COMP90024 Assignment 1

Quanchi Chen
`Student ID: 1358474`

Benjamin Lo
`Student ID: 1171146`

1 Introduction

Twitter is a social networking service that allows friends, family and mutuals to communicate and remain interconnected through frequent exchange of messages, also known as "tweets". To preface the objective, the programming component involves the *bigTwitter.json* file and is broken down into three tasks:

- Identify the top ten Twitter accounts that have made the most tweets
- Count the number of tweets made in the Greater Capital cities of Australia
- Identify the top ten Twitter accounts that have tweeted from the most Greater Capital cities and the number of times they have tweeted from those locations

The aim of this report is to implement a parallelised application leveraging the SPARTAN HPC facility operated by the University of Melbourne to complete the aforementioned objectives. The mpi4py library in Python was used to provide bindings to the Message Parsing Interface (MPI) and allow the program to exploit multiple processors on SPARTAN. This report will investigate the runtimes when executing the program under the following SPARTAN resources:

- 1 node and 1 core
- 1 node and 8 cores
- 2 nodes and 8 cores (with 4 nodes per core)

2 Methodology

2.1 Parallel Programming with MPI

We employ the single program multiple data (SPMD) model to enable multiple processes to analyse the same JSON file in parallel. With MPI, each process executes on a separate computing core and is a running instance of the same program.

Assume the number of processes is N, and the file size is M bytes. The general idea is to split the JSON file into N chunks, each read and processed by one process. p_i denotes the process with rank i, and d_i represents the data chunk that p_i is supposed to analyse, where $i \in [0, N-1]$. Then the range of d_i is $[\frac{M}{N} \cdot i, \frac{M}{N} \cdot (i+1)]$. Each process uses the above formula to calculate its data chunk's start and end positions at the start of execution.

Each process reads its corresponding data chunk line by line and uses regular expressions to extract the author ID and the location's full name from each tweet object. Under such a design, even one process can analyse the entire file without running out of Random Access Memory (RAM). The root process (i.e., the process with rank 0) will gather the results produced by all processes and convert the obtained list of dictionaries into one single dictionary, simplifying the sorting and printing operations for the three programming tasks.

One challenge arises when p_i analyses the last tweet object of d_i, which may be incomplete. If p_i can only extract the author ID of that tweet due to the remaining part residing in d_{i+1}, then p_i will continue reading the file until obtaining the corresponding location. Such an approach avoids skipping several tweets and improves the accuracy of the analytic.

2.2 Preprocessing Tweets - Handling Edge Cases When Searching for Cities

Each process generates a nested dictionary that stores the Greater Capital city counts under each author ID in the structure of {author_ID1: {city1: a, city2: b, ...}, author_ID2: {city1: c, city2: d, ...}, ...}. Below discusses the steps followed by each process to check if the author's location is in one of the Greater Capital cities.

There was a total of 5 locations in Great Other Territories and so these locations were stored in a list. The first step was to check whether the author location matched any of the 5 locations in Great Other Territories. If a match was found, a count was added to "Great Other Territories (9oter)" for the corresponding author and the current iteration would be terminated to prevent the checking of subsequent edge cases.

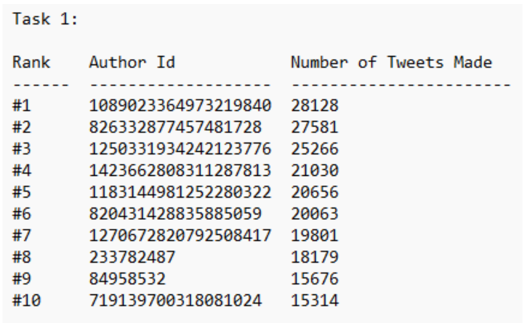
The second step was to identify whether Greater Capital cities were directly found in the provided author location. Some tweets included exact locations of Greater Capital cities which allowed the algorithm to easily identify and add counts to these respective cities.

