# HW 7
# Global Sum

Philip Nelson

2018 October 12

## Introduction

The purpose of this assignment is to write an MPI program that performs a global sum. I have implemented several different variants. The first takes advantage of the cube network communication. Second uses ring communication. Third is a naive master slave method. Fourth uses the building MPI_Allgather function. I made each iteration sleep for one tenth of a second in order to simulate some kind of work being executed.

## Code

```
1  #include "random.hpp"
2  #include <algorithm>
3  #include <iomanip>
4  #include <iostream>
5  #include <mpi.h>
6  #include <unistd.h>
7  #include <vector>
8
9  #define slep 100000
10
11 void print1per(int data, std::string title = "")
12 {
13   int rank;
14   int word_size;
15
16   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
17   MPI_Comm_size(MPI_COMM_WORLD, &word_size);
18
19   if (0 == rank)
20   {
21     int* dArray = new int[word_size];
```

```
22      MPI_Gather(&data, 1, MPI_INT, dArray, 1, MPI_INT, 0, MPI_COMM_WORLD
           );
23
24      std::cout << title << '\n';
25      for (int i = 0; i < word_size; ++i)
26      {
27        std::cout << std::setw(5) << i << std::setw(5) << dArray[i] << "\
             n";
28      }
29      std::cout << std::endl;
30    }
31    else
32    {
33      MPI_Gather(&data, 1, MPI_INT, nullptr, 1, MPI_INT, 0,
           MPI_COMM_WORLD);
34    }
35  }
36
37  int cube(int c, int sendData, int rank)
38  {
39    int recvData;
40    auto dest = rank ^ (1 << c);
41
42    MPI_Send(&sendData, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
43    MPI_Recv(&recvData, 1, MPI_INT, dest, 0, MPI_COMM_WORLD,
         MPI_STATUS_IGNORE);
44
45    return recvData;
46  }
47
48  int ring(int dir, int sendData, int rank, int world_size)
49  {
50    int recvData;
51    auto dest = (rank + 1 * dir) % world_size;
52    auto src = (rank - 1 * dir) % world_size;
53
54    MPI_Send(&sendData, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
55    MPI_Recv(&recvData, 1, MPI_INT, src, 0, MPI_COMM_WORLD,
         MPI_STATUS_IGNORE);
56
57    return recvData;
58  }
59
60  void cubeSum(int num, int rank, int world_size)
61  {
62    int log2n = log2(world_size);
```

```
63    for (auto i = 0; i < log2n; ++i)
64    {
65      num += cube(i, num, rank);
66      usleep(slep);
67    }
68    print1per(num, "cube sum");
69  }
70
71  void ringSum(int num, int rank, int world_size)
72  {
73    int next, prev = num;
74
75    for (auto i = 0; i < world_size - 1; ++i)
76    {
77      next = ring(1, prev, rank, world_size);
78      num += next;
79      prev = next;
80      usleep(slep);
81    }
82    print1per(num, "ring sum");
83  }
84
85  void masterSlaveSum(int num, int rank, int world_size)
86  {
87    if (0 == rank)
88    {
89      int recvData;
90      for (auto i = 1; i < world_size; ++i)
91      {
92        MPI_Recv(&recvData,
93                 1,
94                 MPI_INT,
95                 MPI_ANY_SOURCE,
96                 0,
97                 MPI_COMM_WORLD,
98                 MPI_STATUS_IGNORE);
99        num += recvData;
100       usleep(slep);
101     }
102     std::cout << "master slave sum\n    0   " << num << "\n\n";
103   }
104   else
105   {
106     MPI_Send(&num, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
107   }
108 }
```

```
109
110 void mpiAllReduce(int num)
111 {
112   MPI_Allreduce(&num, &num, 1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
113   usleep(slep);
114   print1per(num, "all reduce");
115 }
116
117 int main(int argc, char** argv)
118 {
119   MPI_Init(&argc, &argv);
120
121   int rank, world_size;
122   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
123   MPI_Comm_size(MPI_COMM_WORLD, &world_size);
124
125   if (0 != (world_size & (world_size - 1)))
126   {
127     if (rank == 0)
128     {
129       std::cerr << "There must be a power of 2 number of threads\n";
130     }
131
132     MPI_Finalize();
133     exit(EXIT_SUCCESS);
134   }
135
136   int num;
137   if (0 == rank)
138   {
139     std::vector<int> data(world_size);
140     random_double_fill(begin(data), end(data), 0, 10);
141     MPI_Scatter(data.data(), 1, MPI_INT, &num, 1, MPI_INT, 0,
          MPI_COMM_WORLD);
142     print1per(num, "original data");
143   }
144   else
145   {
146     MPI_Scatter(nullptr, 1, MPI_INT, &num, 1, MPI_INT, 0,
          MPI_COMM_WORLD);
147     print1per(num);
148   }
149
150   auto t1 = MPI_Wtime();
151   cubeSum(num, rank, world_size);
152   auto t2 = MPI_Wtime();
```

```
153   ringSum(num, rank, world_size);
154   auto t3 = MPI_Wtime();
155   masterSlaveSum(num, rank, world_size);
156   auto t4 = MPI_Wtime();
157   mpiAllReduce(num);
158   auto t5 = MPI_Wtime();
159
160   if (0 == rank)
161     std::cout << "cube: " << t2 - t1 << "\nring: " << t3 - t2
162               << "\nmaster slave: " << t4 - t3 << "\nall reduce: " <<
                     t5 - t4
163               << "\n";
164
165   MPI_Finalize();
166
167   return (EXIT_SUCCESS);
168 }
```

## Output

```
# mpic++ -O3 main.cpp -o release.out

# mpiexec -n 8 --oversubscribe release.out

original data
    0    5
    1    3
    2    0
    3    8
    4    4
    5    8
    6    3
    7    9

cube sum
    0   40
    1   40
    2   40
    3   40
    4   40
    5   40
    6   40
    7   40

ring sum
    0   40
    1   40
    2   40
    3   40
    4   40
    5   40
    6   40
    7   40

master slave sum
    0   40

all reduce
    0   40
    1   40
    2   40
    3   40
    4   40
```

```
    5   40
    6   40
    7   40

cube: 0.305294
ring: 0.708875
master slave: 0.701991
all reduce: 0.101125
```

## Findings

The cube sum was the best performing of my own sorting functions which is unsurprising since it only needs to compute $\log_2 n$ sums where $n$ is the world size. Therefore it is about 43% faster. The ring sum was as performant as the master slave sum however only the master process is left with the sum. If you want all processes to have the global sum, the ring sum would be a better option. Getting a good time estimate for the reduce / all reduce isn't possible using this setup but I image it is the best option for performing global sums.