Stampede All Reduce Benchmarking

Adam Ross
adro4510@colorado.edu

Abstract

This report is set to examine timing performance of various allreduce algorithms including, an allreduce based on tree reduce and tree broadcast, a all to one and one to all allreduce, butterfly allreduce and the MPI implementation of allreduce. Each implementation besides the MPI implementation support two directions of bit transversal, high to low and low to high. I will be comparing increasing message payload sizes up to 1MB as well as 16 processes vs 256 processes split among 16 nodes of 16 processes.

1. Introduction

Passing messages in MPI has overhead performance costs associated for each message sent which can, if used poorly, severely impact the overall performance of a program. Therefor benchmarking various implementations and message sizes in allreduce operations to understand the magnitude and costs associated is valuable to learn parallel optimization. I will first be discussing some of the allreduce implementation details used in this lab, then I analyze data generated by these algorithms via the xsede computing cluster in the form of graphs and command line output.

2. Butterfly Allreduce Algorithm Overview

An implementation of the allreduce method is named the Butterfly Allreduce, which successfully aggregates the same data to all process via “butterfly” looking interprocess communication. This algorithm runs in log2(P) stages in which each node is always exchanging information with another node, i.e. no node is left idle. The flow of both high to low bit transversal and low to high bit transversal is detailed below in figures 2.1 and 2.2.

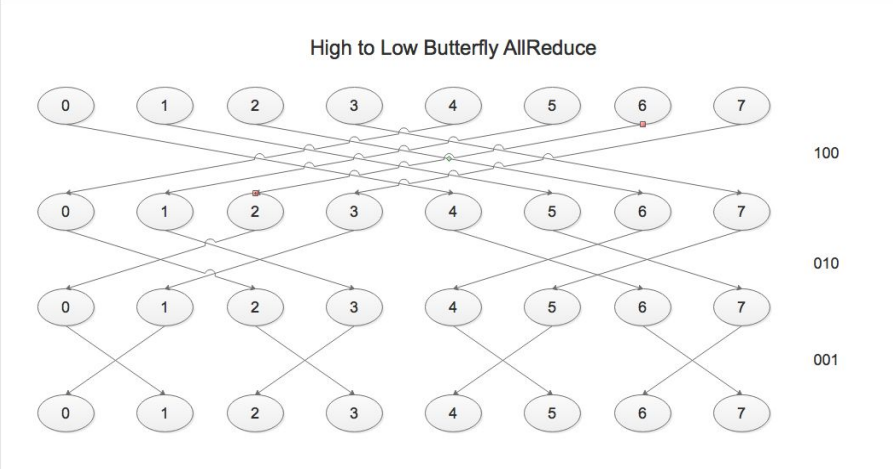


Figure 2.1 - High to low bit transversal butterfly allreduce pattern

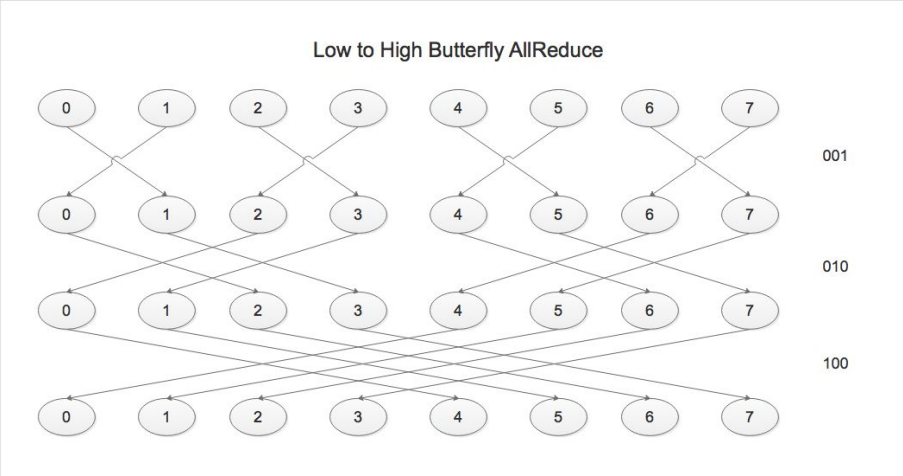


Figure 2.2 - Low to High bit transversal butterfly allreduce pattern

The High to low algorithm is accomplished via the scheduling defined by the following code.

```
current_mask = p;
for (int stage = 0; stage < log2(p); stage++) {
    current_mask = current_mask >> 1;
    dest_source = my_rank ^ current_mask;
    if (my_rank & current_mask) {
        MPI_Send();
        MPI_Recv();
    } else {
        MPI_Recv();
        MPI_Send();
    }
    for (int i = 0; i < count; i++) {
        buffer[i] += recv_buffer[i];
    }
}
```

A similar process is defined for the opposite bit transversal. Additionally you may find the implementations of the other allreduce operations in the code appended to this report.

