CS633: Assignment 1

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

March 21, 2024

1 Code, Methodology, Optimizations

1.1 Initialization

We first assign the arguments given to variables. Then we initialize MPI. We get the host name and print it for all observations.

```
    char hostname[MPI_MAX_PROCESSOR_NAME];
    MPI_Init (&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    MPI_Comm_size(MPI_COMM_WORLD, &P);
    MPI_Status status;
    int len;
    MPI_Get_processor_name (hostname, &len);
    int coreID = sched_getcpu();

    printf("rank %d running on %s\n", myrank, hostname);
```

We observe that processes 0 to 5 run on `csews28` and processes 6 to 11 run on `csews1`. We allocate space for data matrices and initialize the matrix. We compute the existence of all 4 neighbours for all processes.

```
if(myrank % P_x == 0)
        left_n = false;
    else
        left_n = true;
    if(myrank % P_x == P_x - 1)
        right_n = false;
    else
```

```
        right_n = true;
    if(myrank / P_x == 0)
        up_n = false;
    else
        up_n = true;
    if(myrank / P_x == P_y - 1)
        down_n = false;
    else
        down_n = true;
```

We allocate space for send and receive buffers.

```
if(left_n){
    left_recv = (double *)malloc(side * num * sizeof(
       ↪ double));
    left_send = (double *)malloc(side * num * sizeof(
       ↪ double));
}
if(right_n){
    right_recv = (double *)malloc(side * num * sizeof(
       ↪ double));
    right_send = (double *)malloc(side * num * sizeof(
       ↪ double));
}
if(up_n){
    up_recv = (double *)malloc(side * num * sizeof(
       ↪ double));
    up_send = (double *)malloc(side * num * sizeof(
       ↪ double));
}
if(down_n){
    down_recv = (double *)malloc(side * num * sizeof(
       ↪ double));
    down_send = (double *)malloc(side * num * sizeof(
       ↪ double));
}
```

OPTIMIZATIONS

- 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`.

Since all the initializations are done, we move on to communication and computation.

1.2 Communication

We start the time and begin the loop of textttnum_time_steps here.

COMMON ERROR

We should call `MPI_Barrier` and synchronise all the processes before we start the time. Because of a lot of initializations preceding this and lot of memory allocations, it is highly probable that all processes will not synchronously arrive to this loop. If a few processes are later than others, this will lead to blocking in subsequent `MPI_Send` and `MPI_Recv` of earlier processes, thus inflating the maximum completion time.

We pack the data we want to send in send buffers. For `stencil = 9`, we pack both the rows or columns into a single 1D array. While computing the stencil, we will adjust the indices accordingly.

```
1  if(left_n){
2              int l = 0;
3              for(int j = 0; j < num; j++){
4                  for(int i = 0; i < side; i++){
5                      MPI_Pack(&data[i][j], 1, MPI_DOUBLE,
                           ↪  left_send, side * num *
                           ↪ sizeof(double), &l,
                           ↪ MPI_COMM_WORLD);
6                  }
7              }
8          }
9
10
```

```
11
12          if(right_n){
13              int r = 0;
14              for(int j = side - num; j < side; j++){
15                  for(int i = 0; i < side; i++){
16                      MPI_Pack(&data[i][j], 1, MPI_DOUBLE,
                            ↪  right_send, side * num *
                            ↪ sizeof(double), &r,
                            ↪ MPI_COMM_WORLD);
17                  }
18              }
19          }
20
21          if(up_n){
22              int u = 0;
23              for(int i = 0; i < num; i++){
24                  for(int j = 0; j < side; j++){
25                      MPI_Pack(&data[i][j], 1, MPI_DOUBLE,
                            ↪  up_send, side * num * sizeof(
                            ↪ double), &u, MPI_COMM_WORLD);
26                  }
27              }
28          }
29
30          if(down_n){
31              int d = 0;
32              for(int i = side - num; i < side; i++){
33                  for(int j = 0; j < side; j++){
34                      MPI_Pack(&data[i][j], 1, MPI_DOUBLE,
                            ↪  down_send, side * num *
                            ↪ sizeof(double), &d,
                            ↪ MPI_COMM_WORLD);
35                  }
36              }
37          }
```

