

Franklin Algorithm

Project 1

CPSC 474

10 December 2020

Dr. Doina Bein

David Nguyen 891362089

Summary

Franklin algorithm is a leader election algorithm that has each process transmits a value to its left and right neighbors. Then, the process will receive the value from its left and right neighbors. The values will be stored in a temporary list that holds the left, current, and right active processes' values. The program will determine a "maybe" leader if the current active process has a larger value than its left and right neighbors. The "maybe" leaders will remain in the list and all of the non-"maybe" leaders are left out of the list. This way the next iteration of this algorithm will decrease the memory usage and increase the efficiency. This algorithm is similar to divide-and-conquer algorithms, but instead this algorithm is decreasing the problem into smaller problems and will find the leader once the size of this list becomes one. This algorithm was intended to be a faster speed up of the algorithm explained in class.

Pseudocode

```
/*
Each process in the list will have an id and a value.
The id will be used to reference the process's rank just in case if the list gets mutated.
When the list gets mutated, the process's rank becomes lost in the next iteration which
makes sending and receiving impossible.
The value is stored in each process to determine if it should be passive or active on the
next round.
*/
struct Proc {
    int id;
    int val;
};

/*
We have a root process (rank 0). The root process is in charge of controlling the rounds,
mutating the list, and finding the leader.
*/
if rank = 0
{
    print "ROUND: " round
    //sending the data to every existing processes.
    for (i < size of the list) {
        Send the list to every processes.
    }
    //now we need to send the index of the array to every existing processes.
    print the list
    //inspired by raymond algorithm, we need a count to determine the number of
processes leftover.
    Send the count to the first element of the list.
    Receive the count from the last element of the list.

    for (i < count)
    {
        Receive the count from the previous process that held the count.
        Create a temporary list that will hold all of the maybe leader's values
        //sorted insert for idx to avoid out of order comparison.
        if the temp list is not empty, then we will compare the incoming id. {
            if the incoming id is larger than the front of the list,
then we add the incoming process to the back of the tmp list.
            else, then we will add the process to the front of the
list.
        }
        else if the temp list is empty then we will just add to the list.
    }
}
```

```

        If the count of the tmp list becomes one, then we have found a leader.

        Mutate the original list so that next iteration will have to work with the smaller
list.
    }
}

/*
These child processes are in charge of sending/ receiving the process's values.
After receiving the values then it will compare the current process's values with the left
and right processes. Once the program determines if the current process has the highest
values of both its neighbors then it will send the value to the root process.
*/
else{
    if (process is active) {
        Receive the payload or the list from the root process.
        Send the current value to the left neighbor.
        Send the current value to the right neighbor.

        All process must finish sending

        Receive the value from the left neighbor.
        Add to the comparison list.
        Receive the value from the right neighbor.
        Add to the comparison list.

        All process must finish receiving.

        Receive the count from the previous process that held it.

        Compare the values in the comparison list to see if the current process can
go onto the next round. If it does, increment the count.

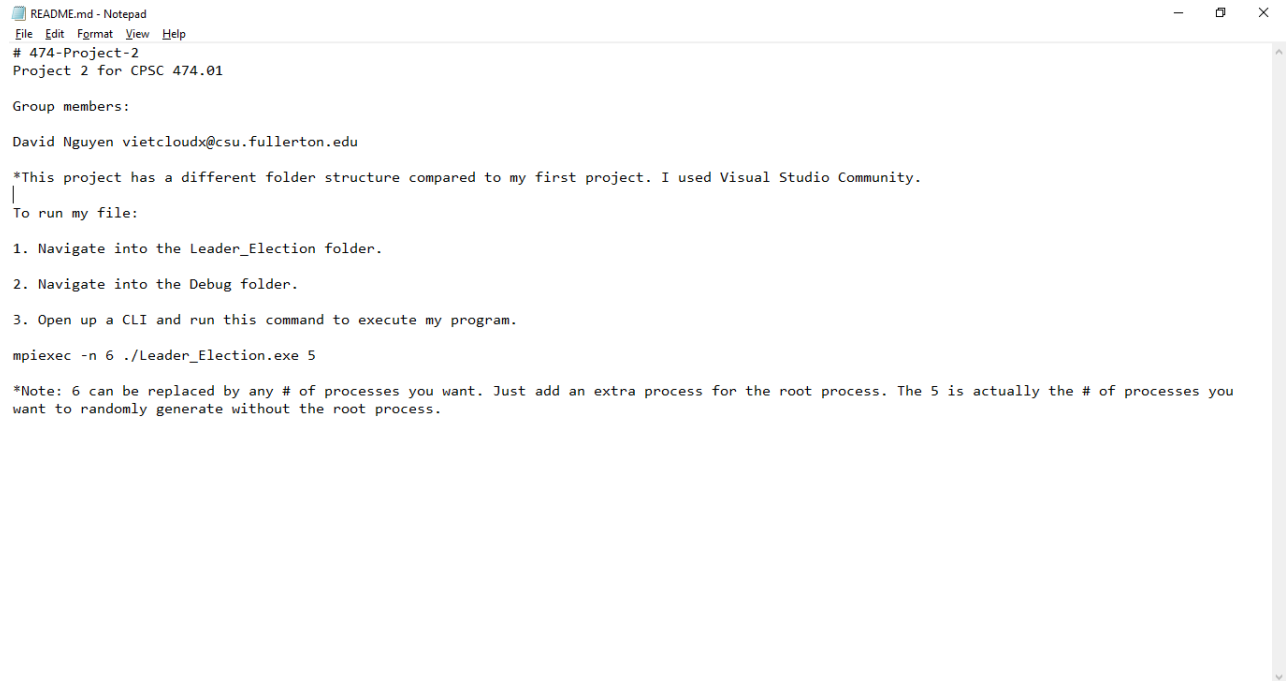
        Send the count to the next process.

        All process must finish determining the maybe leader processes.

        Compare the values in the comparison list again to see if the current process
can send its value and id back to root process and stay for the next round.
    }
}

