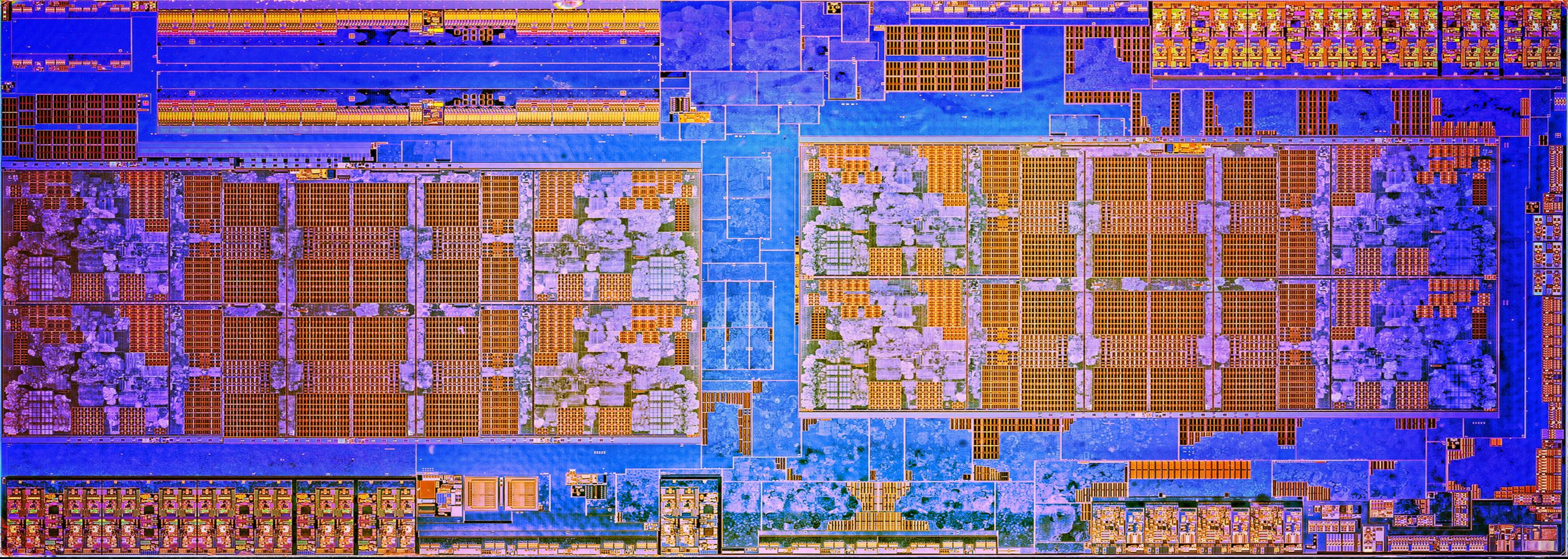


# CFD Lab – Lecture 3

Message Passing Interface - MPI

# Introduction

Message Passing Interface (MPI) is a library that allows communication between different processes

