PROJECT 01 ***alternate***

RING LEADER ELECTION WITH MPI (Message Passing Interface)

DAVID NGUYEN

D_NGUYEN@CSU.FULLERTON.EDU

CALIFORNIA STATE UNIVERSITY, FULLERTON

CPSC 479

HIGH PERFORMANCE COMPUTING

DOINA BEIN

Table of Contents

Abstract

Project 1 of California State University, Fullerton's, CPSC-479 High Performance Computing, named *Introduction to HPC – Variant of Leader Election on a Ring Topology*, nicknamed by David Nguyen, *Parallel Ring Leader Election*.

This is the alternative extra credit problem

Project goal is to implement an algorithm to select a leader on a ring topology based on processe s' exclusive randomly generated computed value compared to its neighbors' values. Using MPI (Message Passing Interface), randomly generating values and neighbor comparisons can be done i n parallel to speed up the processing.

This project showcases one of many tasks that can be achieved using Message Passing Interface to speed up minor tasks and remove overheads that would normally appear in sequential processing.

Pseudocode

Main

```
int rank, size, temp, tempComp, leftNeighbor, rightNeighbor;
// rank and size for MPI values
// temp is to hold each process's randomly generated value
// tempComp is to hold the computed value
// leftNeighbor to hold left neighbor's value
// rightNeighbor to hold right neighbor's value

<Initiate MPI>

if size is less than 6 or greater than 20:
     exit program
else
     Generate random integer and store into temp
     Store computed value into tempComp

     Announce computed value.
     Send computed value to left neighbor and right neighbor.
     Receive computed value from left neighbor and right
neighbor.

     Announce your computed value with received neighbor's
values.
     If tempComp is bigger than leftNeighbor and rightNeighbor,
     Announce current rank as leader of immediate neighbors.

<End MPI block with MPI_Finalize()>
```

randomInt()

```
int randTmp = generate random integer between 0 – 89, then add
10
if randTmp is negative:
     multiply randTmp by -1 to make positive

return randTmp;
```

int concat(int gen, int rank)

```
char s1[5] = "1" // size of 5 because largest concatenation is
1XXYY
char s2[2]; // size of 2 because largest input is double digit
char s3[2];

convert gen into string and store into s2

if(rank is less than 10)
     convert rank into string with a padded 0 and store into s3
else
     convert rank into string and store into s3

concatenate s1, which contains "1", with s2
concatenate s1, which now contains "1" + contents of s2, with s3

return s1 converted back to integer
```

How to run program

To run program:

1. In terminal, change to directory with file **ring_election.c**.
2. Run `**mpicc ring_election.c**` to build the program.
3. Run `**mpirun -n <# of processes wished> ./a.out**` to run program
    a. # of processes must be between [6, 20]
    b. a.out is the default compiled name. If you named the compiled name differently, replace a.out with that name.
4. Success!

Refer to follow pages for screenshots of example outputs.

Screenshots

Credit Proof

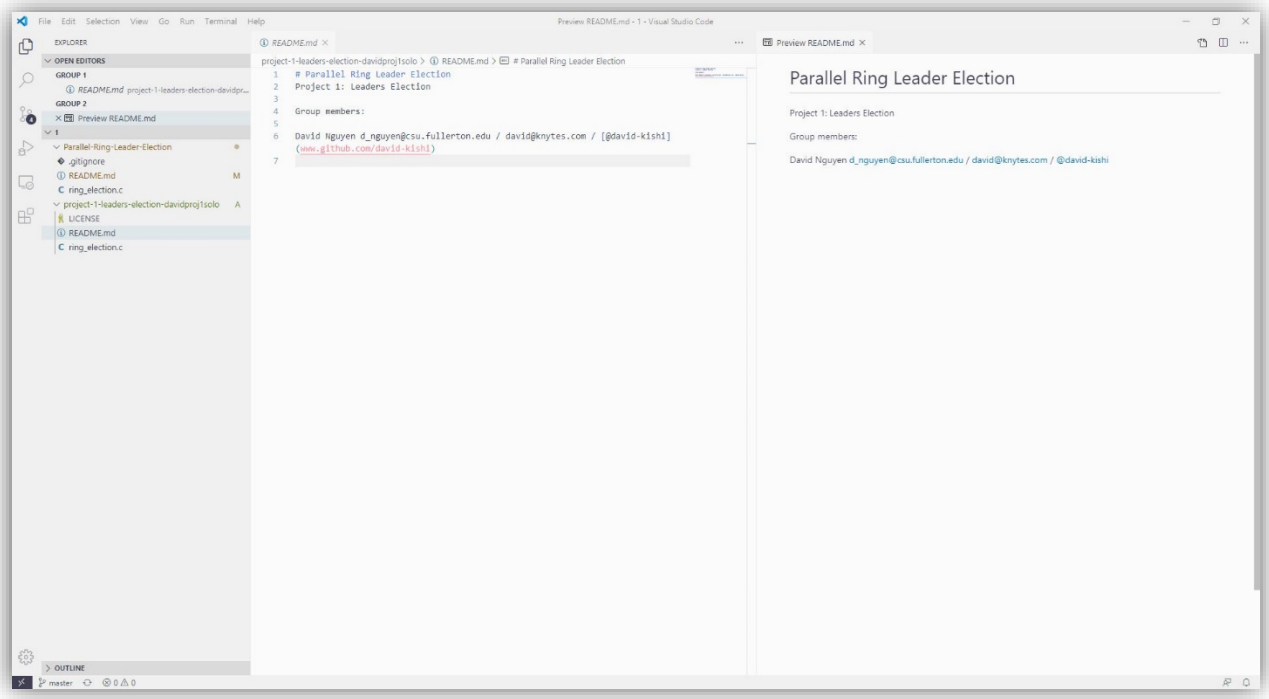

Figure 1. Credits

Example Output

```
[d_nguyen@titanv alternative_problem]$ mpirun -n 6 ./a.out
Process 2 generated random value 84, computed value of 18402
Process 0 generated random value 70, computed value of 17000
Process 5 generated random value 23, computed value of 12305
Process 1 generated random value 83, computed value of 18301
Process 1, My Value: 18301, Left Neighbor: 17000, Right Neighbor: 18402
Process 0, My Value: 17000, Left Neighbor: 12305, Right Neighbor: 18301
Process 4 generated random value 29, computed value of 12904
Process 3 generated random value 21, computed value of 12103
Process 3, My Value: 12103, Left Neighbor: 18402, Right Neighbor: 12904
Process 4, My Value: 12904, Left Neighbor: 12103, Right Neighbor: 12305
I, process 4, am the leader of my immediate neighbors.
Process 2, My Value: 18402, Left Neighbor: 18301, Right Neighbor: 12103
I, process 2, am the leader of my immediate neighbors.
Process 5, My Value: 12305, Left Neighbor: 12904, Right Neighbor: 17000
```

Figure 2. Example output with 6 processes