```

Screenshots



```
README.md - Notepad
File Edit Format View Help
# 474-Project-2
Project 2 for CPSC 474.01

Group members:

David Nguyen vietcloudx@csu.fullerton.edu

*This project has a different folder structure compared to my first project. I used Visual Studio Community.
|
To run my file:

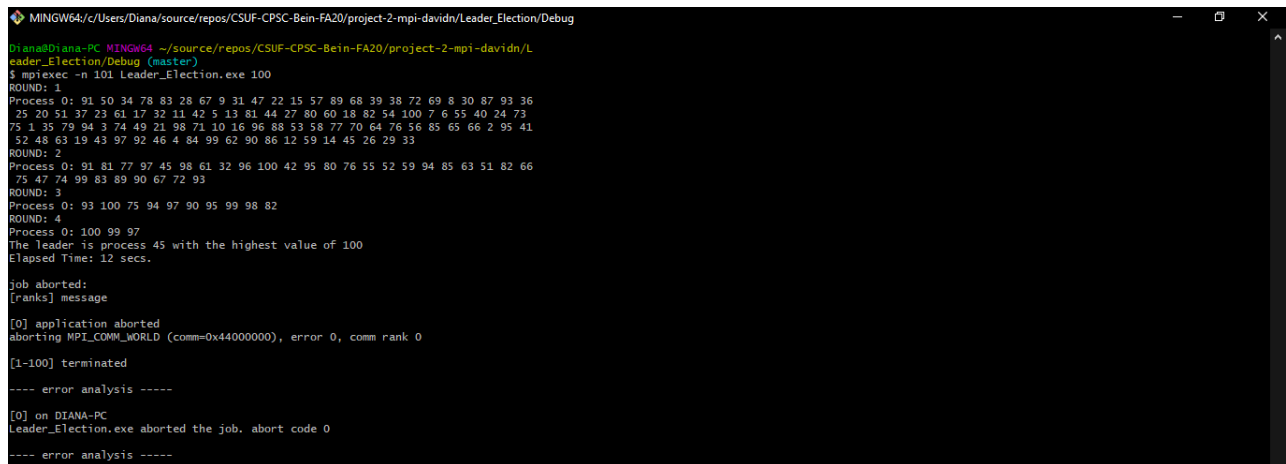
1. Navigate into the Leader_Election folder.

2. Navigate into the Debug folder.

3. Open up a CLI and run this command to execute my program.

mpiexec -n 6 ./Leader_Election.exe 5

*Note: 6 can be replaced by any # of processes you want. Just add an extra process for the root process. The 5 is actually the # of processes you want to randomly generate without the root process.
```



```
MINGW64/c/Users/Diana/source/repos/CSUF-CPSC-Bein-FA20/project-2-mpi-davidn/Leader_Election/Debug
Diana@Diana-PC MINGW64 ~/source/repos/CSUF-CPSC-Bein-FA20/project-2-mpi-davidn/Leader_Election/Debug (master)
$ mpiexec -n 101 Leader_Election.exe 100
ROUND: 1
Process 0: 91 50 34 78 83 28 67 9 31 47 22 15 57 89 68 39 38 72 69 8 30 87 93 36
25 20 51 37 23 61 17 32 11 42 5 13 81 44 27 80 60 18 82 54 100 7 6 55 40 24 73
75 1 35 79 94 3 74 49 21 98 71 10 16 98 88 53 58 77 70 64 76 56 85 65 66 2 95 41
52 48 63 19 43 57 92 46 4 84 99 62 90 86 12 59 14 45 26 29 33
ROUND: 2
Process 0: 91 81 77 97 45 98 61 32 96 100 42 95 80 76 55 52 59 94 85 63 51 82 66
75 47 74 99 83 89 90 67 72 93
ROUND: 3
Process 0: 93 100 75 94 97 90 95 99 98 82
ROUND: 4
Process 0: 100 99 97
The leader is process 45 with the highest value of 100
Elapsed Time: 12 secs.

Job aborted:
[ranks] message

[0] application aborted
aborting MPI_COMM_WORLD (comm=0x44000000), error 0, comm rank 0

[1-100] terminated

---- error analysis ----

[0] on DIANA-PC
Leader_Election.exe aborted the job. abort code 0

---- error analysis ----
```



```
Diana@Diana-PC MINGW64 ~/source/repos/CSUF-CPSC-Bein-FA20/project-2-mpi-davidn/Leader_Election/Debug (master)
$ mpiexec -n 11 Leader_Election.exe 10
ROUND: 1
Process 0: 6 4 1 3 5 9 7 8 2 10
ROUND: 2
Process 0: 9 10 8
The leader is process 10 with the highest value of 10
Elapsed Time: 0 secs.

Job aborted:
[ranks] message

[0] application aborted
aborting MPI_COMM_WORLD (comm=0x44000000), error 0, comm rank 0

[1-10] terminated

---- error analysis ----

[0] on DIANA-PC
Leader_Election.exe aborted the job. abort code 0

---- error analysis ----
```