Log Book

Parallelization of a Brute Force Algorithm

Elaine Motley

University of the West of England

Parallel Computing UFCFFL-15-M

Daily logbook of progress when developing a program written in C to crack a given cipher using BruteForce. The program is then is parallelized by using OpenMP and MPI.

1 CONTENTS

2 OUTLINE

This project is to develop a serial program which can crack a piece of ciphertext. I.e. find the key, if the initial vector (IV), the original plaintext, and the resultant ciphertext produced by using AES-128-cbc encryption is known

The program is then to be parallelized by firstly using shared memory methods (OpenMP) and then using distributed memory methods (MPI).

3 OPENMP

OpenMP is an API used for multi-threaded parallel processing which can be used on shared memory multi-processor computers. Most modern computers have multiple cores and this is one method of utilising all of the processing power of those cores. The number of processors defines the number of maximum possible threads. In this project it is to be used on a SIMD (single instruction, multi-data) process. The data (the potential keys, once generated), can be distributed to the threads and each thread can carry out the same instruction on their share of the data.

Attention needs to be given to how the data is distributed and which variables should be global or private to each thread.

4 MPI

MPI (Message Passing Interface) is a method whereby multiple computers can run a program in parallel by passing messages across distributed memory. This method can also be used on MIMD processes (multi-instruction, multi-data), but for this project only SIMD implementation is required. Again, the focus will be on how to distribute the data between the multiple processors.

5 METHOD

The serial program will first be developed. The main stages will be:

- Develop an algorithm to generate strings of a given length from a set alphabet
- Implement the algorithm in a program written in C
- Test generated strings for match against an input string
- Introduce encryption, so that the generated strings are used as keys in AES encryption, then test for match against a given string (the original ciphertext)

The next stage will be to parallelize the program using OpenMP so that it runs on multiple cores and the number of threads can be specified.

Finally, the serial program will be parallelized using MPI so that it can run on a remote cluster.

Once all versions of the BruteForce program have been successfully developed, testing can then commence and the results analysed. The results and analysis are presented in a separate document.

An activity log will show the stages of development and highlight issues arising during the development process.

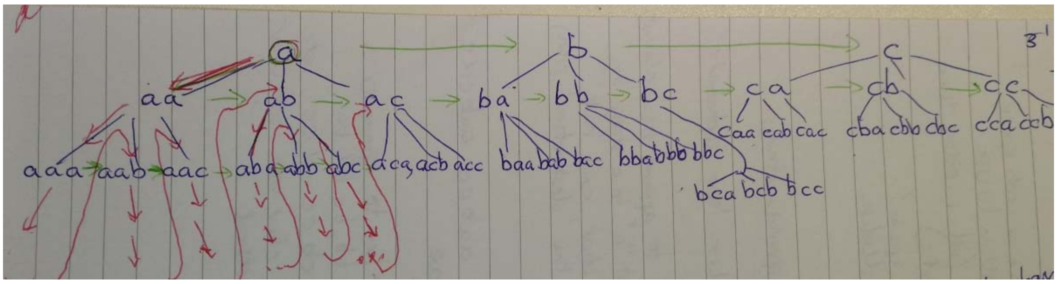
6 Activity Log

04/11/18 Session Time 5:30 hr Cumulative Time = 5:30 hr

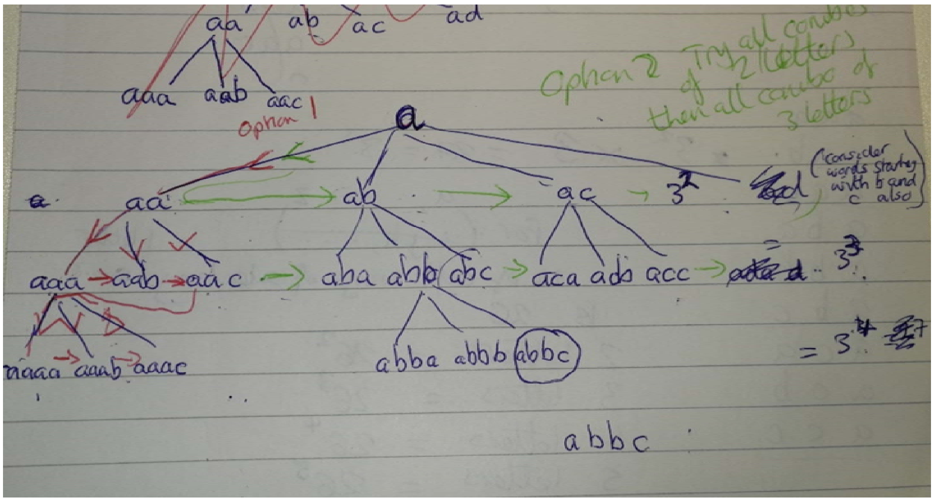
Goals:

To develop basic code to generate strings of given length from alphabet, then compare to input string and find a match.

