Performance Evaluation

Introduction

The goal of this report is to measure how efficient of using pmap over map on row operations in our dataframe. Efficiency will only be measured in the execution time that is required to execute our implemented Rower subclasses on data we provided. In order to ensure an accurate measurement of how long the program will take to map row data in the dataframe in two different ways , our regular map and pmap with multi-threads functions will be tested a number of times with a variety of inputs. Since implementations of each Rower subclass will be different and the second Rower subclass is required to be more expensive than the first one, we split them into two parts when results of performance are analyzed.

Implementation

pmap(Rower* r) method creates a thread pool and initialize two threads. The program will make two deep copies of the given rower, pass them as a parameter of the function passed into one of the threads. Each of the thread will perform the given task on half of the dataframe. Before pmap() returns, we use thread.join() in a for loop to wait for both of them to finish. After both threads have joined, each copy of the original rower calls join_delete(Rower* original) to aggregate their intermediate result.

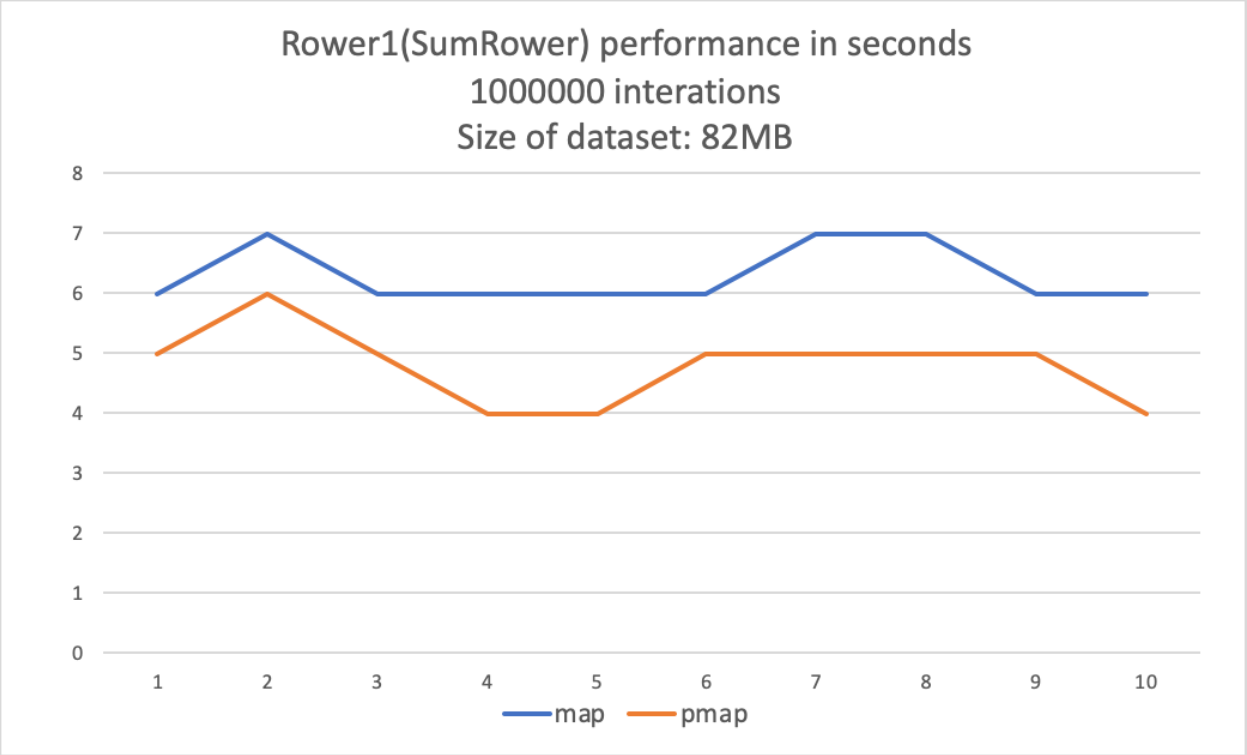
Description of the analysis performed

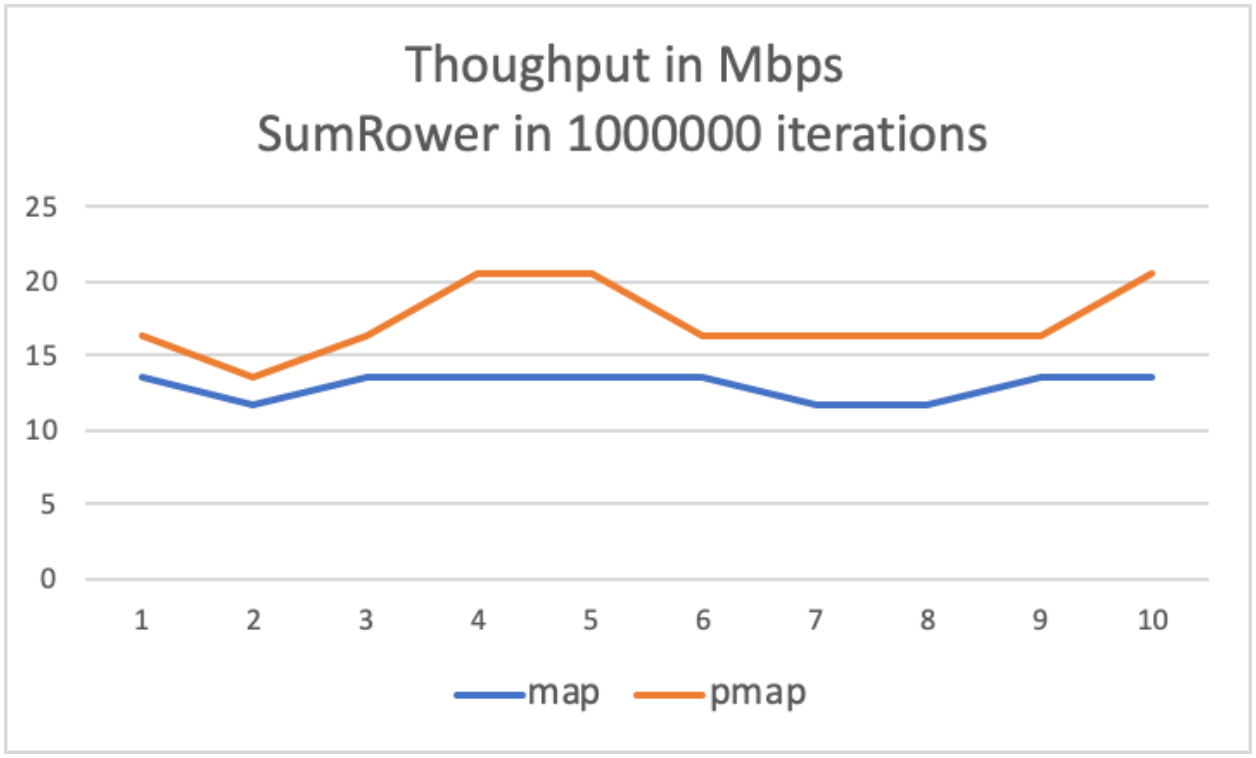
The dataset we used for testing is generated by ourselves. In our test script, we are loading exactly the same data to each row in each iteration before rows are added to our custom dataframe. In order to benchmark performance of both map and pmap functions, we will measure the time that is taken to load data and map all rows that reside in the dataframe. The timer will begin when our test script starts to load data to the first row in the first iteration and terminate the program when all maping operations are done and output time interval that we want to evaluate. Since we have provided implementations of two Rower classes, each map function will be tested 20 times for each Rower class(10 times for each size of dataset) such

that 80 testing will be performed in total. This was done in order to reduce inaccurate variance that any one iteration could produce.