3. Verification/ Procedure

Similar to Figures 2.1 and 2.2 I will be providing output from the Butterfly Allreduce operations to prove the correctness of my implementation. A buffer size of 1MB was used the output below.

```
mpiexec -np 8 RossAdam_HW3 -P 7 -c 36
My rank is 6 and the buffer contains 1.000000
My rank is 1 and the buffer contains 1.000000
My rank is 2 and the buffer contains 1.000000
My rank is 4 and the buffer contains 1.000000
My rank is 5 and the buffer contains 1.000000
```

```
My rank is 0 and the buffer contains 1.000000
My rank is 3 and the buffer contains 1.000000
My rank is 7 and the buffer contains 1.000000
-My rank is 1 - Stage is: 0 - Receiving from and then sending to 5
-My rank is 2 - Stage is: 0 - Receiving from and then sending to 6
+My rank is 4 - Stage is: 0 - Sending to then receiving from 0
+My rank is 5 - Stage is: 0 - Sending to then receiving from 1
+My rank is 6 - Stage is: 0 - Sending to then receiving from 2
-My rank is 0 - Stage is: 0 - Receiving from and then sending to 4
-My rank is 3 - Stage is: 0 - Receiving from and then sending to 7
+My rank is 7 - Stage is: 0 - Sending to then receiving from 3
-My rank is 0 - Stage is: 1 - Receiving from and then sending to 2
-My rank is 1 - Stage is: 1 - Receiving from and then sending to 3
+My rank is 2 - Stage is: 1 - Sending to then receiving from 0
+My rank is 6 - Stage is: 1 - Sending to then receiving from 4
-My rank is 4 - Stage is: 1 - Receiving from and then sending to 6
-My rank is 0 - Stage is: 2 - Receiving from and then sending to 1
-My rank is 4 - Stage is: 2 - Receiving from and then sending to 5
-My rank is 5 - Stage is: 1 - Receiving from and then sending to 7
-My rank is 6 - Stage is: 2 - Receiving from and then sending to 7
@ My rank is 0 and the total is 8.000000
+My rank is 1 - Stage is: 2 - Sending to then receiving from 0
@ My rank is 1 and the total is 8.000000
-My rank is 2 - Stage is: 2 - Receiving from and then sending to 3
@ My rank is 2 and the total is 8.000000
+My rank is 3 - Stage is: 1 - Sending to then receiving from 1
+My rank is 3 - Stage is: 2 - Sending to then receiving from 2
+My rank is 5 - Stage is: 2 - Sending to then receiving from 4
@ My rank is 5 and the total is 8.000000
+My rank is 7 - Stage is: 1 - Sending to then receiving from 5
+My rank is 7 - Stage is: 2 - Sending to then receiving from 6
@ My rank is 4 and the total is 8.000000
@ My rank is 6 and the total is 8.000000
@ My rank is 7 and the total is 8.000000
@ My rank is 3 and the total is 8.000000
```