I considered using a top down approach or a depth search. Option 1 was to consider all string combinations starting with the first letter of the alphabet, before moving on to the next:


a
b
c
3¹
aa ab ac ba bb bc ca cb cc
aaa aab aac aba abb abc aca acb acc baa bab bac bba bbb bbc
bca bcb bcc
caa cab cac cba cbb cbc cca ccb

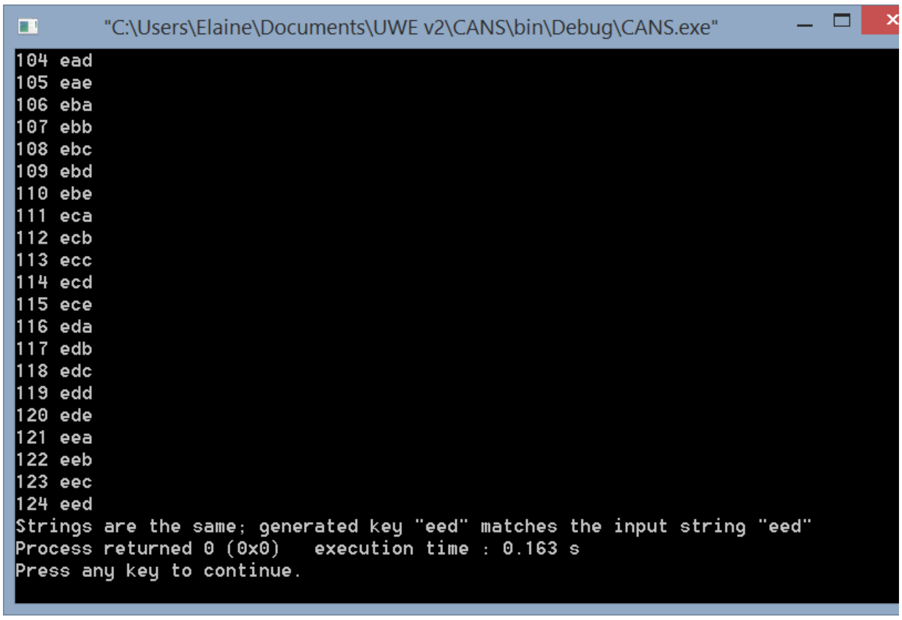
Option 2 was to try all 1 letter combinations, then 2 letter combinations, etc.:


aa ab ac ad
aaa aab aac
Option 1
Option 2 Try all combos of 2 letters then all combo of 3 letters
a
a aa ab ac
3²
(consider words starting with b and c also)
aaa aab aac aba abb abc aca acb acc
= 3³
= 3⁴
aaaa aaab aaac
abba abbb abbc
abbc

The recursive function developed took Option 1 approach with a reduced search alphabet of "abcde". The total no. of strings possible from an alphabet of size 5 is:

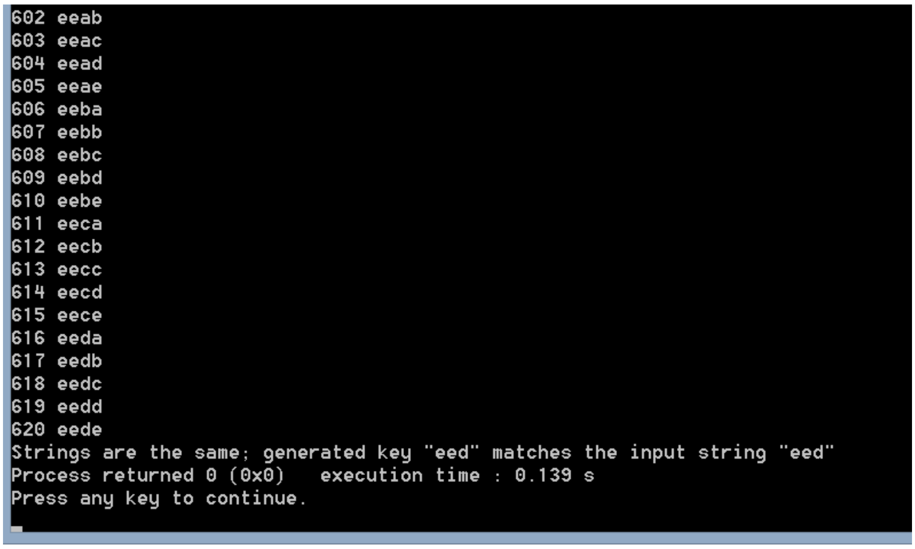
Length of string	No. of permutations		
1	5^1	=	5
2	5^2	=	25
3	5^3	=	125
4	5^4	=	625
5	5^5	=	3125
Total			**3905**

The next stage was to add string matching to a given input (string length of 3):



```
104 ead
105 eae
106 eba
107 ebb
108 ebc
109 ebd
110 ebe
111 eca
112 ecb
113 ecc
114 ecd
115 ece
116 eda
117 edb
118 edc
119 edd
120 ede
121 eea
122 eeb
123 eec
124 eed
Strings are the same; generated key "eed" matches the input string "eed"
Process returned 0 (0x0)   execution time : 0.163 s
Press any key to continue.
```

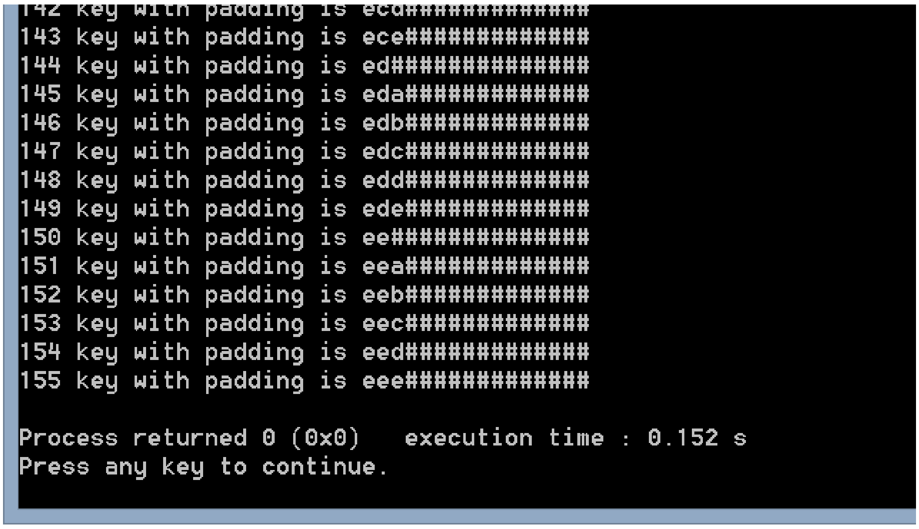
I then increased the generated string size to 4:



```
602 eeab
603 eeac
604 eead
605 eeae
606 eeba
607 eebb
608 eebc
609 eebd
610 eebe
611 eeca
612 eecb
613 eecc
614 eecd
615 eece
616 eeda
617 eedb
618 eedc
619 eedd
620 eede
Strings are the same; generated key "eed" matches the input string "eed"
Process returned 0 (0x0)   execution time : 0.139 s
Press any key to continue.
```

