CS 5500 HW4 : All Reduce Add

Marshal Taylor

February 17, 2022

Abstract

A concise report describing my implementation All Reduce Add

1 Introduction

Each time the program is run, every processor will get a random number. Then, the program will sum up all random numbers for each of the processors, and then let each processor know what the total sum was.

2 Commands

1) Compile: VS code Build

```
2)mpiexec -np {number of processsors} .\HM4AllReduceAdd.exe
```

3) Due to the hypercube, the number of processors needs to be a power of 2

3 Implementation

3.1 All Reduce Add

1) The simplest method. Just use MPI's built in method.

```
void getAllReduce(int rank, int data) {
    int result = 0;
    MPI_Allreduce(&data, &result, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    printTotal(rank, data, result);
    return;
}
```

3.2 Gather and BCast

2) We need to gather all random values. Once we have all random values from all of the processors, if we are rank 0, sum the values up and bcast them back to each of the processors.

```
void getGather(int rank, int data, int size) {
    int Barray[100];
    int result = 0;

    MPI_Gather(&data, 1, MPI_INT, &Barray, 1, MPI_INT, 0, MPI_COMM_WORLD);

    if (!rank) {
        for (int i = 0; i < size; i++) {
            result += Barray[i];
```

```
        }
    }
    MPI_Bcast(&result, 1, MPI_INT, 0, MPI_COMM_WORLD);
    printTotal(rank, data, result);
    return;
}
```

3.3 Send And Receive

3) The naive approach. Send our random number to processor 0, and wait for processor 0 to receive all random numbers. Once it has, sum them up and send them to each of the processors.

```
void getSendAndRecieve(int rank, int data, int size) {
    int result = data;
    if (!rank) {
        int rec_size = 1;
        int rec_total;
        while (1) {
            MPI_Recv(&rec_total, 1, MPI_INT, MPI_ANY_SOURCE, 1, MCW, MPI_STATUS_IGNORE);
            result += rec_total;
            rec_size += 1;
            if (rec_size == size) break;
        }

        for (int i = 1; i < size; ++i) {
            MPI_Send(&result, 1, MPI_INT, i, 1, MCW);
        }
        printTotal(rank, data, result);
    }
    if (rank) {
        MPI_Send(&data, 1, MPI_INT, 0, 1, MCW);

        while (1) {
            MPI_Recv(&result, 1, MPI_INT, MPI_ANY_SOURCE, 1, MCW, MPI_STATUS_IGNORE);
            printTotal(rank, data, result);
            break;
        }
    }
    return;
}
```

3.4 Ring

3) We first need to calculate the next and previous processors in the ring. Then, for each processor, I need to send my information forward, receive the raw information behind me. Add the raw information to my local total, and pass the raw information forward. If I do this size number of times, I will at one point or another get every other processors raw value, so I just need to sum them up locally.

```
void getRing(int rank, int data, int size) {
    int result = 0;
    int recvData = data;
    int receive = 0;
    if ((rank - 1) < 0) {
        receive = size-1;
    }
    else {
        receive = (rank - 1);
```

```
    }

    for (int i = 0; i < size; ++i) {
        MPI_Send(&recvData, 1, MPI_INT, (rank + 1) % size, 0, MCW);
        MPI_Recv(&recvData, 1, MPI_INT, receive, 0, MCW, MPI_STATUS_IGNORE);
        result += recvData;
    }
    printTotal(rank, data, result);

    return;
}
```

3.5 Cube

3) Calculate the dimensionality of the hypercube with the helper function find square. Then, while my dimensionality is greater than 0, parallel sum up the values along each edge. By reducing the dimensionality each time, I change the connections between the processors. They eventually converge on a single value that each processor will have. This value is the sum of the hypercube.

```
void getCube(int rank, int data, int size) {
    int result = data;
    int recvData = 0;
    int d = findSquare(size);
    int dest = rank;

    unsigned int mask = 1;

    while (d>=0) {
        dest = dest ^ (mask << d);
        //lower = std::min(rank, dest);

        MPI_Send(&result, 1, MPI_INT, dest, 0, MCW);
        MPI_Recv(&recvData, 1, MPI_INT, dest, 0, MCW, MPI_STATUS_IGNORE);
        result += recvData;

        d--;
    }

    printTotal(rank, data, result);
    return;
}
```

4 Expected Output

Number of processors lines of printed output. Each printed line will contain the processors rank, its random value, and the sum of all processors. This should happen 5 times as there are 5 ways that we calculate the total rank