The third step was to find state names in author location and append their respective abbreviations to the provided suburb. Some suburb names were repeated many times across several states and this procedure would allow the algorithm to correctly identify the respective Greater Capital city for such suburbs. All states were stored as keys in a dictionary, with their respective abbreviations stored as values. If a state was identified in the author location, their abbreviation was obtained through a key search in the dictionary. The abbreviation was appended to the given suburb through concatenation and the resulting string was searched in the *sal.json* file to find its corresponding Greater Capital city.

If neither the Greater Capital city or state was provided in the author location, then it was determined that the suburb was unique across Australia. The final option was to directly search for the suburb in the *sal.json* file and obtain its Greater Capital city without additional information.

3 Results

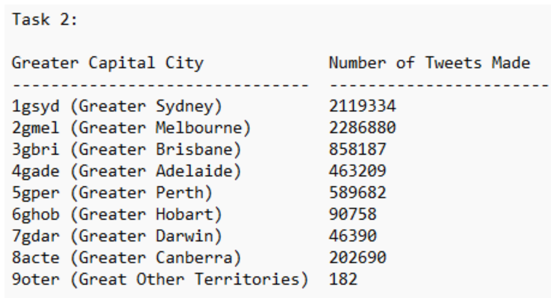
3.1 Results Tables for Tasks 1, 2 and 3



```
Task 1:

Rank    Author Id            Number of Tweets Made
------  -------------------  ----------------------
#1      1089023364973219840  28128
#2      826332877457481728   27581
#3      1250331934242123776  25266
#4      1423662808311287813  21030
#5      1183144981252280322  20656
#6      820431428835885059   20063
#7      1270672820792508417  19801
#8      233782487            18179
#9      84958532             15676
#10     719139700318081024   15314
```

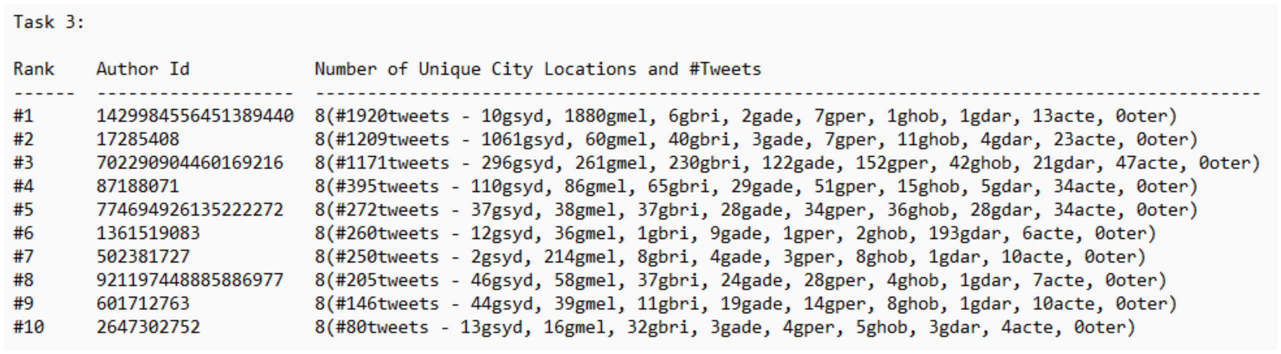
Table 1: Top ten Twitter accounts that have made the most tweets



```
Task 2:

Greater Capital City            Number of Tweets Made
------------------------------  ----------------------
1gsyd (Greater Sydney)          2119334
2gmel (Greater Melbourne)       2286880
3gbri (Greater Brisbane)        858187
4gade (Greater Adelaide)        463209
5gper (Greater Perth)           589682
6ghob (Greater Hobart)          90758
7gdar (Greater Darwin)          46390
8acte (Greater Canberra)        202690
9oter (Great Other Territories) 182
```