Initially had a problem with string size of 4, as the matching occurred for the first 3 chars and so the program would terminate early.

Padding (the '#' character) was then appended to the strings (3 chars long) to make them 16B keys:

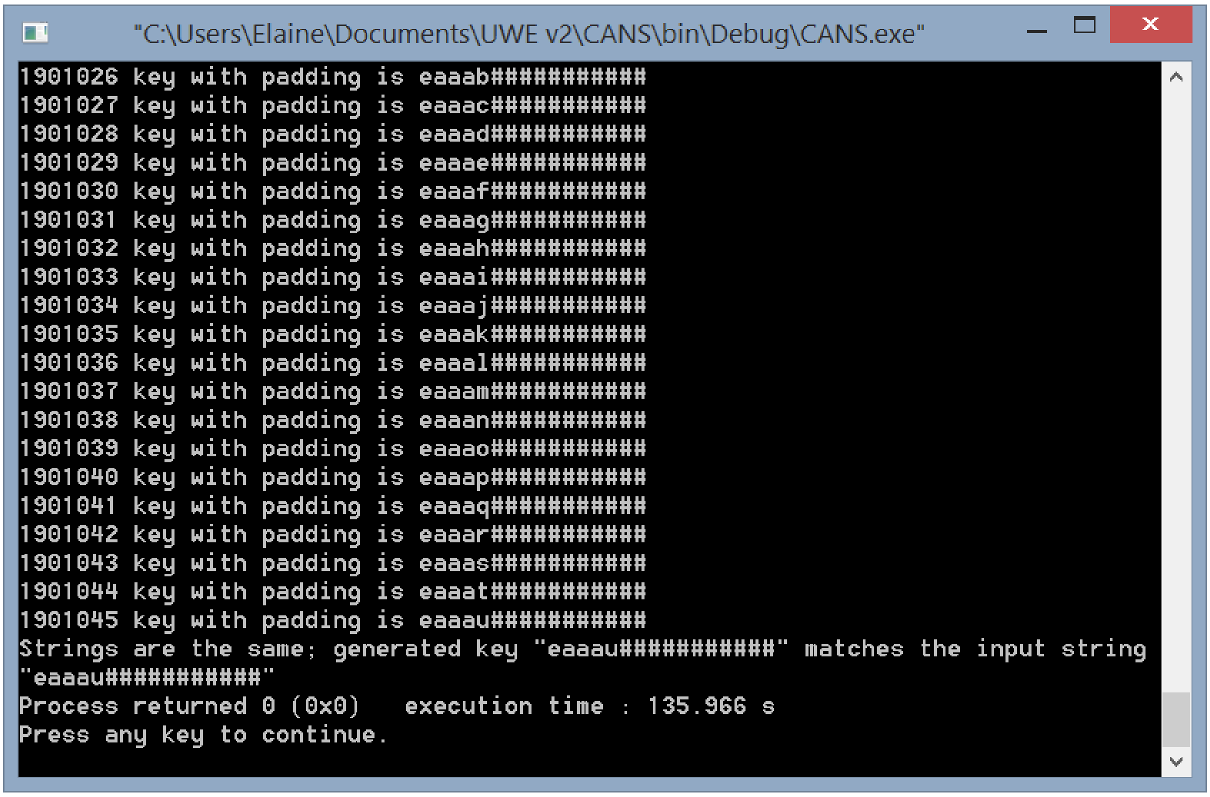


```
142 key with padding is ecd#############
143 key with padding is ece#############
144 key with padding is ed##############
145 key with padding is eda#############
146 key with padding is edb#############
147 key with padding is edc#############
148 key with padding is edd#############
149 key with padding is ede#############
150 key with padding is ee##############
151 key with padding is eea#############
152 key with padding is eeb#############
153 key with padding is eec#############
154 key with padding is eed#############
155 key with padding is eee#############

Process returned 0 (0x0)   execution time : 0.152 s
Press any key to continue.
```

