FIT 3143
Assignment 1 Report
Name:Austin Sing Jet Wong
Id: 32933975

Introduction:

Bloom Filter Algorithm has been chosen as a parallelization study case.
The Bloom Filter is known to be a space-efficient probabilistic data structure which a given data set is hashed and inserted to a hash table, but the special properties of the Bloom Filter is that it could never has False Negative but False Positive as query of the item only determine the possibility existed or definitive not in set . However the downside is that the Item cannot be deleted once inserted as it could be causing the “existence” of the other items. Thus ,the larger the data set, the higher the false positive, but it depends on the size of the set m, and numbers of hash functions, which is further derived to some functions that could obtain an optimised numbers of m and h from is (1+1/m)^h(by the probability of the collision by number of hashing,h) .

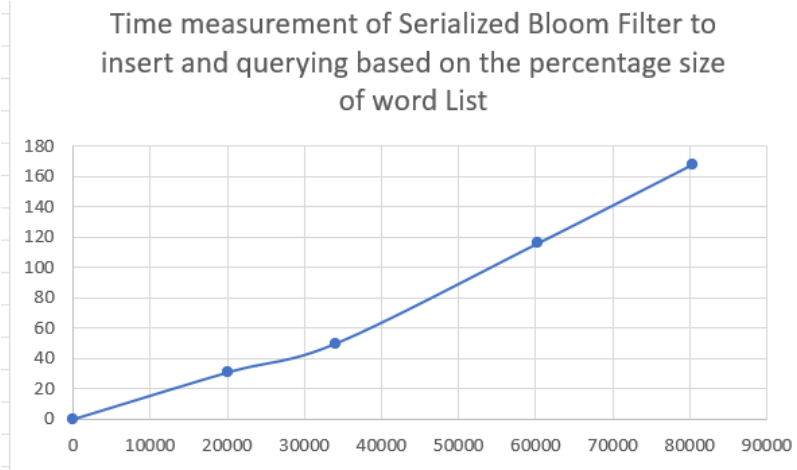
$$m = -\frac{n \ln(p)}{(\ln(2))^2}$$
$$k = \frac{m}{n} \ln(2)$$

The algorithm that I designed will accommodate the equation I used above with the use of a heavily-modified universal hash function could simulates h function and yet getting a Uniform distributed range of numbers . A pseudo-random number generator is also included to increase the uncertainty and yet reliability of a single hashing and also
the padding of each element to increase overall computational cost.

Universal Hash: a set of map that which provided a prime numbers,p >m and random number a that 0< a <= m-1 which results in a uniform distributed set of numbers(hash) to avoid collision which its general equation is

$$h_i = \sum_{i=start}^{i} (h_{i-1} * a_i + x_i) mod p$$

So the overall time and space complexity is a pseudo-polynomial complexity O(H x N X L) and O(M) which m is the size of the bit array and K is the number of hash functions and N is the number of the elements to be inserted or query and L is the maximum length of the string by the Universal Hash Function.
Diagram 1: The diagram based on the percentage of the size of the item to be inserted(from Appendix B)



Pseudocode:

BloomFilter:

```
UniversalHash(stri,seed,primes,index,m)->
        h=seed
        for i to index in stri do
                h+=(h*seed + stri[i]))%primes
        Return h%m;


InsertItem,QueryItem=readFile()
T =  bitArray(m)
MultiInsert():
        C=bitArray(len(Insertwords))
        for words in InsertItem do :
                 isInserted=0
                stri<-padding(words,lengths)
                for i in len(words):
                        h_i=Universal_Hash_i(stri…,m)
                        If (bitArray(h_i)==0:

                                flip(bitArray(h_i))
                        Else:
                                inserted=1

                if (inserted==Nh)
                        flip(C(words))

          return C
```

```
MultiLU():
        LU=bitArray(len(Querywords))
        for words in QueryItem do :
                 isInserted=0
                stri<-padding(words)

                for i in len(words):
                        h_i=Universal_Hash_i(stri…,m)
                        If (bitArray(h_i)==1:


                                inserted=1

                if (inserted==Nh)
                        flip(bitArray(LU_words))
         return LU


 NaiveCompare()
        TT=bitArray(len(Querywords))

        for x in QueryItem do :
                 isInserted=0
                stri<-padding(words)
                for y  in InsertItem
                        If x==y:
                        flip(TT[x])
                       Break
Return TT
False Positive Rate()
        falsep=0
        truene=0
        For words  in TT do{
                if(get_bit(TruthArr,words )==0){

                (get_bit(LU,words)==1)?falsep++:trueneg++;

                }
        Return falsep/(trueneg +falsep)
```

writeFile(LU,TT)

Word List Insertion and Query:

To has a consistency time measurement, each words is considered unique which Obtained through the three file given by the teaching teams, which the one inserting is based on the SHAKESPEARE while Query is based on the combination of the three text files,SHAKESPEARE, MOBY_DICK and LITTLE_WOMEN.
The tools that I used to preprocessing is simple python program,
which ended up with 27289(spUnique) and 53148 words(combination).

The reason the Bloom Filter is parallelism :

1) Least Data dependency - The part of data dependency os least on the long run as one another as long as the Algorithm procedure is followed step-by-step.

2) Scalability of performance
   The reason above identifies that it might be parallelized since it usually deals with a large set of numbers and the insertion and Query can be done in parallel respectively since each hashing of the item is independent of the other item but the content of its own.

3) Complexity Enhancement - The Nature of the algorithm is able to add extra complexity for the bloom filter without screwing up the understructure logic of the algorithm.