Table 2: Number of tweets made in the Greater Capital cities of Australia



```
Task 3:

Rank    Author Id            Number of Unique City Locations and #Tweets
------  -------------------  ---------------------------------------------------------------------------------------
#1      1429984556451389440  8(#1920tweets - 10gsyd, 1880gmel, 6gbri, 2gade, 7gper, 1ghob, 1gdar, 13acte, 0oter)
#2      17285408             8(#1209tweets - 1061gsyd, 60gmel, 40gbri, 3gade, 7gper, 11ghob, 4gdar, 23acte, 0oter)
#3      702290904460169216   8(#1171tweets - 296gsyd, 261gmel, 230gbri, 122gade, 152gper, 42ghob, 21gdar, 47acte, 0oter)
#4      87188071             8(#395tweets - 110gsyd, 86gmel, 65gbri, 29gade, 51gper, 15ghob, 5gdar, 34acte, 0oter)
#5      774694926135222272   8(#272tweets - 37gsyd, 38gmel, 37gbri, 28gade, 34gper, 36ghob, 28gdar, 34acte, 0oter)
#6      1361519083           8(#260tweets - 12gsyd, 36gmel, 1gbri, 9gade, 1gper, 2ghob, 193gdar, 6acte, 0oter)
#7      502381727            8(#250tweets - 2gsyd, 214gmel, 8gbri, 4gade, 3gper, 8ghob, 1gdar, 10acte, 0oter)
#8      921197448885886977   8(#205tweets - 46gsyd, 58gmel, 37gbri, 24gade, 28gper, 4ghob, 1gdar, 7acte, 0oter)
#9      601712763            8(#146tweets - 44gsyd, 39gmel, 11gbri, 19gade, 14gper, 8ghob, 1gdar, 10acte, 0oter)
#10     2647302752           8(#80tweets - 13gsyd, 16gmel, 32gbri, 3gade, 4gper, 5ghob, 3gdar, 4acte, 0oter)
```

Table 3: Top ten Twitter accounts that have tweeted from the most Greater Capital cities and the number of times they have tweeted from those locations

3.2 Table Displaying Time for Execution of Solution on Various SPARTAN Resources

SPARTAN Resource	Time Taken (Seconds)
1 Node and 1 Core	800.7738356590271
1 Node and 8 Cores	108.79034471511841
2 Nodes and 8 Cores	114.48109245300293

Table 4: Runtimes for the execution of programming application on various SPARTAN resources