Note that the total no. of keys is 5^1 + 5^2 + 5^3

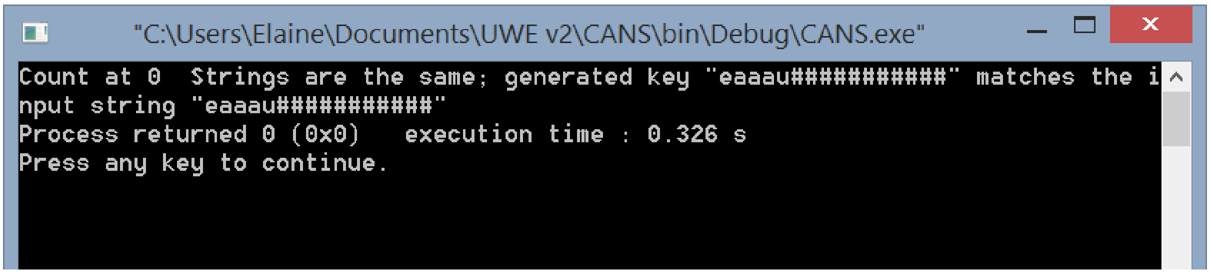
= 155

Increased alphabet to a -z and also tried to match a 5 char long password about a fifth of the way through the search space ("eaaau###########")



```
1901026 key with padding is eaaab###########
1901027 key with padding is eaaac###########
1901028 key with padding is eaaad###########
1901029 key with padding is eaaae###########
1901030 key with padding is eaaaf###########
1901031 key with padding is eaaag###########
1901032 key with padding is eaaah###########
1901033 key with padding is eaaai###########
1901034 key with padding is eaaaj###########
1901035 key with padding is eaaak###########
1901036 key with padding is eaaal###########
1901037 key with padding is eaaam###########
1901038 key with padding is eaaan###########
1901039 key with padding is eaaao###########
1901040 key with padding is eaaap###########
1901041 key with padding is eaaaq###########
1901042 key with padding is eaaar###########
1901043 key with padding is eaaas###########
1901044 key with padding is eaaat###########
1901045 key with padding is eaaau###########
Strings are the same; generated key "eaaau###########" matches the input string
"eaaau###########"
Process returned 0 (0x0)   execution time : 135.966 s
Press any key to continue.
```

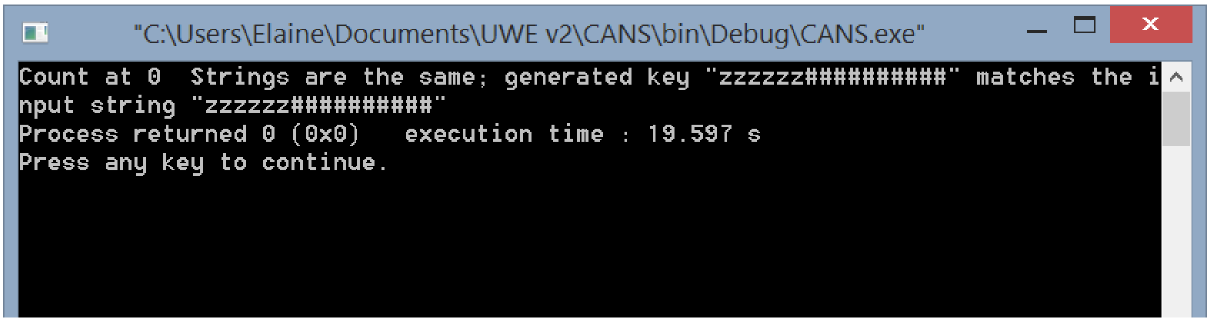
Printing obviously takes time, so have eliminated printing every line, just the count and the key match: (counter not returning expected results, so will have to modify)



```
Count at 0  Strings are the same; generated key "eaaau###########" matches the i
nput string "eaaau###########"
Process returned 0 (0x0)   execution time : 0.326 s
Press any key to continue.
```

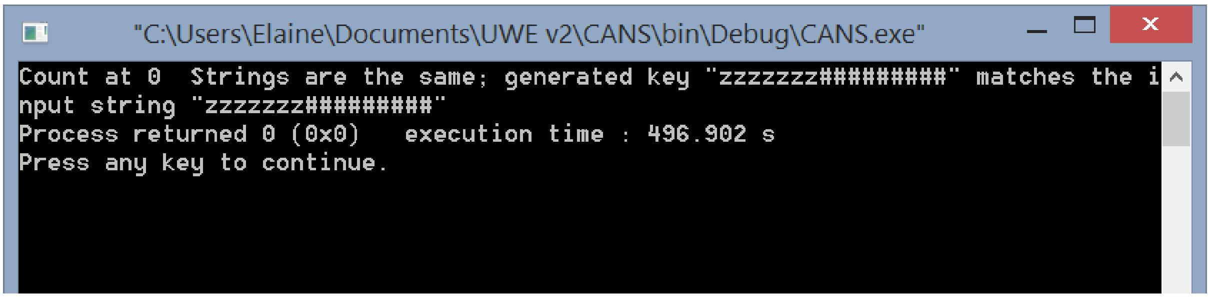
I then experimented by searching for keys at the beginning, and end of the search space (eg “aaaaa”, “”zzzzz”) and for different lengths of keys (before padding).

6 char string at end of search space: (alphabet = a-z) (approx 19.5 s)



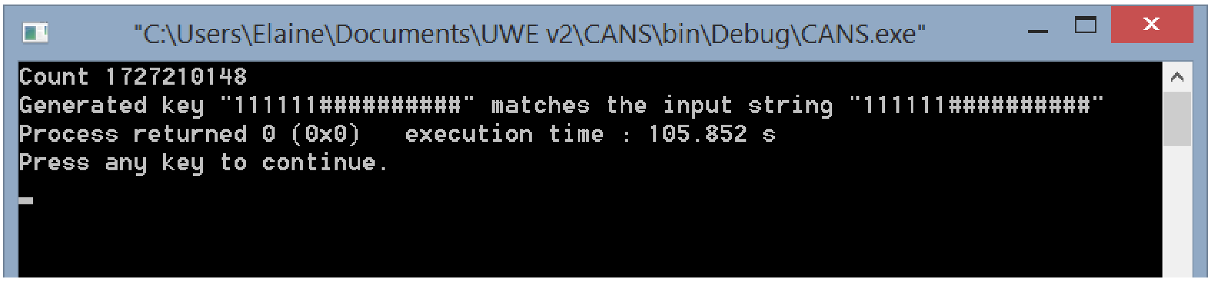
```
Count at 0  Strings are the same; generated key "zzzzzz##########" matches the i
nput string "zzzzzz##########"
Process returned 0 (0x0)   execution time : 19.597 s
Press any key to continue.
```

