Daniel Alejandro Rivera Estrada ID 80699287
Map Reduce MPI Report
CS 1190 Parallel Computing

The purpose of this assignment was to use an MPI mapreduce implementation. The program is supposed to find the number of instances from words in a list by opening and reading eight files.

Some of the problems I encountered while doing this program was that the threads would keep waiting for another thread to perform an action that would never happen, this caused the program to never finish and I had to stop the execution.

Another problem I had was finding a way to send the data to the other threads from thread 0, at first I was trying to use the bcast() function from the communicator. This function would also cause the problem to wait forever and it would not finish the execution. After not being able to solve this problem, I decided to use the send() and recv() functions but I didn't really know how to send all the data to multiple threads at a time.

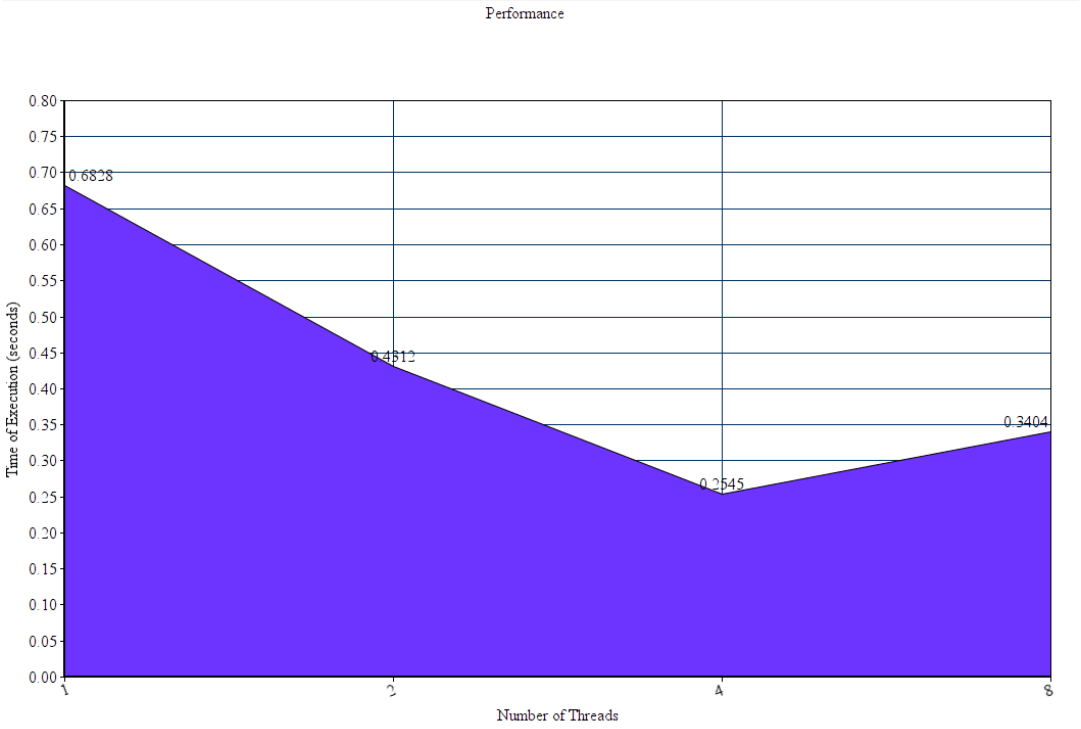
In order to be able to send all the data and receive it, I used the rank variable and for loops to send and receive data from/to all the threads. Although I believe there might be more efficient way to achieve the same thing, the for loops seemed to work well and all the threads got the dictionary and were able to send it back to thread 0.

```
13
14
15  def findAllWords(documents,word_list):      #this will be the parallel section of the program
16      rank = comm.Get_rank()      # get our rank (process #)
17
18      if rank == 0:
19          wordCount = {}
20          for word in word_list:
21              wordCount[word] = 0     # Creating the dictionary for all the threads
22          if threads >=2:# if we have more than one thread it'll send all the data to the other threads
23              for i in range(1,threads):
24                  comm.send(wordCount,dest=i,)
25      else:
26          wordCount={}
27          wordCount = comm.recv(source=0,)#tag=<tag#>
28
29      docs = len(documents)  #number of docs 8
30      perThread = int(docs/threads)#docs per thread
31      current = perThread*rank # current index for the document list
32      limit = int(perThread*(rank+1))      #limit of the part assigned to this thread
33
34      for d in range(current,limit): #each thread gets a certain number of docs depending of the number of threads
35          countInstances(documents[d], word_list, wordCount)#use the auxiliary function
36
37      if rank != 0:# all the other threads are sending their count to thread 0
38          comm.send(wordCount,dest=0,)
39
40      if rank == 0:#we gather all the counts from all the dictionary parts into one
41          result=wordCount
42          if threads >= 2:
43              for i in range(1,threads):
44                  data = comm.recv(source=i,)
45                  for k, v in data.items():
46                      if k not in result:
47                          result[k] = v
48                      else:
49                          result[k] += v
50
51          return result
52
53      #return wordCount#at the end return the whole dictionary with the values
54
```

The error that I wasn't able to solve was the bcast() function causing the threads to wait forever. But since I decided to use recv() and send() instead, it didn't end up affecting the code.

I believe that the project took me about 8 hours to complete adding all the time that I worked on it. At first I was a bit confused on how to implement the code but after talking to the professor and looking for functions online I was able to finish it.

Performance:



Analysis:

I believe this implementation's performance trend was similar to the pyMP implementation. From 1 thread to 4 threads the time of execution is reduced significantly, but when the program starts using more than 4 threads the performance starts diminishing. The more threads the program uses, the more overhead there is and more need of communication, this causes the time of execution to go up.

One thing that I noticed with MPI is that you have more control over the parallel section of the program. The rank variable is very useful to manipulate the threads. It also gives you more control over what data is shared and not. This may have its advantages but it also has problems because you need to keep track of which threads are sending and which will have to wait, missing one of those details may cause the program to wait forever.

Instructions:

To be able to run the program with the desired number of processes, please use this command:

mpirun -n <number of processes> python3 <nameOfFile>.py

The program only handles 1,2,4 and 8 threads. Although I didn't create exceptions to handle other number of threads, the program may not be able to run properly with an odd number of threads

```
student@linux-4oae:~/pythonTestProgs> mpirun -n 1 python3 ./MapReduce2_MPI.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}
Time for execution: %s 0.682883524998033
student@linux-4oae:~/pythonTestProgs> mpirun -n 2 python3 ./MapReduce2_MPI.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}
Time for execution: %s 0.4312627249964862
student@linux-4oae:~/pythonTestProgs> mpirun -n 4 python3 ./MapReduce2_MPI.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}
Time for execution: %s 0.2545843400002923
student@linux-4oae:~/pythonTestProgs> mpirun -n 8 python3 ./MapReduce2_MPI.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}
Time for execution: %s 0.3404618130007293
student@linux-4oae:~/pythonTestProgs>
```