3.3 Bar Chart Displaying Time for Execution of Solution on Various SPARTAN Resources

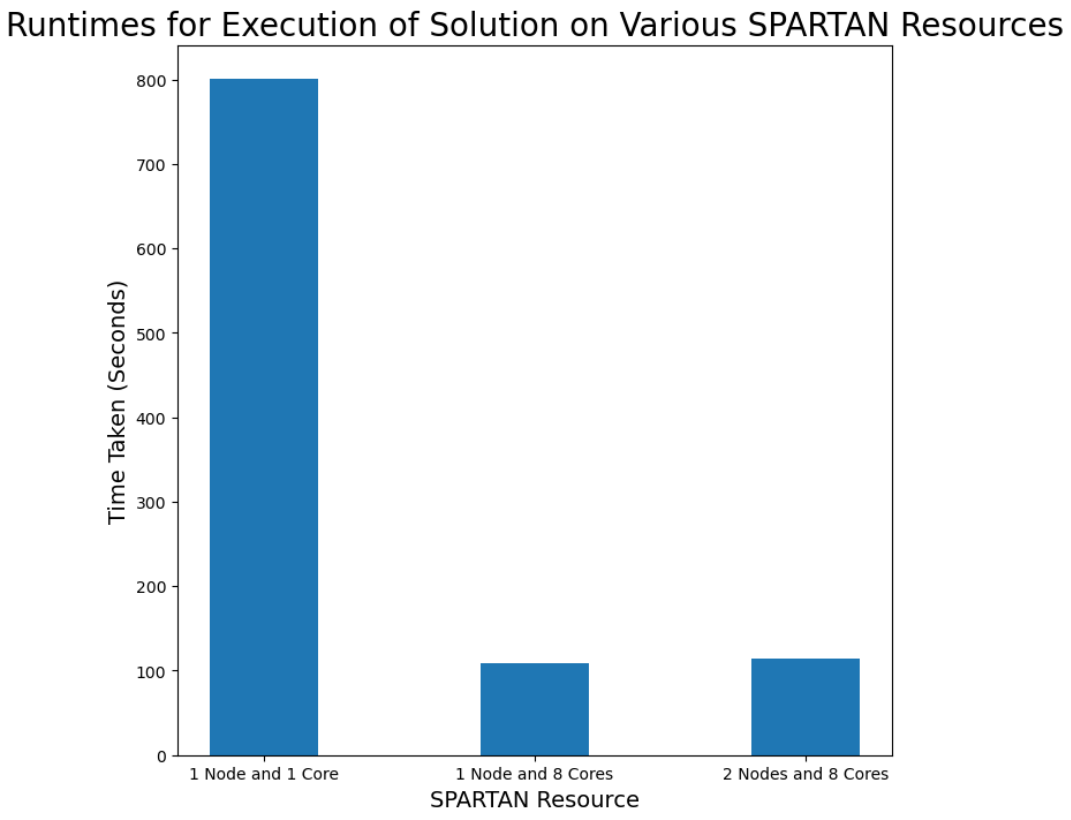


Figure 1: Runtimes for the execution of programming application on various SPARTAN resources

4 Discussion

From *figure 1*, it can be observed that the 1 node and 1 core combination had a significantly longer runtime compared to the other two options, which utilises parallelised computing to take advantage of the multithreaded performance of 8 cores. The two options utilising 8 cores can execute the program in a shorter timeframe as the MPI allows the parallelised sections of the algorithm to distribute work amongst the 8 processors.

Theoretically, it was expected that resources utilising 8 cores would execute the program 8 times faster than the 1 core option. However, upon closer inspection in *table 4*, the 1 node and 8 cores combination was approximately 7.36 times faster than the 1 node and 1 core combination. This is supported by Amdahl's law, which states that the overall performance improvement gained from optimising a particular section of the program is limited to the fraction of time the section is actually used. Therefore, the 1 node and 8 cores combination did not execute the program 8 times faster than the 1 node and 1 core combination as the parallelisation only applied to the preprocessing of twitter data, rather than the entire program itself. The same idea can be applied to the 2 nodes and 8 cores combination, which allocated 4 cores per node. This combination executed the program 6.99 times faster than 1 node and 1 core, but still did not achieve the theoretical prediction of running 8 times faster.

Furthermore, a comparison of runtimes between the two parallelised options in *table 4* demonstrates that the 2 nodes and 8 cores (4 cores per node) option executed the program slightly slower than 1 node and 8 cores. Although the difference of 5.69 seconds appears to be very minimal, it is still worth exploring this discrepancy as both combinations are using the same number of cores. The intra-node communication of using 1 node is faster than the inter-node communication between 2 nodes. When using 2 or more nodes, overhead will inevitably increase computational runtimes as nodes are required to communicate with one another to execute the programming application.

5 Conclusion

High Performance Computing (HPC) is pivotal for the implementation of algorithms which utilise multiple processors to solve larger scale problems, perform complex calculations and improve computational efficiency. Although an increase in the number of cores can drastically reduce computation runtimes, it has been demonstrated that a core count of x (where $x \geq 1$) does not necessarily improve efficiency by a factor of x compared to using 1 core. Furthermore, it is best to pair multiple cores with 1 node compared to several nodes because intra-node communication is faster and more optimised compared to inter-node communication, which introduces more overhead.