Output 3.1 - High to low AllReduce output

```
mpiexec -np 8 RossAdam_HW3 -P 8 -c 36
My rank is 7 and the buffer contains 1.000000
My rank is 3 and the buffer contains 1.000000
My rank is 6 and the buffer contains 1.000000
My rank is 0 and the buffer contains 1.000000
My rank is 2 and the buffer contains 1.000000
My rank is 4 and the buffer contains 1.000000
My rank is 1 and the buffer contains 1.000000
My rank is 5 and the buffer contains 1.000000
My rank is 7 - Stage is: 0 - Sending to then receiving from 6
My rank is 3 - Stage is: 0 - Sending to then receiving from 2
My rank is 6 - Stage is: 0 - Receiving from and then sending to 7
My rank is 0 - Stage is: 0 - Receiving from and then sending to 1
```

```
My rank is 2 - Stage is: 0 - Receiving from and then sending to 3
My rank is 2 - Stage is: 1 - Sending to then receiving from 0
My rank is 4 - Stage is: 0 - Receiving from and then sending to 5
My rank is 3 - Stage is: 1 - Sending to then receiving from 1
My rank is 6 - Stage is: 1 - Sending to then receiving from 4
My rank is 7 - Stage is: 1 - Sending to then receiving from 5
My rank is 1 - Stage is: 0 - Sending to then receiving from 0
My rank is 5 - Stage is: 0 - Sending to then receiving from 4
My rank is 4 - Stage is: 1 - Receiving from and then sending to 6
My rank is 4 - Stage is: 2 - Sending to then receiving from 0
My rank is 0 - Stage is: 1 - Receiving from and then sending to 2
My rank is 0 - Stage is: 2 - Receiving from and then sending to 4
My rank is 1 - Stage is: 1 - Receiving from and then sending to 3
My rank is 1 - Stage is: 2 - Receiving from and then sending to 5
My rank is 5 - Stage is: 1 - Receiving from and then sending to 7
My rank is 5 - Stage is: 2 - Sending to then receiving from 1
My rank is 6 - Stage is: 2 - Sending to then receiving from 2
My rank is 2 - Stage is: 2 - Receiving from and then sending to 6
@ My rank is 0 and the total is 8.000000
@ My rank is 1 and the total is 8.000000
@ My rank is 2 and the total is 8.000000
My rank is 3 - Stage is: 2 - Receiving from and then sending to 7
@ My rank is 3 and the total is 8.000000
@ My rank is 4 and the total is 8.000000
@ My rank is 5 and the total is 8.000000
My rank is 7 - Stage is: 2 - Sending to then receiving from 3
@ My rank is 6 and the total is 8.000000
@ My rank is 7 and the total is 8.000000
```

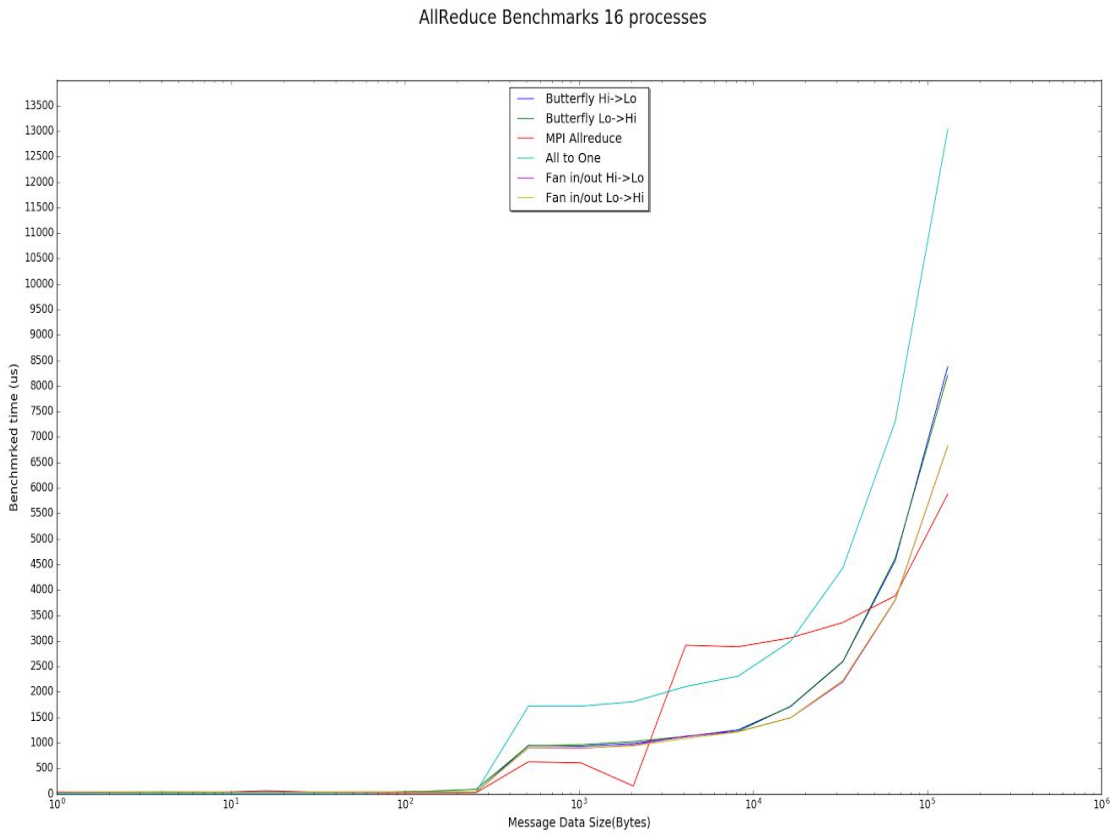
Output 3.2 - Low to High AllReduce output

