Project 1

CSCI 475 – Parallel Computing

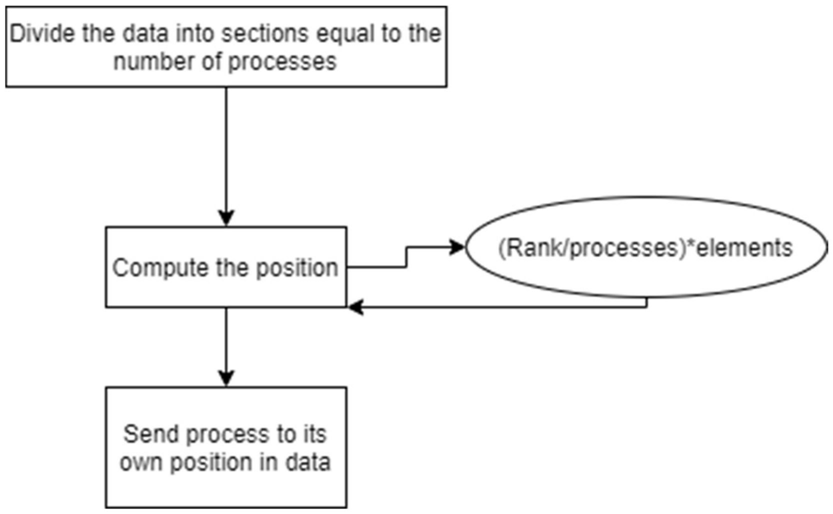
Saint Cloud State University

Mark Christenson

1. FindMin.c

Observations

The minimum for all cases is 500. This program finds the minimum for 2, 4, 8, and 16 processes, but assigning both 64 processes and a single process break this program. The program partitions the data based on the number of processes and each process seeks to their assigned position in the data file. After the seek, then the processes read the data for a number of times equal to the size of each partition.


Divide the data into sections equal to the number of processes
Compute the position
(Rank/processes)*elements
Send process to its own position in data

Output

```
[oz5808sg@csci4 Project 1]$ mpicc -std=c99 -o FindMin.out  FindMin.c
------------------------------------------------------------------------------------------------------------------------------------
[oz5808sg@csci4 Project 1]$ mpiexec -n 1 ./FindMin.out
Process 0 out of 1 started
0 positioned at 0

0 final number is 595
0: 595
Best is 595
------------------------------------------------------------------------------------------------------------------------------------
[oz5808sg@csci4 Project 1]$ mpiexec ./FindMin.out
Process 2 out of 4 started
2 positioned at 20000

2 final number is 500
Process 0 out of 4 started
```

```
0 positioned at 0

0 final number is 595
Process 1 out of 4 started
1 positioned at 10000

1 final number is 553
Process 3 out of 4 started
3 positioned at 30000

3 final number is 552
0: 595
1: 553
2: 500
3: 552
Best is 500
```

```
[oz5808sg@csci4 Project 1]$ mpiexec -n 16 ./FindMin.out
Process 14 out of 16 started
14 positioned at 35000

14 final number is 543
Process 11 out of 16 started
11 positioned at 27500

11 final number is 527
Process 8 out of 16 started
8 positioned at 20000

8 final number is 500
Process 2 out of 16 started
2 positioned at 5000

2 final number is 564
Process 4 out of 16 started
4 positioned at 10000

4 final number is 553
Process 9 out of 16 started
9 positioned at 22500

9 final number is 502
Process 10 out of 16 started
10 positioned at 25000
```

```
10 final number is 582
Process 3 out of 16 started
3 positioned at 7500

3 final number is 548
Process 1 out of 16 started
1 positioned at 2500

1 final number is 512
Process 6 out of 16 started
6 positioned at 15000

6 final number is 553
Process 13 out of 16 started
13 positioned at 32500

13 final number is 540
Process 5 out of 16 started
5 positioned at 12500

5 final number is 526
Process 7 out of 16 started
7 positioned at 17500

7 final number is 551
Process 15 out of 16 started
15 positioned at 37500

15 final number is 526
Process 12 out of 16 started
12 positioned at 30000

12 final number is 552
Process 0 out of 16 started
0 positioned at 0

0 final number is 595
0: 595
1: 512
2: 564
3: 548
4: 553
5: 526
6: 553
7: 551
```

```
8: 500
9: 502
10: 582
11: 527
12: 552
13: 540
14: 543
15: 526
Best is 500
```

```
[oz5808sg@csci4 Project 1]$ mpiexec -n 64 ./FindMin.out
Process 0 out of 64 started
0 positioned at 0

0 final number is 595
Process 11 out of 64 started
11 positioned at 6875
          .
          .
          .
54 final number is 1
0: 595
1: 45
2: 2
3: 1000
4: 512
5: 70
6: 8
7: 1000
8: 564
9: 21
10: 2
11: 1000
12: 548
13: 30
14: 2
15: 1000
16: 553
17: 79
18: 5
19: 1000
20: 526
21: 75
22: 7
23: 1000
```

```
24: 553
25: 13
26: 6
27: 1000
28: 551
29: 61
30: 7
31: 1000
32: 500
33: 41
34: 4
35: 1000
36: 502
37: 5
38: 9
39: 1000
40: 582
41: 72
42: 9
43: 1000
44: 527
45: 81
46: 8
47: 1000
48: 552
49: 6
50: 4
51: 1000
52: 540
53: 92
54: 1
55: 1000
56: 543
57: 60
58: 9
59: 1000
60: 526
61: 36
62: 3
63: 1000
Best is 1
```