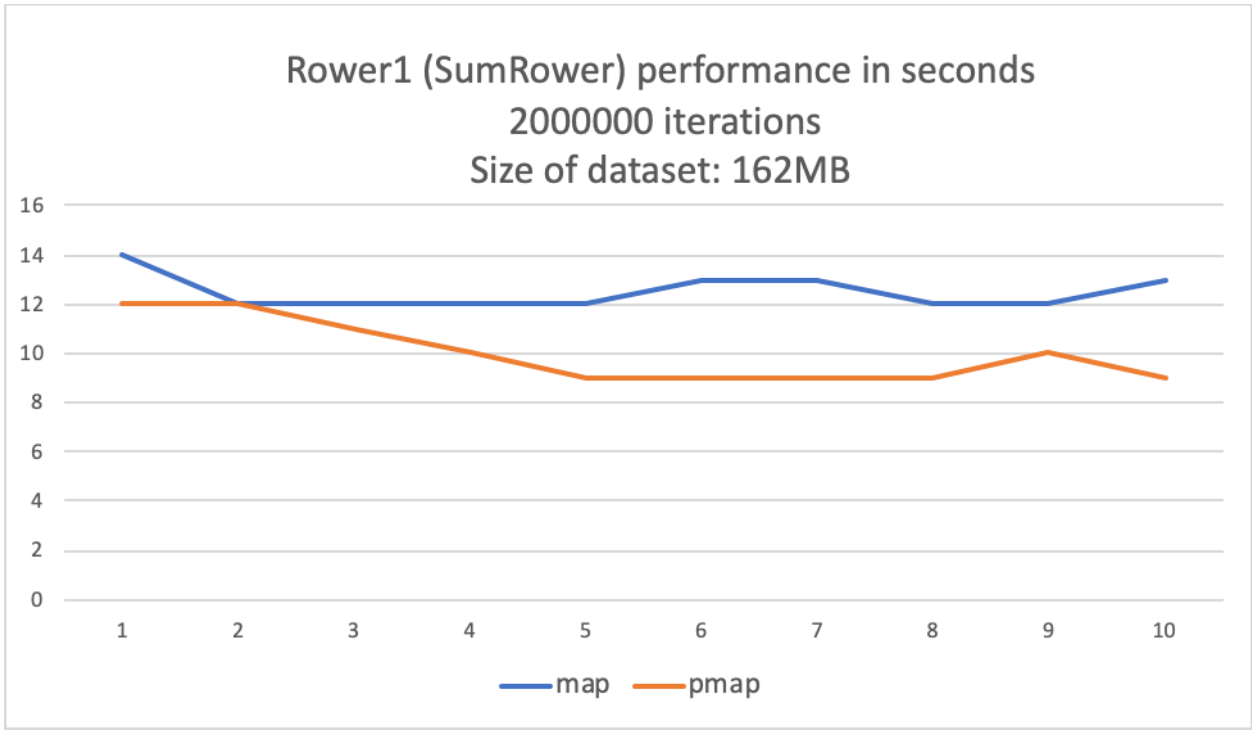
Comparison of experimental results

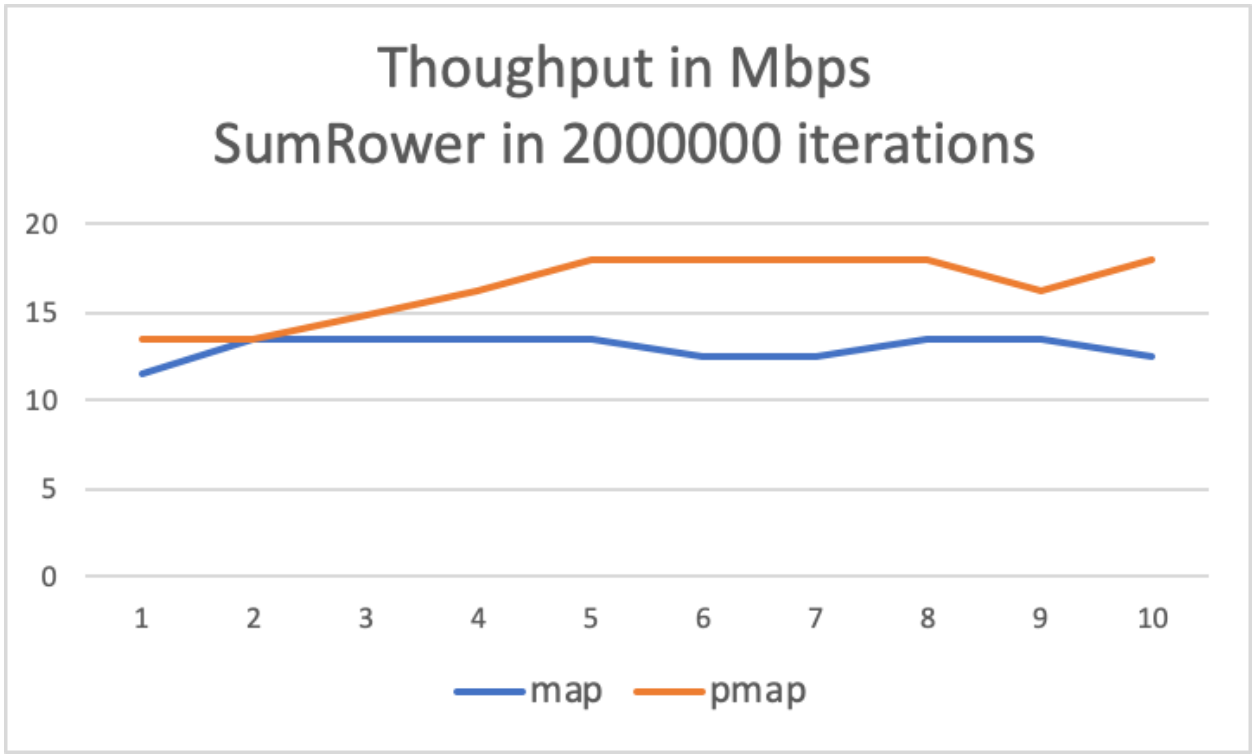
For the implementation of the SumRower subclass, we are basically trying to sum up the first number in each row such that it will end up doing arithmetic 1000000 times since we are repeatedly adding a row to our dataframe with 1000000 intertations in this scenario, which is guarantee that the dataset we work on is big enough to perform measurement precisely. When we are mapping data in the dataframe by using the first Rower object, we found that the use of regular map function is apparently slower than the use of pmap function. The chart graph below shows that time is taken to execute map and pmap separately in 1000000 iterations.



