

# Message Passing Programming

---

Introduction to MPI

**EPSRC**

**NERC** SCIENCE OF THE ENVIRONMENT

 **archer**

**CRAY**  
THE SUPERCOMPUTER COMPANY

**epcc**



# What is MPI?



# MPI Forum

- First message-passing interface standard.
- Sixty people from forty different organisations.
- Users and vendors represented, from the US and Europe.
- Two-year process of proposals, meetings and review.
- *Message Passing Interface* document produced in 1993



# Implementation

- MPI is a *library* of function/subroutine calls
- MPI is *not a language*
- There is *no such thing* as an MPI compiler
- The C or Fortran compiler you invoke knows nothing about what MPI actually does
  - only knows prototype/interface of the function/subroutine calls



# Goals and Scope of MPI

- MPI's prime goals are:
  - To provide source-code portability.
  - To allow efficient implementation.
- It also offers:
  - A great deal of functionality.
  - Support for heterogeneous parallel architectures.



# Header files

- C:

```
#include <mpi.h>
```

- Fortran 77:

```
include 'mpif.h'
```

- Fortran 90:

```
use mpi
```



# MPI Function Format

- C:

```
error = MPI_Xxxxx(parameter, ...);
```

```
MPI_Xxxxx(parameter, ...);
```

- Fortran:

```
CALL MPI_XXXXX(parameter, ..., IERROR)
```



# Handles

- MPI controls its own internal data structures.
- MPI releases `handles' to allow programmers to refer to these.
- C handles are of defined `typedefs`.
- Fortran handles are `INTEGERS`.





# Initialising MPI

- C:

```
int MPI_Init(int *argc, char ***argv)
```

- Fortran:

```
MPI_INIT(IERROR)  
INTEGER IERROR
```

- Must be the first MPI procedure called.
  - but multiple processes are already running before MPI\_Init



# MPI\_Init

```
int main(int argc, char *argv[])  
{  
    ...  
    MPI_Init(&argc, &argv);  
    ...  
}
```

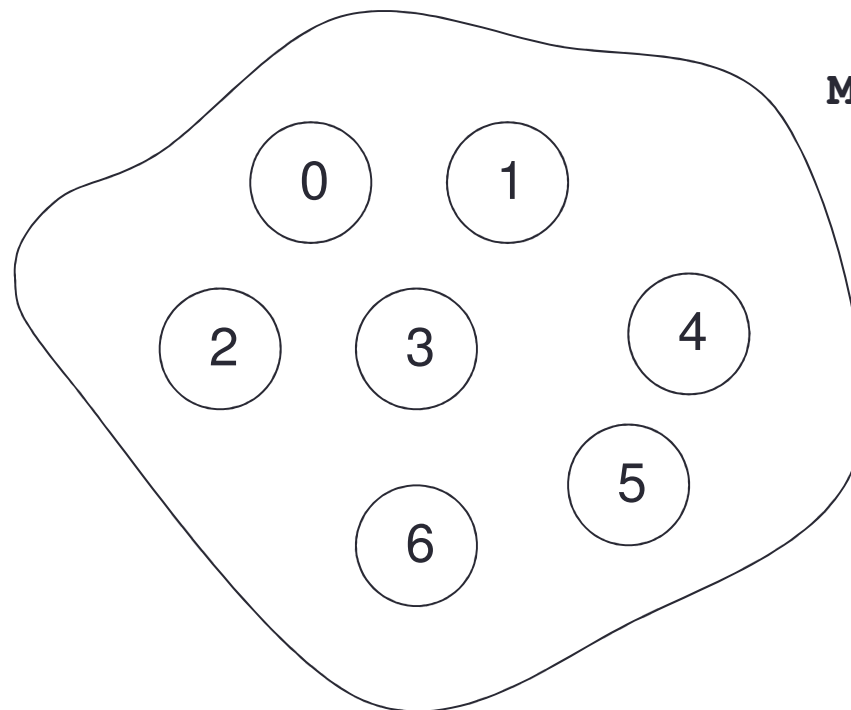
```
int main()  
{  
    ...  
    MPI_Init(NULL, NULL);  
    ...  
}
```

```
program my_mpi_program  
integer :: ierror  
...  
CALL MPI_INIT(IERROR)
```



# MPI\_COMM\_WORLD

Communicators



MPI\_COMM\_WORLD



# Rank

- How do you identify different processes in a communicator?

```
MPI_Comm_rank (MPI_Comm comm, int *rank)
```

```
MPI_COMM_RANK (COMM, RANK, IERROR)  
INTEGER COMM, RANK, IERROR
```

- The rank is not the physical processor number.
  - numbering is always 0, 1, 2, ....., N-1



# MPI\_Comm\_rank

```
int rank;  
...  
MPI_Comm_rank(MPI_COMM_WORLD, &rank);  
printf("Hello from rank %d\n", rank);  
...
```

```
integer :: ierror  
integer :: rank  
...  
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)  
write(*,*) 'Hello from rank ', rank
```



# Size

- How many processes are contained within a communicator?

```
MPI_Comm_size(MPI_Comm comm, int *size)
```

```
MPI_COMM_SIZE(COMM, SIZE, IERROR)  
INTEGER COMM, SIZE, IERROR
```



# Exiting MPI

- C:

```
int MPI_Finalize()
```

- Fortran:

```
MPI_FINALIZE(IERROR)  
INTEGER IERROR
```

- Must be the last MPI procedure called.



# Aborting MPI

- Aborting the execution from any processor (e.g. error condition)

- C:

```
int MPI_Abort(MPI_Comm comm, int errorcode)
```

- Fortran:

```
MPI_ABORT(COMM, ERRORCODE, IERROR)  
INTEGER COMM, ERRORCODE, IERROR
```

- Behaviour

- will abort all processes even if only called by one process
- this is the ONLY MPI routine that can have this effect
- only use as a last-resort “nuclear” option!





# What machine am I on?

- Can be useful on a cluster
  - e.g. to confirm mapping of processes to nodes/processors/cores

```
integer namelen
character*(MPI_MAX_PROCESSOR_NAME) :: procname
...
call MPI_GET_PROCESSOR_NAME(procname, namelen, ierror)
write(*,*) 'rank ', rank, ' is on machine ', procname(1:namelen)
```

```
int namelen;
char procname[MPI_MAX_PROCESSOR_NAME];
...
MPI_Get_processor_name(procname, &namelen);
printf("rank %d is on machine %s\n", rank, procname);
```



# Summary

- Have some covered basic calls
  - but no explicit message-passing yet
- Can still write useful programs
  - eg a task farm of independent jobs
- Need to compile and launch parallel jobs
  - procedure is not specified by MPI
  - next lecture gives machine-specific details