Bernstein condition:

- $I1 \cap O2 = \varnothing$ (Anti dependency)
- $I2 \cap O1 = \varnothing$ (Flow dependency)
- $O1 \cap O2 = \varnothing$ (Output dependency)

Which the input and output of Tn is represented by I n and O n

In Bloom Filter, we will be analysed through the parts that can be parallelized (insertion and query) and cannot be (reading and writing)

Insertion and Query :

```
for(index=0;index<size;index++){

    char*stri=padding(input[index],primes,NumFunc,EXTRAPADDING);
    h=HashInsert(stri,insertTable,primes+primes*m,primes,m,NumFunc);
```

Hash Inserting and Querying

Which given a hashing of the string is independent while inserting and querying which, as the parameter of the hash function , since we just use the given value which is considered constant.

- $I1 \cap O2 = \varnothing$
- $I2 \cap O1 = \varnothing$
- $O1 \cap O2 = \varnothing$

The three dependencies can obviously be nullified since all the parameters are constant as well as the words are also independent .

thus it can indeed be parallelised since the hashing of multiple values can be done parallelise But can the hash function itself can it be parallelize

```
for( int i=0; i<NumFunc*partition;i+=partition){
  int RNa=RN(i*NumFunc,pow(2,31)*(i),(int)stri[i],m);
  int h = UniversalHash(stri,RNa,primes,i,m);
```

```
strawberry
121
126
90
78
13
watermelon
125
217
189
193
36
```

which it has set to the partition and but does it independent to each other, which the hash function is simulate to be a map of function since based on the code:

- $I1 \cap O2 = \varnothing$

Since the second hashing of the function does not depend on the output does not depend on the previous thus ,there is no anti dependency to each other

- $I2 \cap O1 = \varnothing$ (Flow dependency)

It does not has flow dependency since its second input does not has relationship with its first output which there is no flow dependency

- $O1 \cap O2 = \varnothing$ (Output dependency)

It does not have output dependency since the hash function does not build up on the previous hash only correlates with the input of that instance.

Thus, most of the reason is that output is correlated to that instance of input which depends on the correlated index loop which proves that all instances and output are independent of each other as race conditions are not existential in both inserting and querying .

Thus, it is proved that the map of hash function for inserting and querying is independent to each other, which it is indeed can be potentially parallelized

Theoretical Speed Up

To calculate the theoretical speed up, Amdahl's law is used to measure the theoretical speed up of the program.

$$S = \frac{1}{(1-f)+\frac{f}{p}}$$

Where f is the amount of fraction that the program allowed and p is the number of processor

The fraction of the program that can be parallelized in the algorithm is the time for inserting, querying, naive comparison and calculation of false positive rate.
So the fraction that can be parallelized is based on the formula ,

$$f = \frac{t^s}{t^t}$$

t_p= the time taken by the program that can be parallelized and t_t= overall time of the program.

From serial time measurement with several decreasing size of input and output from serial time reading,(appendix C)
f1=156/168 = 0 .92,
f2=108.692/116.47=0.93, with 75% percentage of size
f3=47.71/49.67 = 0.95 with 50% percentage of size

f4=30.94/30.525= 0.97 with 25% percentage of size
Which we can observe that as the data set grows, the portion of fraction time of the parallelized time grows smaller

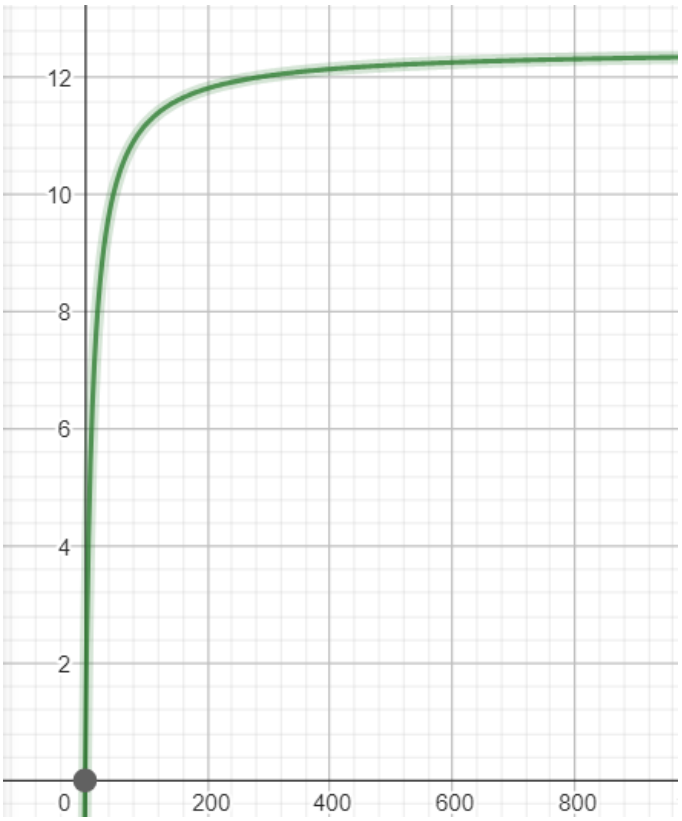
which considers the overall complexity of the algorithm , the minimum fraction is chosen as the amount of program that can be parallelized is 0.9 if given a file with a random data set which consists of a unique data set.
So the theoretical speed up time by Amdahl's law in generalised form is

$$S = \frac{1}{(1-f)+\frac{f}{p}}$$
$$S = \frac{1}{(1-0.92)+\frac{0.92}{p}}$$

which the exact amount of the theoretical speedup is

$$S = \frac{1}{(1-0.92)+\frac{0.92}{p}}$$
$$S = \frac{1}{(1-0.92)+\frac{0.92}{p}}$$

S:speedUp factor



p:Number of processor

So the theoretical speedup based on the equation of above by on my device

$$S = \frac{1}{(1-0.92)+\frac{0.92}{16}}$$
$$p = 16 \qquad \text{from appendix A}$$
$$S = 7.3$$

And the limitation of the theoretical speedup is given with as $p- > \infty$

$$S = \frac{1}{0.08)}$$

$$S = 12.5$$

The limitation speedup given by Amdahl's law is 12.5 as number of processor as $p -> \infty$

Design of the parallel algorithm:

Measurement: appendix C of 3 different cases

So the part that has been parallelized in the Bloom Filter is inserting,querying, naive comparing and the calculation of the false positive rate.