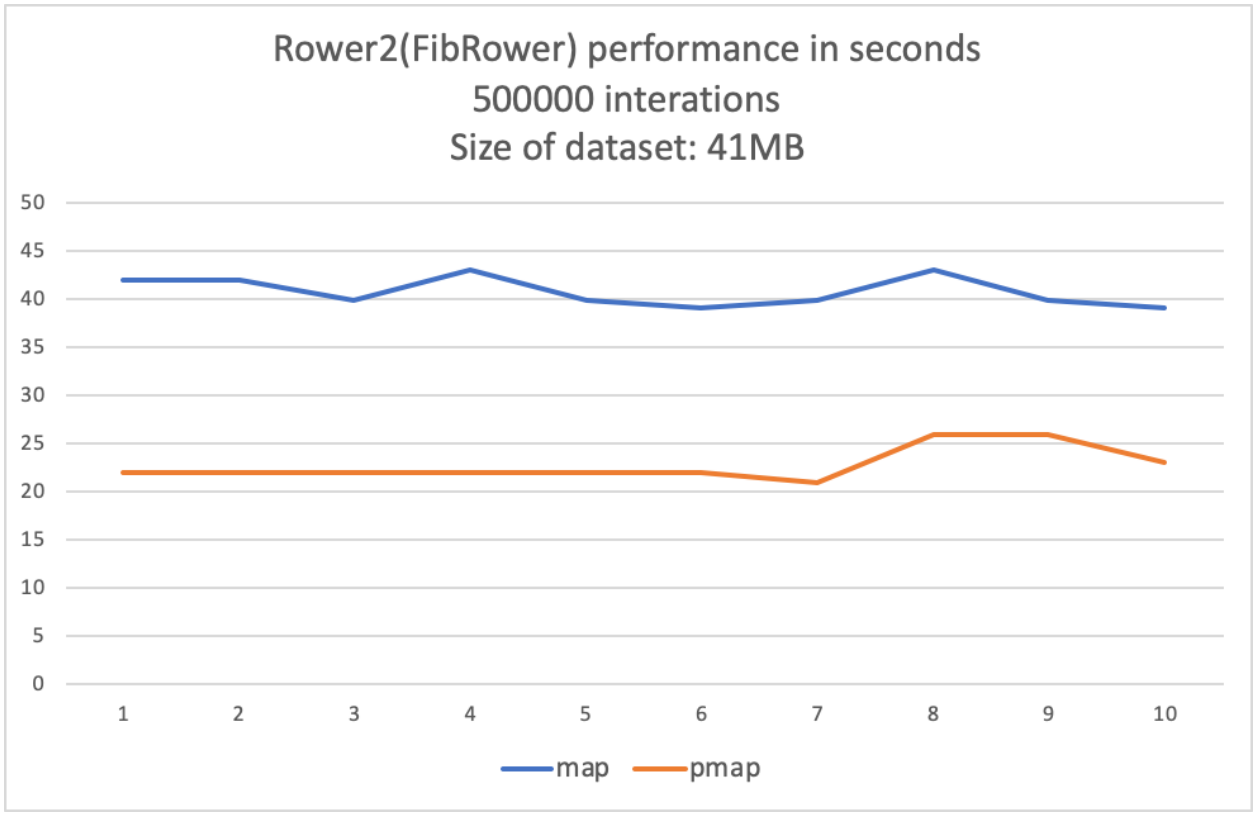


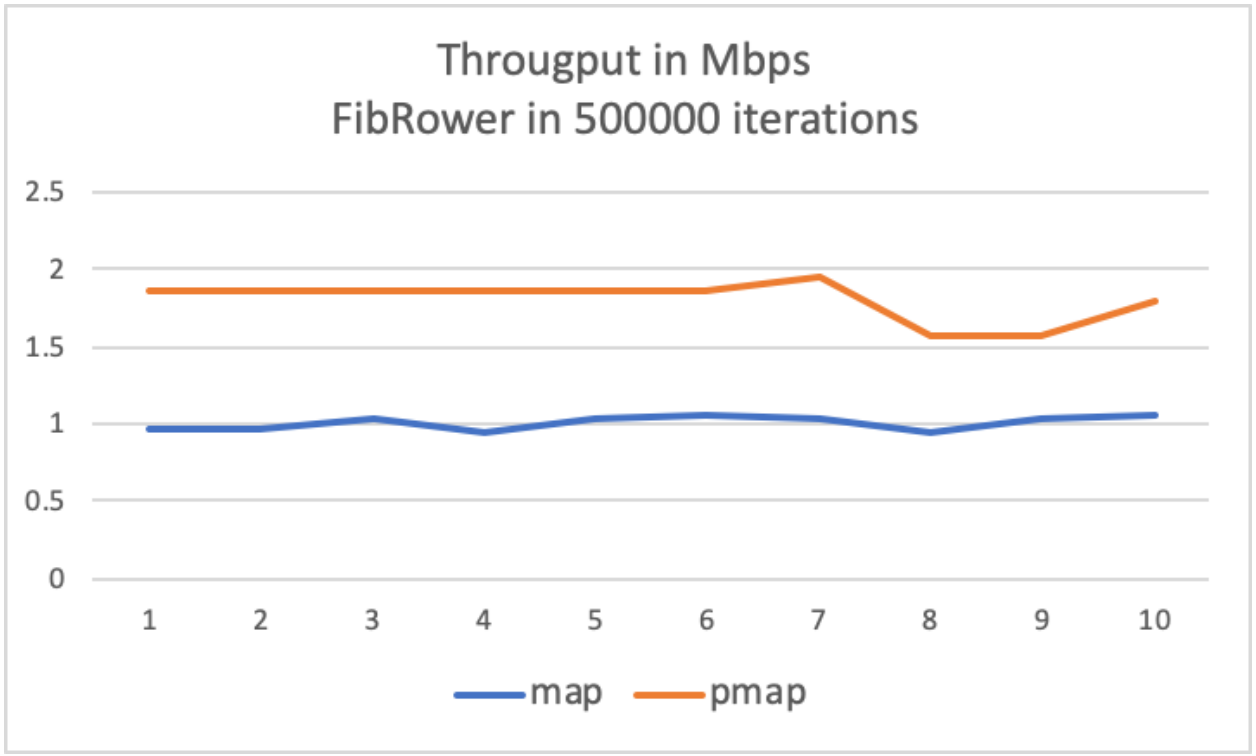
Another char graph below demonstrates the same Rower is used for testing, but with different iterations. (2000000 iterations).



