Further MPI Programming

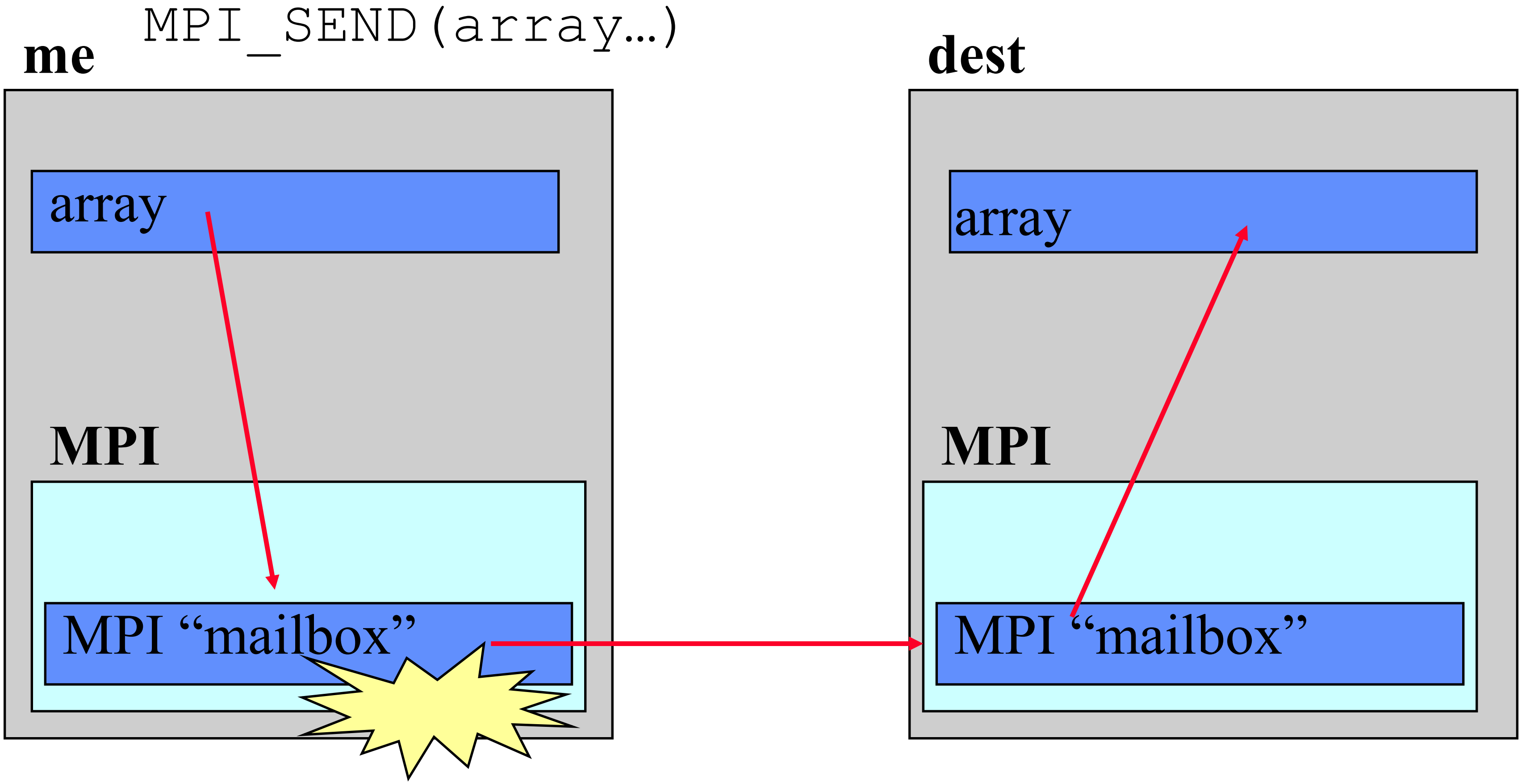
Paul Burton

April 2015

Blocking v Non-blocking communication

- **Blocking communication**
  - **Call to MPI "sending" routine does not return until the "send" buffer (array) is safe to use again**
    - **This does not necessarily mean the data has been sent and received by the remote task**
  - **Call to MPI "receiving" routine does not return until the "receive" buffer has received all the data in the incoming message**
- **Non-blocking communication**
  - **Call to MPI routine returns immediately**
  - **Further MPI calls are required to check the progress of the communication**
  - **Allows other work to be done during communication**
- **Cray's `MPI_SEND` can sometimes be blocking and sometimes non-blocking!**

