HPC4M - ASSIGNMENT 1

KAROLINA BENKOVA

Exercise 2

The task in this exercise is to use MPI functions and collective communication to generate two $N \times N$ (integer) matrices and calculate their product $A \times B$, both using parallel computing (in C++).

1. Set-up for running the job on Cirrus

We are going to use N processes so that the matrices can be split into rows nicely - this is implemented in the `slurm` file `run_job.slurm` : `tasks-per-node=N` with N as a number of our choice (up to 36 as we have up to 36 cores per node and we are using 1 node). The output file is stored in the file `out`. Both the number of processes and the dimension variable is by default set to $N = 3$.

2. Parallel code

In the script `main.cpp`, we start with creating matrix B using the root process only (in our case we have chosen this to be the process number 0). Next, we broadcast B to all the other processes involved in the parallel computation using the `MPI_Bcast` function. This is necessary for the next step - in parallel, all the processes create their corresponding row of matrix A, i.e. process 0 creates the 1st row, process 1 creates the 2nd row etc., all allocating the rows of matrix A into a vector a. Each process then uses its calculated vector a (a row of matrix A) and the whole matrix B in order to calculate $a \times B$. The result is saved into a vector c. Finally, the root process gathers these vectors c from all N processes using the `MPI_Gather` function, and puts them into a matrix D in the order of the processes' ranks, i.e. vector c created by process 0 is allocated into the first N elements of D (1st row), c from process 1 is put into the next N elements of D (2nd row) etc. The output of the script, the matrix D, is the result of multiplying matrix A by B from the right.

The script is compiled on a login node of Cirrus using the script `compile_now` using the Intel compiler.

3. Implementation of the MPI functions

First, we used the function `MPI_Bcast` for one-to-all communication in order to broadcast the matrix B to all processes. This was done using

```
MPI_Bcast(B, N*N, MPI_INT, 0, comm),
```

where B is $N \times N$-dimensional data copied from the memory of the root process to the memory of other processes in the communicator. The datatype of all matrices and vectors in the script is integer, so we used `MPI_INT`.

The vectors c were gathered using the `MPI_Gather` function for all-to-one communication

```
MPI_Gather(c, N, MPI_INT, D, N, MPI_INT, 0, comm),
```

where the N-dimensional vector c is collected from all the processes and reassembled in the root process into a row of matrix D (which is also N-dimensional).

4. Possible changes

If the user is interested in printing the matrices A and B as well, the function `MPI_Gather` can be used to assemble vectors a from all the processes in the same fashion as vectors c.

5. Conclusions

In this exercise we managed to create a script with parallel code using two of the MPI functions and collective communication. By compiling the node on the login node on Cirrus, we were able to submit a job to calculate the matrix product of 2 square matrices with dimension up to 36.