7 char string at end of search space: (alphabet = a-z):

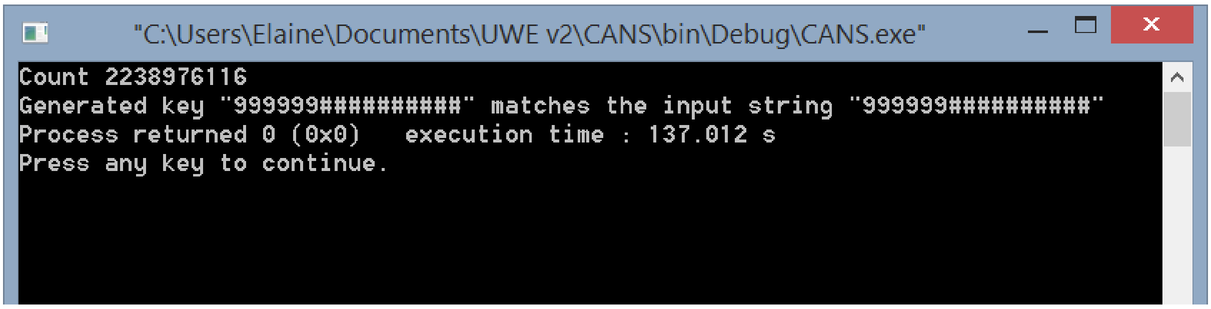


```
Count at 0  Strings are the same; generated key "zzzzzzz#########" matches the i
nput string "zzzzzzz#########"
Process returned 0 (0x0)   execution time : 496.902 s
Press any key to continue.
```

Testing on increased alphabet "abcdefghijklmnopqrstuvwxyz0123456789":



```
Count 1727210148
Generated key "111111##########" matches the input string "111111##########"
Process returned 0 (0x0)   execution time : 105.852 s
Press any key to continue.
```



```
Count 2238976116
Generated key "999999##########" matches the input string "999999##########"
Process returned 0 (0x0)   execution time : 137.012 s
Press any key to continue.
```

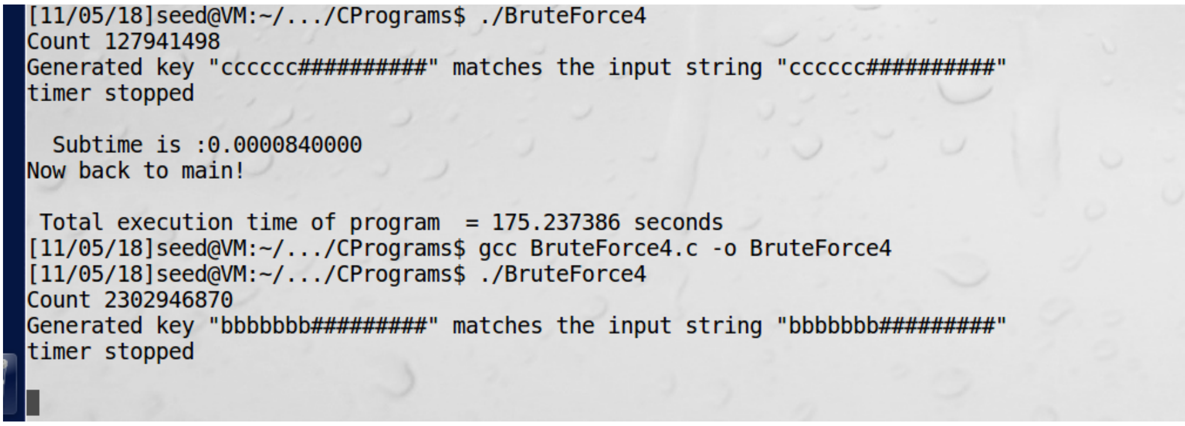
I think this is the limit of the testing for this particular alphabet. The search space grows exponentially, and as am trying to match string at the end of the search space, then the time will increase exponentially too. Also, the program has thus far been tested in a Windows IDE environment, not in Ubuntu. Code also needs to be added to calculate the execution time, as the IDE does this automatically.

05/11/18 Session Time 5:15 hrs Cumulative Time = 10:45 hrs

Goals:

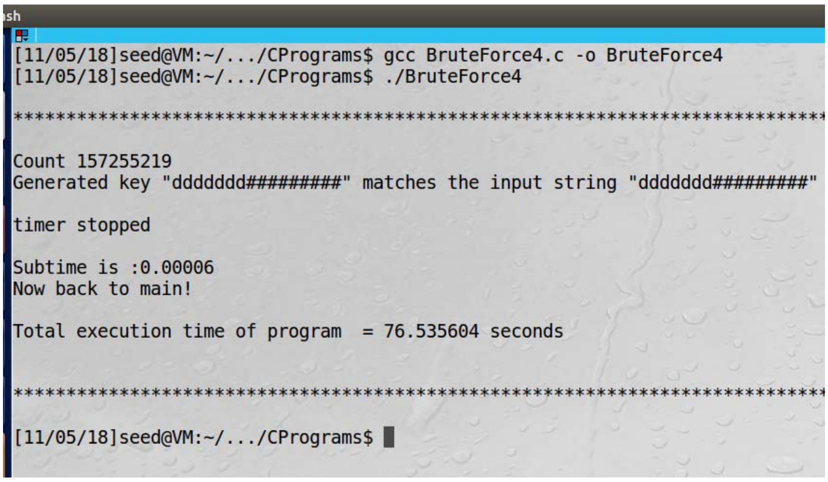
To add timing code and run in Ubuntu.

