MPI in Small Bites

HPC.NRW Competence Network

MPI Concepts

HPC.NRW Competence Network

MPI in Small Bites

Library Initialization (classic MPI – no threads)

HPC.NRW

– Start-up, initialisation, finalization, and shutdown – C

1. Inclusion of the MPI header file
2. Pre-initialisation mode: uncoordinated
   - **No MPI function calls allowed with few exceptions**
   - **All program instances run exactly the same code**
3. Initialisation of the MPI environment with implicit synchronisation
4. Parallel MPI code
5. Finalisation of the MPI environment
6. Post-finalisation mode: uncoordinated
   - **No MPI function calls allowed with few exceptions**

C

```c
#include <mpi.h>

int main(int argc, char **argv)
{
  // … some code …
  MPI_Init(&argc, &argv);



  // … computation & communication …



  MPI_Finalize();
  // … wrap-up …
  return 0;
}
```

Library Initialization (classic MPI – no threads)

HPC.NRW

– Start-up, initialisation, finalisation, and shutdown – Fortran

1. Using the MPI module
2. Pre-initialisation mode: uncoordinated
   - **No MPI function calls allowed with few exceptions**
   - **All program instances run exactly the same code**
3. Initialisation of the MPI environment with implicit synchronisation
4. Parallel MPI code
5. Finalisation of the MPI environment
6. Post-finalisation mode: uncoordinated
   - **No MPI function calls allowed with few exceptions**

Fortran

```
PROGRAM example
  USE mpi_f08 ! USE mpi

!   … some code …
  INTEGER :: ierr
  CALL MPI_Init(ierr)

!  … computation  & communication …

  CALL MPI_Finalize(ierr)

!  … wrap-up …
END
```

Library Initialization (classic MPI – no threads)

- Initialization:

```
C:       ierr = MPI_Init(&argc, &argv);
Fortran: CALL MPI_Init(ierr)
```

  - Initializes the MPI library and makes the process member of `MPI_COMM_WORLD`
  - [C] Both arguments must be either NULL or they *must* point to the arguments of `main()`
  - **May not be called more than once for the duration of the program execution**
  - Error code as return value in [C] and additional parameter in [F]

- Finalization:

```
C:       ierr = MPI_Finalize();
Fortran: CALL MPI_Finalize(ierr)
```

  - Cleans up the MPI library and prepares the process for termination
  - **Must be called once before the process terminates**
  - Having other code after the finalisation call is not recommended

General Structure of an MPI Program

HPC.NRW

– How many processes are there in total?
– Who am I?

1 Obtains the number of processes (ranks) in the MPI program

Example: if the job was started with 4 processes, then **numberOfProcs** will be set to 4 by the call

2 Obtains the identity of the calling process within the MPI program
NB: MPI processes are numbered starting from 0

Example: if there are 4 processes in the job, then **rank** receives the value of 0 in the first process, 1 in the second process, etc.

C

```c
#include <mpi.h>

int main(int argc, char **argv)
{
  // … some code …
  int ierr = MPI_Init(&argc, &argv);
  int numberOfProcs, rank;
  // … more code …
  ierr = MPI_Comm_size(MPI_COMM_WORLD,          // 1
         &numberOfProcs);
  ierr = MPI_Comm_rank(MPI_COMM_WORLD,          // 2
         &rank);
  // … computation  & communication …
  ierr = MPI_Finalize();
  // … wrap-up …
  return 0;
}
```

General Structure of an MPI Program

HPC.NRW

- How many processes are there in total?
- Who am I?

1 Obtains the number of processes (ranks) in the MPI program

Example: if the job was started with 4 processes, then **numberOfProcs** will be set to 4 by the call