4. All Reduce Performance Comparison Graph

As we could have guessed the Open_MPI implementation blows my written AllReduces away. We also would expect the all to one/one to all to perform the worst as it does in the graph. This data was taken with consideration to the MPI tick count, that is the smallest resolution the MPI timer can work with. Each measurement was at least an order of magnitude larger than the given tic count of the stampede machines. If a job was running too quickly for the resolution to keep accurate track of an average was taken from multiple runs.

Each program run generated output into a text file, then fed to a python script using matplotlib to plot the graphs below. We were asked to generate a %95 confidence interval within reasonable run time parameters. I accomplished this by increasing the number of times a program ran until the error margin percentage was below %5 or we had run the program 10 times. Each node generated slightly different results causing some to reach this threshold while others continued and blocked. This was solved by use of MPI_Allreduce to synchronize and average the error margin percentage. This was accomplished by the statement below.

```
while (cont_recv == 0 && n < 10) {}
…
MPI_Allreduce(&cont, &cont_recv, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
```



```c
/* RossAdam_HW4.c
 *
 * Butterfly Allreduce and benchmarking
 *
 * Broadcast and Reduce methods
 * with high to low and low to high
 * bit mask transversal.
 *
 * All_Reduce method compound from the above.
 * All_Reduce method using an all to one alogrithm.
 *
 * Input: -c <times to run programs> -d <size of data for program != 10> -P <program>
 * Output: none.
 *
 *
 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "mpi.h"
#include <getopt.h>
#include <stdlib.h>

#define ONE_MB_BUFFER_SIZE 131072

void My_BroadcastHL(double *buffer, int count, MPI_Comm comm);
void My_BroadcastLH(double *buffer, int count, MPI_Comm comm);
void My_ReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm);
void My_ReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm);
void My_Compound_All_ReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm)
;
void My_Compound_All_ReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm)
;
void My_Compound_All_ReduceHLLH(double *buffer, double *recv_buffer, int count, MPI_Comm com
m);
void My_Compound_All_ReduceLHHL(double *buffer, double *recv_buffer, int count, MPI_Comm com
m);
void My_All_Reduce(double *buffer, double *recv_buffer, int count, MPI_Comm comm);
void Butterfly_AllReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm);
void Butterfly_AllReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm);

void print_usage() {
    printf("Usage: -program or -p to specify which program to run. -data-size or -d to speci
fy data size in bytes.\n");
}

int         my_rank;       /* rank of process      */
int         p;             /* number of processes  */
int         ct;


main(int argc, char* argv[]) {
    int         option = 0;
    int         tag = 0;       /* tag for messages     */
    int         program = -1;  /* from argparse - number of program to run */
    int         data_size = 1; /* the number of byte of doubles to run on */
    double      result;
    ct = 1;

    while ((option = getopt(argc, argv,"P:d:c:")) != -1) {
        switch (option) {
             case 'P' : program = atoi(optarg);
                 break;
             case 'd' : data_size = atoi(optarg);
                 break;
             case 'c' : ct = atoi(optarg);
                 break;
             default: print_usage();
                 exit(1);
        }
    }

    /* Start up MPI */
    MPI_Init(&argc, &argv);

    /* Find out process rank  */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    double *sum_ptr = (double *)calloc(ONE_MB_BUFFER_SIZE, sizeof(double));
    double *recv_buffer = (double *)calloc(ONE_MB_BUFFER_SIZE, sizeof(double));

    if (program == 0 || program == 1) {
        if (my_rank == 0) {
            for(int i = 0; i < ONE_MB_BUFFER_SIZE; i++) {
                sum_ptr[i] = 1.0;
            }

        }
    } else {
        for(int i = 0; i < ONE_MB_BUFFER_SIZE; i++) {
            sum_ptr[i] = 1.0;