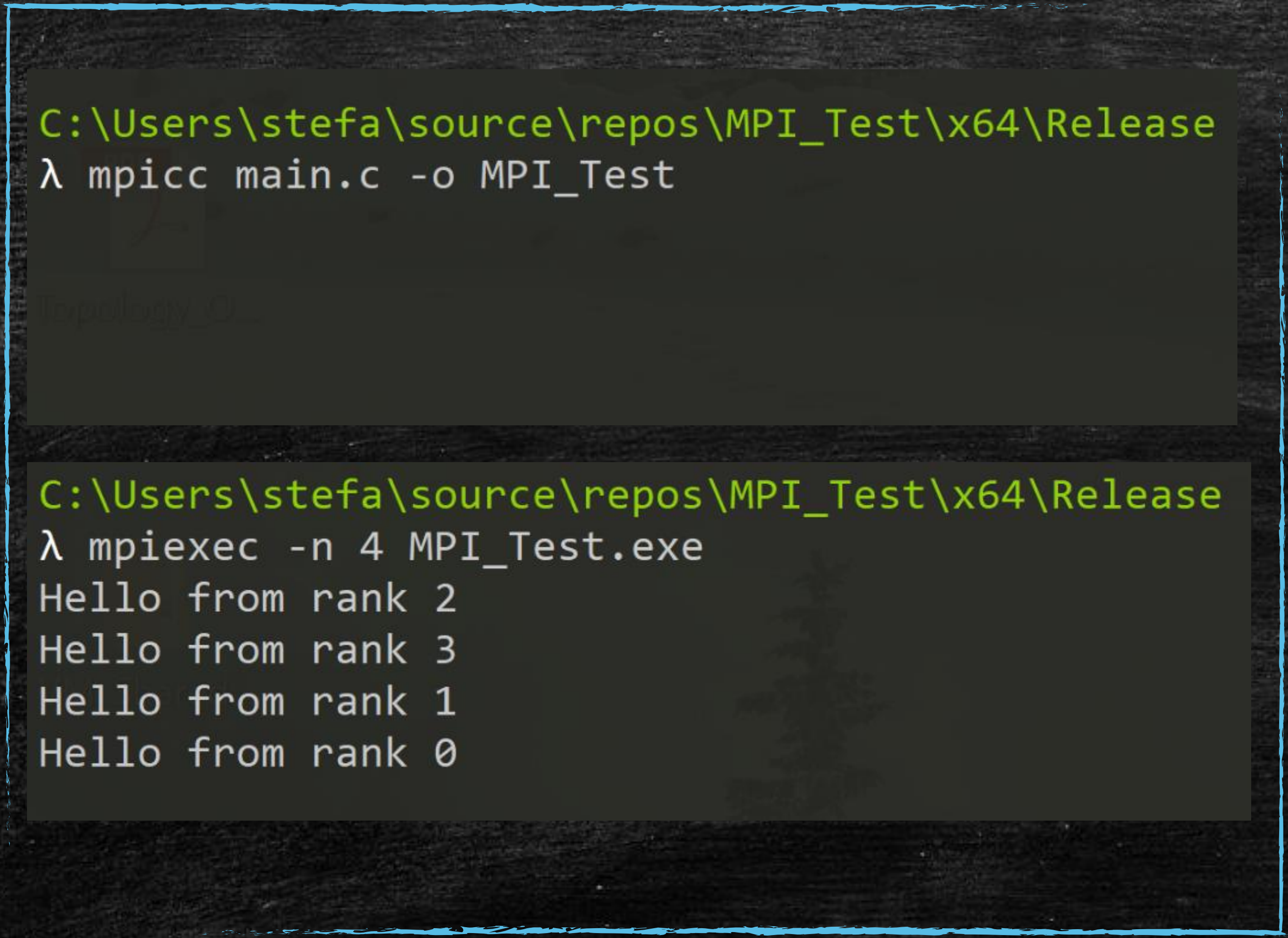
- Enables explicit passing of the data between processes.
- From several cores on a single CPU till thousands of cores on SuperMUC

# Hallo World in MPI

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_id);