MPI_SEND : Eager protocol

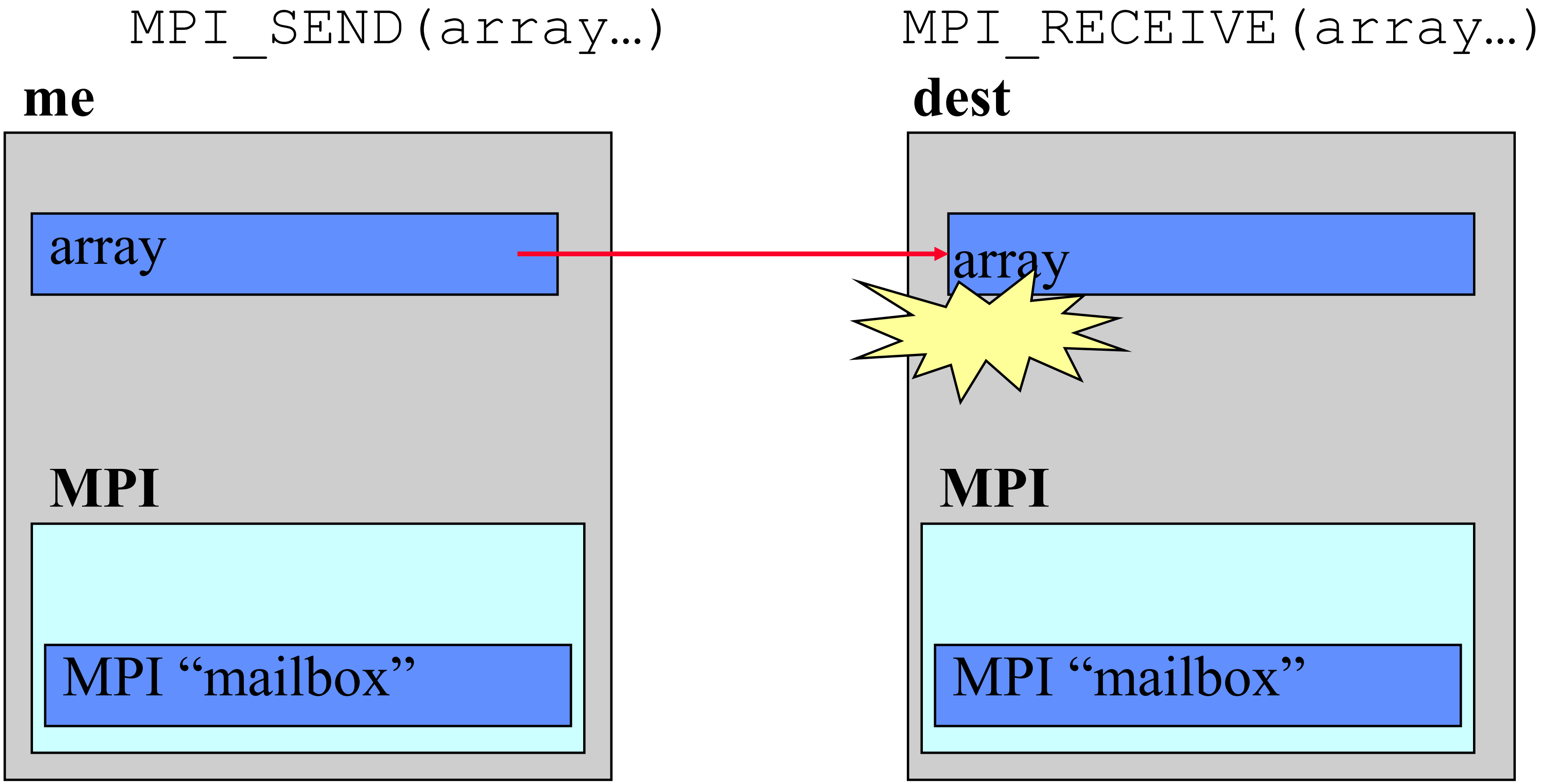


MPI_SEND completes when “array” is copied into “mailbox” on the sending task

MPI_SEND : Eager protocol

- The MPI layer has copied the data elsewhere
  - using internal buffer/mailbox space on the sending task
- `MPI_SEND` returns as soon as the message has been copied
  - The message is then “in transit” but not necessarily in the receivers array
- Used for short messages
  - By default “short” is 8192 bytes (8Kb) on the Cray
  - Can be modified by envioronment vairable
    - `$ export MPICH_GNI_MAX_EAGER_MSG_SIZE=X` *(bytes)*
    - Maximum permitted value 131072 bytes (128Kb)
- No need to worry if the remote task has done an “`MPI_RECEIVE`”

MPI_SEND : Rendezvous protocol



`MPI_SEND` completes when “array” is copied into “array” on the receiving task

MPI_SEND : Rendezvous protocol

- **`MPI_SEND` does not return until the message has been successfully received by the remote task**
- **Used for long messages**
