Assignment 1

Gohil Dwijesh 2017CS50407
Prafful Ravuri 2017CS10369

January 2020

1 Machine specification

- Dual core processor (for Blocking P2P and Non-Blocking communication parts)
- Quad core processor (for Collective Communication)
- Open MPI 2.2.1.

2 Assumptions

- For blocking and non-blocking point to point communication algorithm, number of processes must be a perfect square.
- If the matrix size of matrix A is (n x m) then matrix B size is assumed to be (m x n) so the result of matrix multiplication will be (n x n).
- For blocking and non-blocking P2P communication, size of the input matrix should be divisible by $\sqrt{p}$. Where p is number of processes. There should be such restriction because we are dividing the input matrices into p blocks. Each block being $n/\sqrt{p}$ x $m/\sqrt{p}$.
- In Collective Communication, there are no constraints upon the matrix size n.
- All the three implementations are done using 4 processes.

3 Cannon's Algorithm

For blocking and non-blocking point-to-point communication we have implemented Cannon's algorithm. We chose to implement this algorithm because it is memory efficient. If we have matrix A of size (n x m) then Cannon's algorithm uses only O(2*n*m + n*n) memory. n*m for each A, B and n*n for

answer matrix. Which is equivalent to the memory required in the sequential implementation.

How does Cannon's algorithm work?

- $\begin{array}{|c|c|}\hline A_{00} & A_{01} \\ \hline A_{10} & A_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline B_{00} & B_{01} \\ \hline B_{10} & B_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline P_{00} & P_{01} \\ \hline P_{10} & P_{11} \\ \hline \end{array}$

  Consider dividing matrix A and B in 4 block matrix as shown. Also consider the 2D grid of processes.

- Process P_{ij} initially holds A_{ij} and B_{ij}.

- We define an initial reordering of blocks so that each process gets their intended blocks. After reordering, each process will have the following blocks.

$\begin{array}{|c|c|}\hline A_{00} & A_{01} \\ \hline A_{11} & A_{10} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline B_{00} & B_{11} \\ \hline B_{10} & B_{01} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline P_{00} & P_{01} \\ \hline P_{10} & P_{11} \\ \hline \end{array}$

- Now in the first iteration, each process does matrix multiplication of the allotted block matrices A_{ij} and B_{pq}. And it looks as the following.

$\begin{array}{|c|c|}\hline A_{00} & A_{01} \\ \hline A_{11} & A_{10} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline B_{00} & B_{11} \\ \hline B_{10} & B_{01} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline P_{00} & P_{01} \\ \hline P_{10} & P_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline A_{00}*B_{00} & A_{01}*B_{11} \\ \hline A_{11}*B_{10} & A_{10}*B_{01} \\ \hline \end{array}$

- Now from second iteration onward each process will send their corresponding block matrix A_{ij} to their immediate left process and will receive $A_{i(j+1)}$ from the immediate right process. Each process will send their corresponding block matrix B_{pq} to immediate upper process and will receive $B_{(p+1)q}$ from immediate below process. And the configuration now looks as the following.

$\begin{array}{|c|c|}\hline A_{01} & A_{00} \\ \hline A_{10} & A_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline B_{10} & B_{01} \\ \hline B_{00} & B_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline P_{00} & P_{01} \\ \hline P_{10} & P_{11} \\ \hline \end{array}$

- Again each process will multiply corresponding block matrix and add it to the previous results. Final configuration is given below.

$\begin{array}{|c|c|}\hline P_{00} & P_{01} \\ \hline P_{10} & P_{11} \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline A_{00}*B_{00}+A_{01}*B_{10} & A_{01}*B_{11}+A_{00}*B_{01} \\ \hline A_{11}*B_{10}+A_{10}*B_{00} & A_{10}*B_{01}+A_{11}*B_{11} \\ \hline \end{array}$

which is equivalent to multiplying matrix A with matrix B.

$\begin{array}{|c|c|}\hline A_{00} & A_{01} \\ \hline A_{10} & A_{11} \\ \hline \end{array} * \begin{array}{|c|c|}\hline B_{00} & B_{01} \\ \hline B_{10} & B_{11} \\ \hline \end{array} = \begin{array}{|c|c|}\hline A_{00}*B_{00}+A_{01}*B_{10} & A_{01}*B_{11}+A_{00}*B_{01} \\ \hline A_{11}*B_{10}+A_{10}*B_{00} & A_{10}*B_{01}+A_{11}*B_{11} \\ \hline \end{array}$

- In the assignment, we had implemented Cannon's algorithm in the blocking P2P communication and non blocking communication parts.

