CS1190 Parallel Computing
Daniel Alejandro Rivera Estrada ID#80699287
Assignment 2 Parallel Map Reduce

For this assignment I had to use the re library for the find all function and the iterator. It was interesting to see how the function could find the instances of the words on the documents.
In my implementation I used two functions, one that initializes the parallel section and contains the critical section with the lock. And the other function will be the one that uses the iterator to open all the documents and uses find all to find the occurrence of the words on the list

```
import pymp
import time
import threading
import re

def countInstances(document,word_list,wordCount): #this is an auxiliary function that helps count all the instances of the words
    f = open(document, 'r')#open the file in read mode
    for w in word_list: #find all the words in the list and add it to the count on the dictionary in lowercaps
        wordCount[w] += len(re.findall(w,f.read().lower()))
        f.seek(0)#return to the beginning of the file for the next word search

def findAllWords(documents,word_list):#this will be the parallel section of the program
    wordCount = pymp.shared.dict()#shared dictionary
    with pymp.Parallel(8) as pimp:
        for word in word_list:
            wordCount[word] = 0
        safe = pimp.lock#declaring the lock for the threads
        for d in pimp.iterate(documents):
            safe.acquire()
            countInstances(d, word_list, wordCount)#use the auxiliary function
            safe.release()
    print(wordCount)#at the end print the whole dictionary with the values
```

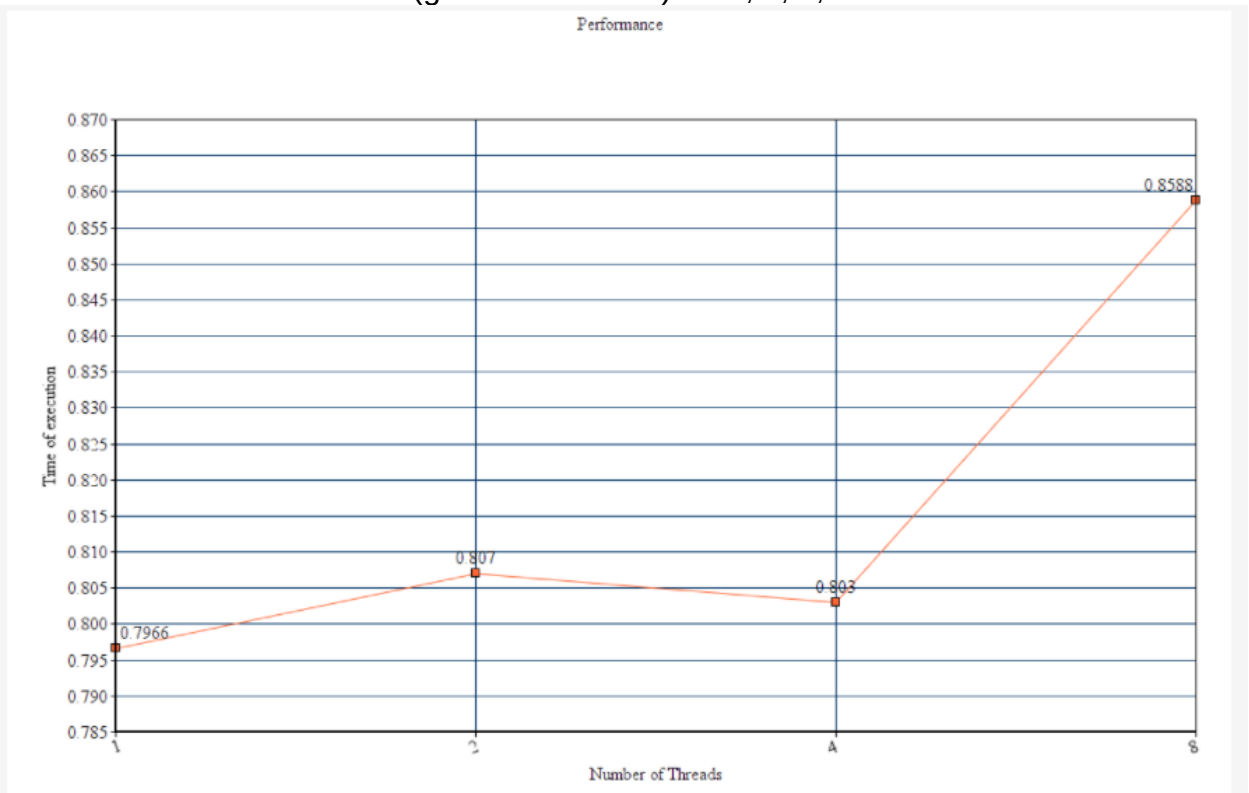
I had problems at first understanding how the regular expression “find all” worked, but after looking at some tutorials I understood how to use it. The main problem I had was deadlocks, the program would start and would stay like that until I stopped the execution, I also had problems updating the counts of the words, because they would appear incomplete.
In the end the reason why I had deadlocks and the incomplete counts for the words was because I was trying to use a lock from the threading library. Once I changed to a pymp lock the counts started being complete, the problems of deadlock were also solved.

The assignment took me about 5 hours to finish including all the time I worked on it throughout the week. I believe that it may have taken me longer than I expected even though I anticipated it would take me longer to understand the map reduce technique.

All in all, I think that the program works as intended because the counts were the correct amounts as shown on the assignment page

```
student@linux-4oae:~/pythonTestProgs> python3 ./ParallelMapReduce.py
{'hate': 332, 'love': 3070, 'death': 1016, 'night': 1402, 'sleep': 470, 'time': 1806, 'henry': 661, 'hamlet': 475, 'you': 23306, 'my'
: 14203, 'blood': 1009, 'poison': 139, 'macbeth': 288, 'king': 4545, 'heart': 1458, 'honest': 434}
student@linux-4oae:~/pythonTestProgs>
```

Performance measurements (given in seconds) for 1, 2, 4, and 8 threads



I did not expect these results from a parallel program, it seems like the time of execution is roughly the same until the number of threads gets to 8. I believe this happens because I am using a lock for the critical section and only one thread is allowed to update the count at time. With 8 threads there is just too much overhead for it to get faster.

The CPU info of my machine

output.txt

```
model name» : Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
      4      36     216
```