  - **By default “long” is >8192 bytes on the Cray**
- **Need to ensure that remote task is doing an “`MPI_RECEIVE`” otherwise we may deadlock…**
  - **Easily done!**
  - **eg. ping-pong example – 2 tasks exchanging messages…**

```
if(me .eq.0) then
  other=1
else
  other=0
endif

call MPI_SEND(sbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,ierror)
call MPI_RECV(rbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,stat,ierror)
```

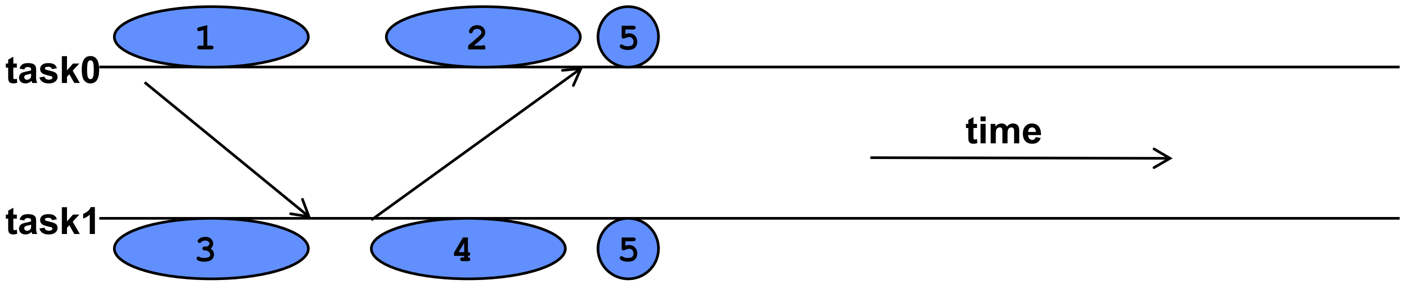
Solutions to Send/Send deadlocks

- **My advice – avoid `MPI_SEND/MPI_RECV`!**
  - Behaviour is implementation dependent – code may work, but then stop working when message size changes or move to another platform
- **Pair up sends and receives (next slide shows how…)**
  - But this is not very efficient
- **use `MPI_SENDRECV`**
  - Hopefully more efficient
- **use a buffered send (like the eager protocol, but user space buffering)**
  - `MPI_BSEND`
- **use asynchronous sends/receives**
  - `MPI_ISEND/MPI_IRECV`