2 Obtains the identity of the calling process within the MPI program
NB: MPI processes are numbered starting from 0

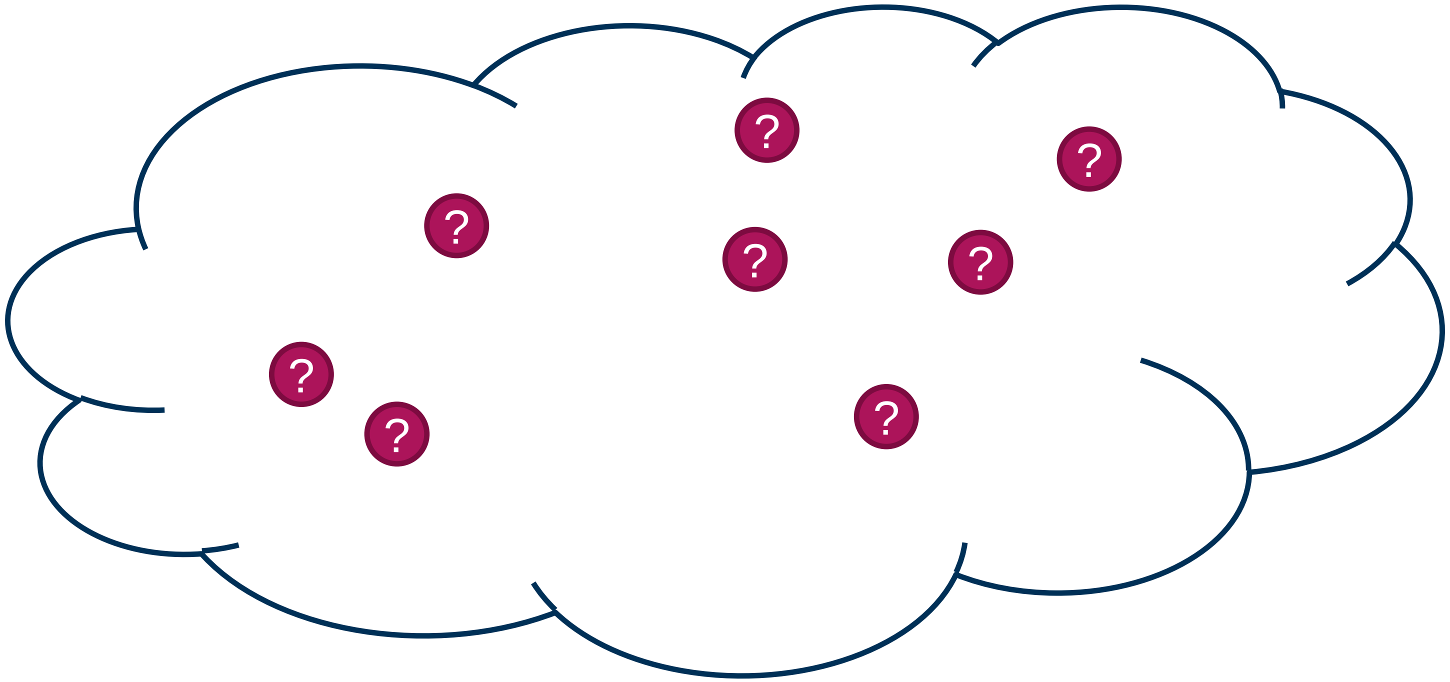
Example: if there are 4 processes in the job, then **rank** receives the value of 0 in the first process, 1 in the second process, etc.

Fortran

```
PROGRAM example
  USE mpi_f08 ! USE mpi
  INTEGER :: rank, numberOfProcs, ierr
!   … some code …
  CALL MPI_Init(ierr)
!   … other code …
  CALL MPI_Comm_size(MPI_COMM_WORLD,&
         numberOfProcs, ierr)
  CALL MPI_Comm_rank(MPI_COMM_WORLD,&
         rank, ierr)
!   … computation  & communication …
  CALL MPI_Finalize(ierr)
!   … wrap-up …
END PROGRAM example
```

