INSTITUTE FOR MICROELECTRONICS
nssc@iue.tuwien.ac.at

Numerical Simulation and Scientific Computing II
Exercise 1

Submission is due on Tuesday, April 13th 2021, 8am (submit to nssc@iue.tuwien.ac.at)

- Include the name of all group members (min. 2, max. 3) in your submission.
- The binaries should be callable with command line parameters as specified below.
- Submit everything (Makefiles, Shell-Scripts, Sources, Binaries, PDF with Plots/Discussion) as one zip-file per task.

General information

- Use the GNU compiler collection for the submitted binaries using at least the following compiler flags: `-std=c++11 -Wall -pedantic` .
- Use (and assume) the `double` precision floating point representation.
- Test your parallel implementation on your local machine before you benchmark on the cluster.
- Compare the results (i.e., residual and error) with your serial implementation to ensure a correct implementation.
- Use only a small number of Jacobi iterations to benchmark your code, convergence is not required during benchmarking.

Parallel Stencil-Based Jacobi Solver

In this excercise your task is to parallelize a stencil-based Jacobi solver for a 2D elliptic PDE (1) with analytic solution (2) and right-hand-side (3) discretized on a regular finite-difference grid with fixed boundary conditions (5)-(8) for the domain (4). As a staring point, you can use your own implementation of the Jacobi solver from the NSSC I exercises or use the source-code distributed with this exercise.

$$-\Delta u(x,y) + k^2 u(x,y) = f(x,y) \quad ,\text{with } k = 2\pi \tag{1}$$

$$u_p(x,y) = sin(kx)sinh(ky) \tag{2}$$

$$f(x,y) = k^2 u_p(x,y) \tag{3}$$

$$\Omega = [0,2] \times [0,1] \tag{4}$$

$$u(0,y) = 0 \tag{5}$$

$$u(2,y) = 0 \tag{6}$$

$$u(x,0) = 0 \tag{7}$$

$$u(x,1) = sin(kx)sinh(k) \tag{8}$$

Domain Decomposition

Your task is to decompose the finite-difference grid into domain regions so that multiple MPI-processes can independently perform an iteration on each region. The decoupling of the regions is achieved by introducing a "ghost layer" of grid points which surrounds each region. The values in the ghost layer of a region are not updated during an iteration. Instead, after an iteration is finished the updated values for the ghost layer are received from the neighbouring regions, and the boundary layer is sent to the neighouring regions (see Figure 1). For global properties, the required information must be communicated to a master process (or between all processes), e.g., to compute the global residual or error norms.

Task 1: Questions (3 Points)

(a) Describe the advantages/disadvantages of a two-dimensional decomposition (compared to a one-dimensional decomposition).

(b) Using a ghost layer based decomposition, how could multiple independent Jacobi iterations be achieved before communication has to happen? Comment in which situation (w.r.t the available bandwidth or latency between processes) multiple independent iterations are potentially advantageous.

(c) Describe the conceptual differences between an hybrid OpenMP/MPI-parallelization over a pure MPI-parallelization.

(d) How big is the sum of all L2 caches for 2 nodes of the IUE-cluster[1]?