Successfully ran on Ubuntu, but had problems with the timer. Program prints the matched string to the screen, but then doesn't return to the main function for quite a while. Waited over an hour for "bbbbbbb#########" to return to main, before eventually terminating program manually.



```
[11/05/18]seed@VM:~/.../CPrograms$ ./BruteForce4
Count 127941498
Generated key "cccccc##########" matches the input string "cccccc##########"
timer stopped

  Subtime is :0.0000840000
Now back to main!

 Total execution time of program  = 175.237386 seconds
[11/05/18]seed@VM:~/.../CPrograms$ gcc BruteForce4.c -o BruteForce4
[11/05/18]seed@VM:~/.../CPrograms$ ./BruteForce4
Count 2302946870
Generated key "bbbbbbb#########" matches the input string "bbbbbbb#########"
timer stopped

```

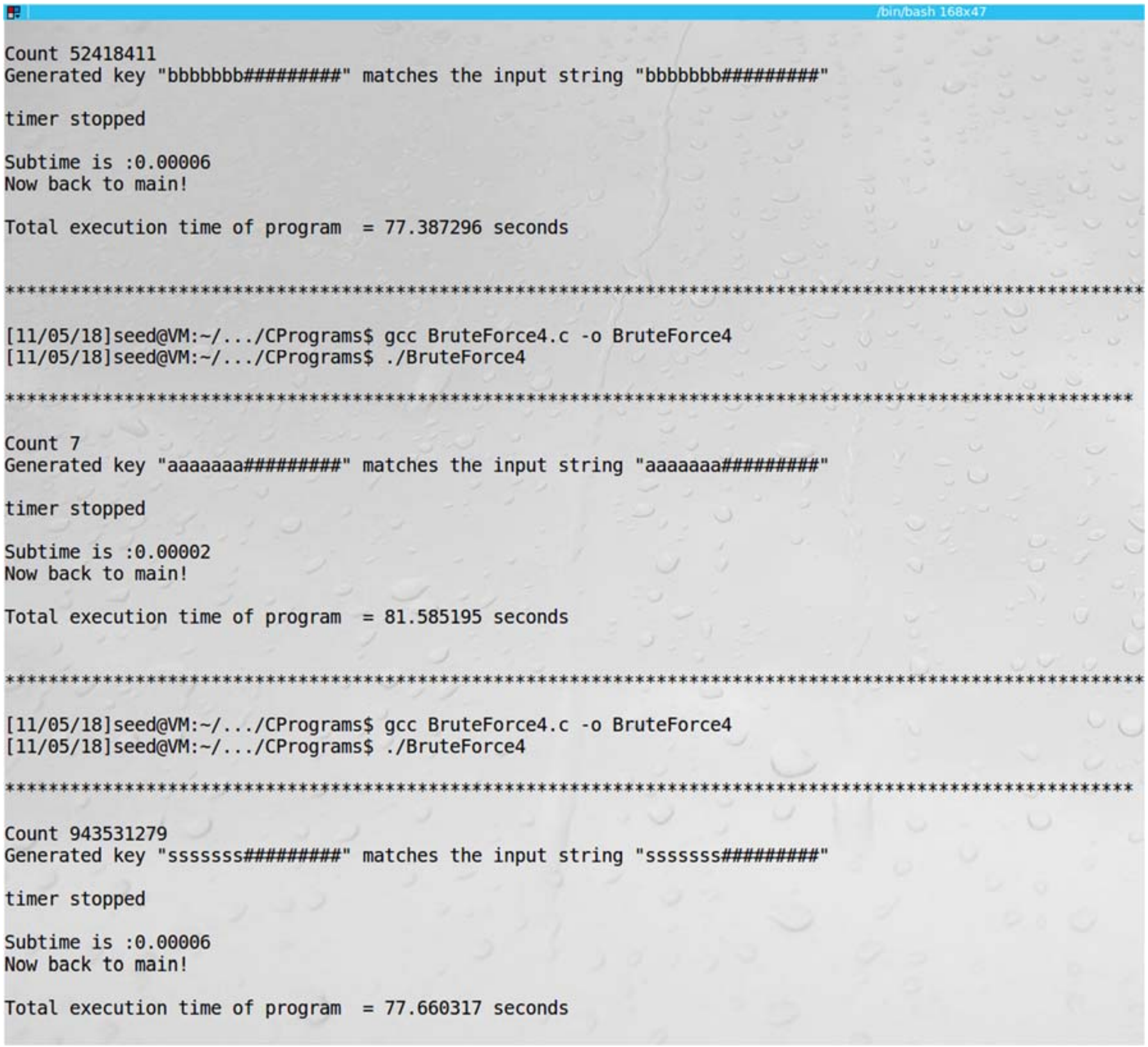
I manually timed the program and a match was found after approx 15 secs and the "timer stopped" line printed, but the actual timer information wasn't dumped to the screen until the termination of the program. The subtimer is definitely wrong! (Note reduced alphabet a-s used)



```
[11/05/18]seed@VM:~/.../CPrograms$ gcc BruteForce4.c -o BruteForce4
[11/05/18]seed@VM:~/.../CPrograms$ ./BruteForce4

*****************************************************************************

Count 157255219
Generated key "ddddddd#########" matches the input string "ddddddd#########"

timer stopped

Subtime is :0.00006
Now back to main!

Total execution time of program  = 76.535604 seconds


*****************************************************************************

[11/05/18]seed@VM:~/.../CPrograms$
```

