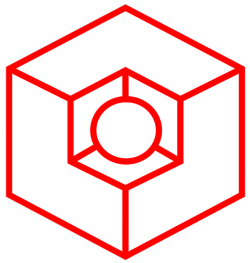

BremHLR

Kompetenzzentrum für Höchstleistungsrechnen Bremen

Writing Message-Passing Parallel Programs with MPI

Lars Nerger (BremHLR & Alfred Wegener Institute)

AWI

Universität Bremen

CONSTRUCTOR UNIVERSITY

HSB

Hochschule Bremerhaven

NHR NATIONALES HOCHLEISTUNGS RECHNEN

Computer setup

for compiling and running parallel programs

http://bremhlr.uni-bremen.de/mpi-openmp-course

Computer Setup

- We will do several hands-on exercises
  - Practice implementation of different MPI functions

- Required
  - C or Fortran compiler (typically GCC – GNU compiler collection)
  - MPI library and runtime (typically OpenMPI)
  - Plot tool (typically gnuplot)
  - Software availability
    - Linux: available as packages of Linux distribution
    - Windows: available as packages in Cygwin
    - MacOS: available with Homebrew, Fink, orMacports

MPI Programs

http://bremhlr.uni-bremen.de/mpi-openmp-course

MPI Preliminaries

- MPI comprises a library.
  - A precompiled archive providing routines with specified interface
  - Library needs to be linked when compiling a program
- MPI process is a program (C / C++ / Fortran) that communicates with other MPI programs by calling MPI routines.
  - Still the MPI-parallelized program is a single executable program that is started using a single command
- The is a special starter command for MPI-parallel programs (usually `mpirun`)

The minimal MPI program: "Hello world" (in Fortran)

```
PROGRAM hello

  USE mpi

  IMPLICIT NONE

  INTEGER :: ierror

  CALL MPI_INIT(ierror)

  WRITE(*,*) 'Hello world!'

  CALL MPI_FINALIZE(ierror)

END
```

The minimal MPI program: "Hello world" (in C)

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

int main(int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  printf("Hello world!\n");
  MPI_Finalize();
}
```

Compiling and Linking MPI programs

Using OpenMPI on course computers

- C

```
mpicc -o simple simple.c
```

- Fortran

```
mpif90 -o simple simple.f90
```

mpicc/mpif90 are wrappers;
without them one needs to explicitly link the MPI library:

- C

```
gcc -o simple simple.c -lmpi
```

- Fortran

```
gfortran -o simple simple.f90 -lmpi
```

Running MPI programs

Running MPI programs on the course computers

- In the shell:

```
mpirun -np TASKS EXE
```

  - `TASKS` a number specifying the number of processes
  - `EXE` name of the executable (program)

Size

- How many processes are running in my program?
  - C:

```
int MPI_Comm_size(MPI_COMM_WORLD, int *size)
```

  - Fortran:

```
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, SIZE, IERROR)
        INTEGER :: SIZE, IERROR
```

Rank

- How do you identify different processes?
  - Process number within the group
  - `rank = 0, 1, ... size-1`
  - C:

```
int MPI_Comm_rank(MPI_COMM_WORLD, int *rank)
```

  - Fortran:

```
CALL MPI_COMM_RANK(MPI_COMM_WORLD, RANK, IERROR)
        INTEGER :: RANK, IERROR
```

Exercise 1

- Type (do not use cut and paste!) the Hello-world program using an editor on the course computers (use either C or Fortran)
- Compile the program
- Run the program
- Modify your program, so that only the process with rank 0 prints out
- Extend the Hello-world program, so that it also writes the total number of processes and the rank of the process in each output line
- Compile and run the program with different numbers of processes

The minimal MPI program: "Hello world"

```
PROGRAM hello
  USE mpi
  IMPLICIT NONE
  INTEGER :: ierror

  CALL MPI_INIT(ierror)

  WRITE(*,*) 'Hello world!'

  CALL MPI_FINALIZE(ierror)

END
```

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

int main(int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  printf("Hello world!\n");
  MPI_Finalize();
}
```

Time for programming…

Hello-World Program in Python

```
from mpi4py import MPI

if not MPI.Is_initialized():
   MPI.Init()

my_rank = MPI.COMM_WORLD.Get_rank()
npes = MPI.COMM_WORLD.Get_size()

print("Hello world rank ", my_rank, ' npes ', npes)
```

Run with `mpiexec -n 4 python -u hello.py`

Programming exercises of the course should also be possible in Python

- Syntax is different from that for C and Fortran (class-based)
- Some functions (e.g. MPI_Gather work differently)
- Not sure how complete the mpi4py library is

The more complete Hello-World Program in C

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

int main(int argc, char *argv[])  /* hello world */
{
  int size;
  int rank;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("MPI: size = %3d rank = %3d", size, rank);

  MPI_Finalize();
}
```

The more complete Hello-World Program in Fortran

```
PROGRAM main  ! hello world

  IMPLICIT NONE
  INCLUDE 'mpif.h'
  INTEGER :: size, rank, ierr

  CALL MPI_Init(ierr)
  CALL MPI_Comm_size(MPI_COMM_WORLD, size, ierr)
  CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)

  WRITE(*, *) " MPI: size = ", size, rank = ", rank

  CALL MPI_Finalize(ierr)

END
```

The more complete Hello-World Program in C

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

int main(int argc, char *argv[])  /* hello world */
{
  int size;
  int rank;
  int len_name;
  char name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(name, &len_name);

  printf("MPI: size = %3d rank = %3d name = %s\n", size, rank, name);

  MPI_Finalize();
}
```