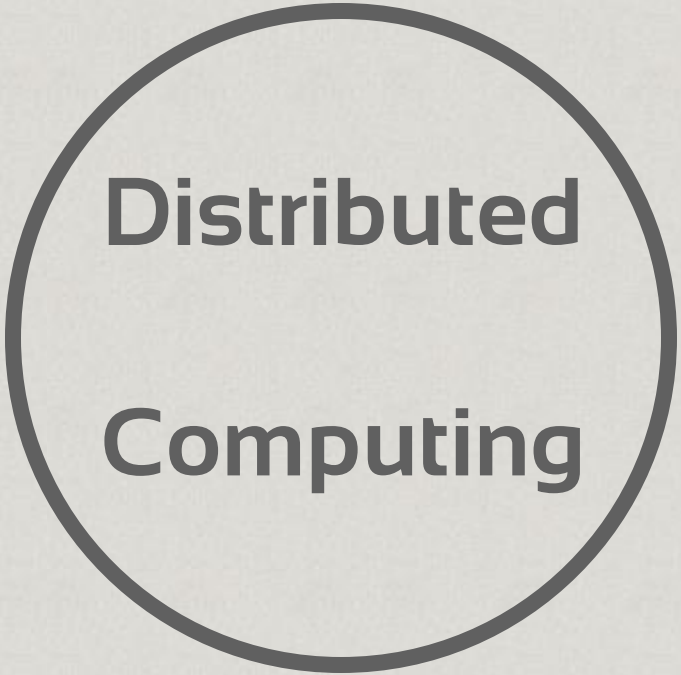

MPI

Mahmoud Badry, mma18@fayoum.edu.eg

AGENDA

Non-blocking Operations

PTP deadlocks

End

Questions

Mahmoud Badry, mma18@fayoum.edu.eg

Deadlocks with Point-to-Point Communications

Mahmoud Badry, mma18@fayoum.edu.eg

What happends while sending and receiving ?

1. The data elements to be sent are **copied** from the send buffer **smessage** specified as parameter into a **system buffer** of the MPI runtime system. The **message** is **assembled** by adding a **header** with information on the **sending process**, the **receiving process**, the **tag**, and the **communicator** used.
2. The **message** is sent via the **network** from the **sending** process to the **receiving** process.
3. At the **receiving** side, the data entries of the **message** are **copied** from the **system buffer** into the **receive buffer** rmessage specified by MPI_Recv().

Mahmoud Badry, mma18@fayoum.edu.eg

Deadlocks with Point-to-Point Communications

- Send and receive operations must be used with care, since deadlocks can occur in ill-constructed programs.

```
/* program fragment which always causes a deadlock */
MPI_Comm_rank (comm, &my_rank);
if (my_rank == 0) {
   MPI_Recv (recvbuf, count, MPI_INT, 1, tag, comm, &status);
   MPI_Send (sendbuf, count, MPI_INT, 1, tag, comm);
}
else if (my_rank == 1) {
   MPI_Recv (recvbuf, count, MPI_INT, 0, tag, comm, &status);
   MPI_Send (sendbuf, count, MPI_INT, 0, tag, comm);
}
```


Mahmoud Badry, mma18@fayoum.edu.eg

Deadlocks with Point-to-Point Communications cont.

```
/* program fragment for which the occurrence of a deadlock
   depends on the implementation */
MPI_Comm_rank (comm, &my_rank);
if (my_rank == 0) {
   MPI_Send (sendbuf, count, MPI_INT, 1, tag, comm);
   MPI_Recv (recvbuf, count, MPI_INT, 1, tag, comm, &status);
}
else if (my_rank == 1) {
   MPI_Send (sendbuf, count, MPI_INT, 0, tag, comm);
   MPI_Recv (recvbuf, count, MPI_INT, 0, tag, comm, &status);
}
```

- Message transmission is performed correctly here without deadlock, if the MPI runtime system uses system buffers. But a deadlock occurs, if the runtime system does not use system buffers or if the system buffers used are too small.

Secure implementation

- A secure implementation which does not cause deadlocks even if no system buffers are used.

```
/* program fragment that does not cause a deadlock */
MPI_Comm_rank (comm, &myrank);
if (my_rank == 0) {
   MPI_Send (sendbuf, count, MPI_INT, 1, tag, comm);
   MPI_Recv (recvbuf, count, MPI_INT, 1, tag, comm, &status);
}
else if (my_rank == 1) {
   MPI_Recv (recvbuf, count, MPI_INT, 0, tag, comm, &status);
   MPI_Send (sendbuf, count, MPI_INT, 0, tag, comm);
}
```


Mahmoud Badry, mma18@fayoum.edu.eg

Secure implementation Cont.

- An MPI program is called secure if the correctness of the program does not depend on assumptions about specific properties of the MPI runtime system, like the existence of system buffers or the size of system buffers.

Mahmoud Badry, mma18@fayoum.edu.eg

Send and recieve operation?!

