Prof. Dr. O. Rheinbach
Dipl.-Math. F. Röver
Institut für Numerische Mathematik und Optimierung
TU Bergakademie Freiberg

4th Exercise in HPC

Preliminary: Connecting to the cluster

When using the cluster, instead of opening your local terminal or ssh-ing to the usual machine, log on to the *login node*

```
login01.hrz.tu-freiberg.de
```

MPI can be used on the login node but this is not recommended. Use the compute nodes instead. You can start an interactive session on a compute node by

```
[login01 ~]$ qsub -I -l select=1:ncpus=12:mpiprocs=12 -q teaching
```

Here, `-I` is a upper case "i" and `-l` is a lower case "L". This reserves a single "chunk" of 12 processors cores to run up to 12 MPI processes. You can obtain a larger number of processor cores by increasing the `select`-value.

You can view the allocated resources by

```
[node139 ~]$ cat $PBS_NODEFILE
```

Before compiling, you have to load the OpenMPI module

```
[node139 ~]$ module add openmpi/gcc/7.1.0/2.1.2
```

The URZ has an extensive guide how to use the module system and the queue system with your software at `http://tu-freiberg.de/en/urz/dienste/hpc`

Instructions: Using MPI (on your machines and on the cluster)

Compile your program using

```
$ mpicc -O mpi_bcast.c -o mpi_bcast
```

Then start the program by

```
$ mpirun -np 2 ./mpi_bcast
```

Here `-np` is the number of MPI processes.

If you are on the cluster and get the error message

```
-bash: mpirun: command not found
```

you have not loaded the MPI module (see above).

As usual you can run and time the command

```
$ time mpirun -np 1 ./mpi_bcast
$ time mpirun -np 4 ./mpi_bcast
```

or you can use the timing functions in the MPI library.

Exercise 1
Use the following program `mpi_first.c` to test if everything works.

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

int main(int argc, char **argv)
{
  int rank, size, err, n;
  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  printf("rank=%d size=%d\n",rank,size);

  MPI_Finalize();
  return 0;
}
```

Exercise 2
Use the following program to test and understand the behavior of MPI Broadcast.

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

int main(int argc, char **argv)
{
  int rank, size, err, n;
  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  printf("rank=%d size=%d\n",rank,size);

  err=MPI_Barrier(MPI_COMM_WORLD);

  n=(rank+1)*4711;
  err=MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
  printf("   Received =%d\n",n);
  MPI_Finalize();
  return 0;
}
```

Exercise 3
The command

```
cat /proc/cpuinfo | less
```

gives you information on the CPU. Find out what CPU type is used on the compute nodes. Find information on the internet how to interpret the `physical id`, `core id`, and `siblings` field in the `cpuinfo`. Remark: Note that the Core ID is not consecutive.

Exercise 4
Use the following program to study the use of MPI reductions

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

int main(int argc, char **argv)
{
  int rank, size, err, n, sum;
  MPI_Init(&argc, &argv);

  MPI_Comm_size(MPI_COMM_WORLD,&size);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);

  printf("rank=%d size=%d\n",rank,size);

  n=rank+1;
  sum=4710;

  MPI_Barrier(MPI_COMM_WORLD);

  MPI_Reduce(&n,&sum,1,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
  printf("   Received in reduction=%d\n",sum);

  MPI_Barrier(MPI_COMM_WORLD);

  sum=3710;
  MPI_Allreduce(&n,&sum,1,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
  printf("   Received in allreduction =%d\n",sum);

  MPI_Finalize();
  return 0;
}
```

Exercise 5
Recall the computation of an approximation of π by numerical integration using trapezoid rule for

$$\int_0^1 \frac{1}{1+x^2} = \arctan(1) - \arctan(0) = \pi/4. \tag{1}$$

Write an MPI program using this approach. How much (real) time do you need to compute 7 digits of π? What speedup do you achieve if you use $1, 2, 4, 8, 12$ MPI processes?

Hints: You may want to use a MPI reduction, i.e.,

```
MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

Remember that the number of MPI processes should be smaller or equal to the number of cores.