What if two hashes of two items flip the bit in the same position since the other hash position is occupied other than this, which might lead to data Corruption due to race condition, while inserting which I added an exclusive lock (critical).

However, there is a part which is an integer variable and the calculation of false positive which determines if the item is inserted or not, which to avoid race condition if it is parallelized, a reduction, atomic or critical is needed but I used reduction.

So, There is a 3 design of the parallel architecture for the parallelized Bloom Filter for inserting and querying which is based on the

1. Data - partition parallelism
   The straightforward design, using the full capabilities of the maximum threads on the device and each threads worked on each string as which ideally each will be working amount of n/p in both insertion, which it will works on the data set independently
2. Nested parallelism
   The amount of threads for working of each item is limited but the threads will have the same amount of its own slave threads which assists the hashing of each item which since the hashing of the words will be independent, the hash function will run longer as the partitions is getting larger from index 0 to partition, which it will run with Initialise the value by parallel region of 4 which rest of them are slave threads to 4 while inserting

3. Parallelised hashing for inserting and querying.
   So a thread will be working on a data set, which following the hashing and querying, The master thread will spawn the amount of threads left available and working on the Item which involving on the hashing

So the guided has chosen as my schedule since the partition of the hash starting from 0 to Index of partition which the numbers of loop will keep getting, which guided help the program by starting off with a larger set of partitions of a divided word and decremental as which could help with the overall overhead parallelism as the size of the chunk is decreasing

In Naive Comparison Function it is run by nested for loop, which it can be parallelized using the outer loop since the inner loop has a break, which even if the #omp did not restrict, it will be pointless to run full course given if it breaks early, so only the outer for loop is parallelized.

The parallelization of false positive calculation can be done by simply using #omp reduction to compute its tn and tp to obtain the same result to avoid race condition while the for loop to run is parallelized

Pseudocode:
Parallelized Bloom Filter based on the above design:

```
Input(Choose 1 between 4):
switchCase{
        Case1:
                Total _threads= max threads()
                Thread per region = 1
        Case 2:
                Total _threads= 4
                Thread per region = thread per region/4
        Case 3:
                Total _threads= 1
                Thread per region =max threads()

}
InsertItem,QueryItem=readFile()

MultiInsert():
        C=bitArray(len(Insertwords))
        #pragma omp schedule(dynamics)  num_threads(total_threads)
        for words in InsertItem do :
                 isInserted=0
                stri<-padding(words,lengths)
                #pragma omp schedule(guided,len(stri)/threads_per_region)
num_threads(threads_per_region)
                for i in len(words):
                        h_i=Universal_Hash_i(stri…,m)
                        If (bitArray(h_i)==0:

                                flip(bitArray(h_i))
                        Else:
                                inserted=1

                if (inserted==Nh)
```

```
                    flip(C(words))

        return C
MultiLU():
    LU=bitArray(len(Querywords))
    #pragma omp schedule(dynamics)  num_threads(total_threads)
    for words in QueryItem do :
            isInserted=0
            stri<-padding(words)
            #pragma omp schedule(guided,len(stri)/threads_per_region)
    num_threads(threads_per_region)

            for i in len(words):
                    h_i=Universal_Hash_i(stri…,m)
                    If (bitArray(h_i)==1:


                            inserted=1

            if (inserted==Nh)
                    flip(LU(words))


        return LU



NaiveCompare()
    TT=bitArray(len(Querywords))
    chunk= ceil(NumLookUpSts/total_thread);
    #pragma omp parallel for schedule(static,chunk)

    for x in QueryItem do :

            isInserted=0
            stri<-padding(words)
            for y  in InsertItem
                    If x==y:
                    flip(TT[x])
```

```
                    Break
        return TT
False Positive Rate()
        falsep=0
        truene=0
         #pragma omp parallel for schedule(dynamic) shared(i) reduction(+ :
falsep,trueneg)
        for words  in TT do{
                if(get_bit(TruthArr,words )==0)

                                (get_bit(LU,words)==1)?falsep++:trueneg++;

        Return falsep/(trueneg +falsep)

writeFile(LU,CC,QueryItem)
```

Analyse and evaluate the overall performance of algorithm

The formula actual speedup time of speed up

$$S = \frac{t^s}{t^p}$$

Where t^s is the total time taken to run in serial time and
t^p is the total time taken to run in parallel time
Based on the Parallelized time measurement (appendix C)

From appendix B and C
Which 2nd cases is based on the improved 2nd case

DataSet percentage	Time Completed /Serial Time	1st Case	2nd Case	3rd Case	Actual RunSpeed ,$S = \frac{t^s}{t^p}$
Total SIze: 80437					
100%	168.04	25.473	36.35	191.45	S1: 6.58 S2: 4.6 S3: 0.87
75%	116.47	16.11	24.439	164.89	S1: 7.27

					S2: 4.765 S3: 0.7
50%	60.91	6.98	13.94	69.99	S1: 8.81 S2: 4.336 S3: 0.88
25%	30.998	3.34	5.82	39.959	S1:9.12 S2:5.23 S3:0.766