4 Algorithm

4.1 Blocking Point-to-Point Communication

- Declaring the data-structure: We used pointer to a double element for 2D matrix (double *) which points to a contiguous memory block, storing the matrix in row major order.

- *Process 0*: Initializing the matrix A, and B. Sending the corresponding block matrix to other processes using MPI_Send function.
  Other processes: Waiting to receive corresponding block matrix.

- All processes create the 2D mesh type topology and send the initial intended block matrix to each other. Process P_{ij} will send the block matrix A_{ij} to left by shift of i amount, and B_{ij} to up by shift of j amount. It will also receive accordingly from right and lower processes.

- All processes start calculating corresponding answer matrix. Process P_{ij} will calculate ans_{ij} block matrix. Recall that we divided original matrix into total of p blocks, with $\sqrt{p}$ rows and $\sqrt{p}$ columns. So now consider the multiplication at the block level, where to calculate one block of answer matrix we have to multiply $\sqrt{p}$ block matrices and sum sum them up.

- *Process 0*: Wait for all other processes to send the their corresponding answer block matrix.
  Other processes: Send the answer block matrix to process 0.

4.2 Non-Blocking Point-to-Point Communication

- While distributing the work to other processes, we used non-blocking send at process 0 and blocking receive at other processes.

- When the processes are evaluating the block of answer matrix, they have to shift their allotted block matrix A_{ij} and B_{pq} at corresponding processes. They have to do both send those matrices and receive both from other processes. So to avoid the **dead locks** we used send-recv and recv-send alternatively in neighbouring processes. In case of recv-send, once receiving the new block matrix, the process should start multiplying and intermediately check for the acceptance of send request. So we replaced recv-send with **recv-Isend**.

4.3 Collective Communication

- The root process takes user input for size of the matrices and randomly initializes the matrices A and B.

- The root process 0 then broadcasts the matrix sizes (m,n) and also the whole matrix B to all processes in the global communicator using **MPI Broadcast**.
- The root then distributes the n number of rows of matrix A equally among all the p number of processes using **MPI Scatter**. (What if n isn't divisible by p? Discussed below).
- Each process then performs the sequential multiplication of their corresponding sub matrix of A and B (in parallel!) and obtains the result, a sub matrix of final result A*B.
- Using **MPI Gather**, all the sub matrices of the final output C, calculated in different processes are gathered to the root process 0.
- If n isn't divisible by p i.e say remainder = r, then a new group is created fro the processes with rank in 0,1...,r-1 using the MPI function **MPI Group incl**. Then each row of the remaining r rows of matrix A are sent to each of the process in the new group, multiplications are performed done in parallel, and then sub matrices are gathered back to the root. This is achieved by creating a new communicator using the **MPI Comm Create** function.
- Note: In the first two parts, datatype of elements is double and in collective communication it's float.

5 Specific Optimizations

- **Communication Optimization:** In case of blocking and non-blocking communication we used 2D grid topology for representing the processes.
- **Memory Efficient Algorithm:** For blocking and non-blocking communication we have implemented the Cannon's algorithm, which is memory efficient. It means that the memory requirement by such concurrent programs is same as that of the sequential programs.
- **Disadvantages of using Cannon's Algorithm:** Cannon's algorithm is memory efficient at the cost of introducing more block matrix transactions which could have been avoided by allowing to use more memory storage. It becomes slower than the later case.
- Tried to allocate memory in the heap using malloc separately for each of the row/column (float**) thinking that searching for a contiguous block of memory by the system would be easier (and less chance of segmentation fault) if it had to allocate n separate blocks instead of a contiguous n*n memory block. Implemented using this data structure for collective communication at first, but later realized that the original data structure is giving much better results.

6 Principle and Implementation Difference among three Algorithms

1. **Blocking vs Non-blocking P2P Communication** Calculating the block of the answer matrix involves sending and receiving of block matrices to and from neighbouring processes. Each process calls send followed by receive or receive followed by send to avoid dead-locks. Now consider a case when a process is calling receive followed by a send.

```
1          P00                              P01
2          MPI_Recv(A_dash)      <----      MPI_Send(A_dash)
3 <----    MPI_Send(A)                      MPI_Recv(A)          <----
4          C += A_dash*B                    C += A*B
```

   Now just consider P_{00} has received block matrix from P_{01} but the process left to it is not ready to receive from P_{00} then P_{00} will keep waiting for other left process to receive it. But later in the computation it is using A_dash only so it does not necessarily has to wait for the other process to receive the message. So in the **non-blocking P2P communication** we eliminated such situation as the following.

```
1          P00                              P01
2          wait()
3          MPI_Recv(A_dash)      <----      MPI_Send(A_dash)
4 <----    MPI_Isend(A)                     MPI_Recv(A)          <----
5          C += A_dash*B                    C += A*B
```

   So the principle difference between blocking and non-blocking P2P communication is that sometimes a process doesn't need to wait for a completion of a communication. The process can leave it as a request, do some work and then wait for the the completion.