Ranks

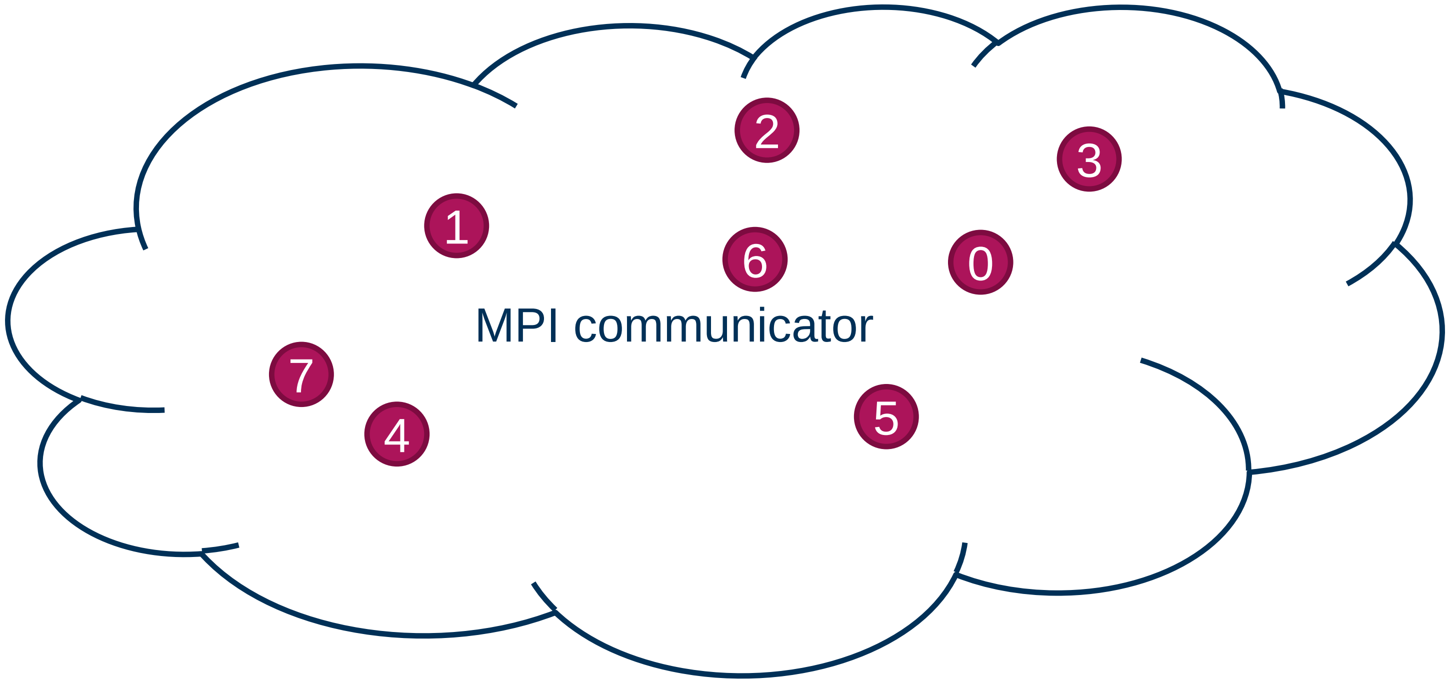
- The processes in any MPI program are initially indistinguishable
- MPI assigns each process a unique identity (rank) in a communication context (communicator)



Ranks

HPC.NRW

- The processes in any MPI program are initially indistinguishable
- MPI assigns each process a unique identity (rank) in a communication context (communicator)



Ranks

HPC.NRW

- The processes in any MPI program are initially indistinguishable
- MPI assigns each process a unique identity (rank) in a communication context (communicator)
- Ranks
  - Range from `0` to `n-1` (with n processes in the communicator)
  - An MPI process can have different ranks in different communicators
- Communicators
  - Logical contexts where communication takes place
    - Comprises a group of MPI processes with some additional information
  - `MPI_COMM_WORLD` is implicitly available
    - Comprises all processes initially started with the MPI program

MPI as an SPMD Environment

HPC.NRW

1. ✓ Provide dynamic identification of all peers
   - Who am I and who else is also working on this problem?
2. Provide robust mechanisms to exchange data
   - Whom to send data to / From whom to receive the data?
   - How much data?
   - What kind of data?
   - Has the data arrived?
3. Provide synchronisation mechanisms
   - Have all processes reached same point in the program execution flow?
4. Provide methods to launch and control a set of processes
   - How do we start multiple processes and get them to work together?
5. Portability

Compiling MPI Programs

HPC.NRW

– MPI is a typical library with C header files, Fortran modules, etc.
– Most MPI vendors provide convenience compiler wrappers (names are not standardized!)

```
cc   → mpicc
c++  → mpic++
f90  → mpif90
```

– On the RWTH Aachen Compute Cluster:

```
$CC  → $MPICC
$CXX → $MPICXX
$FC  → $MPIFC
```

Specific compilers called automatically change depending on the module loaded.

Example: MPI Compiler Wrappers

HPC.NRW

– RWTH Aachen Cluster defines additional environment variables to minimize confusion

```
cluster:~[1]$ $MPICC --show                              # instruct wrapper to show compile line
icc \
 -I/opt/MPI/openmpi-4.0.3/linux/intel_19.0.1.144/include \
 -pthread \
 -Wl,-rpath \
 -Wl,/opt/MPI/openmpi-4.0.3/linux/intel_19.0.1.144/lib \
 -Wl,--enable-new-dtags \
 -L/opt/MPI/openmpi-4.0.3/linux/intel_19.0.1.144/lib \
 -lmpi
cluster:~[1]$ echo $MPICC                                # check compiler wrapper name
mpicc
cluster:~[1]$ module switch openmpi intelmpi             # switch MPI implementation
cluster:~[1]$ echo $MPICC                                # check compiler wrapper name again
mpiicc
```