From Appendix D:
Diagram For Case 1

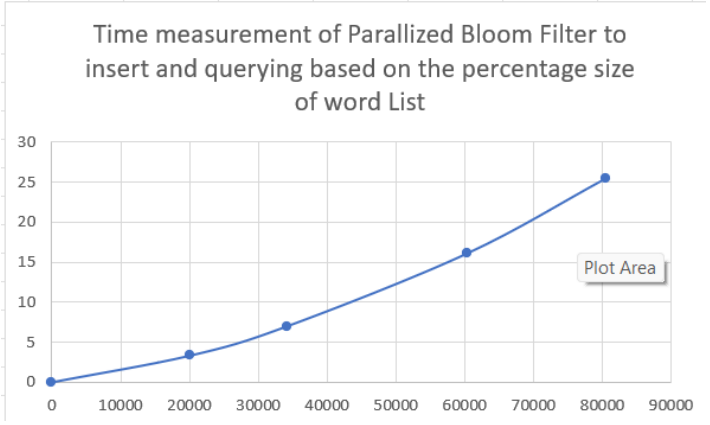


Diagram for Case 2

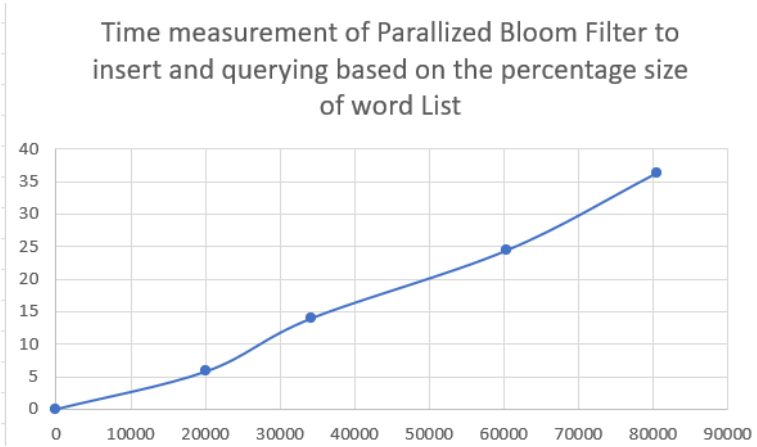
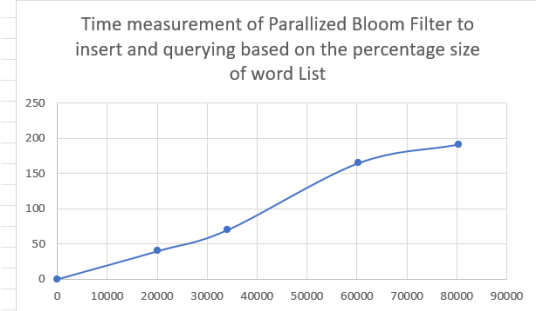


Diagram for Case 3



So the actual speed up of the overall , that we will based on the largest set , which usually is the worst case) i

in case 1:

which is by $S = \frac{t_s}{t_p}$ = 6.58 where t_s =168.04 and t_p=191.45

in case 2:

which is by $S = \frac{t_s}{t_p}$ =4.6 where t_s =168.04 and t_p=36.35

In case 3:

The speed up factor is neglected since there is parallel overhead and inefficient use of cases in these cases.

Analysis and Output

We can observe that as the data grows, the actual speed up is smaller as the data is larger. It could be caused by the potential overhead as it tries to synchronise the threads and then the scalability of the program keeps decreasing , thus the smaller the data, the smaller the significance of the overhead, since it did not contribute much to the cause. which according to the theoretical speedup, which as serial parts grows, lower its speed up Which overhead parallel time accumulate outweigh the par

Case 1: the most common with not much adjustment as well as and let it run dynamically as extra computational uncertainty complexity of the hash functions can contribute the different amount of hashing time between asynchronous of the timing of the data so it could be better to set it up as dynamically which as the dataset has a different length which threads of another may result to be finish earlier

Case 2 :

It is able to achieve a somewhat good speed up factor t since it needs to deal with parallel overhead, which could stagnant the overall performance, which further optimization is needed in terms of controlling its number of threads spawn and the partition of chuck according to h and m, which a small optimization is needed

Case 2:

```
if(NumsHashF<16){
        total_thread=omp_get_max_threads()/2;
        thread_per_region=2;
        }
```

```
else{
        total_thread=4;
        thread_per_region=omp_get_max_threads()/4;
       }
```

```
Enter the number betweem 1 and 4 :   2
total number of threads:             4
total number of threads per region:  4
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   53148
SIZE Of BIT ARRAY:                   254714
Number of Hash Function:             7
Next Prime Value:                    254729
Overall Reading Files time (s):              13.176398
Overall Insert time (s):                      13.376195
Overall Look Up time (s):                     28.614381
Overall Naive Comparison time (s):             0.835461
Number of False Positive::           615
Number of True Negative ::           25368
Rate of False Positive::             0.023669
Overall False Positive Computational time (s):      0.835461
Overall Parallilized Bloom Filter time (s):        43.661499
Number of False Positive::           615
Number of True Negative ::           25368
Rate of False Positive::             0.023669
Overall Writing Files time (s):              0.138449
Overall time (s):                             56.976345
```

Actual Speed Up Sp= 161/56=2.875

In case 2: Actual Speed Up Sp=4.6

Overall, with different master threads and slave threads, the overall improvement speedup is 1.7 which is quite a huge feat which if we know the Number of hash Function we could possibly control the number of master threads and its slave threads .