The next step is sending and receiving data to and from neighbouring processes. Here we employ two algorithms discussed in class.
We first try out odd-even method of neighbour communication.

```
1  if(((myrank % P_x) % 2) == 0 && right_n){
2              MPI_Send(right_send, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + 1,
                  ↪ myrank + 1, MPI_COMM_WORLD);
3              MPI_Recv(right_recv, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + 1,
                  ↪ myrank, MPI_COMM_WORLD, &status);
4          }
5          else if (((myrank % P_x) % 2) == 1 && left_n){
6              MPI_Recv(left_recv, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank - 1,
                  ↪ myrank, MPI_COMM_WORLD, &status);
7              MPI_Send(left_send, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank - 1,
                  ↪ myrank-1, MPI_COMM_WORLD);
8          }
9          if(((myrank % P_x) % 2) == 0 && left_n){
10             MPI_Send(left_send, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank - 1,
                  ↪ myrank - 1, MPI_COMM_WORLD);
11             MPI_Recv(left_recv, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank - 1,
                  ↪ myrank, MPI_COMM_WORLD, &status);
12         }
13         else if (((myrank % P_x) % 2) == 1 && right_n){
14             MPI_Recv(right_recv, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + 1,
                  ↪ myrank, MPI_COMM_WORLD, &status);
15             MPI_Send(right_send, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + 1,
                  ↪ myrank + 1, MPI_COMM_WORLD);
16         }
17         if(((myrank / P_x) % 2 )== 0 && down_n){
18             MPI_Send(down_send, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + P_x,
                  ↪ myrank + P_x, MPI_COMM_WORLD);
19             MPI_Recv(down_recv, side * num * sizeof(
                  ↪ double), MPI_PACKED, myrank + P_x,
                  ↪ myrank, MPI_COMM_WORLD, &status);
```

```
20          }
21          else if (((myrank / P_x) % 2) == 1 && up_n){
22              MPI_Recv(up_recv, side * num * sizeof(double
                    ↪ ), MPI_PACKED, myrank - P_x, myrank,
                    ↪ MPI_COMM_WORLD, &status);
23              MPI_Send(up_send, side * num * sizeof(double
                    ↪ ), MPI_PACKED, myrank - P_x, myrank -
                    ↪ P_x, MPI_COMM_WORLD);
24          }
25          if(((myrank / P_x) % 2) == 0 && up_n){
26              MPI_Send(up_send, side * num * sizeof(double
                    ↪ ), MPI_PACKED, myrank - P_x, myrank -
                    ↪ P_x, MPI_COMM_WORLD);
27              MPI_Recv(up_recv, side * num * sizeof(double
                    ↪ ), MPI_PACKED, myrank - P_x, myrank,
                    ↪ MPI_COMM_WORLD, &status);
28          }
29          else if (((myrank / P_x) % 2) == 1 && down_n){
30              MPI_Recv(down_recv, side * num * sizeof(
                    ↪ double), MPI_PACKED, myrank + P_x,
                    ↪ myrank, MPI_COMM_WORLD, &status);
31              MPI_Send(down_send, side * num * sizeof(
                    ↪ double), MPI_PACKED, myrank + P_x,
                    ↪ myrank + P_x, MPI_COMM_WORLD);
32          }
```

We record multiple observation times of stencil communication and computation for 10 times **(Table 1)**.

We next try out Right Sends followed by Left Sends algorithm.

```
1  if((myrank % P_x) < (P_x - 1)){
2      MPI_Send(right_send, side * num * sizeof(double),
           ↪ MPI_PACKED, myrank + 1, myrank + 1,
           ↪ MPI_COMM_WORLD);
3      MPI_Recv(right_recv, side * num * sizeof(double),
           ↪ MPI_PACKED, myrank + 1, myrank, MPI_COMM_WORLD
           ↪ , &status);
4  }
5
6  if((myrank % P_x) > 0){
7      MPI_Recv(left_recv, side * num * sizeof(double),
```

N = 512*512 Stencil = 5	N = 512*512 Stencil = 9	N = 2048*2048 Stencil = 5	N = 2048*2048 Stencil = 9
0.129758	0.255263	2.642078	3.889563
0.319764	0.446686	2.723942	5.204010
0.157748	0.407793	3.505116	4.112610
0.082223	0.124064	1.423459	2.083425
0.053595	0.131683	1.100825	1.953753
0.187250	0.311018	1.441511	2.419764
0.114963	0.203370	1.412683	2.856533
0.065813	0.097893	1.536768	1.665227
0.210653	0.378807	2.601703	4.497096
0.235704	0.289712	1.940953	2.645182
0.094637	0.119297	1.666537	2.559862
0.103017	0.118323	1.662261	2.344135
0.084956	0.149314	1.204297	3.131902
0.080584	0.226755	1.656593	2.185397
0.075815	0.103943	0.976338	1.787833
0.076607	0.139601	1.886889	2.850758
0.087761	0.173939	1.077444	3.976789
0.080959	0.146329	2.158231	2.983053