Paired Sends and Receives

- **More complex code, and close synchronisation**
- **Less efficient**
  - **task 1 has to wait until it has received message from task 0 before it can send its message**

```
  if (me .eq. 0) then
     other=1
1    call MPI_SEND(sbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,ierror)
2    call MPI_RECV(rbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,stat,ierror)
  else
     other=0
3    call MPI_RECV(rbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,stat,ierror)
4    call MPI_SEND(sbuff,n,MPI_REAL8,other,tag,MPI_COMM_WORLD,ierror)
  endif
5
```



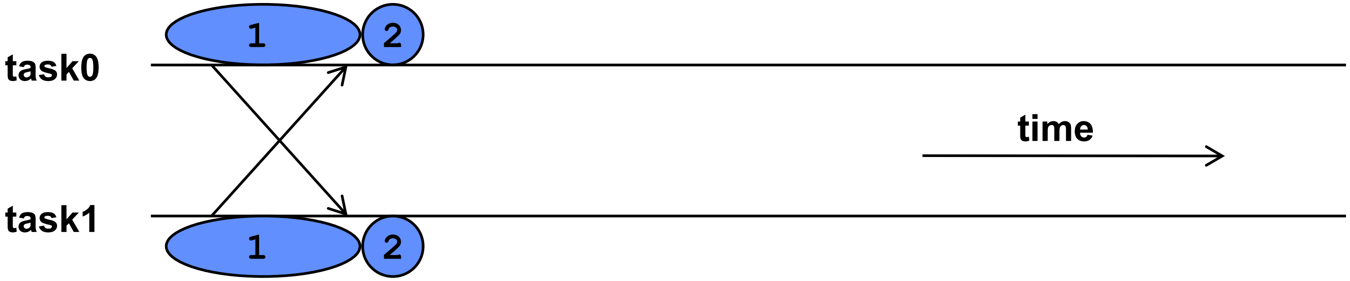
MPI_SENDRECV

- **Simpler to code & hopefully more efficient**
- **Still implies close synchronisation**

1

```
call MPI_SENDRECV(sbuff,n,MPI_REAL8,other,1, &
                  rbuff,n,MPI_REAL8,other,1, &
                  MPI_COMM_WORLD,stat,ierror)
```

2



MPI_BSEND

- This performs a send using an additional buffer
  - the buffer is allocated by the program via `MPI_BUFFER_ATTACH`
  - done once as part of the program initialisation
  - `MPI_BSEND` completes as soon as message is copied into buffer
- Typically quick to implement
  - add the `MPI_BUFFER_ATTACH` call
    - how big to make the buffer?
  - change `MPI_SEND` to `MPI_BSEND` everywhere
- But introduces additional memory copy
  - extra overhead
  - not recommended for production codes
  - One day your buffer won’t be big enough!

MPI_IRECV / MPI_ISEND

- **Uses Non Blocking Communications**
- **“I” stands for immediate**
    - **the call returns immediately**
- **Routines return without completing the operation**
    - **the operations run asynchronously (in the background)**
    - **Must NOT reuse the buffer (send/receive array) until safe to do so**
- **Later test that the operation completed**
    - **via an integer identification handle passed to `MPI_WAIT`**

```
call MPI_IRECV(rbuff,n,MPI_REAL8,other,1,MPI_COMM_WORLD,request,ierror)
call MPI_SEND (sbuff,n,MPI_REAL8,other,1,MPI_COMM_WORLD,ierror)
call MPI_WAIT(request,stat,ierr)
```

- **Alternatively could have used `MPI_ISEND` and `MPI_RECV`**

Non Blocking Communications

- **Routines include**
  - `MPI_ISEND`
  - `MPI_IRECV`
  - `MPI_WAIT`
  - `MPI_WAITALL`
    - **Waits for a number of outstanding communications to complete**

Final Practical

- **exercise2**
- **A simple model**
- **See the README for details**
- **See copies of MPI standard for details of arguments required for various MPI routines you might want to use.**
- **Ask if you need help or don’t understand anything!**