Executing MPI Programs

HPC.NRW

- Most MPI implementations provide a special launcher program:

```
mpiexec -n nprocs … program <arg1> <arg2> <arg3> …
```

  - Launches `nprocs` instances of **`program`** with command-line arguments **`arg1`**, **`arg2`**, **`...`**
- The standard specifies the **`mpiexec`** program, but does not require it:
  - IBM BG/Q: **`runjob --np 1024 …`**
  - SLURM resource manager: **`srun …`**

Executing MPI Programs

- The launcher often performs more than simply launching processes:
  - Helps MPI processes find each other and establish the world communicator
  - Redirects the standard output of all ranks to the terminal
  - Redirects the terminal input to the standard input of rank 0
  - Forwards received signals (Unix-specific)

MPI as an SPMD Environment

- ✓ Provide dynamic identification of all peers
  - Who am I and who else is also working on this problem?
2. Provide robust mechanisms to exchange data
   - Whom to send data to / From whom to receive the data?
   - How much data?
   - What kind of data?
   - Has the data arrived?
3. Provide synchronisation mechanisms
   - Have all processes reached same point in the program execution flow?
- ✓ Provide methods to launch and control a set of processes
  - How do we start multiple processes and get them to work together?
- ✓ Portability

Error handling

- Error codes indicate the success of the operation:
  - Failure is indicated by error codes other than **`MPI_SUCCESS`**

C

```c
if (MPI_SUCCESS != MPI_Init(&argc, &argv))
 …
```

Fortran

```fortran
CALL MPI_Init(ierr)
IF (ierr /= MPI_SUCCESS) …
```

- An MPI error handler is called first before the call returns
  - **The default error handler for non-I/O calls aborts the entire MPI program!**
  - Error checking in simple programs is redundant
- Actual MPI error code values are implementation specific
  - Use **`MPI_Error_string`** to derive human readable information

Handles to Opaque Objects

- MPI objects (e.g., communicators) are referenced via handles
  - Process-local values
    - Cannot be passed from one process to another
  - Objects referenced by handles are opaque
    - Structure is implementation dependent
    - Blackbox for the user

- C (mpi.h)
  - typedef'd handle types: `MPI_Comm`, `MPI_Datatype`, `MPI_File`, etc.

Handles to Opaque Objects II

- Fortran (USE mpi)
  - All handles are INTEGER values
  - Easy to pass the wrong handle type

- Fortran 2008 (USE mpi_f08)
  - Wrapped INTEGER values: TYPE(MPI_Comm), TYPE(MPI_File), etc.
  - The INTEGER handle is still available: comm%MPI_VAL

Datatype Handles

HPC.NRW

- MPI is a library
  - Cannot infer datatypes of supplied buffers at runtime
  - User needs to provide additional information on buffer type

- MPI datatype handles tell the MPI library how to:
  - read binary values from the send buffer
  - write binary values into the receive buffer
  - correctly apply value alignments
  - convert between machine representations in heterogeneous environments

Datatype Handles II

- MPI datatypes are **handles**
  - Cannot be used to declare variables of a specific language type
  - `sizeof(MPI_INT)` provides the size of a datatype handle **NOT** the size of an `int` in C

- Type Signatures
  - Sequence of basic datatypes in a buffer
  - Basic datatypes correspond to native language datatypes

- Type Maps
  - Sequence of basic datatypes **AND** their location in a buffer

- **Type signatures of associated operations have to match; Type map may differ!**

Basic MPI Datatypes

HPC.NRW

– MPI provides predefined datatypes for each language binding:

MPI data type	C data type
MPI_CHAR	char
MPI_SHORT	short
MPI_INT	int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_UNSIGNED_INT	unsigned int
…	…
MPI_BYTE	-

MPI data type	Fortran data type
MPI_INTEGER	INTEGER
MPI_REAL MPI_REAL8	REAL REAL(KIND=8)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
…	…
MPI_BYTE	-

8 binary digits
no conversion
used for untyped data

Blocking vs. Non-blocking vs. Asynchronous

HPC.NRW

- Blocking procedures return when the associated operation is complete locally
  - Any input argument can be safely reused or deallocated
  - Operation may not be completed remotely
- Non-blocking procedures return before associated operation is complete locally
  - One or more additional calls are needed to complete operation
  - Input arguments may not be written or deallocated until operation is complete
- Synchronous operations complete locally only with specific remote intervention
  - Asynchronous operations may complete locally without remote intervention

MPI Communication Paradigms

Point-to-Point
Communication

Collective
Communication

One-sided
Communication