HW4

1. Greene network test

The latency and bandwidth test on greene different nodes are as follows:

```
pingpong latency: 1.707000e-06 ms
pingpong bandwidth: 6.062443e+05 GB/s
```

2. MPI RING Communication

Start with

By start with integer and check correctness, we can get the following result:

```
for (int repeat = 0; repeat < Nrepeat; repeat++)
    {
        MPI_Status status;
        if (rank == 0)
        {
            MPI_Send(&sum, 1, MPI_INT, 1, repeat, MPI_COMM_WORLD);
            MPI_Recv(&sum, 1, MPI_INT, size - 1, repeat, MPI_COMM_WORLD,
&status);
            printf("Iteration %d: sum = %d\n", repeat, sum);
        }
        else
        {

            MPI_Recv(&sum, 1, MPI_INT, rank - 1, repeat, MPI_COMM_WORLD,
&status);
            sum += rank;
            MPI_Send(&sum, 1, MPI_INT, (rank + 1) % size, repeat,
MPI_COMM_WORLD);
        }
    }
```

And run it with 4 processes and N = 10, we get the following result:

```
Iteration 0: sum = 6
Iteration 1: sum = 12
Iteration 2: sum = 18
Iteration 3: sum = 24
Iteration 4: sum = 30
Iteration 5: sum = 36
Iteration 6: sum = 42
Iteration 7: sum = 48
Iteration 8: sum = 54
Iteration 9: sum = 60
```

Each time is 0 + 1 + 2 + 3 = 6. The code is correct.

when run with 6 processes and N = 10, we get the following result:

```
Iteration 0: sum = 15
Iteration 1: sum = 30
Iteration 2: sum = 45
Iteration 3: sum = 60
Iteration 4: sum = 75
Iteration 5: sum = 90
Iteration 6: sum = 105
Iteration 7: sum = 120
Iteration 8: sum = 135
Iteration 9: sum = 150
```

Correct.

Test Latency

Then run on Large N = 1000000, and 4 node in Greene cluster, we get the following result:

```
Process 0 on cs424.hpc.nyu.edu out of 4
Process 2 on cs426.hpc.nyu.edu out of 4
Process 1 on cs425.hpc.nyu.edu out of 4
Process 3 on cs427.hpc.nyu.edu out of 4
ring latency: 3.628616e-03 ms
Total time: 3.628616
```

Test Bandwidth

Then run on Large N = 1000000, and 4 node in Greene cluster, we get the following result:

```
Process 1 on cs429.hpc.nyu.edu out of 4
Process 0 on cs428.hpc.nyu.edu out of 4
Process 3 on cs431.hpc.nyu.edu out of 4
Process 2 on cs430.hpc.nyu.edu out of 4
Total time: 3.754824
ring bandwidth: 4.750672e-01 GB/s
```

3. MPI Scan

I choose to write a MPI scan function, which is a MPI version of prefix sum. The code is as follows:

```
int main(int argc, char **argv)
{
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_rank(comm, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    srand(42);
    int Nsize = 1000000000;
    int *data = NULL, *local_data = NULL, *local_scan = NULL, *offsets =
NULL;

    if (rank == 0)
    {
        // Fill data on rank 0
        data = (int *)malloc(Nsize * sizeof(int));
        for (int i = 0; i < Nsize; i++)
        {
            data[i] = rand() % 10000;
            printf("%d : %d\n", i, data[i]);
        }
    }

    local_data = (int *)malloc(Nsize / size * sizeof(int));
    MPI_Scatter(data, Nsize / size, MPI_INT, local_data, Nsize / size,
MPI_INT, 0, comm);

    local_scan = (int *)malloc(Nsize / size * sizeof(int));
    local_scan[0] = local_data[0];
    for (int i = 1; i < Nsize / size; i++)
    {
        local_scan[i] = local_scan[i - 1] + local_data[i];
    }

    offsets = (int *)malloc(size * sizeof(int));
    MPI_Allgather(&local_scan[Nsize / size - 1], 1, MPI_INT, offsets, 1,
MPI_INT, comm);

    int pre_sum = 0;
    for (int i = 0; i < rank; i++)
    {
        pre_sum += offsets[i];
    }
    for (int i = 0; i < Nsize / size; i++)
    {
        local_scan[i] += pre_sum;
    }

    MPI_Gather(local_scan, Nsize / size, MPI_INT, data, Nsize / size,
MPI_INT, 0, MPI_COMM_WORLD);
    if (rank == 0)
    {
        for (int i = 0; i < Nsize; i++)
        {
            printf("%d ", data[i]);
        }
    }
    MPI_Finalize();
    return 0;
}
```

And I run it in N = 16 and processes = 2 to valid the correctness, and get the following result:

```
0 : 6166
1 : 2740
2 : 8881
3 : 3241
4 : 1012
5 : 6758
6 : 9021
7 : 4940
8 : 535
9 : 3743
10 : 5874
11 : 3143
12 : 9717
13 : 4504
14 : 3696
15 : 9762
6166 8906 17787 21028 22040 28798 37819 42759 43294 47037 52911 56054
65771 70275 73971 83733
```

The result is correct.

Then I run it in N = 16 and processes = 4 to valid the correctness, and get the following result:

```
0 : 6166
1 : 2740
2 : 8881
3 : 3241
4 : 1012
5 : 6758
6 : 9021
7 : 4940
8 : 535
9 : 3743
10 : 5874
11 : 3143
12 : 9717
13 : 4504
14 : 3696
```

```
15 : 9762
6166 8906 17787 21028 22040 28798 37819 42759 43294 47037 52911 56054
65771 70275 73971 83733
```

It is correct too.

Run in large N

Then I run it in N = 10000000 and processes = 16 (4 node and 4 task per node) to valid the correctness, and get the following result:

```
Process 0 on cs471.hpc.nyu.edu out of 16
Process 1 on cs471.hpc.nyu.edu out of 16
Process 2 on cs471.hpc.nyu.edu out of 16
Process 3 on cs471.hpc.nyu.edu out of 16
Process 6 on cs472.hpc.nyu.edu out of 16
Process 10 on cs473.hpc.nyu.edu out of 16
Process 13 on cs474.hpc.nyu.edu out of 16
Process 4 on cs472.hpc.nyu.edu out of 16
Process 8 on cs473.hpc.nyu.edu out of 16
Process 5 on cs472.hpc.nyu.edu out of 16
Process 14 on cs474.hpc.nyu.edu out of 16
Process 11 on cs473.hpc.nyu.edu out of 16
Process 12 on cs474.hpc.nyu.edu out of 16
Process 9 on cs473.hpc.nyu.edu out of 16
Process 15 on cs474.hpc.nyu.edu out of 16
Process 7 on cs472.hpc.nyu.edu out of 16
time: 2
```

4. Pitch your final project.

I am going to do my project with Letao Chen

We plan to use OpenMP and CUDA to solve high-level Sudoku problems.First, we will write a correct sequential version, and then use it as a baseline to compare the performance improvements brought by OpenMP and CUDA parallelization, and explore both strong scalability and weak scalability.