- Suppose that we made a program which has some processes. Each process is sending and receiving a message in the same time. If we've imagined a secure implementation for this, the phases would be like this:

Phase	Process 0	Process 1	Process 2
1	MPI_Send() to 1	MPI_Recv() from 0	MPI_Send() to 0
2	MPI_Recv() from 2	MPI_Send() to 2	-wait-
3		-wait-	MPI_Recv() from 1

Mahmoud Badry, mma18@fayoum.edu.eg

MPI_Sendrecv operation

- ```
  int MPI Sendrecv (void *sendbuf,
                    int sendcount,
                    MPI Datatype sendtype,
                    int dest,
                    int sendtag,
                    void *recvbuf,
                    int recvcount,
                    MPI Datatype recvtype,
                    int source,
                    int recvtag,
                    MPI Comm comm,
                    MPI Status *status);
  ```


Mahmoud Badry, mma18@fayoum.edu.eg
  ```

MPI_Sendrecv operation cont.

- Using MPI_Sendrecv(), the programmer does not need to worry about the order of the send and receive operations. The MPI runtime system guarantees deadlock freedom, also for the case that no internal system buffers are used.
- The parameters sendbuf and recvbuf, specifying the send and receive buffers of the executing process, must be disjoint, non-overlapping memory locations.
- There is a variant of MPI_Sendrecv() for which the send buffer and the receive buffer are identical.

Mahmoud Badry, mma18@fayoum.edu.eg

MPI_Sendrecv_replace operation

```
int MPI_Sendrecv_replace (void *buffer,
                          int count,
                          MPI Datatype type,
                          int dest,
                          int sendtag,
                          int source,
                          int recvtag,
                          MPI Comm comm,
                          MPI Status *status).
```

- Here, buffer specifies the buffer that is used as both send and receive buffer. For this function, count is the number of elements to be sent and to be received; these elements now should have identical type.

Mahmoud Badry, mma18@fayoum.edu.eg

Running „Send recieve operation“ example

Mahmoud Badry, mma18@fayoum.edu.eg

Non-blocking Operations in MPI

Mahmoud Badry, mma18@fayoum.edu.eg

Why Non-blocking Operations?

- The use of **blocking** communication operations can lead to **waiting** times in which the blocked process does not perform **useful** work.
- For **example**, a process executing a blocking **send** operation must **wait** until the send buffer has been copied into a system buffer or even until the message has completely arrived at the receiving process if no system buffers are used.

Mahmoud Badry, mma18@fayoum.edu.eg

Non-blocking Send

- A non-blocking send operation initiates the sending of a message and returns control to the sending process as soon as possible. Upon return, the send operation has been started, but the send buffer specified cannot be reused safely.
-
```
int MPI Isend (void *buffer,
               int count,
               MPI Datatype type,
               int dest,
               int tag,
               MPI Comm comm,
               MPI Request *request).
```


Mahmoud Badry, mma18@fayoum.edu.eg

Non-blocking Send cont.

- There is an additional parameter of type MPI_Request which denotes an opaque object that can be used for the identification of a specific communication operation.

Mahmoud Badry, mma18@fayoum.edu.eg

Non-blocking Recieve

- A non-blocking receive operation initiates the receiving of a message and returns control to the receiving process as soon as possible. Upon return, the receive operation has been started and the runtime system has been informed that the receive buffer specified is ready to receive data.
-
```
int MPI Irecv (void *buffer,
               int count,
               MPI Datatype type,
               int source,
               int tag,
               MPI Comm comm,
               MPI Request *request).
```

MPI Test

- We can call MPI_Test to check the status of send or receive operation.
- ```
  int MPI Test (MPI Request *request,
                int *flag,
                MPI Status *status)
  ```
- The call returns flag = 1 (true), if the communication operation identified by request has been completed. Otherwise, flag = 0 (false) is returned. If request denotes a receive operation and flag = 1 is returned, the parameter status contains information on the message received as described for MPI Recv().

Mahmoud Badry, mma18@fayoum.edu.eg
  ```

MPI Wait

- The MPI_Wait function can be used to wait for the completion of a non-blocking communication operation.
- ```
  int MPI_Wait (MPI Request *request,
                MPI Status *status)
  ```
- Blocking and non-blocking operations can be mixed, i.e., data sent by MPI Isend() can be received by MPI Recv() and data sent by MPI Send() can be received by MPI Irecv().

Mahmoud Badry, mma18@fayoum.edu.eg
  ```

Non blocking example

```
void Gather_ring_nb (float x[], int blocksize, float y[])
{
   int i, p, my_rank, succ, pred;
   int send_offset, recv_offset;
   MPI_Status status;
   MPI_Request send_request, recv_request;

   MPI_Comm_size (MPI_COMM_WORLD, &p);
   MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
   for (i=0; i<blocksize; i++)
      y[i+my_rank * blocksize] = x[i];
   succ = (my_rank+1) % p;
   pred = (my_rank-1+p) % p;
   send_offset = my_rank * blocksize;
   recv_offset = ((my_rank-1+p) % p) * blocksize;
   for (i=0; i<p-1; i++) {
      MPI_Isend (y+send_offset, blocksize, MPI_FLOAT, succ, 0,
         MPI_COMM_WORLD, &send_request);
      MPI_Irecv (y+recv_offset, blocksize, MPI_FLOAT, pred, 0,
         MPI_COMM_WORLD, &recv_request);
      send_offset = ((my_rank-i-1+p) % p) * blocksize;
      recv_offset = ((my_rank-i-2+p) % p) * blocksize;
      MPI_Wait (&send_request, &status);
      MPI_Wait (&recv_request, &status);
   }
}
```


Mahmoud Badry, mma18@fayoum.edu.eg

QUESTIONS

Mahmoud Badry, mma18@fayoum.edu.eg

THANK YOU!

Mahmoud Badry, mma18@fayoum.edu.eg