Università della Svizzera italiana

Institute of Computational Science ICS

High Performance Computing **2019**

Instructor: Prof. Olaf Schenk TA: Juraj Kardos, Radim Janalik, Aryan Eftekhari

---

Mini-project 6

Due date: 3 December 2019, 23:55

Adding MPI to the mini-app

In assignment 4 we have added OpenMP to the mini-app, so that we could use all cores on one node. The goal of this assignment is to add MPI to the OpenMP version, so we are able to use multiple nodes.

Look at the directory `Mini-project6/miniapp_mpi`. The source codes are almost equivalent to the OpenMP version you have already implemented. You should modify the same files as before. There are comments that will help you.

Before you start

Download the source codes from git

```
$ cd HPC_2019
$ git pull
```

Allocate as many nodes as you need. Let's say 4.

```
$ salloc -N 4 --ntasks-per-node=1
```

For debugging you can use 1 node and execute multiple processes there. But when your code works well and you are running benchmarks, use 1 process per node.
Load openmpi and python modules

```
$ module load openmpi python/2.7.12
```

Domain decomposition

In serial and OpenMP version there is only one process that has all the data. In MPI version we have many processes (ranks). The grid on which we compute the stencil is devided into equal parts and every part is assigned to one process. Each process has only its part of the grid and cannot access data that belong to other processes.

But in order to compute the new value of some point, we need to know values of all of its neighbors. If the point is on the boundary, we need to get value from the process that has this data. So before every iteration, processes first exchange "halo cells" and save values from neighbors to boundary buffers (bndN, bndS, bndE, bndW). After the exchange every process has all the data it needs to compute the next iteration.

Hints

Note that at the beginning the code does not compile. It is because you first have to initialize MPI and there are also some errors because of missing parts you should fill in.

When you work on ghost cells exchange (`operators.cpp`), look at the resulting image. You will see that only part of the image is correct, or parts of the image are flipped in N-S or E-W direction. Think why this happens. It will help you to find what is wrong in your code.

For testing you can use the same parameters as for OpenMP version, e.g. `./main 128 128 100 0.01`

Don't forget that you cannot start MPI application the same way as serial or OpenMP version. You have to use `srun` or `mpirun`.

```
$ export OMP_NUM_THREADS=10
$ mpirun -np 4 ./main 128 128 100 0.01
```

If you want to run all processes on one node. It is useful for debugging, because you don't block many nodes. But use it **only for debugging**.

```
$ salloc -N 1 --ntasks-per-node=4
```

For measurements to your report use **one process per node**.

```
$ salloc -N 4 --ntasks-per-node=1
```

1. Initialize and finalize MPI *(5 Points)*

In file `main.cpp`

- Initialize MPI
- Get current rank and number of ranks
- Finalize MPI

2. Create a Cartesian topology *(10 Points)*

Mini-app uses a 2D grid. Each rank will work on a part of the grid. Make a 2D domain decomposition of the grid depending on the number of ranks.
In file `data.cpp`

- Create the dimension of the decomposition depending on the number of ranks
- Create a non-periodic Cartesian topology for the grid of domains
- Identify coordinates of the current rank in the domain grid
- Identify neighbours of the current rank: east, west, north and south directions

3. Change linear algebra functions *(5 Points)*

Make the dot product and the computation of the norm over all ranks.

Why only these two functions and not the others?
In file `linalg.cpp`

- Add a collective operation to compute the dot product
- Add a collective operation to compute the norm

4. Exchange ghost cells *(60 Points)*

Each rank has only it's own part of the grid. Before every iteration it is necessary to exchange ghost cells between the neighbors, so every process has all the data it needs for the stencil computation.

Use point to point communication to exchange ghost cells among neighbours.
In file `operators.cpp`

- Add point-to-point communication for all neighbours in all directions
- Use Non-blocking communication
- Try to overlap computation and communication

Be careful to send first / last row / column. Before you send the data, copy it to the send buffers (buffN, buffS, buffE, buffW) and receive to ghost cells (bndN, nbdS, bndE, bndW). Because you copy the data to the 1D array fist, you don't need any custom data types for send and receive. Look at Figure 1 to see an example.

