Stephanie Galvan
Parallel Computing
MW 4:30PM – 5:50PM

Lab 3: Map Reduce

Description

The following lab counts the number of occurrences of a word from a given list found in Shakespeare's works.

Challenges

At some point during the implementation of the program, the console threw an error regarding being unable to update the value of the dictionary. This was fixed by restarting the virtual machine; thus, I do not know what really caused this issue. Another challenge I faced was loading the files before doing the parallel section; I first tested the parallel region by opening the files while in there just to verify that I was able to run in parallel. After some reading on the internet, I was able to find how to fix my issue by using the read() function. I modified my program so that it could load the files beforehand and improve performance.

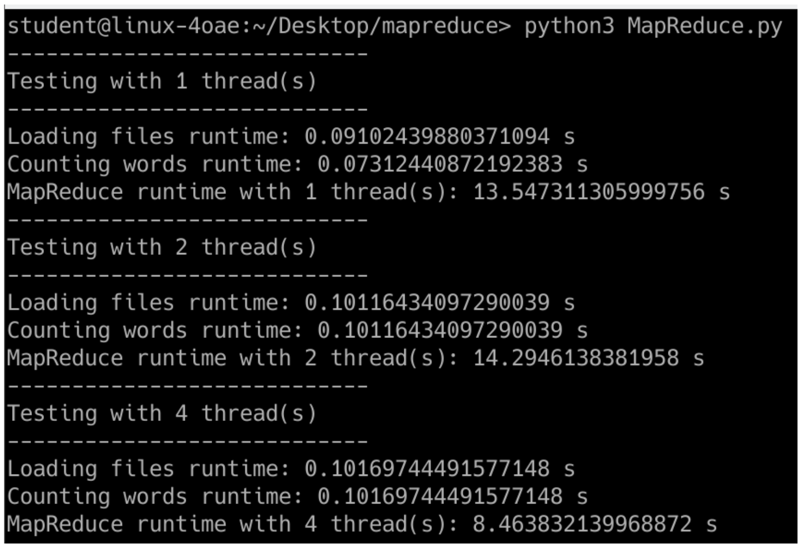
Bugs

A possible bug is that since I allow the user to input a number of threads on the console, the user could potentially input an invalid number for threads (such as a negative value or zero); however, I also have a default run of the program in which the Map Reduce will be tested with 1, 2, 4, and 8 threads which fulfills the requirements for this lab.

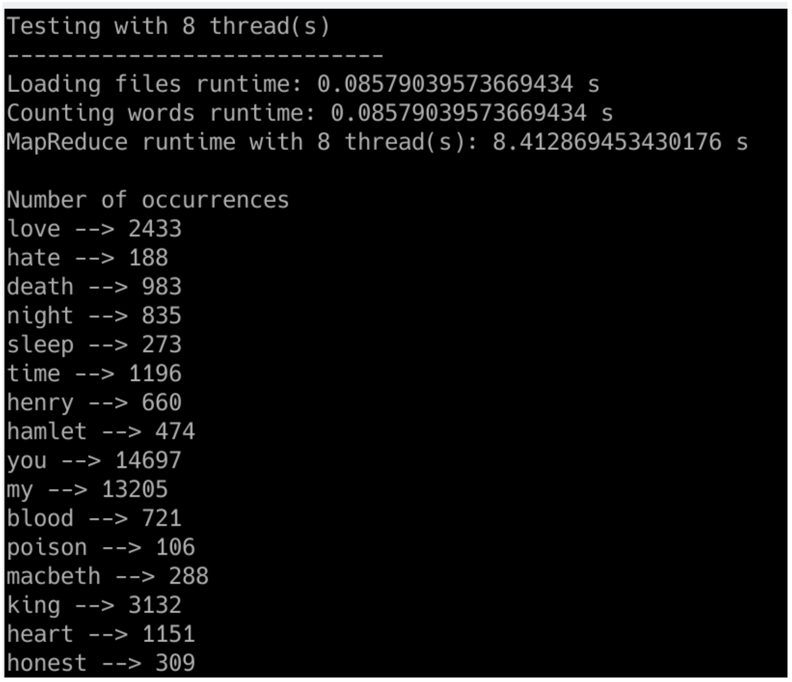
Time Completion

Overall, the implementation and testing of the program took around three to four hours as I was trying to debug it and correctly implement the parallel region.

Performance Measurements



```
student@linux-4oae:~/Desktop/mapreduce> python3 MapReduce.py
----------------------------
Testing with 1 thread(s)
----------------------------
Loading files runtime: 0.09102439880371094 s
Counting words runtime: 0.07312440872192383 s
MapReduce runtime with 1 thread(s): 13.547311305999756 s
----------------------------
Testing with 2 thread(s)
----------------------------
Loading files runtime: 0.10116434097290039 s
Counting words runtime: 0.10116434097290039 s
MapReduce runtime with 2 thread(s): 14.2946138381958 s
----------------------------
Testing with 4 thread(s)
----------------------------
Loading files runtime: 0.10169744491577148 s
Counting words runtime: 0.10169744491577148 s
MapReduce runtime with 4 thread(s): 8.463832139968872 s
```



```
Testing with 8 thread(s)
----------------------------
Loading files runtime: 0.08579039573669434 s
Counting words runtime: 0.08579039573669434 s
MapReduce runtime with 8 thread(s): 8.412869453430176 s

Number of occurrences
love --> 2433
hate --> 188
death --> 983
night --> 835
sleep --> 273
time --> 1196
henry --> 660
hamlet --> 474
you --> 14697
my --> 13205
blood --> 721
poison --> 106
macbeth --> 288
king --> 3132
heart --> 1151
honest --> 309
```

Analysis

As shown on the screenshots above, a single thread would not take longer than 20s to finalize the program. In this instance, having two threads did not result in an improvement, and although in some instances having two threads was faster than a single thread, the difference was too small to be significant. However, once four threads are used, the program would take almost half the time to run. Similar to adding two threads, having 8 threads didn't significantly improved the runtime compared to four threads. When running parallel programs, one must consider the overhead which can affect the performance and sometimes lead to worse runtime. Overall, increasing the number of threads usually leads to shorter runtime although there is a limit as to how many threads make a difference; in this instance, the best performance is best achieved by using 4 threads.

Observations/Comment

This was fun lab to do; I started testing it with other words and files too. The output was pretty consistent which made me glad.

Execution

For default run: python3 MapReduce.py

For custom input: python3 MapReduce.py --t #

In which # represents any positive number bigger than 1

Output from the dumpCPUInfo.sh program

Intel 4 Core i7-6500U CPU @ 2.50GHz