Case 3:

It oversubscribed since the number of Hash function is smaller than < p

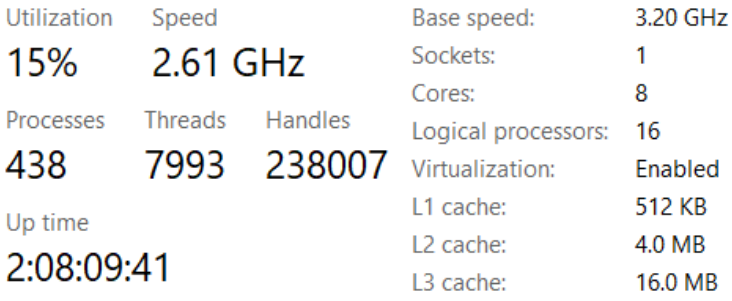
Which lead to waste of resources, and that also lead to parallel overhead since would need to create the parallel region everytime starting a word even if the number of Hash functions is >=p. Thus it will cause deteriorated speed up due to oversubscribe and overhead by each threads are inactive

Thus, Case 1 is the most efficient one and the most least resources usage since it create all the threads at one goes without recreating the threads in-between process which ti could save energy and time which the scalability of the case 1 is among the best between 2 and 3 which case 2 and case 3 have an unreliable runtime

Appendix A: Devices

Number of processor:16(threads)

Cores: 8



Appendix B:

Serial Time Measurement
Serial Time Reading with 27289 unique words to insert and 53148 to query

```
Overall Reading Files time (s):             11.722747
Number of Unique words to Insert:           27289
Number of Unique words to Look Up:          53148
SIZE Of BIT ARRAY:                          254714
Number of Hash Function(Times to be Hashed):7
Next Prime Value:                           254729
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   53148
SIZE Of BIT ARRAY:                   254714
Number of Hash Function:             7
Next Prime Value:                    254729
Overall Reading Files time (s):             11.722747
Overall Insert time (s):                         38.016762
Overall Look Up time (s):                        103.765894
Overall Naive Comparison time (s):                7.200847
Number of False Positive::           14
Number of True Negative ::           26544
Rate of False Positive::             0.000527
Overall False Positive Computational time (s):      7.200847
Overall Parallilized Bloom Filter time (s):       156.184349
Number of False Positive::           14
Number of True Negative ::           26544
Rate of False Positive::             0.000527
Overall Writing Files time (s):               0.135959
Overall time (s):                                168.043060
```

Serial Time Reading with 25% percentage less to insert and query

```
Overall Reading Files time (s):             7.675719
Number of Unique words to Insert:           27289
Number of Unique words to Look Up:          39861
SIZE Of BIT ARRAY:                          191035
Number of Hash Function(Times to be Hashed):5
Next Prime Value:                           191039
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   39861
SIZE Of BIT ARRAY:                   191035
Number of Hash Function:             5
Next Prime Value:                    191039
Overall Reading Files time (s):             7.675719
Overall Insert time (s):                          33.967887
Overall Look Up time (s):                         64.906154
Overall Naive Comparison time (s):                 4.909376
Number of False Positive::           130
Number of True Negative ::           18873
Rate of False Positive::             0.006841
Overall False Positive Computational time (s):      4.909376
Overall Parallilized Bloom Filter time (s):       108.692793
Number of False Positive::           130
Number of True Negative ::           18873
Rate of False Positive::             0.006841
Overall Writing Files time (s):               0.104790
Overall time (s):                                 116.473305
```

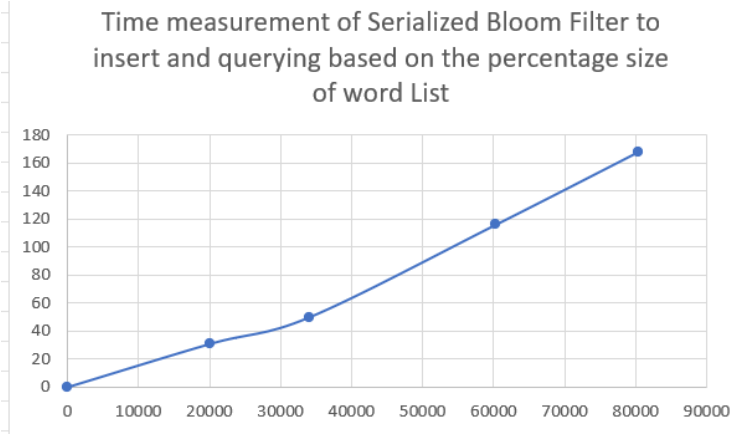
Serial time reading with 50% percentage to insert and query

```
Overall Reading Files time (s):             1.900901
Number of Unique words to Insert:           13645
Number of Unique words to Look Up:          20465
SIZE Of BIT ARRAY:                          98080
Number of Hash Function(Times to be Hashed):5
Next Prime Value:                           98081
Number of Unique words to Insert:    13645
Number of Unique words to Look Up:   20465
SIZE Of BIT ARRAY:                   98080
Number of Hash Function:             5
Next Prime Value:                    98081
Overall Reading Files time (s):             1.900901
Overall Insert time (s):                          15.057957
Overall Look Up time (s):                         30.044239
Overall Naive Comparison time (s):                 1.305678
Number of False Positive::           68
Number of True Negative ::           14342
Rate of False Positive::             0.004719
Overall False Positive Computational time (s):      1.305678
Overall Parallilized Bloom Filter time (s):       47.713551
Number of False Positive::           68
Number of True Negative ::           14342
Rate of False Positive::             0.004719
Overall Writing Files time (s):               0.059162
Overall time (s):                                 49.673615
```

