PROJECT 01

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*.

Project goal is to implement an algorithm to select a President and a Vice-President on a ring topology based on processes' exclusive randomly generated computed value, smallest odd and smallest even respectively. Using MPI (Message Passing Interface), results and comparisons can be checked and passed among processes.

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 results[4], rank, size, temp;
// results integer array of size 4 to hold results of election
// rank and size for MPI values
// temp is to hold each process's randomly generated value

<Initiate MPI>

if size is less than 6 or greater than 20:
     exit program
else if rank is not 0
     MPI Receive results array from previous process
     Generate random integer and store into temp
     if temp is even:
          if temp is less than even in results:
               replace even in results with temp
               replace process number in results
     else if temp is less than odd in results:
               replace odd in results with temp
               replace process number in results
else
     Generate random integer and store into temp
     If temp is even:
          set results array to {19999, 0, temp, 0}
     else:
          set results array to {temp, 0, 19990, 0}

MPI Send results to next process, else if last process, send
back to process 0

if rank is 0:
     MPI receive finalized results from final process
     Print results

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

randomInt(int rank)

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

return concat(randTmp, rank);
```

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
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

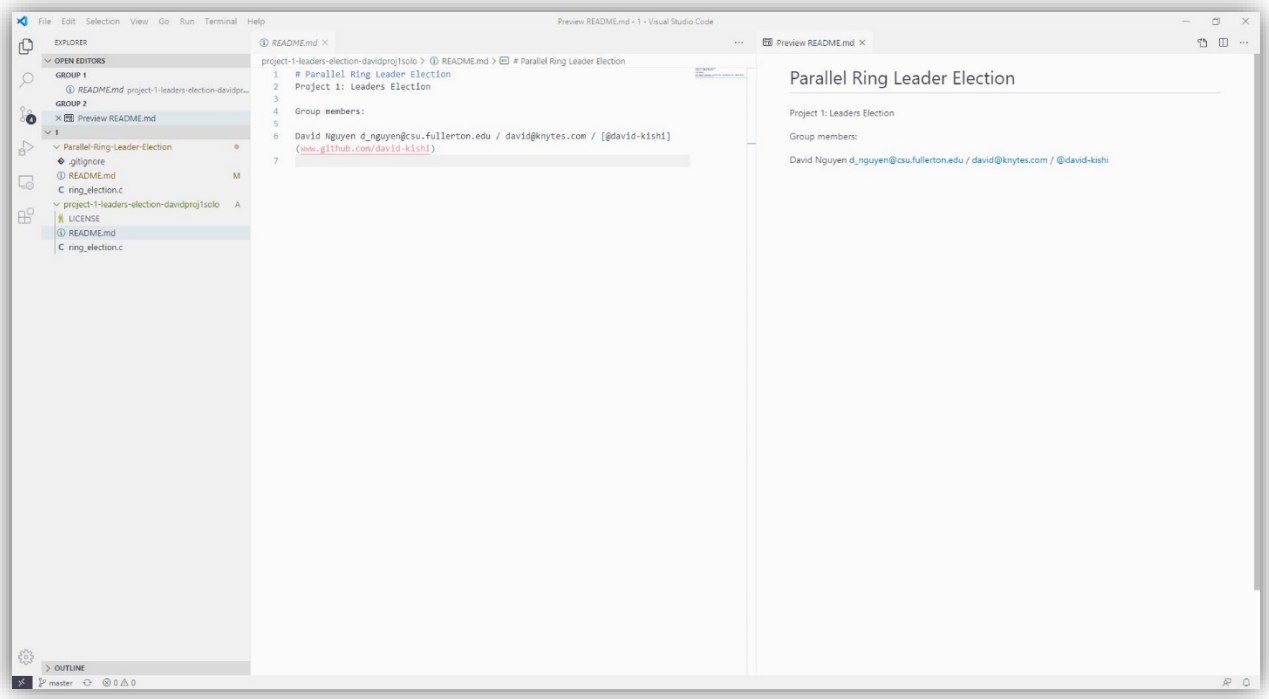

Figure 1. Credits

Example Outputs

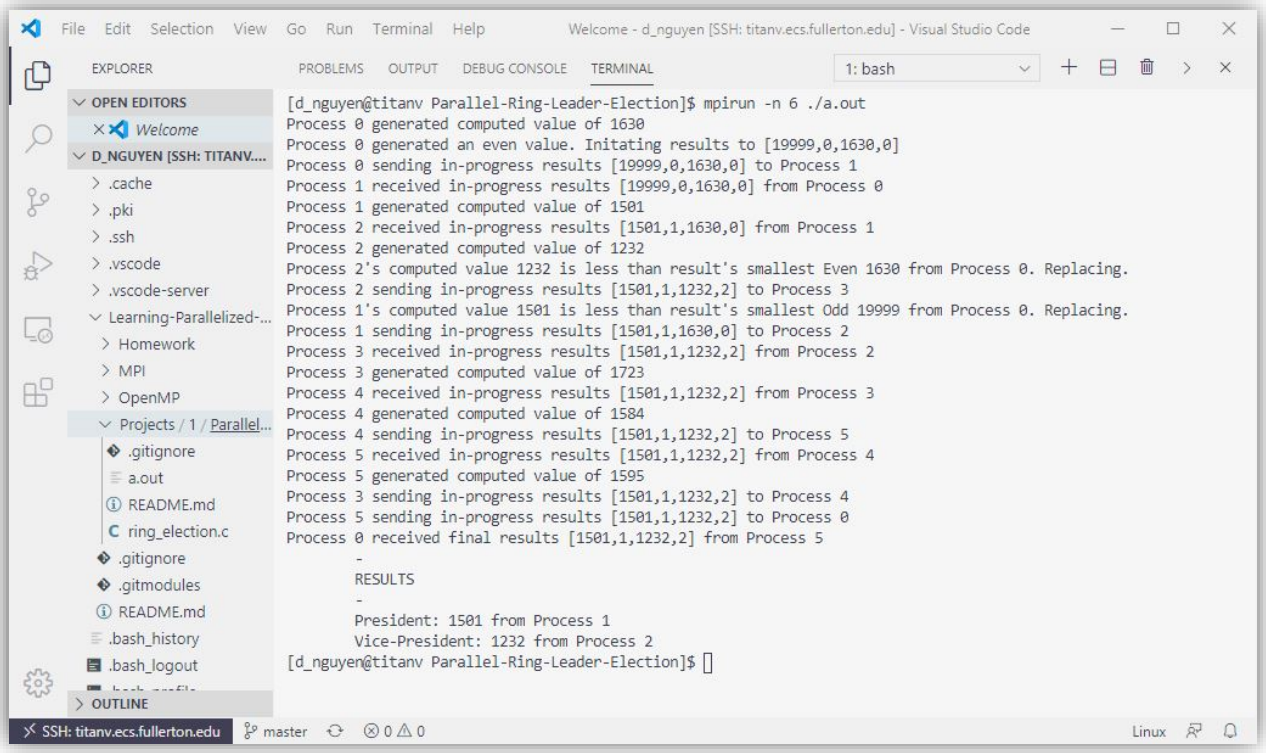

Figure 2. Example output with 6 processes

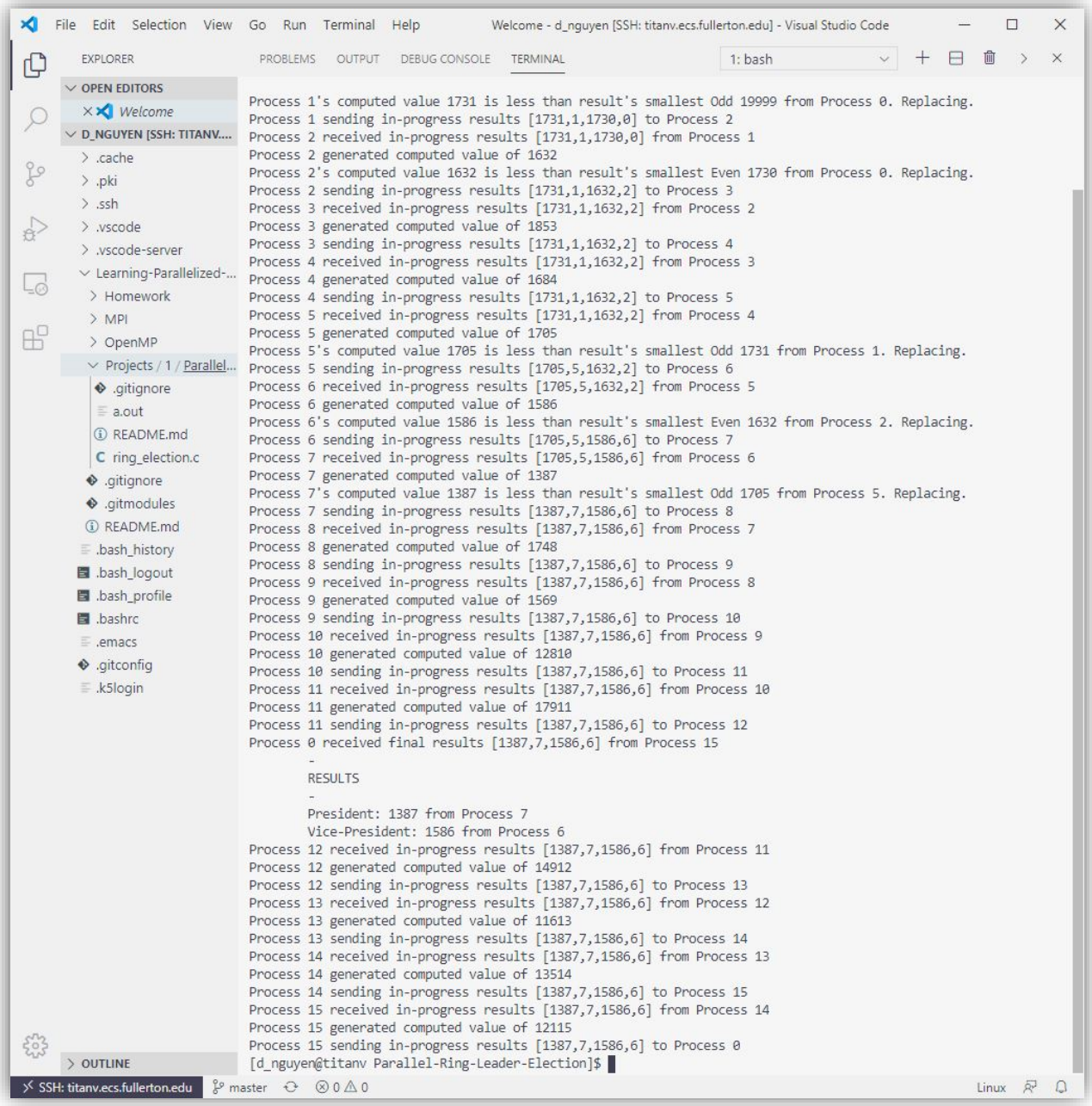

Figure 3. Example output with 16 processes