Source

```c
/*
FindMin.c by Mark Christenson finds the minimum number in a set of numbers saved as Data.csv
*/

#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"


int main(int argc, char *argv[])
{
    int npes, myrank;
    int totalElements = 10000;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&npes);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

    printf("Process %d out of %d started\n", myrank, npes);

  //Open file
  FILE *data;
  data = fopen("Data.csv", "r");
  //Compute my position = (my rank * total data elements) / number of processes
  int myPosition = 4 * (myrank * totalElements)/npes;

  printf("%d positioned at %d\n", myrank, myPosition);
  //Go to my position
  fseek(data, myPosition, 0);
  char currentChar = getc(data);
  //printf("%d found %c", myrank, currentChar);
  //consider trying to find the previous comma if my rank != 0
  //read numbers
  char str[] = "";
  int best = 1000;
  for(int i = 0; i < 4*(totalElements/npes); i++){
    int tBest = 0;
    while(currentChar!=',' && currentChar!=EOF){
      tBest = 10 * tBest + (currentChar - '0');
      //printf("%c", currentChar);
      currentChar = getc(data);
    }
    //printf("\n%d current number is %d", myrank, tBest);
    if(best > tBest && tBest > 0){best = tBest;}
  }
  printf("\n%d final number is %d\n", myrank, best);

  int size, grp_size;
  int *buf;

  MPI_Barrier(MPI_COMM_WORLD);

```

```c
      if (myrank == 0){
        MPI_Comm_size(MPI_COMM_WORLD, &grp_size);
        size = grp_size*sizeof(int);
        buf = (int *)malloc(size);
      }

      int mBest[npes];
      mBest[0] = best;
      MPI_Gather(mBest, 1, MPI_INT, buf, 1, MPI_INT, 0, MPI_COMM_WORLD);
      if(myrank == 0){
        for (size_t i = 0; i < npes; i++) {
          printf("%d: %d\n", i, buf[i]);
          if(best>buf[i]){best=buf[i];}
        }
        printf("Best is %d\n", best);
      }
      /*
      //Send best to 0
      int mBest[npes];
      mBest[myrank] = best;
      if(myrank > 0){
        printf("%d sending message\n", myrank);
        MPI_Send(mBest, 1, MPI_INT, 0, myrank, MPI_COMM_WORLD);
        printf("%d sent message\n", myrank);
      }
      //0 output best
      else if(myrank == 0){
        int inbox[npes+1];
        MPI_Status status[npes+1];
        inbox[0] = best;
        for (size_t i = 1; i < npes; i++) {
          printf("waiting for messages\n");
          MPI_Gather(inbox, npes, MPI_INT, buf, npes, MPI_INT, 0,
MPI_COMM_WORLD);
          if(inbox[0]>inbox[i]){inbox[0]=inbox[i];}
          printf("current inbox best: %d\n", inbox[0]);
        }
        printf("final inbox best: %d\n", inbox[0]);
      }
      */

        MPI_Finalize();

      //close(data);
        return 0;
    }//end FindMin.c
```