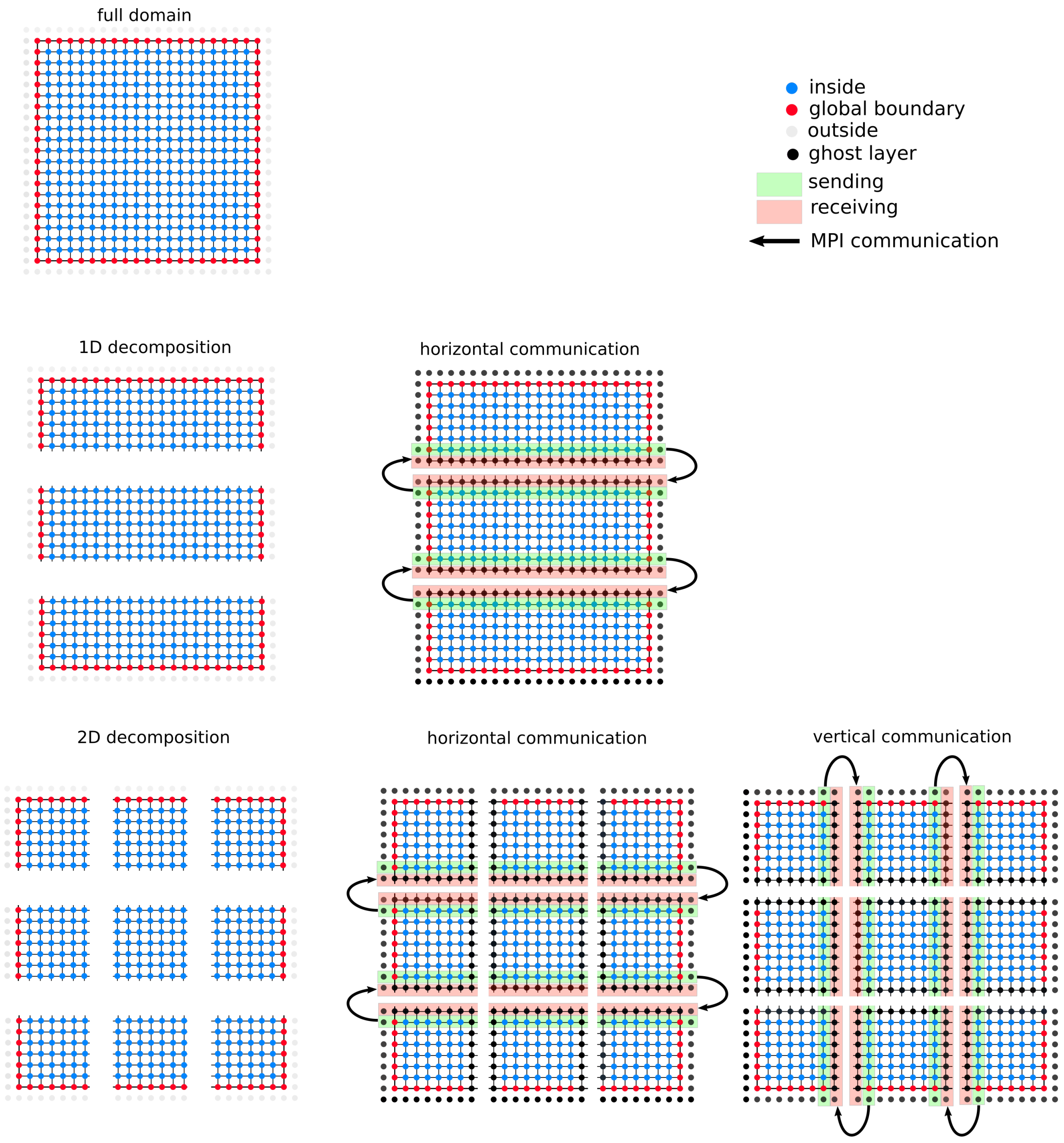

Figure 1: Square domain discretized using a regular finite difference grid indicating a 1D/2D decomposition using a ghost layer.

[1]`https://www.iue.tuwien.ac.at/research/computing-infrastructure/`

Task 2: One-Dimensional Decomposition (3 points)

Your task is to implement a one-dimensional decomposition using a ghost layer and MPI-communication to update the ghost layers. Create a program which is callable by

```
mpirun -n NUMMPIPROC ./jacobiMPI resolutionY iterations
```

e.g., `mpirun -n 4 ./jacobiMPI 250 30`

where `NUMMPIPROC` is the number of MPI-processes to launch,
`resolutionY` implies a `resolutionX` $= 2 \times$ `resolutionY` $- 1$,
the grid spacing is defined as $h_x = h_y = h = 1.0/($`resolutionY-1`$)$, and
`iterations` defines the number of Jacobi iterations to perform.
Further and more specifically, your program should

- use $\bar{u}_h = \mathbf{0}$ as initial approximation to u, and (after finishing all iterations)
- print the gobal Euclidean $\| \cdot \|_2$ and Maximum $\| \cdot \|_\infty$ norm of the residual $\| A_h\bar{u}_h - b_h \|$ and of the total error $\| \bar{u}_h - u_p \|$ to the console, and
- print the average runtime per iteration to the console, and
- produce the same results as a serial run.

Finally, benchmark the parallel performance of your program `jacobiMPI` using 2 nodes of the IUE-Cluster for 4 different `resolutionYs`$=\{125, 500, 2000, 4000\}$ using between 1 and 80 MPI-processes (`NUMMPIPROC`). More specifically, you should

- create a plot of the parallel speedup and a plot of the parallel efficiency for each `resolutionY`, and
- discuss the results in detail.

Notes

- On your local machine, you can also launch MPI-runs using `mpirun` (after installing MPI, see Makefile for install commands on Ubuntu)
- An example of a "MPI"-Makefile and of a job submission script are provided with this exercise.
- The use of `MPI_Cart_create`, `MPI_Cart_coords`, and `MPI_Cart_shift` for setup of the communication paths is recommended.
- Your implementation should work for any positive integer supplied for `NUMMPIPROC` (e.g., 1,2,3,4,...) and also utilize this number of processes for the decomposition.

Task 3: Single-Precision Data Representation (1 Points)

Adopt your program `jacobiMPI` from above by changing the underlying floating point data type of the direcrtization from `double` to `float`.
Specifically,

- ensure a correct implementation (by comparing results to a serial run), and
- compare the performance results to with the results from Task 2 in a suitable plot and dicuss your results.

Task 4: Two-Dimensional Decomposition (3 points)

Extend your program from Task 2 by implementing a two-dimensional decomposition using a ghost layer and MPI-communication to update the ghost layers. Create a program which is callable by

```
mpirun -n NUMMPIPROC ./jacobiMPI DIM resolutionY iterations
```
e.g.,
```
mpirun -n 4 ./jacobiMPI 2D 125 200
```
where the command line parameters have the same meaning as bove and `DIM` switches between `1D` and `2D` decomposition.

Ensure a correct implementation by comparing your results to a serial run. Benmarking on the cluster is not required.

Notes

- Your implementation should work for any positive integer supplied for `NUMMPIPROC` (e.g., 1,2,3,4,...) and also utilize this number of processes for the 2D decomposition. If a the 2D composition is not possible with the supplied number of processes (i.e., a prime number), your program should resort to a 1D decomposition.