printf("Hello from rank %d.\n", my_id);
MPI_Finalize();
}
```

1. Include <mpi.h> header file
2. Initialize MPI
3. Assign IDs to each process using MPI_Comm_rank
4. Each process prints its ID
5. Finalize MPI

# Output of the program



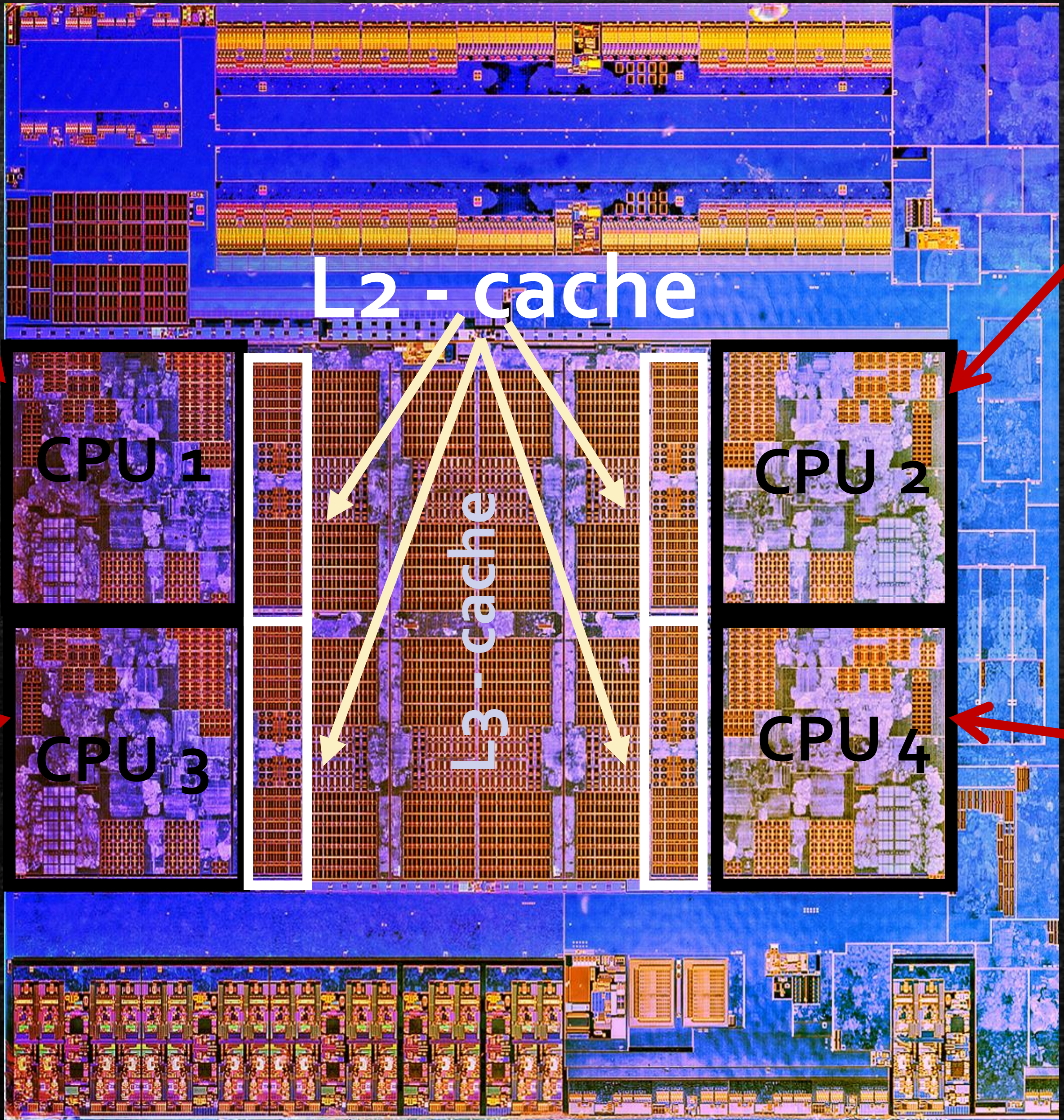
```
C:\Users\stefa\source\repos\MPI_Test\x64\Release
λ mpicc main.c -o MPI_Test
```

```
C:\Users\stefa\source\repos\MPI_Test\x64\Release
λ mpiexec -n 4 MPI_Test.exe
Hello from rank 2
Hello from rank 3
Hello from rank 1
Hello from rank 0
```

- Code is compiled using **mpicc** flag
- In order to run our program we need to use **mpiexec** command
- Order of execution is not ensured
- As soon as process get assigned hardware resources it will perform its task

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
&my_id);

printf("Hello from rank %d.\n", my_id);
MPI_Finalize();
}
```

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
&my_id);

printf("Hello from rank %d.\n", my_id);
MPI_Finalize();
}
```



```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
&my_id);

printf("Hello from rank %d.\n", my_id);
MPI_Finalize();
}
```

```
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,
&my_id);

printf("Hello from rank %d.\n", my_id);
MPI_Finalize();
}
```

# What to do when code should differ in each process?

- Each process has its own unique ID which help to differentiate between them
- For process dependent tasks we can use branching

```
if ( my_id = 0 )
        // Routine_Calculate A
else if ( my_id = 1 )
        // Routine_Calculate B
else if ( my_id = 2 )
        // Routine_Calculate C
```

# Sending and Receiving Messages

- Hello World example is easy, but there is no message exchange yet.
- Simplest Message Passing program can be implemented for only two processes ( P0 and P1)
- Goal is to send number 23 from P1 to P0

# MPI_Send(…)

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype,
             int dest, int tag,  MPI_Comm comm)
```

