CS633: Assignment 1

Shrilakshmi S K (211012) Prashant Sharma (210750) Venkaat Balaje (211159)

March 21, 2024

1 Code Explanation

For more details, please view `code_documentation.pdf` Code is divided into 3 parts: Initialization, Preprocessing, Communication and Computation. In preprocessing, we calculate various variables needed further such as existence of neighbours for processes. We call `MPI_Barrier` here to not consider the effect of heavy preprocessing and memory allocations in time. We start the time here. In communication, we first pack and then use Right Sends followed by Left Sends algorithm for Nearest Neighbour communication. In computation, we calculate the new value for each cell in the `data` matrix. We end the time. We then report the maximum time taken for stencil communication and computation among all processes. We run the code on Prutor and plot the boxplot for 3 observations.

2 Optimizations

For more detailed methodology, please view `code_documentation.pdf`

- We allocate space only if the neighbouring processes exist.
- We allocate space depending upon stencil. So, `stencil = 5` takes up half as much as space as `stencil = 9`.
- We observed (The supporting data is attached in `code_documentation.pdf`) that the average time is lesser in Right Sends followed by Left Sends Algorithm compared to Odd Even Algorithm. Hence we choose the former algorithm to communicate in the final code.
- We have also not used `MPI_Unpack` after `MPI_Recv`. We have adjusted the indices accordingly in stencil computation part to eliminate the use

of `MPI_Unpack`, hence removing the copying overhead and optimize the code further.

3 Performance Observations

We picked the best 3 recorded times to plot the boxplot as follows.

`N = 512*512` `Stencil = 5`	`N = 512*512` `Stencil = 9`	`N = 2048*2048` `Stencil = 5`	`N = 2048*2048` `Stencil = 9`
0.069165	0.095521	0.843737	1.744045
0.083086	0.107675	0.87256	1.761941
0.065187	0.11778	0.949845	1.893914

Table 1: Time in seconds for `P_x = 4, P = 12, num_time_steps = 10`

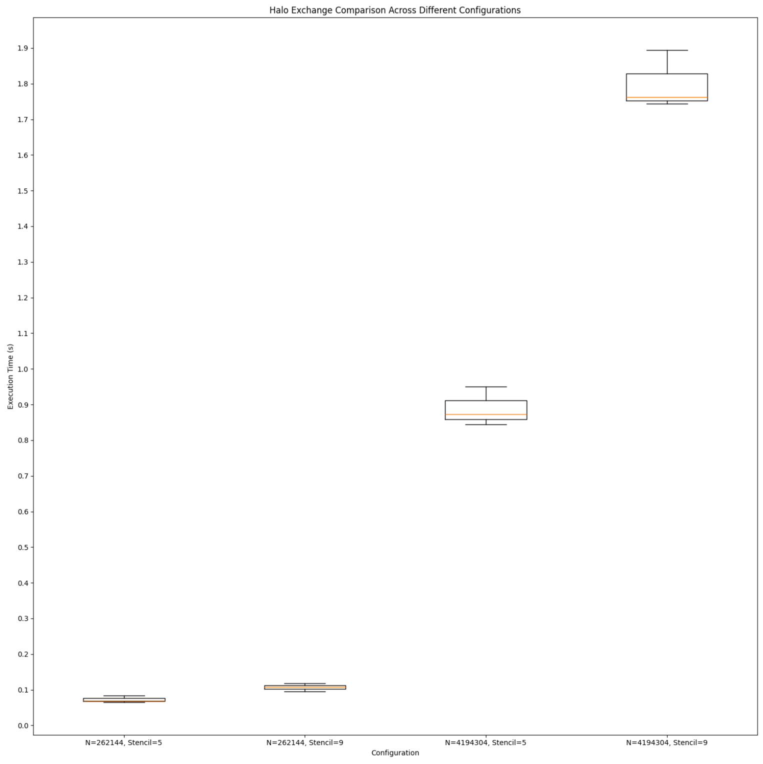

Figure 1: Boxplot of Halo Exchange Across Different Configurations

3.1 Complexities

		Maximum communication data size sent or received by a single process (bytes)	Maximum number of additions to calculate sum for stencil computation by a single process
1	`N = 512*512, Stencil = 5`	2048	2048
2	`N = 512*512, Stencil = 9`	4096	4096
3	`N = 2048*2048, Stencil = 5`	8192	8192
4	`N = 2048*2048, Stencil = 9`	16384	16384

Table 2: Stencil communication & computation complexity of a single process

3.2 Observations

- Both communication and computation complexities increase by 200% from Case 1 to Case 2. Despite this, the time only increases by 40-80%. This can be attributed to packing. Because in both cases, despite different communication data sizes, a single `MPI_Send` and `MPI_Recv` is called, hence reducing overheads associated with another send and receive call. Other factors for lesser time than expected could be load imbalance and lack of synchronization after each iteration in the loop.

- Both communication and computation complexities increase by 400% from Case 1 to Case 3. Despite this, the time increases by 800-1200%. Both Case 1 and Case 3 are latency limited as they are sending and receiving less than 128KB data in each call. The reason could be that, both communication and computation requires memory access of `data` matrix. These huge number of memory accesses may not be linearly scalable as nodes have different level of cache (NUMA architecture). Other reasons also could be load imbalance (This is particularly important as the data initialization depends on i and j which in turn depends on N) and resource contention. One more reason could be that all processes do not synchronize after an iteration and hence in next iteration, the earlier processes are blocked by `MPI_Send` and `MPI_Recv` calls. This could propagate more delay in further iterations.

- Both communication and computation complexities increase by 200% from Case 3 to Case 4. Surprisingly, the time also increases by 200%. The observed linear increase may be attributable to the fact that, while in previous 2 points, the overheads we discussed were minor and had a consequential impact, the significant rise in complexity for this case renders these overheads negligible by comparison.

- **Limitation in our observations:** Different process placements or network topologies have not been compared as for all execution on Prutor, we observed that that processes 0 to 5 ran on `csews28` and processes 6 to 11 ran on `csews1`. We could not obtain the data from different processors and process placements.

4 Group Details

Group: 43
Shrilakshmi S K - 211012 - shrilakshm21@iitk.ac.in
Prashant Sharma - 210750 - kprashant21@iitk.ac.in
Venkaat Balaje - 211159 - venkaatb21@iitk.ac.in