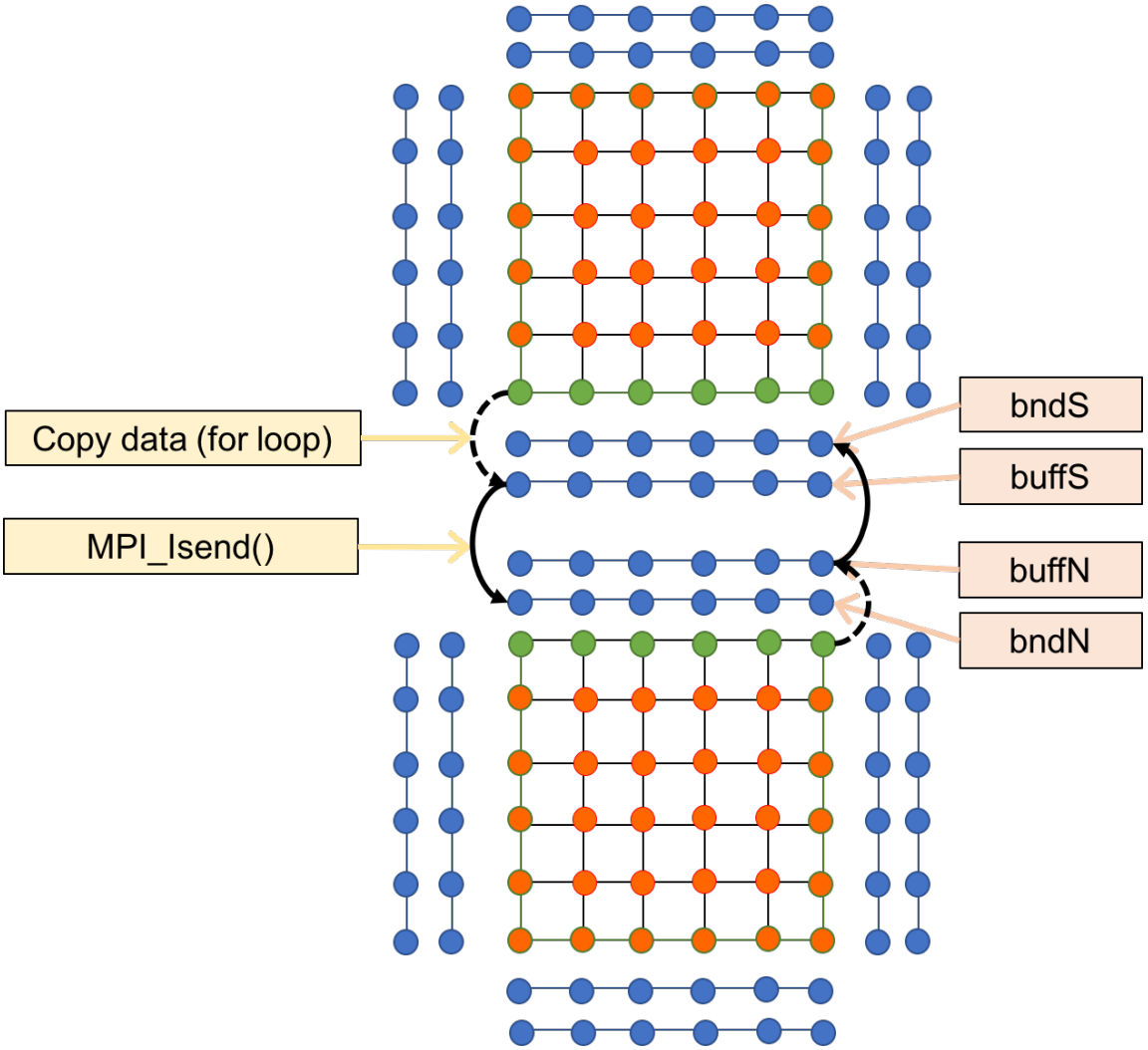

Figure 1. Ghost cell exchange [1]. Copy the North (South) row to buffN (buffS), send buffN (buffS) to the neighbor, receive to bndS (bndN).

5. Testing *(20 Points)*

How does it scale at different resolutions?
Plot time to solution for these grid sizes for 1-10 threads and 1, 2, 4 MPI ranks.

- 128x128
- 256x256
- 512x512
- 1024x1024
- You can also use larger grid sizes.

Submission:

Submission: Submit the source code files in an archive file (tar, zip, etc) and show the TA the results. Furthermore, summarize your results and the observations for all exercises by writing an extended Latex summary. Use the Latex template from the webpage and upload the extended Latex summary as a PDF to iCorsi.

References

[1] T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, Addison-Wesley Professional, first ed., 2004.