        }
    }

    if (my_rank == 0) {
        printf("Number of program %d, number of datasize %d\n", program, data_size);
        double tick = MPI_Wtick();
        printf("Resolution of MPI wtime: %1.20f\n", tick);
        printf("----------------------------------------\n");
    }

    int dat_size = 1;

    double x[10];
    int n = 0;
    double confidence_int = 0.0;
    double x_sum = 0.0;
    double mean = 0.0;
    double std_dev = 0.0;
    double marg_err = 0.0;
    double marg_perc = 100.0;
    int cont = 0;
    int cont_recv = 0;


    /* Comment or un comment to run specific programs */
    double finish;
    double start;

    start = MPI_Wtime();
    switch (program) {
        case 0 :
            My_BroadcastHL(sum_ptr, data_size, MPI_COMM_WORLD);
```

```
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_BroadcastHL(sum_ptr, data_size, MPI_COMM_WORLD);
        }
        break;
    case 1 :
        My_BroadcastLH(sum_ptr, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_BroadcastLH(sum_ptr, data_size, MPI_COMM_WORLD);
        }
        break;
    case 2 :
        My_ReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_ReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 3 :
        My_ReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        //for (int i = 0; i < ct; i++) {
            //My_ReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //}
        break;
    case 4 :
        My_Compound_All_ReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_Compound_All_ReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 5 :
        My_Compound_All_ReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_Compound_All_ReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 6 :
        My_All_Reduce(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            My_All_Reduce(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 7 :
        Butterfly_AllReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            Butterfly_AllReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 8 :
        Butterfly_AllReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            Butterfly_AllReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        }
        break;
    case 9 :
        MPI_Allreduce(sum_ptr, recv_buffer, data_size, MPI_DOUBLE, MPI_SUM, MPI_COMM_WOR
LD);
        MPI_Barrier(MPI_COMM_WORLD);
        start = MPI_Wtime();
        for (int i = 0; i < ct; i++) {
            MPI_Allreduce(sum_ptr, recv_buffer, data_size, MPI_DOUBLE, MPI_SUM, MPI_COMM
_WORLD);
        }
        break;
    case 10 :
        //My_Compound_All_ReduceHL(sum_ptr, recv_buffer, dat_size, MPI_COMM_WORLD);
        //My_Compound_All_ReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //My_Compound_All_ReduceHLLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //My_Compound_All_ReduceLHHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //My_All_Reduce(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //Butterfly_AllReduceHL(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        //Butterfly_AllReduceLH(sum_ptr, recv_buffer, data_size, MPI_COMM_WORLD);
        MPI_Allreduce(sum_ptr, recv_buffer, data_size, MPI_DOUBLE, MPI_SUM, MPI_COMM_WOR
LD);

        MPI_Barrier(MPI_COMM_WORLD);
        while(dat_size <= ONE_MB_BUFFER_SIZE) {
            n = 0;
            confidence_int = 0.0;
            x_sum = 0.0;
            mean = 0.0;
            std_dev = 0.0;
            marg_err = 0.0;
            marg_perc = 100.0;
            cont = 0;
            cont_recv = 0;
            while (cont_recv == 0 && n < 10) {
                MPI_Barrier(MPI_COMM_WORLD);
                start = MPI_Wtime();
                for (int i = 0; i < ct; i++) {
                    //My_Compound_All_ReduceHL(sum_ptr, recv_buffer, dat_size, MPI_COMM_
WORLD);
                    //My_Compound_All_ReduceLH(sum_ptr, recv_buffer, dat_size, MPI_COMM_
WORLD);
                    //My_Compound_All_ReduceHLLH(sum_ptr, recv_buffer, data_size, MPI_CO
MM_WORLD);
                    //My_Compound_All_ReduceLHHL(sum_ptr, recv_buffer, data_size, MPI_CO
MM_WORLD);
                    //My_All_Reduce(sum_ptr, recv_buffer, dat_size, MPI_COMM_WORLD);
