Assignment 1 COMP4300

Abhaas Goyal (u7145384)

April 26, 2021

Table 1: Table abbreviations

$\mathbf{A_T}$	Advection Time
$\mathbf{N_{Send}}$	Blocking Send
$\mathbf{I_{Send}}$	Non-Blocking Send
$\mathbf{G_F}$	Gigaflops per second
$\mathbf{P_C}$	Per core (with G_F)
NP	Number of processes

1 Deadlock issues

The prototype function `parAdvect()` can fail for total buffer length $> 2^{15}$ because there are 2 `MPI_Send` happening at the same time. When buffer length is small then the send is locally blocking (so it copies the message into the local buffer). But for larger value of total buffer length it becomes globally blocking (so both the messages now are stuck in the first `MPI_Send` waiting for the other process to receive). Hence, we face a deadlock situation here.
The simplest solution is to divide up the ranks into even and odd, and switch up the sending and receiving these classes.

```
if (rank % 2 == 0) {
    MPI_Send(&V(u, M_loc, 1), N_loc, MPI_DOUBLE, botProc, HALO_TAG, comm);
    MPI_Recv(&V(u, 0, 1), N_loc, MPI_DOUBLE, topProc, HALO_TAG, comm,
             MPI_STATUS_IGNORE);
    MPI_Send(&V(u, 1, 1), N_loc, MPI_DOUBLE, topProc, HALO_TAG, comm);
    MPI_Recv(&V(u, M_loc+1, 1), N_loc, MPI_DOUBLE, botProc, HALO_TAG,
             comm, MPI_STATUS_IGNORE);
} else {
    MPI_Recv(&V(u, 0, 1), N_loc, MPI_DOUBLE, topProc, HALO_TAG, comm,
             MPI_STATUS_IGNORE);
    MPI_Send(&V(u, M_loc, 1), N_loc, MPI_DOUBLE, botProc, HALO_TAG, comm);
    MPI_Recv(&V(u, M_loc+1, 1), N_loc, MPI_DOUBLE, botProc, HALO_TAG,
             comm, MPI_STATUS_IGNORE);
    MPI_Send(&V(u, 1, 1), N_loc, MPI_DOUBLE, topProc, HALO_TAG, comm);
}
```

Now it seems to be safely working for large values of M and N.

2 The effect of non-blocking communication

2.1 Implementing Non-blocking communication

- `MPI_ISend` and `MPI_IRecv` were implemented with the same style as given in the original question with `MPI_ISend` and `MPI_IRecv`, which form the basis for later questions.
- The above was done keeping in mind that there were no race conditions during top/bottom and left/right halo exchange

2.2 Timing of Blocking vs Non-blocking communication rationale

- The timing was defined by using the difference of `M_Wtime()` between top and bottom halo sending and receiving of messages to maximize preciseness.
- In each repetition the total time accumulator was increased and in the end average time to update was taken. The source could be found in a previous commit here. The basic methodology to reduce and put the answer is here as follows

```
avg_time = total_time / reps;
MPI_Reduce(&avg_time,&root_avg , 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (rank == 0) {
  printf("%d: Average time to update boundary wrt reps: %.1e\n", rank,
         root_avg/n_ranks);
}
```

- The tests were done with small values of M and huge values of N since we need to focus on timings of communication here and in top/bottom exchange large values of N would mean a larger chunk of block to sent/receive.
- It is also to be noted that the value of M should atleast be the number of working nodes for all nodes participating together in alteast one send/receive. Hence, I chose M as {48, 192} for tests
- Another advanctage of keeping the value of M small is that we can still exploit the L3 cache with larger values of N
- Number of reps were taken to be 100 for making the sending/receiving average more consistent
- The tests were done 3 times and we were provided with consistent values.

2.3 Results

- For small values of N (Table 2), got varied results with in some cases blocking being faster and in others non-blocking being faster

For large values of N (Table 3), one could clearly see that non-blocking sends/received proved to be consistently faster to a noticeable extent. Hence, we would be using that for the following set of questions.