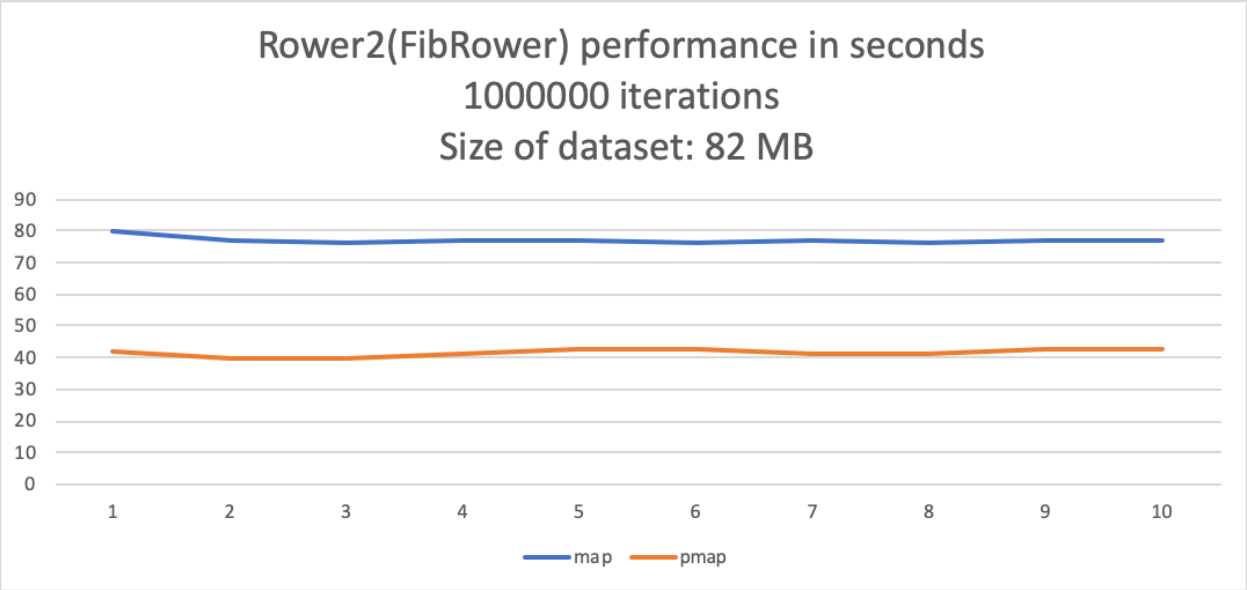


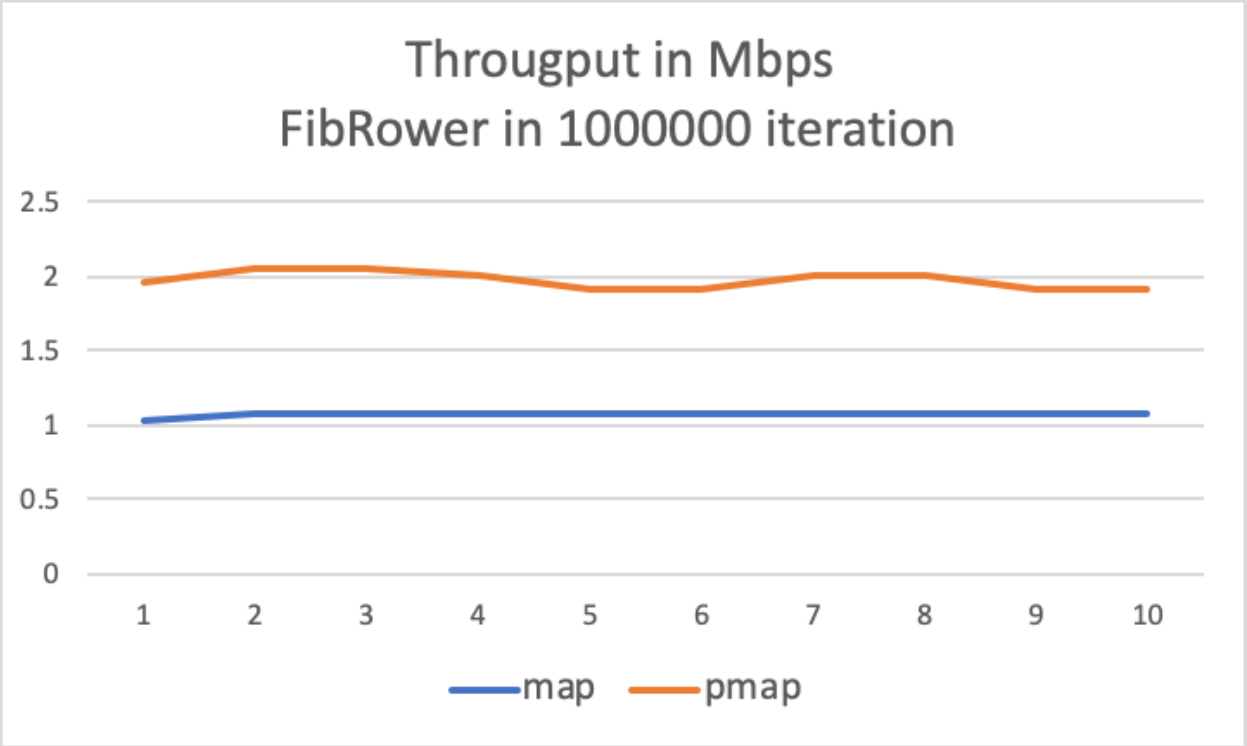
Although multithreading pmap allows our program to process slightly faster than using regular map overall, there is no huge difference between each other.
The implementation of the second Rower(FibRower) is more expensive than the first one. It is achieved by calculating the sum of Fibonacci numbers of the first integer in each row and then sum them up when the script is iterating rows. Because it is way more costly than the first Rower class, performance of using pmap will obviously outperform when using map function. In the chart graph below, we can tell that by using multi-threading pmap, on average, time intervals are shorten to half of original time spans that are taken to load and map data in the dataframe.


Rower2(FibRower) performance in seconds
500000 interations
Size of dataset: 41MB
50
45
40
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
map
pmap


Througput in Mbps
FibRower in 500000 iterations
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
8
9
10
map
pmap

In the same way, in order to make sure the measurements of performance are accurate, we choose two different sizes of dataset to work on. In this scenario, two functions have been tested under 1000000 iterations. See chart graph below.





Threats to validity

1) Unit of Measurement

The test script use time() method to measure execution time. The function returns execution time in second, as a result, the execution time might be a little off from the actual CPU time by less than 2 seconds. The fact that we can only measure execution time in seconds would make our final result deviate from the actual ones, the degree of deviation is negatively related to the input file's size. The larger the input file and time complexity of the rower's operation, the less noticeable the deviation.

2) Testing Environment

We used a 2 Core MacBook as our test environment. This test has not been done on a single-core computer or a computer with more than 2 cores. However, we can expect that for multi-core computers, the result would not deviate a lot. As for computer with a single-core, using multi-threading might not bring huge performance improvement.

Conclusion

In general, after the completion of this performance measurement, we found that a function which is executed with multithreading will be approximately 50%faster than other regular mapping functions if the dataset that program used to process has adequate contents. In other words, the dataset needs to be big enough to be performed measurements, otherwise, there would be no significant improvement on run-time speed