                    //Butterfly_AllReduceHL(sum_ptr, recv_buffer, dat_size, MPI_COMM_WOR
LD);
                    //Butterfly_AllReduceLH(sum_ptr, recv_buffer, dat_size, MPI_COMM_WOR
LD);
                    MPI_Allreduce(sum_ptr, recv_buffer, dat_size, MPI_DOUBLE, MPI_SUM, M
PI_COMM_WORLD);
                }
                finish = MPI_Wtime();
                x[n] = (finish - start)/(double)ct;
                n++;
                if (n > 1) {
```

```
                    for (int i = 0; i < n; i++) {
                        x_sum += x[i];
                    }
                    mean = x_sum / n;
                    x_sum = 0.0;
                    for (int i = 0; i < n; i++) {
                        x_sum += pow(x[i] - mean, 2);
                    }
                    std_dev = sqrt(x_sum / n);
                    marg_err = 1.96 * (std_dev / sqrt(n));
                    marg_perc = (marg_err / mean) * 100;
                }
                if (marg_perc < 5.0) {
                    cont = 1;
                }
                MPI_Allreduce(&cont, &cont_recv, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
                //printf("%d %d\n", my_rank, cont_recv);
            }
            if (my_rank == 0) {
                printf("%d\t%d\t%d\t%f\t%1.10f\t%1.10f\t%f\t%d\t%d\n", program, 0, (dat_
size * sizeof(double)), mean, std_dev, marg_err, marg_perc, n, ct);
            }
            dat_size = dat_size * 2;
            if (ct > 1) {
                ct -= 1;
            }
        }
        break;
    default:
        exit(1);
}
finish = MPI_Wtime();
// program, data-iteration, packet size, mean, std-dev, marg_err

//printf("+ My rank is %d\t buffer is %f\t recv_buffer is %f\n", my_rank, sum_ptr[0], re
cv_buffer[0]);
if (my_rank == 0 && program != 10) {
    //printf("%d\t%d\t%d\t%f\t%1.10f\t%1.10f\t%f", program, 0, (data_size * sizeof(doubl
e)), mean, std_dev, marg_err, marg_perc);
    printf("%d\t%1.20f \n", my_rank,  (finish-start)/(double)ct);
}

/* Shut down MPI */
MPI_Finalize();
} /* main */


// low to high bit transversal
// 001 -> 010 -> 100
//  1      2      4

// hight to low bit transversal
// 100 -> 010 -> 001
//  4      2      1

void My_BroadcastHL(double *buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest;
    int source;
    MPI_Status status;
    //printf("+ My rank is %d and the buffer contains %f\n", my_rank, buffer[0]);
    current_mask = p;
    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = current_mask >> 1;
        dest = my_rank | current_mask;
        if ((my_rank % current_mask == 0) && (my_rank != dest)) {
            //printf("My rank is %d - Stage is: %d - Sending to %d\n", my_rank, stage, dest)
;
            //send
            MPI_Send(buffer, count, MPI_DOUBLE, dest, 0, comm);
        } else if ((my_rank + current_mask) % (2 * current_mask) == 0) {
            //recv
            source = my_rank ^ current_mask;
            //printf("My rank is %d - Stage is: %d - Receiving from %d\n", my_rank, stage, s
ource);
            MPI_Recv(buffer, count, MPI_DOUBLE, source, 0, comm, &status);
        }
    }
    //printf("+ My rank is %d and the buffer contained %f\n", my_rank, buffer[0]);
}

void My_BroadcastLH(double *buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest;
    int source;
    MPI_Status status;
    //printf("+ My rank is %d and the buffer contains %f\n", my_rank, buffer[0]);
    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = 1 << stage;
        if (my_rank < current_mask) {
            dest = my_rank | current_mask;
            //printf("My rank is %d - Stage is: %d - Sending to %d\n", my_rank, stage, dest)
;
            // send
            MPI_Send(buffer, count, MPI_DOUBLE, dest, 0, comm);
        } else if ((my_rank >= current_mask) && (my_rank < (current_mask * 2))) {

            // recv
            source = my_rank - current_mask;
            //printf("My rank is %d - Stage is: %d - Receiving from %d\n", my_rank, stage, s
ource);
            MPI_Recv(buffer, count, MPI_DOUBLE, source, 0, comm, &status);
        }
    }
    //printf("+ My rank is %d and the buffer contained %f\n", my_rank, buffer[0]);
}

void My_ReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest;
    int source;