2. **P2P vs Collective Communication** The core algorithm used itself is different among the two implementations. P2P communication is implemented using Cannon's algorithm and Collective Communication is done using a simple algorithm where a matrix is broadcasted and work is divided among the other matrix's rows. The former is a better algorithm compared to the latter in terms of memory efficiency etc. But, in order to demonstrate the usage of all collective communication methods, we had used this algorithm

7 Empirical Results

N	B P2P time(2 core)	Non-B P2P time(2 core)	Seq(2 core)	Collective time(4 core)	Seq(4 core)
12000	11.76	11.93	43.13	16.60	61.34
10000	8.36	8.25	29.26	11.39	42.70
8000	5.41	5.55	18.95	7.310	26.55
6000	2.93	3.01	10.21	4.084	15.13
4000	1.35	1.35	4.60	1.82	6.59
2000	0.34	0.35	1.13	0.46	1.66
1000	0.09	0.08	0.28	0.11	0.42
800	0.05	0.05	0.18	0.073	0.27
600	0.03	0.03	0.104	0.041	0.15
400	0.014	0.01	0.047	0.0174	0.062
200	0.0038	0.0038	0.011	0.0048	0.0166
100	0.0013	0.0013	0.002	0.0013	0.00412
50	0.0013	0.0019	0.0006	0.00055	0.00096
32	0.0011	0.0007	0.0002	0.00022	0.0004

Table 1: Exec. Time(sec) of different approaches for varying size of N

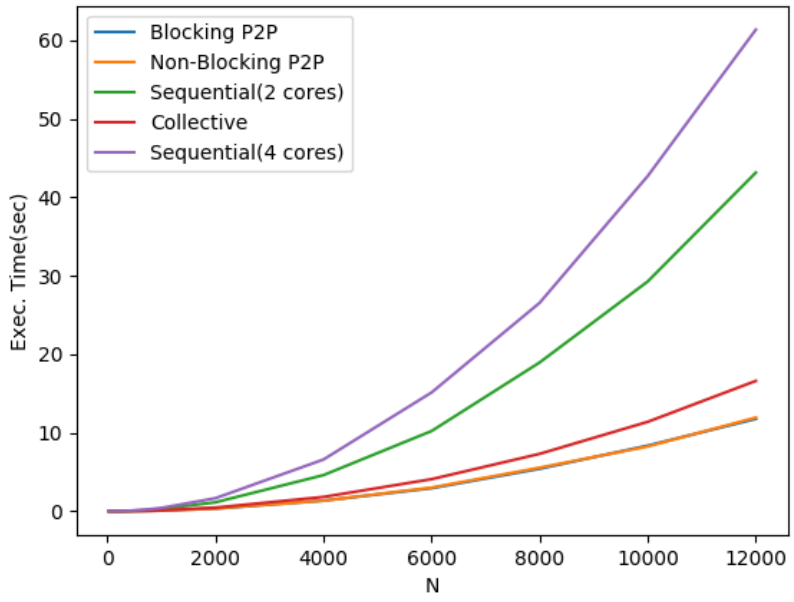

Figure 1: Execution time(sec) vs N

8 Observation

- Any general matrix multiplication is of $O(n^3)$ order. We had one dimension of the matrix to be constant, equal to 32. So theoretical time complexity is $O(n^2)$. The empirical results also ensures the quadratic (time vs n) curve.
- Blocking and non-blocking execution time is almost the same for all n. It is probably because we are using only four processes. Non-blocking is better when a process is not waiting for a completion of communication and starts doing some work and later it waits for the completion. For such small number of processes it is more likely that a communication completes immediately. In this case it does not matter if a process waits or starts some work, so there won't be much difference between blocking and non-blocking communication.
- In collective communication implementation, the speed up is increasing as the size of matrix increases, and for N as high as 12000, it gives a speedup of about 3.69. A probable explanation for this could be that - for low N, communication overhead is being significant and as N increases the effect of the communication overhead diminishes.
- Theoretically, the optimal speed up possible is 4 in our case because we are using 4 processes. We are getting an optimal speed up of around 3.69. Which is reasonable because of the inter-process communication overhead.