**COMP90024 Cluster and Cloud Computing 2018 Semester 1**

**Assignment 1 – HPC Instagram Geoprocessing**

**Application Report**

**Ivan Ken Weng Chee**

**736901**

**Objectives**

To implement a parallelized application to parse and analyse a large Instagram dataset to determine usage across Melbourne.

To learn about the different functions and uses of Message Passing Interface (MPI).

To familiarize myself with working on High Performance Computing (HPC) Systems such as SPARTAN.

**Scripts Used for Submitting Jobs to SPARTAN**

#!/bin/bash

#SBATCH --time=00:04:00

#SBATCH --nodes=2

#SBATCH --ntasks-per-node=4

#SBATCH --partition=physical

#SBATCH --output=python-2n4c.out

# Load required modules

module load Python/2.7.10-goolf-2015a

# Launch python code

echo "Processing bigInstagram.json"

time mpirun python hpc\_instagram\_geoprocessing.py melbGrid.json bigInstagram.json

**Approach I Took to Parallelize My Code**

There are multiple ways of approaching this problem and arriving at a conclusion.

My approach

* Master reads melbGrid coordinates into a list and sends this to all worker processes.
* Master then opens bigInstagram and, reading line by line, converts them to json format (python dictionary), and extracts the corresponding x and y coordinates.
* These coordinates are sent workers in an alternating manner, whereby each worker receives coordinates in each line in order (W1 < L1, W2 < L2, W3 < L3, W1 < L4, and so on).
* Upon receiving coordinates, each worker checks whether the coordinates are in a grid, and if so, count the number of coordinates found via maintaining three dictionary counters for posts in each grid, posts in each row, and posts in each column.
* Once master reaches the end of file, a terminating signal is sent to all workers.
* All processes then call the reduce operation with a custom function to merge the count dictionaries together, which is then sorted and printed by master
* If there is only one process, master does all the work

However, while this method does parallelize work, it has proven to be rather slow, running at least 150 seconds on 1node1core, and slightly slower on 1node8cores and 2nodes4cores respectively. An explanation of this would be that, as a send and receive call is made for each line, this results in approximately 18 million calls to these functions throughout the program, which ultimately results in higher overhead. 2nodes4cores is the slowest as message passing between different nodes is generally slower as the systems are physically apart.

A faster approach

Another method would be where master first reads in the file, extracting coordinates and storing them in a list, then scatters the list to all workers to process. This has the added advantage of having far less message passing calls, as well as having master do a portion of the work. While storage of coordinates is small, for larger files this method may eventually run out of memory as the list becomes larger and larger.

An alternative approach

Each process reads in bigInstagram, interleaving a set number of lines each process based on their rank. Example with 100 lines: (P1 < L1-100, P2 < L101-200, P3 < L201-300, P4 < L301-400, P1 < L401-500, and so on). This method would offer a slight improvement over one process processing the entire file. However, this method would not be able to reach its full potential as there is no way to ‘jump’ or ‘seek’ to corresponding lines with files containing arbitrary line lengths. It can be modified to have processes read n number of lines according to patterns such as the Fibonacci numbers, which would scale better with smaller files.

**Difficulties**

* Size of the dataset which inhibits reading into memory
* Irregularity of json formatting of files
* Missing coordinates [NULL, NULL]

**Variations in its Performance on Different Numbers of Nodes and Cores**

1n1c

Processing bigInstagram.json

// Rank by Unit

C2: 175234 posts

B2: 22797 posts

C3: 17167 posts

B3: 5703 posts

C4: 4077 posts

B1: 3311 posts

C5: 2613 posts

D3: 2333 posts

D4: 1903 posts

C1: 1595 posts

B4: 950 posts

D5: 717 posts

A3: 495 posts

A2: 479 posts

A1: 262 posts

A4: 103 posts

// Rank by Row

C-Row: 200686 posts

B-Row: 32761 posts

D-Row: 4953 posts

A-Row: 1339 posts

// Rank by Column

Column 2: 198510 posts

Column 3: 25698 posts

Column 4: 7033 posts

Column 1: 5168 posts

Column 5: 3330 posts

Time: 154.41s

real 2m34.794s

user 2m31.114s

sys 0m3.564s

1n8c

Processing bigInstagram.json

// Rank by Unit

C2: 175234 posts

B2: 22797 posts

C3: 17167 posts

B3: 5703 posts

C4: 4077 posts

B1: 3311 posts

C5: 2613 posts

D3: 2333 posts

D4: 1903 posts

C1: 1595 posts

B4: 950 posts

D5: 717 posts

A3: 495 posts

A2: 479 posts

A1: 262 posts

A4: 103 posts

// Rank by Row

C-Row: 200686 posts

B-Row: 32761 posts

D-Row: 4953 posts

A-Row: 1339 posts

// Rank by Column

Column 2: 198510 posts

Column 3: 25698 posts

Column 4: 7033 posts

Column 1: 5168 posts

Column 5: 3330 posts

Time: 156.38s

real 2m37.013s

user 20m49.565s

sys 0m3.025s

2n4c

Processing bigInstagram.json

// Rank by Unit

C2: 175234 posts

B2: 22797 posts

C3: 17167 posts

B3: 5703 posts

C4: 4077 posts

B1: 3311 posts

C5: 2613 posts

D3: 2333 posts

D4: 1903 posts

C1: 1595 posts

B4: 950 posts

D5: 717 posts

A3: 495 posts

A2: 479 posts

A1: 262 posts

A4: 103 posts

// Rank by Row

C-Row: 200686 posts

B-Row: 32761 posts

D-Row: 4953 posts

A-Row: 1339 posts

// Rank by Column

Column 2: 198510 posts

Column 3: 25698 posts

Column 4: 7033 posts

Column 1: 5168 posts

Column 5: 3330 posts

Time: 162.48s

real 2m43.982s

user 6m46.014s

sys 4m5.151s

**Single Bar Chart Showing Times for Execution vs. Numbers of Nodes and Cores**

d

// Report no longer than 3 pages

// Submission <Joe-Smith-0123456>.zip