CS 5500
Spring 2021
Homework 4
By Griffin Hackley
A02224681

At the bottom of each function is a line that will print the final result of each process. This can be commented in or out based on the desired output. This can be used to confirm that each process ended up with the correct result. In my implementations the numbers that were being added together are the rank of each process. So the expected results would be as follows

2 processes : 1
4 processes : 6
8 processes : 28
16 processes: 120
etc

The command to compile the code is:
mpic++ hw4.cpp

The command to run the code is:
mpirun -np (num) -oversubscribe a.out

where (num) is the number of processes (which must be a power of 2)

All Reduce:
This is the simplest solution to this problem. Each process sets the data equal to their rank. Then the allreduce function gathers the data from each process and adds them together using MPI SUM and returns it to each process. Each process already has the result so we do not need to send any more data.

```
Allreduce : 28
process 0 got 28
process 1 got 28
process 2 got 28
process 3 got 28
process 4 got 28
process 5 got 28
process 6 got 28
process 7 got 28
```

Gather:
This implementation uses gather to put the ranks of each process into an array. the 0^{th} process then adds them together and a Bcast is used to return the results to the other processes.

```
gather    : 28
process 1 got 28
process 2 got 28
process 3 got 28
process 4 got 28
process 5 got 28
process 6 got 28
process 7 got 28
process 0 got 28
```

Leader:

Each process (except the 0^{th}) sends its rank to process 0. Process 0 then computes the result and sends it back to each process, one after the other.

```
leader    : 28
process 0 got 28
process 1 got 28
process 2 got 28
process 3 got 28
process 4 got 28
process 5 got 28
process 6 got 28
process 7 got 28
```

Ring:

Starting at process 0, each process sends data to the process that is their rank+1 where their rank is added to the data. The final process sends the data back to process 0, closing the ring. This process is repeated except their rank is not added to the data. This ends with every process having the correct result.

```
ring      : 28
process 1 got 28
process 2 got 28
process 0 got 28
process 3 got 28
process 4 got 28
process 5 got 28
process 6 got 28
process 7 got 28
```

Hypercube:

Each process trades information across an axis with a partner process and adds each others data together. Then the same step is done across a different axis. This is repeated for each axis. In the end each process will have the final result and it can be printed out.

```
hypercube : 28
process 0 got 28
process 1 got 28
process 2 got 28
process 3 got 28
process 4 got 28
process 5 got 28
process 6 got 28
process 7 got 28
```

Below is the results from all implementations

```
gather    : 28
Allreduce : 28
leader    : 28
ring      : 28
hypercube : 28
```

Full code is on the following pages

```
void allReduce(){
    int rank, data, size, value = 0;
    MPI_Comm_rank(MCW, &rank);

    data = rank;
    MPI_Allreduce(&data, &value, 1, MPI_INT, MPI_SUM, MCW);

    //print one
    if(rank == 0){
        cout << "Allreduce : " << value << endl;
    }

    //print all
    cout << "process " << rank << " got " << value << endl;
}

void gather(){
    //initialize
    MPI_Barrier(MCW);
    int rank, data, size, value, result = 0;
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    int recvData[size];


    //gather rank from every process
    data = rank;
    MPI_Gather(&data,1,MPI_INT,recvData,1,MPI_INT,0,MCW);

    //add all of the gathered data together
    if(rank == 0){
        for(int i = 0; i < size; i++){
            value += recvData[i];
        }
        cout << "gather    : " << value << endl;
    }

    // //broadcast result to all
    data = value;
    MPI_Bcast(&data,1,MPI_INT,0,MCW);
    MPI_Barrier(MCW);

    //print from all
    cout << "process " << rank << " got " << data << endl;
}

void leader(){
    MPI_Barrier(MCW);
    int rank, data, size, value = 0;
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    if(rank == 0){
        //get messages from all processes and add them together
        for(int i = 0; i < size-1; i++){
            MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
            value += data;
        }

        //send result to other processes
        data = value;
        for(int i = 1; i < size; i++){
            MPI_Send(&data, 1, MPI_INT, i, 0, MCW);
        }

        //print result
        cout << "leader    : " << value << endl;

    } else {
        //if not rank 0 send rank to process 0
        data = rank;
        MPI_Send(&data, 1, MPI_INT, 0, 0, MCW);

        //recieve result
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
    }
    //print from all
    cout << "process " << rank << " got " << data << endl;
}
```

```cpp
void ring(){
    MPI_Barrier(MCW);
    int rank, size, data;
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    //start at rank 0 and send data to next process
    if(rank == 0){
        data = rank;
        MPI_Send(&data, 1, MPI_INT, (rank+1)%size, 0, MCW);
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);

        //after 0 has recieved a message from the last process, print the results
        cout << "ring      : " << data << endl;
    } else {
        //once you have recieved the data send to the next one
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
        data += rank;
        MPI_Send(&data, 1, MPI_INT, (rank+1)%size, 0, MCW);
    }

    //send result to all
    if(rank == 0){
        MPI_Send(&data, 1, MPI_INT, (rank+1)%size, 0, MCW);
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
    } else {
        //once you have recieved the data send to the next one
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
        MPI_Send(&data, 1, MPI_INT, (rank+1)%size, 0, MCW);
    }
    //print from all
    cout << "process " << rank << " got " << data << endl;
}

void hyperCube(){
    MPI_Barrier(MCW);
    int rank, data, size, value, dest, times = 0;
    unsigned int mask = 1;
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    value = rank;

    //add up across each axis
    for(int i = 0; i < log2(size); i++){
        dest = rank^(mask<<i);
        data = value;
        MPI_Send(&data, 1, MPI_INT, dest, 0, MCW);
        MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, MPI_STATUS_IGNORE);
        value += data;
        MPI_Barrier(MCW);
    }

    //print only one value
    if(rank == 0){
        cout << "hypercube : " << value << endl;
    }
    //print from all
    cout << "process " << rank << " got " << value << endl;
}

int main(int argc, char **argv){
    int size;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MCW, &size);


    //run all
    // gather();
    allReduce();
    // leader();
    // ring();
    // hyperCube();

    MPI_Finalize();
    return 0;
}
```