Prof. Dr. O. Rheinbach
Dr. Matthias Brändel
Dr. Friederike Röver
Institut für Numerische Mathematik und Optimierung
TU Bergakademie Freiberg

Parallel Simulation with MPI, Trilinos and FROSch - Hands on MPI

Note: Code in this document `<text>` need user-specific input.
Note: For external particpants the user name is `u<PCNumber>`. For the next days please use the same workstation.
Note: On the cluster the *tab*-key autocompletes commands, paths, ..., reducing typing.

Access to the compute cluster
Cluster refers to an accumulation of computing nodes. The computing nodes are not accessible externally.
Login on the login node of the compute cluster

```
ssh <username>@mlogin01.hrz.tu-freiberg.de
```

The first login will trigger a warning for connecting to a new device, e. g.,

```
The authenticity of host 'mlogin01.hrz.tu-freiberg.de (139.20.68.154)'
can't be established.
...
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Confirm with `yes`.
You can close the connection with `exit` (but maybe you should not to so right now).

Copying course material
The hands on course material is available at `/projects/mpi-trilinos-frosch/`. You can inspect it using

```
cd  /projects/mpi-trilinos-frosch/
ls
```

In this directory you have no editing rights. Please copy the files to your home directory

```
cp -r /projects/mpi-trilinos-frosch/ ~/
cd ~/mpi-trilinos-frosch/HandsOn-MPI/
```

Code compilation
Always compile on the login node. Never compile on the compute nodes.
To compile code the paths to the installed compilers must be known.
You can inspect available software on the compute cluster using

```
module available
```

Load the modules for `MPI` and `python`

```
module load openmpi/gcc/11.4.0/4.1.6
module load python/gcc/11.4.0/3.11.7
```

(The loading commands can also be stored in the `.bashrc`.)
`module list` shows loaded modules.
Compile `C` code using

```
cd ~/mpi-trilinos-frosch/HandsOn-MPI/01-first-example/
mpicc first-example.c
```

Compile `C++` code using `mpiCC`, `mpicxx`, or `mpic++`.
`python` code is interpreted.

Program execution, i. e., job submission
Only ever run code on the compute nodes. (Except for small tests.)
The compute nodes for your program execution are assigned by the Portable Batch System (PBS) management system. You submit a job to PBS using a job script.
Look at the job script for the `C` program

```
cd ~/mpi-trilinos-frosch/HandsOn-MPI/01-first-example/
cat submit-c.script
```

The contents are listed here

```
#!/bin/bash
#PBS -N first-example
#PBS -q teachingq
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -l walltime=00:05:00
#PBS -o log.out
#PBS -e log.err

PBS_O_WORKDIR=$HOME/mpi-trilinos-frosch/HandsOn-MPI/01-first-example/
cd $PBS_O_WORKDIR

mpiexec a.out
```

It it a bash script that will be interpreted by PBS. Lines starting with `#PBS` are PBS options. Access to compute nodes is managed via queues. In this course only use the `teachingq`. Always edit the `PBS_O_WORKDIR` to match the path to your executable. More Information here (https://tu-freiberg.de/en/urz/service-portfolio/high-performance-computing-hpc/user-documentation/job-submission).
Submit the job using

```
qsub submit-c.script
```

Your job will be assigned an id. `qstat` will show running jobs. `qdel <job-id>` will delete a specific job. Look at the job script for the `python` program

```
cat submit-py.script
```

The contents are listed here

```
#!/bin/bash
#PBS -N first-example
#PBS -q teachingq
#PBS -l select=1:ncpus=4:mpiprocs=4
#PBS -l walltime=00:05:00
#PBS -o log.out
#PBS -e log.err

module load openmpi/gcc/11.4.0/4.1.6
module load python/gcc/11.4.0/3.11.7

PBS_O_WORKDIR=$HOME/mpi-trilinos-frosch/HandsOn-MPI/01-first-example/
cd $PBS_O_WORKDIR

mpiexec python3 first-example.py
```

Note that for `python` the modules are loaded on the compute nodes as well. Also note that the number of `MPI` ranks is not specified by the `mpiexec`. The `MPI` environment is defined by PBS. Compare with the output (`log.out`) you would expect.

Exercises

Exercise names are consistent with directory names.
Skeleton files are provided for each exercise. Fill in the gaps.
Solutions are given in `solutions`-subdirectories.
Write your own job submission scripts.
Work through the exercises at your own speed.

01-first-example
In this first example a simple `MPI` program is given.

(a) Compile and execute it on the cluster.

(b) Understand the `MPI` commands and modify the number of `MPI` ranks and n and understand the changes in the output.

02-hello-world
Write your first hello world program using `MPI`.

(a) Start and end `MPI` in the code. Every rank should output `Hello World!`.

(b) Assign `rank` and `size` and print this information to the command line.

(c) Only `rank` 0 should say `Hello World!` and every other process behaves as described in (b).

03-bcast-reduce
Implement `MPI` barriers, broadcast and reduce variables.

(a) In `bcast.*` insert a barrier and broadcast n. Try different ranks as the root for the broadcast.

(b) In `reduce.*` insert a barrier and use a reduction as well as an allreduction of n. Also comprehend the difference in the reduction of a single variable and an array.

04-send-recv
In this example you will use the send and receive functions of `MPI`.

(a) In `ping.*` perform a single sending operation from `rank` 0 that is received by `rank` 1.

(b) In `pingpong.*` additionally the operation is reversed.

(c) In `pingpong-bench.*` repeat the `pingpong` in a loop of 50 repetitions. Measure the wall time. `Rank` 0 then puts out the transfer time of one message in microseconds, i. e., $\delta t/(2 \cdot 50) \cdot 1000000$.

05-deadlock-and-non-blocking
Here halo communication is modeled. In a ring an integer is sent and modified. Depending on which send routine gets chosen by the `MPI` library, the implementation may result in a deadlock.

(a) Execute `ring-WRONG2.*` for various number of ranks. Does it result in a deadlock? Why/why not?

(b) In `ring-WRONG1.*` replace the send function with the synchronous send function. What do you observe?

(c) In `ring.*` replace either the send or the receive function with a non-blocking variant.