    MPI_Status  status;
    current_mask = p;
    //printf("+ My rank is %d and the buffer contains %f\n", my_rank, *buffer);
    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = current_mask >> 1;
        if ((my_rank >= current_mask) && (my_rank < current_mask * 2)) {
            dest = my_rank ^ current_mask;
            //printf("My rank is %d - Stage is: %d - Sending to %d\n", my_rank, stage, dest)
;
            // send
            MPI_Send(buffer, count, MPI_DOUBLE, dest, 0, comm);
        } else if (my_rank < current_mask) {
            // recv
            source = my_rank + current_mask;
            //printf("My rank is %d - Stage is: %d - Receiving from %d\n", my_rank, stage, s
```

```
ource);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, source, 0, comm, &status);
            for (int i = 0; i < count; i++) {
                buffer[i] += recv_buffer[i];
            }
        }
    }
}

void My_ReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest;
    int source;
    MPI_Status  status;

    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = 1 << stage;
        if ((my_rank - current_mask) % ((stage + 1) * 2) == 0) {
            dest = my_rank ^ current_mask;
            printf("My rank is %d - Stage is: %d - Sending to %d\n", my_rank, stage, dest);
            // send
            MPI_Send(buffer, count, MPI_DOUBLE, dest, 0, comm);
        } else if ((my_rank % (current_mask * 2)) == 0) {
            // recv
            source = my_rank | current_mask;
            printf("My rank is %d - Stage is: %d - Receiving from %d\n", my_rank, stage, sou
rce);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, source, 0, comm, &status);
            for (int i = 0; i < count; i++) {
                buffer[i] += recv_buffer[i];
            }
        }
    }
    printf("+ My rank is %d and the total is %f\n", my_rank, buffer[0]);
}

void My_Compound_All_ReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm)
 {
    My_ReduceHL(buffer, recv_buffer, count, comm);
    //printf("my rank is %d, buffer contains %f", my_rank, buffer[0]);
    My_BroadcastHL(buffer, count, comm);
    //printf("- MY rank is %d, and the number I have is %f\n", my_rank, a);
}

void My_Compound_All_ReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm)
 {
    My_ReduceLH(buffer, recv_buffer, count, comm);
    //printf("my rank is %d, buffer contains %f", my_rank, buffer[0]);
    My_BroadcastLH(buffer, count, comm);
    //printf("- MY rank is %d, and the number I have is %f\n", my_rank, a);
}

void My_Compound_All_ReduceHLLH(double *buffer, double *recv_buffer, int count, MPI_Comm com
m) {
    My_ReduceHL(buffer, recv_buffer, count, comm);
    //printf("my rank is %d, buffer contains %f", my_rank, buffer[0]);
    My_BroadcastLH(buffer, count, comm);
    //printf("- MY rank is %d, and the number I have is %f\n", my_rank, a);
}

void My_Compound_All_ReduceLHHL(double *buffer, double *recv_buffer, int count, MPI_Comm com
m) {
    My_ReduceLH(buffer, recv_buffer, count, comm);
    //printf("my rank is %d, buffer contains %f", my_rank, buffer[0]);
    My_BroadcastHL(buffer, count, comm);
    //printf("- MY rank is %d, and the number I have is %f\n", my_rank, a);
}

void My_All_Reduce(double *buffer, double *recv_buffer, int count, MPI_Comm comm) {
    int dest;
    int source;
    MPI_Status  status;

    // Reduce
    if (my_rank != 0) {
        dest = 0;
        //printf("My rank is %d, sending to %d\n", my_rank, dest);
        MPI_Send(buffer, count, MPI_DOUBLE, dest, 0, comm);
    } else {
        for (int i = 1; i < p; i++) {
            source = i;
            //printf("My rank is %d, receiving from %d\n", my_rank, source);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, source, 0, comm, &status);
            for (int i = 0; i < count; i++) {
                buffer[i] += recv_buffer[i];
            }
        }
        //printf("- My rank is %d and the reduced value I have is %f\n",my_rank, total);
    }

    // Broadcast
    if (my_rank != 0) {
        source = 0;
        //printf("My rank is %d, receiving from %d\n", my_rank, source);
        MPI_Recv(recv_buffer, count, MPI_DOUBLE, source, 0, comm, &status);
        for (int i = 0; i < count; i++) {