Serial time reading with 75% percentage to insert and query

```
student@3057014857f8:~/project$ ./a.out
Overall Reading Files time (s):                 0.699175
Number of Unique words to Insert:               6822
Number of Unique words to Look Up:              13287
SIZE Of BIT ARRAY:                              63679
Number of Hash Function(Times to be Hashed):7
Next Prime Value:                               63689
Number of Unique words to Insert:       6822
Number of Unique words to Look Up:      13287
SIZE Of BIT ARRAY:                      63679
Number of Hash Function:                7
Next Prime Value:                       63689
Overall Reading Files time (s):                 0.699175
Overall Insert time (s):                              8.129110
Overall Look Up time (s):                             21.163730
Overall Naive Comparison time (s):                     0.479265
Number of False Positive::              23
Number of True Negative ::              9936
Rate of False Positive::                0.002309
Overall False Positive Computational time (s):       0.479265
Overall Parallilized Bloom Filter time (s):          30.251369
Number of False Positive::              23
Number of True Negative ::              9936
Rate of False Positive::                0.002309
Overall Writing Files time (s):                 0.048141
Overall time (s):                                     30.998684
```

Approximate measurement between serial time based on number of elements to insert and query



Appendix C

Parallislied design run time:
Case 1:
Parallelized Time Reading with 27289 unique words to insert and 53148 to query

```
Enter the number betweem 1 and 4 :   1
total number of threads:             16
total number of threads per region:  1
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   53148
SIZE Of BIT ARRAY:                   254714
Number of Hash Function:             7
Next Prime Value:                    254729
Overall Reading Files time (s):              11.572689
Overall Insert time (s):                      3.894805
Overall Look Up time (s):                     8.299826
Overall Naive Comparison time (s):             0.787304
Number of False Positive::           451
Number of True Negative ::           25510
Rate of False Positive::             0.017372
Overall False Positive Computational time (s):      0.787304
Overall Parallilized Bloom Filter time (s):        13.769240
Number of False Positive::           451
Number of True Negative ::           25510
Rate of False Positive::             0.017372
Overall Writing Files time (s):            0.131224
Overall time (s):                         25.473154
```

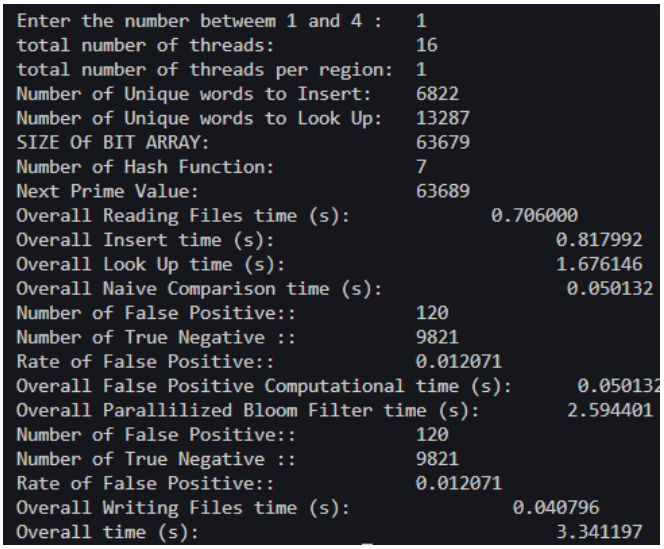
Parallelized Time Reading with 25% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   1
total number of threads:             16
total number of threads per region:  1
Number of Unique words to Insert:    20466
Number of Unique words to Look Up:   39861
SIZE Of BIT ARRAY:                   191035
Number of Hash Function:             7
Next Prime Value:                    191039
Overall Reading Files time (s):             6.273277
Overall Insert time (s):                    2.685077
Overall Look Up time (s):                   6.209715
Overall Naive Comparison time (s):          0.416009
Number of False Positive::           356
Number of True Negative ::           19321
Rate of False Positive::             0.018092
Overall False Positive Computational time (s):      0.416009
Overall Parallilized Bloom Filter time (s):         9.726810
Number of False Positive::           356
Number of True Negative ::           19321
Rate of False Positive::             0.018092
Overall Writing Files time (s):             0.115441
Overall time (s):                           16.115528
```

Parallelized Time Reading with 50% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   1
total number of threads:             16
total number of threads per region:  1
Number of Unique words to Insert:    13645
Number of Unique words to Look Up:   20466
SIZE Of BIT ARRAY:                   98084
Number of Hash Function:             5
Next Prime Value:                    98101
Overall Reading Files time (s):             1.954464
Overall Insert time (s):                    1.703489
Overall Look Up time (s):                   2.940386
Overall Naive Comparison time (s):          0.161892
Number of False Positive::           601
Number of True Negative ::           13632
Rate of False Positive::             0.042226
Overall False Positive Computational time (s):      0.161892
Overall Parallilized Bloom Filter time (s):         4.967659
Number of False Positive::           601
Number of True Negative ::           13632
Rate of False Positive::             0.042226
Overall Writing Files time (s):             0.058516
Overall time (s):                           6.980639
```

Parallelized Time Reading with 75% percentage less to insert and query



Case 2

Parallelized Time Reading with 27289 unique words to insert and 53148 to query

```
Enter the number betweem 1 and 4 :   2
total number of threads:             8
total number of threads per region:  2
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   53148
SIZE Of BIT ARRAY:                   254714
Number of Hash Function:             7
Next Prime Value:                    254729
Overall Reading Files time (s):              11.014742
Overall Insert time (s):                      7.315828
Overall Look Up time (s):                    16.041683
Overall Naive Comparison time (s):            0.922873
Number of False Positive::           498
Number of True Negative ::           25482
Rate of False Positive::             0.019169
Overall False Positive Computational time (s):      0.922873
Overall Parallilized Bloom Filter time (s):        25.203258
Number of False Positive::           498
Number of True Negative ::           25482
Rate of False Positive::             0.019169
Overall Writing Files time (s):              0.133571
Overall time (s):                            36.351570
```