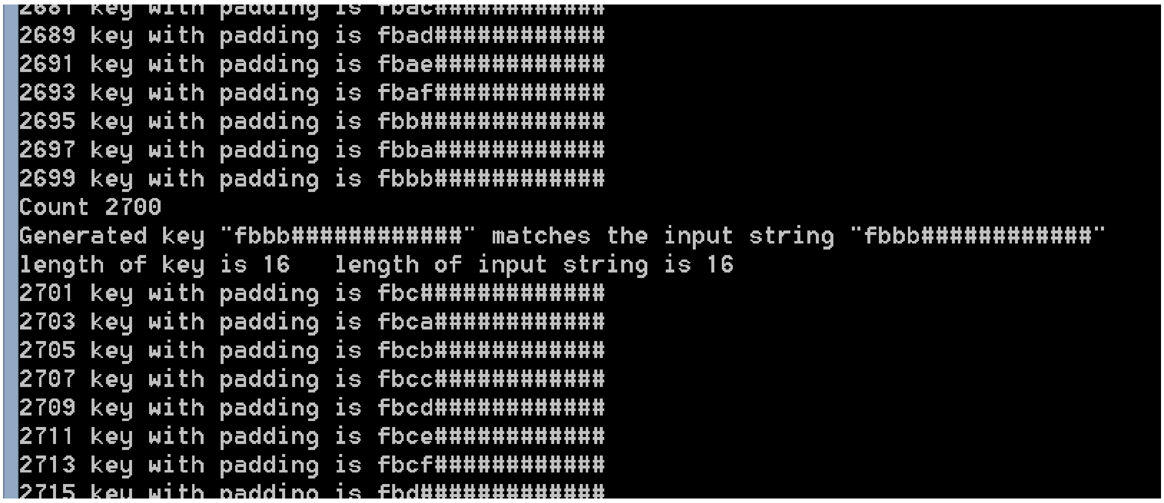
Suspect that even after match found, the function continues until the end of the search space, then returns to main.

Confirmed as all trials produce total execution times of the same magnitude.



```
Count 52418411
Generated key "bbbbbbb#########" matches the input string "bbbbbbb#########"

timer stopped

Subtime is :0.00006
Now back to main!

Total execution time of program  = 77.387296 seconds

************************************************************************************************************

[11/05/18]seed@VM:~/.../CPrograms$ gcc BruteForce4.c -o BruteForce4
[11/05/18]seed@VM:~/.../CPrograms$ ./BruteForce4

***********************************************************************************************************

Count 7
Generated key "aaaaaaa#########" matches the input string "aaaaaaa#########"

timer stopped

Subtime is :0.00002
Now back to main!

Total execution time of program  = 81.585195 seconds

************************************************************************************************************

[11/05/18]seed@VM:~/.../CPrograms$ gcc BruteForce4.c -o BruteForce4
[11/05/18]seed@VM:~/.../CPrograms$ ./BruteForce4

***********************************************************************************************************

Count 943531279
Generated key "sssssss#########" matches the input string "sssssss#########"

timer stopped

Subtime is :0.00006
Now back to main!

Total execution time of program  = 77.660317 seconds
```

Printing every generated key confirms that even though a match has been found, the program continues until all possible keys have been generated:



```
2687 key with padding is fbac############
2689 key with padding is fbad############
2691 key with padding is fbae############
2693 key with padding is fbaf############
2695 key with padding is fbb#############
2697 key with padding is fbba############
2699 key with padding is fbbb############
Count 2700
Generated key "fbbb############" matches the input string "fbbb############"
length of key is 16   length of input string is 16
2701 key with padding is fbc#############
2703 key with padding is fbca############
2705 key with padding is fbcb############
2707 key with padding is fbcc############
2709 key with padding is fbcd############
2711 key with padding is fbce############
2713 key with padding is fbcf############
2715 key with padding is fbd#############
```

I don't know if the function is returning to main, and then the function is continually being re-called, or it is not returning until all loops are completed. I cannot proceed with integrating encryption until this is resolved as no accurate timings will be achieved.

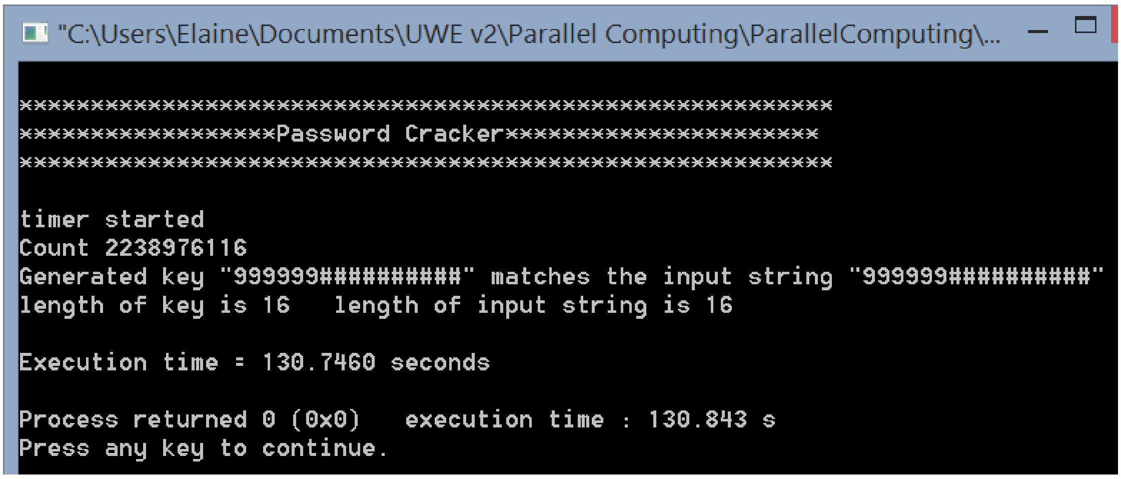
07/11/18 **Session Time 1:35 hr** **Cumulative Time = 12:20 hr**

With a bit of advice, the issue was that the program was only breaking out of its current recursion call, but continued recursively calling the function and only stopped when the search was exhausted.

