Parallel Programming Exercise 6.8 - Estimate the network latency and bandwidth

Author:	張凱捷 (R08922054@ntu.edu.tw)
Student ID	R08922054
Department	資訊工程所

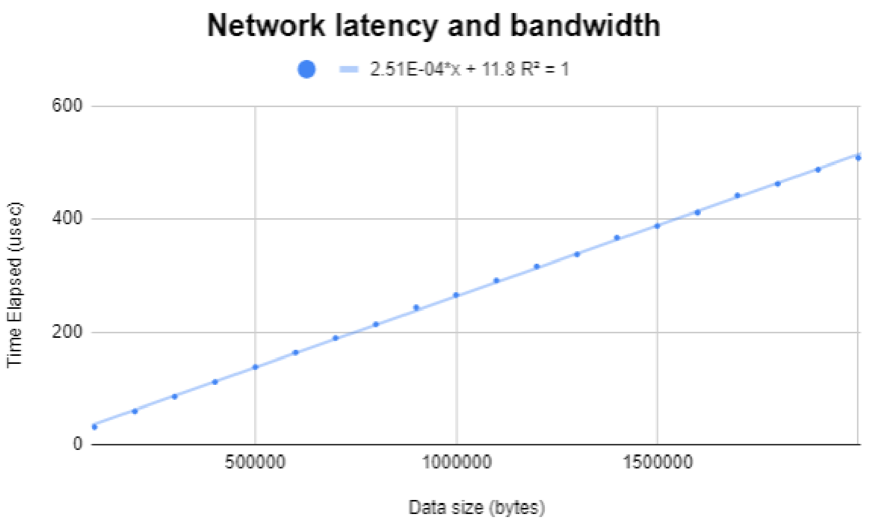
1 Problem and Proposed Approach

Problem: 嘗試使用 MPI_Send, MPI_Recv 來估計 Lambda 和 Beta。

Approach: 傳送 100000 * [1, 20] 大小的資料，每種大小傳送 10 次，扣除掉最大和最小的執行時間各 2 筆，取中間 6 筆的平均當作總傳輸時間 Lambda + Data Size / Beta。接著把得到的資料作線性回歸得到參數。

2 Result

Data Size (Bytes)	Transmission Time (Second)
100000	0.00003163
200000	0.000059585
300000	0.000085255
400000	0.0001115
500000	0.000138203
600000	0.000163297
700000	0.000189245
800000	0.000213822
900000	0.000243326
1000000	0.00026534
1100000	0.000291348
1200000	0.000316421
1300000	0.000337084
1400000	0.000367085
1500000	0.000387728
1600000	0.00041159
1700000	0.000441809
1800000	0.000462433
1900000	0.000487745
2000000	0.000507832



Lambda: 1.18 * 10^-5 second

Beta: 4 * 10^9 Bytes / Second

Appendix(optional):

Code:

```
#include <bits/stdc++.h>
using namespace std;

#include "mpi.h"

int main(int argc, char** argv) {
    int id;
    int psize;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &psize);

    int dataSize = 100000;
    int maxRound = 20;
    int avgRound = 10;
    int delRound = 2;

    char *arr = new char[dataSize * maxRound];
    MPI_Status status;

    if (id == 0) {
        for (int round = 1; round <= maxRound; round++) {
            printf("[Info] Round %d started\n", round);
            // get 5 average time
            vector<double> timeArr;
            double avgTime = 0;
            int avgCnt = 0;
            for (int i = 0; i < avgRound; i++) {
                double time = -MPI_Wtime();
                MPI_Send(arr, dataSize * round, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
                MPI_Recv(arr, dataSize * round, MPI_CHAR, 1, 1, MPI_COMM_WORLD,
&status);
                time += MPI_Wtime();
                // printf("[Run #%d] Receive size: %d, time elapse %10f\n", i,
dataSize * round, time / 2);
                timeArr.push_back(time / 2);
            }
            sort(timeArr.begin(), timeArr.end());
            for (int i = delRound; i < avgRound - delRound; i++) {
                avgTime += timeArr[i];
                avgCnt += 1;
            }
            avgTime /= avgCnt;
            printf("[Average] Receive size: %d, time elapse %.9f\n", dataSize * round, avgTime);
        }
    } else {
        for (int round = 1; round <= maxRound; round++) {
            for (int i = 0; i < avgRound; i++) {
                MPI_Recv(arr, dataSize * round, MPI_CHAR, 0, 1, MPI_COMM_WORLD, &status);
                MPI_Send(arr, dataSize * round, MPI_CHAR, 0, 1, MPI_COMM_WORLD);
            }
        }
    }

    MPI_Finalize();
    return 0;
}
```