Table 2: Blocking vs Non Blocking Send for Large values of N

M	**N**	**NP**	$\mathbf{N_{Send}}$	$\mathbf{I_{Send}}$
48	1000	3	1.0e-06	2.6e-06
48	1000	6	7.0e-06	4.2e-06
48	1000	12	8.9e-06	9.3e-06
48	1000	24	9.3e-05	7.4e-05
48	1000	48	9.3e-05	5.4e-05
192	1000	48	1.4e-05	6.8e-06
192	1000	96	9.9e-05	7.6e-05
192	1000	192	1.1e-05	6.6e-05

Table 3: Blocking vs Non Blocking Send for Large values of N

M	**N**	**NP**	$\mathbf{N_{Send}}$	$\mathbf{I_{Send}}$
48	100000	3	1.0e-03	3.6e-04
48	100000	6	7.0e-04	4.2e-04
48	100000	12	8.9e-04	9.3e-04
48	100000	24	9.3e-04	7.4e-04
48	100000	48	9.3e-04	5.4e-04
192	100000	48	1.4e-03	6.8e-04
192	100000	96	9.9e-04	7.6e-04
192	100000	192	1.1e-03	6.6e-04

3 TODO Make Performance modelling and calibration

- Performance model was tested in one node
- The goal was to minimize top and bottom halo exchange time. Hence, like in question 2, a large value of N and small value of M was taken. In this case `M = 48` and `N = 1000000`. They remain unchanged for increasing NP in this case because we want to do strong scaling.
- Number of reps was taken to be 100

3.1 Results

NP	$\mathbf{A_T}$	$\mathbf{G_F}$	$\mathbf{P_C}$
3	6.01e-01	1.60e+01	5.32e+00
6	5.11e-01	1.88e+01	3.13e+00
12	5.35e-01	1.79e+01	1.50e+00
24	2.40e-01	4.01e+01	1.67e+00
48	1.28e-01	7.53e+01	1.57e+00

- When going from 6 to 12 processors, I was surprised to see the advection time increasing instead of decreasing and for initial values of `NP` $\mathbf{A_T}$ doesn't seem to decrease that much. I don't know of the reason but it's got to do with various layers of the memory hierarchy. Investigate further for support.
- Other than that, we see a consistent result of $\mathbf{A_T}$ almost halving after N=12.

4 The effect of 2D process grids

M = N = 2000 (2 * L_3 cache has around 70 MB memory)

P	Q	NP	$\mathbf{A_T}$	$\mathbf{G_F}$	$\mathbf{P_C}$
1	12	12	2.70e-01	2.97e+01	2.47e+00
1	24	24	8.19e-02	9.76e+01	4.07e+00
1	48	48	4.42e-02	1.81e+02	3.77e+00
2	6	12	2.69e-01	2.98e+01	2.48e+00
2	12	24	7.92e-02	1.01e+02	4.21e+00
2	24	48	3.35e-02	2.38e+02	4.97e+00
3	4	12	2.64e-01	3.03e+01	2.53e+00
3	8	24	8.08e-02	9.90e+01	4.12e+00
3	12	48	3.30e-02	2.42e+02	5.05e+00
6	2	12	2.65e-01	3.02e+01	2.52e+00
6	4	24	8.38e-02	9.55e+01	3.98e+00
6	8	48	3.06e-02	2.61e+02	5.44e+00
12	1	12	2.65e-01	3.02e+01	2.52e+00
12	2	24	7.74e-02	1.03e+01	4.31e+00
12	4	48	2.91e-02	2.75e+02	5.73e+00
16	3	48	2.35e-02	3.40e+02	7.08e+00
24	2	48	3.20e-02	2.50e+02	5.51e+00

From the above table, best ratio of P:Q is either between 3 and 5.2 (Because Q is stranded it needs some more time to send than P) On further investigation an estimate was found to around 4 (by looking at the above data and additional exp)

P	Q	NP	A_T	G_F	P_C
48	1	48	2.70e-01	2.97e+01	2.47e+00
16	3	48	2.35e-02	3.40e+02	7.08e+00
96	1	96	4.97e-02	1.61e+02	1.68e+00
16	6	96	1.73e-02	4.62e+02	4.81e+00
192	1	192	4.40e-02	1.81e+02	9.47e-01
16	12	192	1.45e-02	5.51e+02	2.87e+00
24	8	192	1.37e-02	5.85e+02	3.05e+00
32	6	192	2.19e-02s	3.66e+02	1.90e+00

Till Q3 Q was till 1 only so performance improvement in optimal values of P and Q are vv impressive

5 Overlapping communication with computation

M = 1000000, N = 48 or 192
In this question, we should capitalize on LR exchange Q4 models are bettret than q5 tho except when Q = 1 Where Q = 1