Parallelized Time Reading with 25% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   2
total number of threads:             8
total number of threads per region:  2
Number of Unique words to Insert:    20466
Number of Unique words to Look Up:   39860
SIZE Of BIT ARRAY:                   191031
Number of Hash Function:             7
Next Prime Value:                    191033
Overall Reading Files time (s):              6.388226
Overall Insert time (s):                     5.389768
Overall Look Up time (s):                   11.645827
Overall Naive Comparison time (s):           0.452037
Number of False Positive::           321
Number of True Negative ::           19365
Rate of False Positive::             0.016306
Overall False Positive Computational time (s):      0.452037
Overall Parallilized Bloom Filter time (s):        17.939669
Number of False Positive::           321
Number of True Negative ::           19365
Rate of False Positive::             0.016306
Overall Writing Files time (s):              0.111641
Overall time (s):                           24.439537
```

Parallelized Time Reading with 50% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   2
total number of threads:             8
total number of threads per region:  2
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   20466
SIZE Of BIT ARRAY:                   130784
Number of Hash Function:             3
Next Prime Value:                    130787
Overall Reading Files time (s):              4.070974
Overall Insert time (s):                     5.219015
Overall Look Up time (s):                    3.895865
Overall Naive Comparison time (s):           0.348821
Number of False Positive::           1995
Number of True Negative ::           10154
Rate of False Positive::             0.164211
Overall False Positive Computational time (s):      0.348821
Overall Parallilized Bloom Filter time (s):         9.812522
Number of False Positive::           1995
Number of True Negative ::           10154
Rate of False Positive::             0.164211
Overall Writing Files time (s):              0.059972
Overall time (s):                           13.943469
```

Parallelized Time Reading with 75% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   2
total number of threads:             8
total number of threads per region:  2
Number of Unique words to Insert:    6822
Number of Unique words to Look Up:   13287
SIZE Of BIT ARRAY:                   63679
Number of Hash Function:             7
Next Prime Value:                    63689
Overall Reading Files time (s):              0.740608
Overall Insert time (s):                     1.602070
Overall Look Up time (s):                    3.299328
Overall Naive Comparison time (s):           0.067990
Number of False Positive::           150
Number of True Negative ::           9790
Rate of False Positive::             0.015091
Overall False Positive Computational time (s):      0.067990
Overall Parallilized Bloom Filter time (s):         5.037378
Number of False Positive::           150
Number of True Negative ::           9790
Rate of False Positive::             0.015091
Overall Writing Files time (s):              0.048709
Overall time (s):                            5.826695
```

Case 3

Parallelized Time Reading with 27289 unique words to insert and 53148 to query

```
Enter the number betweem 1 and 4 :   3
total number of threads:             1
total number of threads per region:  16
Number of Unique words to Insert:    27289
Number of Unique words to Look Up:   53148
SIZE Of BIT ARRAY:                   254714
Number of Hash Function:             7
Next Prime Value:                    254729
Overall Reading Files time (s):              11.415912
Overall Insert time (s):                     57.795421
Overall Look Up time (s):                    123.012678
Overall Naive Comparison time (s):           0.894210
Number of False Positive::           292
Number of True Negative ::           25662
Rate of False Positive::             0.011251
Overall False Positive Computational time (s):      0.894210
Overall Parallilized Bloom Filter time (s):         182.596519
Number of False Positive::           292
Number of True Negative ::           25662
Rate of False Positive::             0.011251
Overall Writing Files time (s):              0.140536
Overall time (s):                            194.152969
```

Parallelized Time Reading with 25% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   3
total number of threads:             1
total number of threads per region:  16
Number of Unique words to Insert:    20466
Number of Unique words to Look Up:   39861
SIZE Of BIT ARRAY:                   191035
Number of Hash Function:             7
Next Prime Value:                    191039
Overall Reading Files time (s):              6.274230
Overall Insert time (s):                    53.763288
Overall Look Up time (s):                  103.623364
Overall Naive Comparison time (s):           0.534167
Number of False Positive::           265
Number of True Negative ::           19400
Rate of False Positive::             0.013476
Overall False Positive Computational time (s):      0.534167
Overall Parallilized Bloom Filter time (s):       158.454987
Number of False Positive::           265
Number of True Negative ::           19400
Rate of False Positive::             0.013476
Overall Writing Files time (s):              0.163775
Overall time (s):                          164.892990
```

Parallelized Time Reading with 50% percentage less to insert and query

```
Next Prime Value:                    98101
Enter the number betweem 1 and 4 :   3
total number of threads:             1
total number of threads per region:  16
Number of Unique words to Insert:    13645
Number of Unique words to Look Up:   20466
SIZE Of BIT ARRAY:                   98084
Number of Hash Function:             5
Next Prime Value:                    98101
Overall Reading Files time (s):              2.025275
Overall Insert time (s):                    26.221683
Overall Look Up time (s):                   41.386304
Overall Naive Comparison time (s):           0.154291
Number of False Positive::           551
Number of True Negative ::           13696
Rate of False Positive::             0.038675
Overall False Positive Computational time (s):      0.154291
Overall Parallilized Bloom Filter time (s):        67.916569
Number of False Positive::           551
Number of True Negative ::           13696
Rate of False Positive::             0.038675
Overall Writing Files time (s):              0.058019
Overall time (s):                           69.999863
```