Table 1: Time in seconds for Odd Even Algorithm

```
        ↪ MPI_PACKED, myrank - 1, myrank, MPI_COMM_WORLD
        ↪ , &status);
8       MPI_Send(left_send, side * num * sizeof(double),
        ↪ MPI_PACKED, myrank - 1, myrank - 1,
        ↪ MPI_COMM_WORLD);
9   }
10
11  if((myrank / P_x) < (P_y - 1)){
12      MPI_Send(down_send, side * num * sizeof(double),
        ↪ MPI_PACKED, myrank + P_x, myrank + P_x,
        ↪ MPI_COMM_WORLD);
13      MPI_Recv(down_recv, side * num * sizeof(double),
        ↪ MPI_PACKED, myrank + P_x, myrank,
        ↪ MPI_COMM_WORLD, &status);
14  }
```

```
15
16 if((myrank / P_x) > 0){
17     MPI_Recv(up_recv, side * num * sizeof(double),
          ↪ MPI_PACKED, myrank - P_x, myrank,
          ↪ MPI_COMM_WORLD, &status);
18     MPI_Send(up_send, side * num * sizeof(double),
          ↪ MPI_PACKED, myrank - P_x, myrank - P_x,
          ↪ MPI_COMM_WORLD);
19 }
```

We record multiple observation times of stencil communication and computation for 10 times **(Table 2)**.

N = 512*512 Stencil = 5	N = 512*512 Stencil = 9	N = 2048*2048 Stencil = 5	N = 2048*2048 Stencil = 9
0.083086	0.107675	0.872560	1.761941
0.065187	0.117780	0.949845	1.893914
0.118668	0.225680	1.427137	2.504941
0.078703	0.143986	1.030153	1.874408
0.082954	0.234168	1.123390	2.439716
0.072093	0.124028	0.804028	4.376888
0.069165	0.095521	0.843737	1.744045
0.066180	0.134722	0.885873	2.341248
0.064257	0.186773	1.004169	1.756409
0.085684	0.177666	1.018166	3.870621
0.071120	0.120894	1.012264	3.558501
0.117432	0.172093	1.009631	2.024556
0.090082	0.147758	1.983103	3.616979
0.086070	0.139175	0.895644	2.812353

Table 2: Time in seconds for Right Sends followed by Left Sends Algorithm

OPTIMIZATIONS

- We observe 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.

1.3 Computation

In this step, we have computed the new value of each cell of data matrix. We decrement the value of denominator if neighbour does not exist.

```
for(int i = 0; i < side; i++){
    for(int j = 0; j < side; j++){
        double sum = data[i][j];
        double num_n = (double)stencil;
        if(stencil == 5){
            if(j == 0){
                if(left_n)
                    sum += left_recv[i];
                else
                    num_n-=1;
            }

            if(j == side - 1){
                if(right_n)
                    sum += right_recv[i];
                else
                    num_n-=1;
            }

            if(i == 0){
                if(up_n)
                    sum += up_recv[j];
                else
```

```
                    num_n-=1;
            }

            if(i == side - 1){
                if(down_n)
                    sum += down_recv[j];
                else
                    num_n-=1;
            }

            sum = sum + mat(i, j - 1, side) + mat(i, j +
               ↪  1, side) + mat(i - 1, j, side) + mat(
               ↪ i + 1, j, side);

            data2[i][j] = sum/num_n;
        }
        if(stencil == 9){
            if(j == 0){
                if(left_n){
                    sum += left_recv[i];
                    sum += left_recv[i + side];
                }
                else
                    num_n-=2;
            }

            if(j == 1){
                if(left_n){
                    sum += left_recv[i + side];
                }
                else
                    num_n-=1;
            }

            if(j == side - 1){
                if(right_n){
                    sum += right_recv[i];
                    sum += right_recv[i + side];
                }
                else
```

```
62                      num_n-=2;
63              }
64
65              if(j == side - 2){
66                  if(right_n){
67                      sum += right_recv[i];
68                  }
69                  else
70                      num_n-=1;
71              }
72
73              if(i == 0){
74                  if(up_n){
75                      sum += up_recv[j];
76                      sum += up_recv[j + side];
77                  }
78                  else
79                      num_n-=2;
80              }
81
82              if(i == 1){
83                  if(up_n){
84                      sum += up_recv[j + side];
85                  }
86                  else
87                      num_n-=1;
88              }
89
90              if(i == side - 1){
91                  if(down_n){
92                      sum += down_recv[j];
93                      sum += down_recv[j + side];
94                  }
95                  else
96                      num_n-=2;
97              }
98
99              if(i == side - 2){
100                 if(down_n){
101                     sum += down_recv[j];
```

```
                }
                else
                    num_n-=1;
            }
            sum = sum + mat(i, j - 1, side) + mat(i, j -
               ↪  2, side) + mat(i, j + 1, side) + mat(
               ↪ i, j + 2, side) + mat(i - 1, j, side)
               ↪ + mat(i - 2, j, side) + mat(i + 1, j,
               ↪ side) + mat(i + 2, j, side);

            data2[i][j] = sum/num_n;
        }
    }
}
```

We store the new value of cell in another matrix `data2`. We avoid copying back so many grid cells by just exchanging the pointers.

```
void swap_matrices (double ***data, double ***data2){
    double **temp = *data;
    *data = *data2;
    *data2 = temp;
}

swap_matrices(&data, &data2);
```

We end the loop and stop the time here. We report the maximum time taken among all processes.