NP	G_F	G_R	P_C	G_F (-o)	G_R	P_C
48	3.78e-01s	1.02e+02	2.12e+00	3.60e-01	1.07e+02	2.22e+00
96	1.64e-01s	2.34e+02	2.44e+00	1.21e-01	3.17e+02	3.31e+00
192	5.33e-02s	7.21e+02	3.75e+00	3.95e-02	9.72e+02	5.06e+00

For large number of processes with high left and right halo exchange (it works nice)

6 Wide halo transfers

6.1 Gracefully exiting on w > m || w > n

- For `Q or P > 1` Since the ranks which touch the right and bottom of the field may not satisfy the condition of `w > M_loc || w > N_loc` (because tho blocks size may be smaller there during division of work), normally checking this wouldn't work for all ranks.
- However, we know that for `rank == 0`, if this condition holds true then all the blocks are in danger.
- Hence, we broadcast the value to exit in this scenario
- I also changed the return type of `checkHaloSize()` to int so that all the processes could gracefully exit from `main()`

```
if (rank == 0) {
  if (w > M_loc || w > N_loc) {
    halo_error = 1;
  }
}
MPI_Bcast(&halo_error, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (halo_error == 1) {
  if (rank == 0) {
    printf("%d: w=%d too large for %dx%d local field! Exiting...\n",
           rank, w, M_loc, N_loc);
  }
  return -1; // Could have used exit(0); here but didn't to gracefully exit from main
}
```

6.2 Implementation of wide halo technique

A wide halo technique would be useful when it's applied in conjunction with overlapping (since you need to do some extra computation) 4 * w^2 ?? How do you make up for it? Functionally and algorithmically it is correct however maybe the implemation is wrong for some waits waiting thing is there which idk. To be repeated It is functionally correct though.
M = N = 2000 reps = 100

6.3 Results

w	P	Q	NP	G_R	G_R	P_C
1	3	4	12	2.64e-01	3.03e+01	2.53e+00
1	16	3	48	2.35e-02	3.40e+02	7.08e+00
1	24	8	192	1.37e-02	5.85e+02	3.05e+00
2	3	4	12	2.70e-01	2.97e+01	2.47e+00
2	16	3	48	3.01e-02	2.66e+02	5.55e+00
2	24	8	192	2.66e-01	3.01e+01	2.51e+00
3	3	4	12	2.70e-01	2.97e+01	2.47e+00
3	16	3	48	3.14e-02	2.55e+02	5.31e+00
3	24	8	192	2.68e-01	2.99e+01	2.49e+00
4	3	4	12	2.70e-01	2.97e+01	2.47e+00
4	16	3	48	2.99e-02	2.68e+02	5.58e+00
4	24	8	192	2.70e-01	2.96e+01	2.47e+00

- Sample values of tiled stencil with optimum values of P and Q in Q4
- Can suprisingly see increasing trend of w because of the reason above and also because cache locality is not sustained in increasing order of lines. Maybe the implementation is lacking but my assumption is that the following approach is limited.

7 TODO Tiled Stencil Technique

A tiling transformation can be used to improve locality of data, hence improving cache performance by a huge margin. In a 9 point stencil the distance to be traveled between the central to bottom right corner is N_{loc} + 2 * w + 1. The papers mention that tiling techniques could improve reuse, with array padding.

8 TODO Combination of Wide halo and stencil technique

9 Extra optimizations

- A potential optimization was seen when I saw an opportunity to explore Q5 further to work with values of Q > 1. For that, I also referenced my doubts and got it clarified on Piazza (link).
- The algorithm works to overlap both row and column exchange with computation in the case that
  1. The left and right exchange only happens when the values of the corners of the top and bottom halos are exchanged
  2. The final updation of the inner halo only happens after receiving all the messages.

9.1 Results

P	Q	NP	G_R	G_R	P_C
16	3	48	2.37e-02	3.38e+02	7.05e+00
16	6	96	1.69e-02	4.68e+02	4.81e+00
16	12	192	1.44e-02	5.57e+02	2.87e+00
24	8	192	1.33e-02	5.97e+02	3.11e+00

We see a Similar performance to Q4 with it being slightly slow at some points (but and improved version of Q5 nonetheless). I would have liked to know how I could have improved this model w.r.t the part on exchanging corners.

10 References

[1] Tiling optimizations for 3D scientific computations `https://dl.acm.org/doi/10.5555/370049.370403`