

University of Ottawa

CSI5308 course

Course Project

Election in Ring

Emad Badawi

Ebada090@uottawa.ca

https://github.com/Ebadawi/CSI5308-Project

**Idea**

In this project, the C++ and Message Passing Interface were exploits to implementing minmax, all the way and as far election algorithm in unidirectional rings, and capture the number of exchanged messages in each one of the algorithms, then compare the acquired messages number with the worst case complexity for each of the algorithms.

**System Configuration and Platform Details**

In the conducted experiments, a cluster of 6 computers were used. Each of the cluster nodes has different HW/SW specifications. Table 1 contains the used cluster HW specifications.

Table 1System Configuration & Platform Details

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Computer name** | **Kingpenguin** | **Dune-970** | **Dune-Titan** | **Dune-770** | **Dune-Frg** | **Setup-T3600** |
| **CPU** | Intel(R) Xeon(R) CPU E5-2640 @ 2.50GHz | Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz | Intel(R) Core(TM) i7-5930K CPU @ 3.50GHz | Intel(R) Core(TM) i7-4820K CPU @ 3.70GHz | Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz | Intel(R) Xeon(R) CPU E5-1620 0 @ 3.60GHz |
| **CPU Cores** | 12 | 12 | 6 | 4 | 6 | 4 |
| **Threads / Core** | 2 | 2 | 2 | 2 | 2 | 2 |
| **RAM** | 64 GB | 32 GB | 64 GB | 32 GB | 32 GB | 16 GB |
| **GPU** | - | GeForce GTX 970 | GeForce GTX TITAN X | GeForce GTX 770 | - | Quadro 2000 |
| **GPU Cores** | - | 1664 | 3072 | 1536 | - | 192 |
| **GPU RAM** | - | 4GB | 12GB | 2GB | - | 1GB |
| **Compute Capability** | - | 5 | 5 | 3 | - | 2 |
| **Number of GPUs** | - | 2 | 1 | 2 | - | 1 |

The software environment is the same for all the test beds in **Table** 1 and is mentioned below:

* Operating System: Ubuntu 16.04.1 LTS (64 bit)
* The GNU C++ 4.9.3 compiler.

Election algorithm was executed using MPI parallel implementation on a NoW formed by the machines shown in Table 1.

**Implemented Algorithms**

In all of the implemented algorithms, the number of lunched processes represents the number of nodes in the rings. The connections between the nodes is identified by the fact that MPI processes have distinct IDs (process rank), in which these IDs can be viewed as a ring. Figure 1 represents 8 lunched processes connected with each other in such a way to create ring. The processes IDs can be used as a distinct IDs for the corresponding nodes, but this will limit the simulation with one case, which is the ascending order or descending order. Another way to assign distinct IDs is to use a master process to generate random distinct IDs and distribute them between the other processes. The later way of assigning IDs was used.

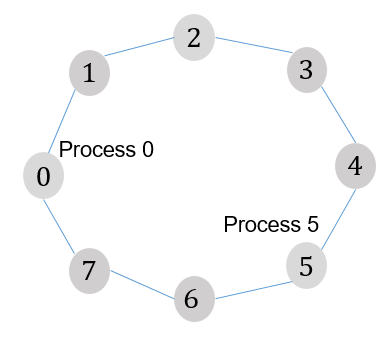


Figure 1: Ring constructed using MPI processes

1. **All the way**

This is the easiest algorithm, here each node sends two values to the neighbor node, the first is the node ID and the second as a counter to count how many nodes the ring consists from. Each node has a local counter to count how many nodes passed by, and has a minimum value equals to its ID initially, this minimum value will be updated if a node that has smaller value passed by. When a node receives its value back, it will know how many nodes in the ring using the counter value traversed with the ID. When the number of passed nodes equals the number of nodes in the ring, I’ll be the leader if my min value equals my ID and I’ll be follower otherwise.

1. **As Far**

This is the a variation of All the Way algorithm, in which each node sends its value to its neighbor, but the main difference that the message will be stopped if its counter a node with smaller ID in the way. The termination in this algorithm is achieved when a node receives its value back, which means that I did not counter any node with smaller ID than mine, thus I am the leader. Here a termination message is required to be send to the other nodes telling them that the algorithm terminated and I am the leader.

1. **Min Max**

This is the a variation of electoral stages, in which each node sends its value to its neighbor, but the main difference that the nodes elimination depends on two factors, the nodes ID’s as well as the current stage, thus in the message we send the stage in addition to the ID. In particular, in odd stages I’ll be eliminated if I receive a message from node with ID smaller than mine, while in even stages I’ll be eliminated if I receive a message from node with ID bigger than mine while. In this algorithm, it’s not necessary for the leader to be the maximum or minimum value, any value might survive until the last stage and become the leader. Here a termination message is required to be send to the other nodes telling them that the algorithm terminated and I am the leader.

**Algorithms Execution**

To build the algorithm:

mpic++ ./code\_file.cpp -o executable\_file

For Example, to build AS Far code: mpic++ ./AsFar.cpp -o asfar

To run the algorithm:

mpirun -np N --hostfile hosts ./ executable\_file

For Example, to run AS Far executable file: mpirun -hostfile hosts -np 30 ./asfar

Where:

N: represent the number of lunched processes (the number of nodes)

Hosts: a file that contains the IPs of the used cluster (1 IP per line)