Parallelized Time Reading with 75% percentage less to insert and query

```
Enter the number betweem 1 and 4 :   3
total number of threads:             1
total number of threads per region:  16
Number of Unique words to Insert:    6822
Number of Unique words to Look Up:   13287
SIZE Of BIT ARRAY:                   63679
Number of Hash Function:             7
Next Prime Value:                    63689
Overall Reading Files time (s):              0.742919
Overall Insert time (s):                    12.709382
Overall Look Up time (s):                   26.257689
Overall Naive Comparison time (s):           0.099372
Number of False Positive::           95
Number of True Negative ::           9846
Rate of False Positive::             0.009556
Overall False Positive Computational time (s):      0.099372
Overall Parallilized Bloom Filter time (s):        39.165815
Number of False Positive::           95
Number of True Negative ::           9846
Rate of False Positive::             0.009556
Overall Writing Files time (s):              0.050512
Overall time (s):                           39.959248
```

Appendix D:

Diagram For Case 1

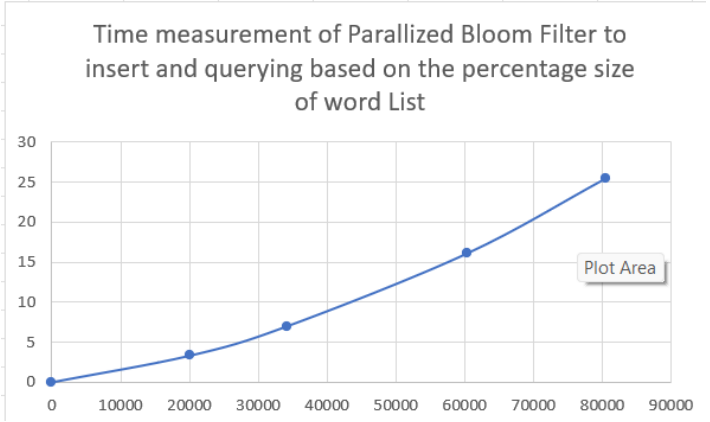


Diagram for Case 2

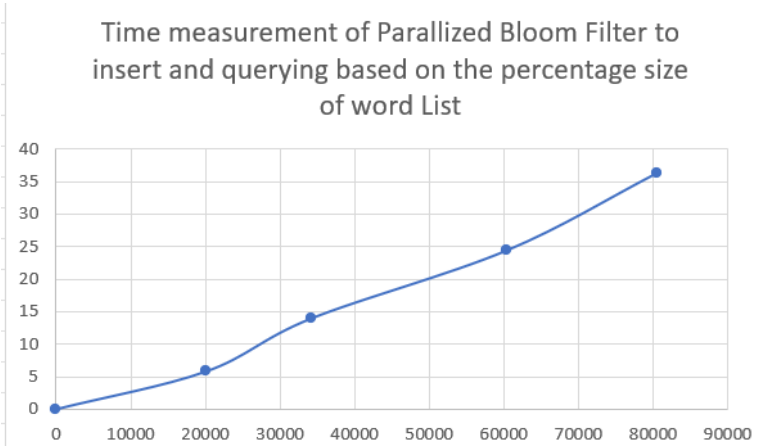
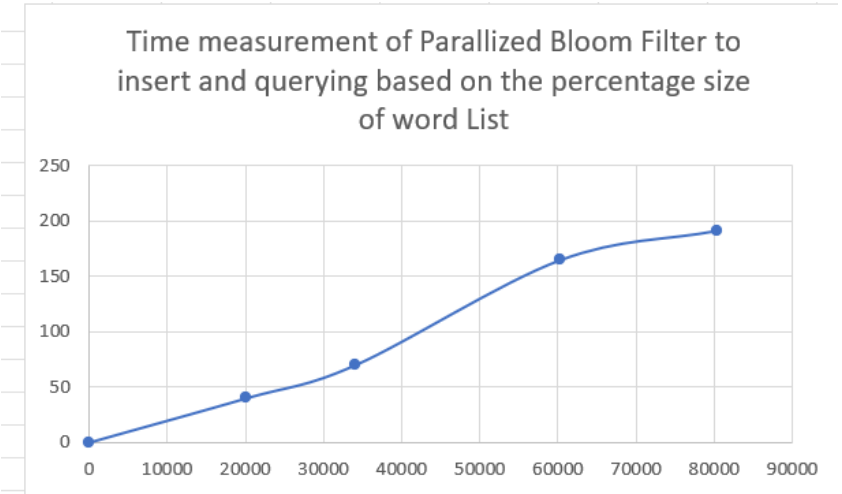


Diagram for Case 3



Reference

Leiserson, Prof. C. E. (2005, October 5). *Introduction to algorithms (SMA 5503): Electrical Engineering and Computer Science*. MIT OpenCourseWare. https://ocw.mit.edu/courses/6-046j-introduction-to-algorithms-sma-5503-fall-2005/resources/lecture-8-universal-hashing-perfect-hashing/

Wikimedia Foundation. (n.d.-b). *Universal hashing*. Wikipedia. https://en.wikipedia.org/wiki/Universal_hashing

Wikimedia Foundation. (n.d.). *Bloom Filter Algorithm*. Wikipedia. https://en.wikipedia.org/wiki/Bloom_filter

Manish Kumar Bhojasia. (2022, May 13). *Bit array in C*. Sanfoundry.

https://www.sanfoundry.com/founder/

Tools:

Implementation : VS, C and omp
Graph Generator: Excel and Geogebra
File Merging: Python