            buffer[i] = recv_buffer[i];
        }
    } else {
        for (int i = 1; i < p; i++) {
            //printf("My rank is %d, sending to %d\n", my_rank, i);
            MPI_Send(buffer, count, MPI_DOUBLE, i, 0, comm);
        }
    }

    //printf("+ My rank is %d, and the value I have is %f\n", my_rank, total);
}

void Butterfly_AllReduceHL(double *buffer, double *recv_buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest_source;
    MPI_Status  status;
    //printf("My rank is %d and the buffer contains %f\n", my_rank, *buffer);

    current_mask = p;
    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = current_mask >> 1;
        dest_source = my_rank ^ current_mask;
        if (my_rank & current_mask) {
            //printf("+My rank is %d - Stage is: %d - Sending to then receiving from %d\n",
my_rank, stage, dest_source);
            MPI_Send(buffer, count, MPI_DOUBLE, dest_source, 0, comm);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, dest_source, 0, comm, &status);
        } else {
            //printf("-My rank is %d - Stage is: %d - Receiving from and then sending to %d\
n", my_rank, stage, dest_source);
```

```
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, dest_source, 0, comm, &status);
            MPI_Send(buffer, count, MPI_DOUBLE, dest_source, 0, comm);
        }
        for (int i = 0; i < count; i++) {
            buffer[i] += recv_buffer[i];
        }
    }
    //printf("@ My rank is %d and the total is %f \n", my_rank, recv_buffer[0]);
}

void Butterfly_AllReduceLH(double *buffer, double *recv_buffer, int count, MPI_Comm comm) {
    int current_mask;
    int dest_source;
    MPI_Status  status;
    //printf("My rank is %d and the buffer contains %f\n", my_rank, *buffer);

    for (int stage = 0; stage < log2(p); stage++) {
        current_mask = 1 << stage;
        dest_source = my_rank ^ current_mask;
        if (my_rank & current_mask) {
            //printf("My rank is %d - Stage is: %d - Sending to then receiving from %d\n", m
y_rank, stage, dest_source);
            MPI_Send(buffer, count, MPI_DOUBLE, dest_source, 0, comm);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, dest_source, 0, comm, &status);
        } else {
            //printf("My rank is %d - Stage is: %d - Receiving from and then sending to %d\n
", my_rank, stage, dest_source);
            MPI_Recv(recv_buffer, count, MPI_DOUBLE, dest_source, 0, comm, &status);
            MPI_Send(buffer, count, MPI_DOUBLE, dest_source, 0, comm);
        }
        for (int i = 0; i < count; i++) {
            buffer[i] += recv_buffer[i];
        }
    }
    //printf("@ My rank is %d and the total is %f \n", my_rank, recv_buffer[0]);
}
```

```
#!/usr/bin/python

import matplotlib.pyplot as plt
import matplotlib.ticker as mtick
import numpy as np

name_map = {'output_BFHL' : 'Butterfly Hi->Lo',
'output_BFLH' : 'Butterfly Lo->Hi',
'output_MPI' : 'MPI Allreduce',
'output_MY' : 'All to One',
'output_COMP_HL' : 'Fan in/out Hi->Lo',
'output_COMP_LH' : 'Fan in/out Lo->Hi'}

#output_file_names = np.genfromtxt('output_files',delimiter=" ")
with open('output_files') as f:
    content = f.readlines()
content = [x.strip() for x in content]

i = 0
mean_max = 0.0

byte_size = []
mean = []
std = []
err_marg = []
err_marg_perc = []

for filename in content:
    output = np.genfromtxt(filename, delimiter="\t", unpack=True, skip_header=3)
    byte_size.append([x/8 for x in output[2]])
    mean.append([x * 1000000 for x in output[3]])
    std.append([x * 1000000 for x in output[4]])
    err_marg.append([x * 1000000 for x in output[5]])
    err_marg_perc.append(output[6])

#plt.errorbar()
fig = plt.figure()
fig.suptitle('AllReduce Benchmarks 16 processes', fontsize=20)
ax = fig.add_subplot(1,1,1)
for it in range(0, len(mean)):
    ax.plot(byte_size[it], mean[it], label=name_map[content[it]])
    if max(mean[it]) > mean_max:
        mean_max = max(mean[it])
legend = ax.legend(loc='upper center', shadow=True)
ax.set_yticks(np.arange(0, mean_max+500, 500))
ax.set_xscale("log", nonposy='clip')
#ax.yaxis.set_major_formatter(mtick.FormatStrFormatter('%.2e'))
plt.xlabel('Message Data Size(Bytes)', fontsize=14)
plt.ylabel('Benchmrked time (us)', fontsize=14)
plt.show()
```