Running MPI programs

Lars Pastewka

Table of contents

Installation

To install the MPI libraries, run

```
sudo apt install openmpi-bin libopenmpi-dev
```

on latest Ubuntu. This also works if you are running Ubuntu through the Windows Subsystem for Linux (WSL).

Compiling

`CMake` will take care of all your compilation needs when you use `find_package(MPI REQUIRED)`. However, should you need to compile an MPI parallel code manually, you need to use the MPI compiler wrappers. Those are `mpicc` for standard C code and `mpic++` for C++ code.

Try compiling the following program

```
#include <iostream>

#include <mpi.h>

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

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

```
    std::cout << "rank = " << rank << "/" << size << std::endl;

    MPI_Finalize();
}
```

with

```
mpic++ hello_world.cpp -o hello_world
```

Running

You need to run your program through the `mpirun` command. It allows you to specify on how many processes (typically the number of cores) you want to run. For example,

```
mpirun -n 2 ./hello_world
```

runs our `hello_world` program on 2 processes.

Note that for testing purposes, you may want to run on more processes than you have cores in your system. Some MPI libraries prevent this by default. You can override this behavior by adding the `--oversubscribe` option, i.e. by running

```
mpirun -n 16 --oversubscribe ./hello_world
```

The output you see should look as follows:

```
rank = 14/16
rank = 15/16
rank = 1/16
rank = 2/16
rank = 3/16
rank = 6/16
rank = 9/16
rank = 10/16
rank = 0/16
rank = 11/16
rank = 12/16
rank = 7/16
rank = 5/16
rank = 4/16
rank = 13/16
```

The order is random as the output command will be issued at slightly different (and random) times.

Running on the cluster

On HPC systems such as bwUniCluster, you do not launch MPI programs directly with `mpirun` on the login node. Instead, jobs run under the SLURM batch system: you submit a job script with `sbatch` (or launch the program with `srun`), and the scheduler allocates the requested cores and passes this information to the MPI runtime. See bwUniCluster for details and an example job script.