## Input Parameters

**buf** - initial address of send buffer (choice)

**count** - number of elements in send buffer (nonnegative integer)

**datatype** - datatype of each send buffer element (handle)

**dest** - id of destination process (integer)

**tag** - message tag (integer)

**comm** - communicator (handle)

# MPI_Recv(…)

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype,
                int source, int tag, MPI_Comm comm, MPI_Status *status)
```

## Input Parameters

**count** - maximum number of elements in receive buffer (integer)

**datatype** - datatype of each receive buffer element (handle)
**source** - rank of source (integer)
**tag** - message tag (integer)
**comm** - communicator (handle)

## Output Parameters

**buf** - initial address of receive buffer (choice)

**status** - status object (Status)

# Send-Receive Example Code

```
#include <stdio.h>
#include <mpi.h>
void main(int argc, char* argv[])
{
int my_id, number_to_receive, number_to_send = 23;
MPI_Status status;
MPI_Init (&argc, &argv);
MPI_Comm_rank( MPI_COMM_WORLD, &my_id );

if ( my_id == 0 )
{
    MPI_Recv( &number_to_receive, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
    printf( "Number received is: %d\n",  number_to_receive );
}
else
MPI_Send( &number_to_send, 1, MPI_INT, 0, 10, MPI_COMM_WORLD );

MPI_Finalize();
}
```

# Pitfalls

- Previously used Receive function call is so called blocking call
  - Receive will wait until message matching its requirements has been detected
- Send functions will try not to block, but this behavior is not guaranteed
  - With  the growing size of data that is sent, this blocking behavior emerges
- For every MPI_Send() there must be pairing MPI_Recv()
- Otherwise Deadlock can occur

# Deadlock Example

- Process with id = 0 will wait to match receive from process id =1
- Process with id = 1 will wait to match receive from process id =0
- Both processes blocked
- By switching statements for one of the processes problem is resolved

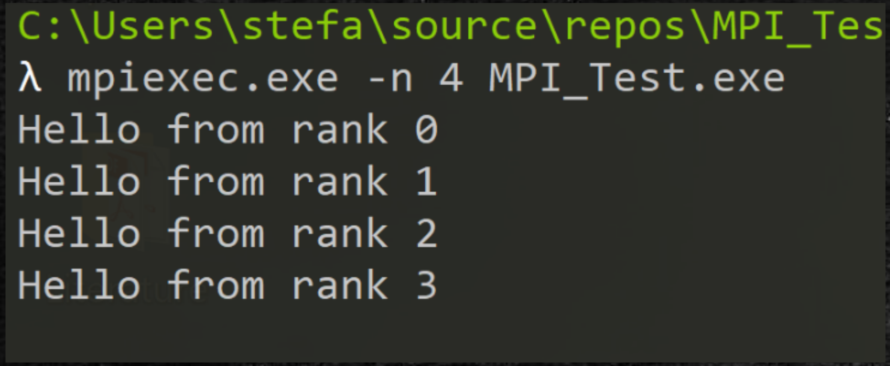
```
MPI_Comm_rank (comm, &my_id);

if (my_id == 0) {
  MPI_Send (sendbuf, count, MPI_INT, 1, tag, comm);
  MPI_Recv (recvbuf, count, MPI_INT, 1, tag, comm, &status);
} else if (my_id == 1) {
  MPI_Send (sendbuf, count, MPI_INT, 0, tag, comm);
  MPI_Recv (recvbuf, count, MPI_INT, 0, tag, comm, &status);

}
```

# Useful Functions

```
int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
          MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
               MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
        MPI_Comm comm )
```

# Synchronization helps to align all the processes

- In order to align all the processes at a certain point MPI_Barrier() can be used
- Hallo World with ordered output



```
C:\Users\stefa\source\repos\MPI_Tes
λ mpiexec.exe -n 4 MPI_Test.exe
Hello from rank 0
Hello from rank 1
Hello from rank 2
Hello from rank 3
```

```c
#include <stdio.h>
#include <mpi.h>
main(int argc, char* argv[])
{
int my_id;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_id);

for (int index = 0; index<4; index++)
{
    MPI_Barrier(MPI_COMM_WORLD);
    if (index == rank)
        printf("Hello from rank %d\n",my_id);
}
MPI_Finalize();
}
```