Subject: CAAM 420/520 – Homework 4
Date Posted: March 24, 2023
Date Due: 7:00pm, March 31, 2023 via Canvas

Total number of points available: 58 (CAAM 420 and 520)

Problem 1 : Overlapping Communication and Stencil Dependency *(16 pts)*

Suppose we were working on the FD problem from HW2 with the backwards referenced stencil. The subdomain on an given rank for a fully embedded block would look as shown in Figure 1.

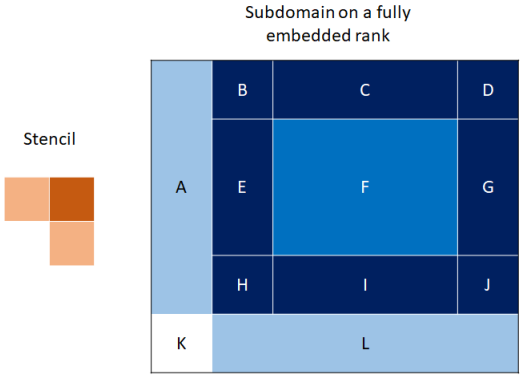

Figure 1. Example subdomain and halo for a single rank with a fully embedded block.

- *(4pts)* What regions would be sent by fully-embedded ranks?
- *(4pts)* What regions would be received by fully-embedded ranks?
- *(8pts)* Can you overlap communication and computation in this case? If so, to what extent? If not, why not? Use the regions given in Figure 1 when justifying your answer.

Problem 2 : Halo Exchange *(42 pts)*

Associated Files: `main.cpp, halo_exchange.h`

Name your files: `<netID>_halo_exchange.cpp`, `<netID>_set_blocks.cpp`

Expected compile command:

```
mpic++ -o halo -std=c++11 main.cpp <netID>_halo_exchange.cpp
```

Running the program (note the export commands only need to be run when intially setting the variables or when changing their values): `export NX=<number of blocks in x>`

```
export NY=<number of blocks in y>
export HALO_RADIUS=<radius of the halo>
mpirun -n $(($NX*$NY)) ./halo
```

WARNING: do not modify the provided files. For testing you can write your own main file if you like and compile your program using the same command as above with your main file in place of `main.cpp`.

Here we will only consider the mechanics of the halo exchange; for the sake of testing we will not implement the FD method.

In place of the FD update, every rank should assign their rank ID to their subdomain as illustrated in Figure 2. The halo exchange will then serve to update each rank's halos.

Rank's should be assigned to blocks using row-major indexing. In the code, the global domain references the entire computational domain, which is split across ranks; local entities, such as the local domain, is the portion of the domain on the current rank.

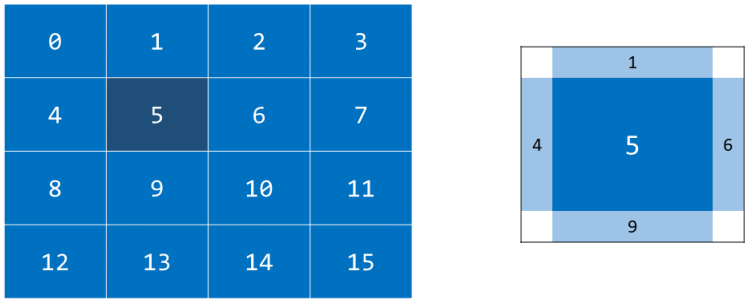

Figure 2. Global domain and memory on each rank showing what values should be placed in each region.

(a) **Computation and Order Dependency** *(5 pts)*

In class we had mentioned that *some* of the computation(the inner halos) must be completed before the halo exchange communication takes place. We may actually

be able to do better still than the algorithm given in the slides when we have two arrays (`u_new` and `u_old`). The communication can be broken into sends and receives.

Do both the sends and the receives need to wait for the computation of the inner halo when we have two arrays? Use the diagram in Figure 3 for referencing regions on a rank when justifying your answer.

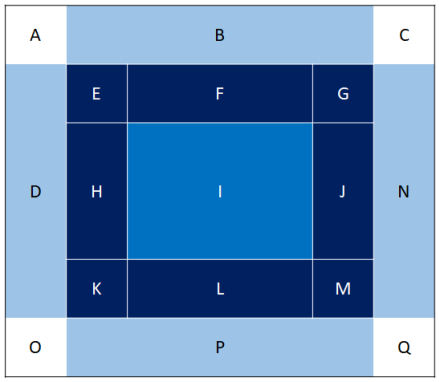

Figure 3. Regions associated with a fully embedded rank.

(b) **Halo Packing** *(3 pts)*
This code will use row-major (i.e. x-major) indexing. Which halos will need to be packed into separately allocated send/recv buffers vs using a pointer to the rank's main data? For the halo's that do not need to be packed, will they contain extra data? (Again, use the regions in Figure 3 when referencing regions)

(c) **Overlapping Communication and Order Dependency** *(4 pts)* Note that here, there is only one array being acted on. What is the order that a rank's regions must be processed/exchanged to ensure correctness and maximally hide communication?

(d) **Halo-Exchange Code** *(30 pts)*
Implement the functions in `halo_exchange.h`:

- *(4 pts)* `pack`
  Call this function from `process_block` when packing the halo/inner halo for the direction that needs it.
- *(4 pts)* `unpack`
  Call this function from `process_block` when unpacking the halo/inner halo for the direction that needs it.
- *(22 pts)* `process_block`
  Implement the halo exchange and assignment of values here. Note that N_x or N_y (the number of blocks in the x and y direction; these will be passed into the program via environment variables) can be 1; in these cases there are no neighbors in that direction so no halo exchanges will take place in that direction. For other cases have edge blocks wrap around. For instance, in Figure 2, rank 0 would exchange information for its top halo with rank 12. Information for its left halo it would get from rank 3.

Make sure to test a variety of N_x, N_y combinations and values of `HALO_RADIUS`, which is the width of the halos. These are all set before running your program using the `export` commands given above. Note that N_x and N_y need to evenly divide n_x, and n_y, which are both set to 128. The code will print the data on all ranks (including their halos, so it will actually print more than the global domain) for you to check via inspection.