nchang1 - 858604

COMP90024 Assignment1 Report

1 Programme Specification

1.1 File List

Name	**Description**
distributor.py	Handle multi-process function, responsible to distribute tasks, collect response, and calculate result.
grid.py	Handle data processing, responsible to check the coordinates, assign to the corresponding grid, and accumulate grid numbers.
one_node_1_core.slurm	Script for 1 node 1 core job (see Appendix)
one_node_8_core.slurm	Script for 1 node 8 cores job (see Appendix)
two_node_4_core.slurm	Script for 2 nodes 4 cores job (see Appendix)

1.2 Programme Detail

The programme supports multi-process with distributed memory.

Master Process

It starts by scan through the file, and generates a list of start-end position for every Instagram post. Then, it split the position list into several equal parts, and scatter the start-end position to worker process.
Once the tasks are distributed, the master process also acts as a worker process, reading through data.
Finally, master process gathers the results, calculates the ranking, and prints out the required output.

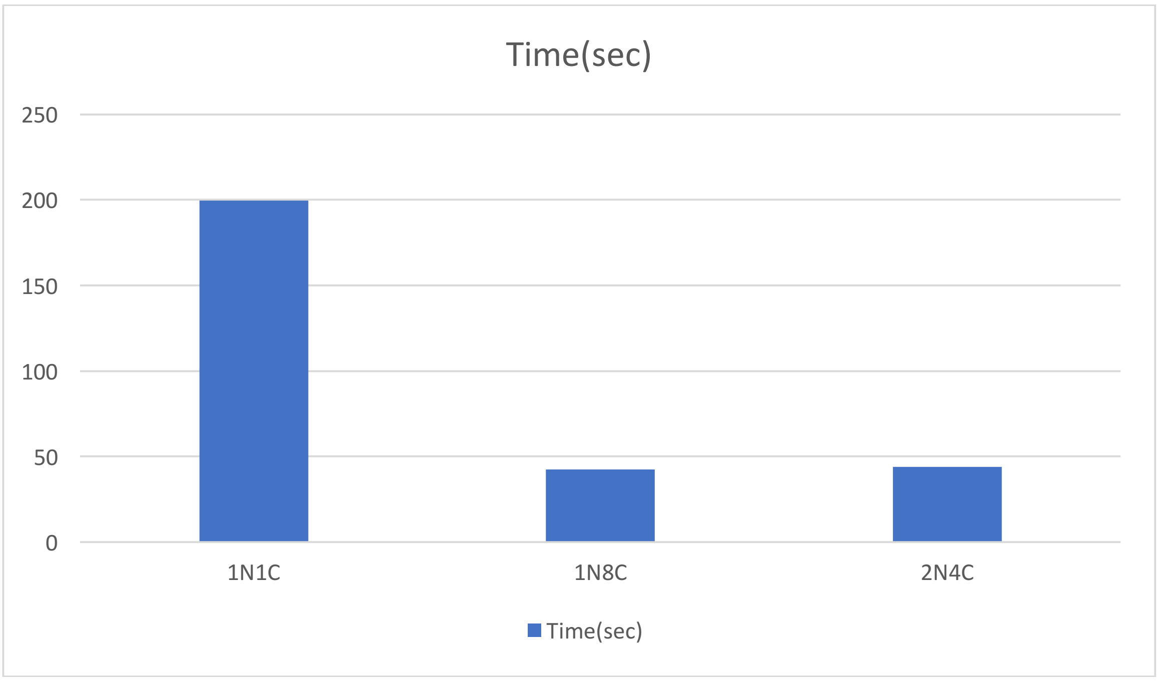
Worker Process

Upon receiving the tasks from the master, the workers will proceed to read the coordinates according to the assigned position, and process the posts accordingly. A grid record with number of posts is maintained on each worker process, and returned to master process for final processing.

2 Programme Run Time

According to the run time of different configurations, the 1 node 1 core task takes roughly 5 times longer than 1 node 8 cores.
The time consumption for 2 nodes 4 cores, comparing to 1 node 8 cores, despite of the same core numbers, is slightly higher, which might be due to the communication loss between different nodes.
The detail time table and bar chart is as below.

Configuration	**Time**
1 Node 1 Core	3m19.402s
1 Node 8 Cores	0m42.261s
2 Nodes 4 Cores	0m43.953s



3 Reflection on Parallel Method

The bottleneck of parallel code is that the master process reads through the file to obtain the split point of file distribution, at which time the other worker processes idle. An alternative solution (as inspired by posts on discussion board) is to let different processes read different lines according to their rank number. For example, among 4 processes, process 0 reads line 0, 4, 8, process 1 reads line 1, 5, 9, and so on.

Appendix

one_node_1_core.slurm

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --time=00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mail-type=ALL
#SBATCH --mail-user=n.chang1@student.unimelb.edu.au

module load Python/2.7.11-goolf-2015a
echo "1 node 1 core"
time mpirun python distributor.py
```

one_node_8_core.slurm

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --time=00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=1
```

```
#SBATCH --mail-type=ALL
#SBATCH --mail-user=n.chang1@student.unimelb.edu.au

module load Python/2.7.11-goolf-2015a
echo "1 node 8 core"
time mpirun python distributor.py
```

two_node_4_core.slurm

```
#!/bin/bash
#SBATCH -p physical
#SBATCH --time=00:15:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
#SBATCH --mail-type=ALL
#SBATCH --mail-user=n.chang1@student.unimelb.edu.au

module load Python/2.7.11-goolf-2015a
echo "2 nodes 4 core"
time mpirun python distributor.py
```