**PROJECT 01 *alternate***

**RING LEADER ELECTION WITH MPI (Message Passing Interface)**

DAVID NGUYEN

[D\_NGUYEN@CSU.FULLERTON.EDU](mailto:D_NGUYEN@CSU.FULLERTON.EDU)

CALIFORNIA STATE UNIVERSITY, FULLERTON

CPSC 479

HIGH PERFORMANCE COMPUTING

DOINA BEIN

Table of Contents

[**Abstract** 1](#_Toc38163075)

[**Pseudocode** 2](#_Toc38163076)

[**Main** 2](#_Toc38163077)

[**randomInt()** 3](#_Toc38163078)

[**int concat(int gen, int rank)** 3](#_Toc38163079)

[**How to run program** 4](#_Toc38163080)

[**Screenshots** 5](#_Toc38163081)

[**Credit Proof** 5](#_Toc38163082)

[**Example Output** 5](#_Toc38163083)

# **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 processes’ exclusive randomly generated computed value compared to its neighbors' values. Using MPI (Message Passing Interface), randomly generating values and neighbor comparisons can be done in 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
   1. # of processes must be between [6, 20]
   2. 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**

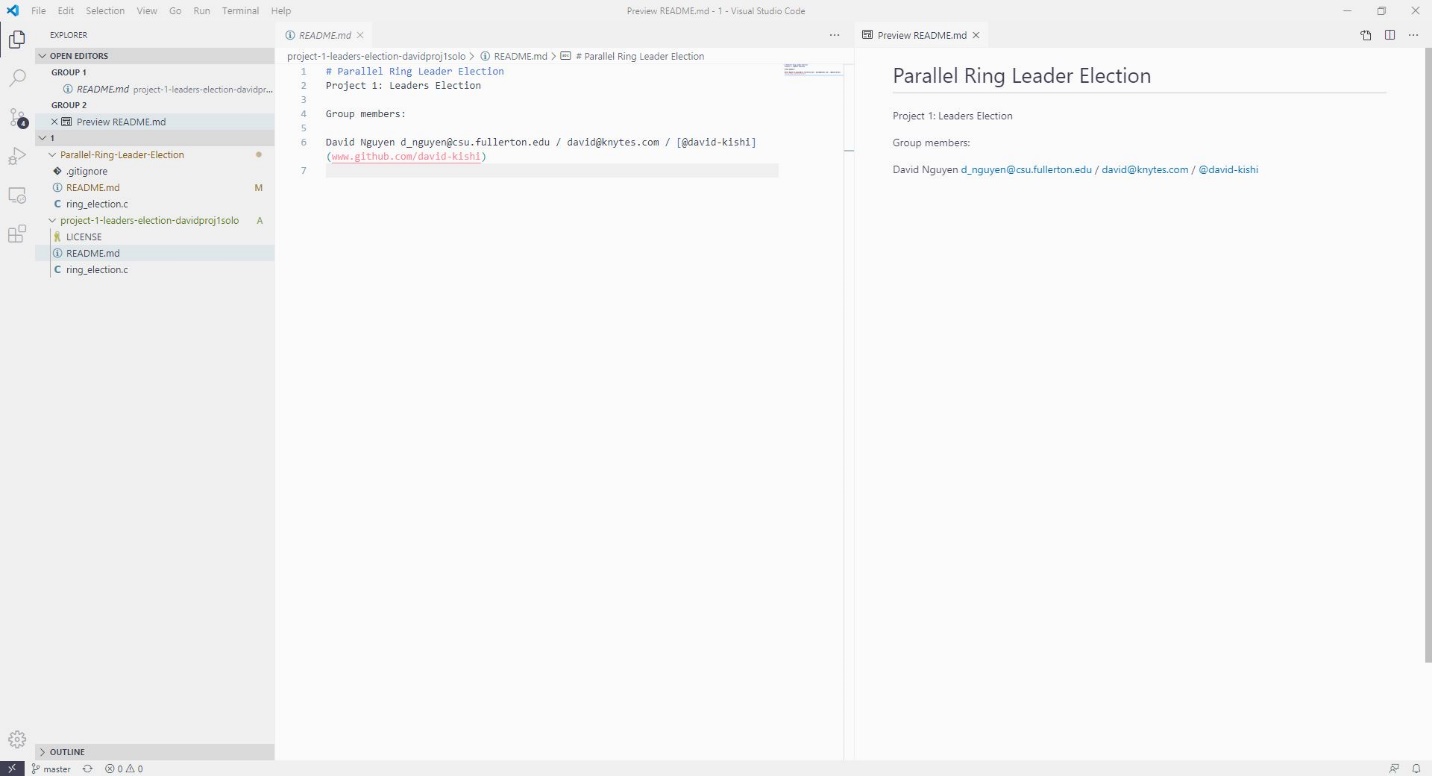


Figure 1. Credits

## **Example Output**

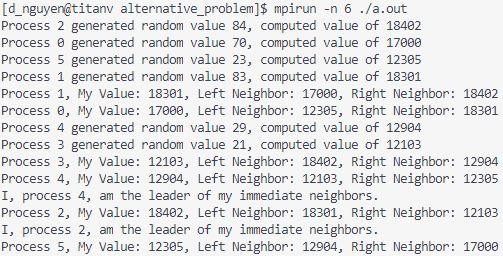


Figure 2. Example output with 6 processes