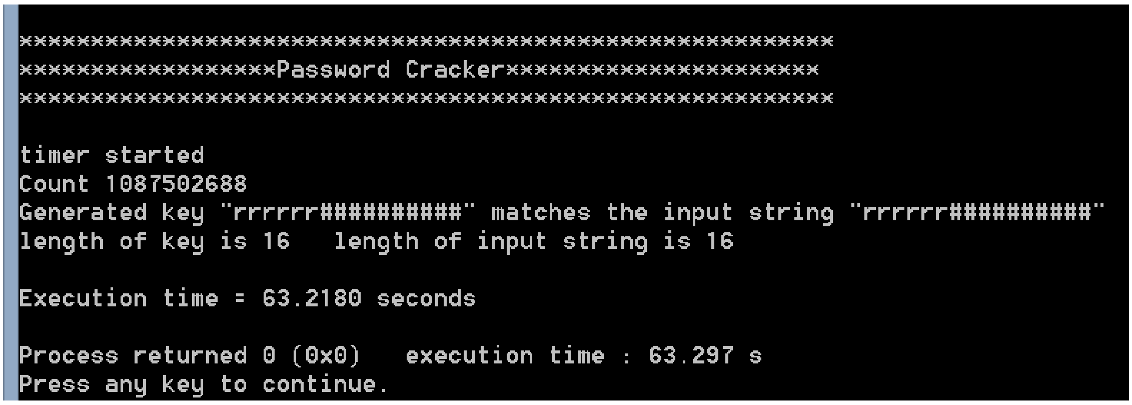
However, another problem arose, in that the recursion was being applied so that excessive keys were being generated. eg if generated strings were of length 4, then each key was generated 4 times.

08/11/18 **Session Time 4:15 hr** **Cumulative Time = 16:35 hr**

I finally got the program to successfully match the string without any duplicates being produced. The program was tested to match strings at the middle and end of the search space (“rrrrrr##########” & "999999##########")


"C:\Users\Elaine\Documents\UWE v2\Parallel Computing\ParallelComputing\...

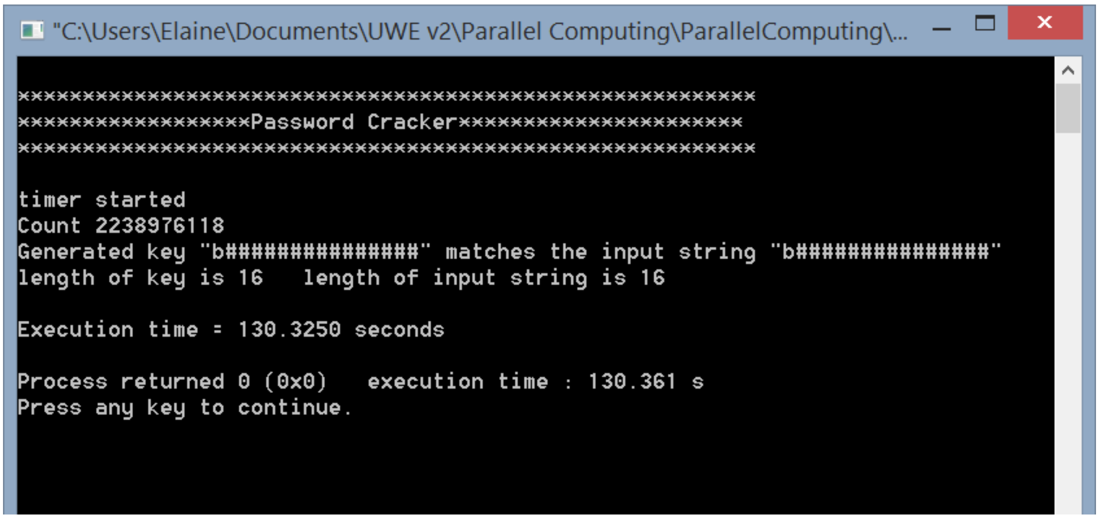
```
*********************************************************
******************Password Cracker**********************
*********************************************************

timer started
Count 2238976116
Generated key "999999##########" matches the input string "999999##########"
length of key is 16   length of input string is 16

Execution time = 130.7460 seconds

Process returned 0 (0x0)   execution time : 130.843 s
Press any key to continue.
```



```
*********************************************************
******************Password Cracker**********************
*********************************************************

timer started
Count 1087502688
Generated key "rrrrrr##########" matches the input string "rrrrrr##########"
length of key is 16   length of input string is 16

Execution time = 63.2180 seconds

Process returned 0 (0x0)   execution time : 63.297 s
Press any key to continue.
```

Further tests were run on different lengths of initial string generated and at different positions within the search alphabet.

This key “b###############” is approx 1/36 of the way through the search space, occurring after all the possible passwords (of length 1,2,3,4,5,6,7) beginning with "a" have been generated.

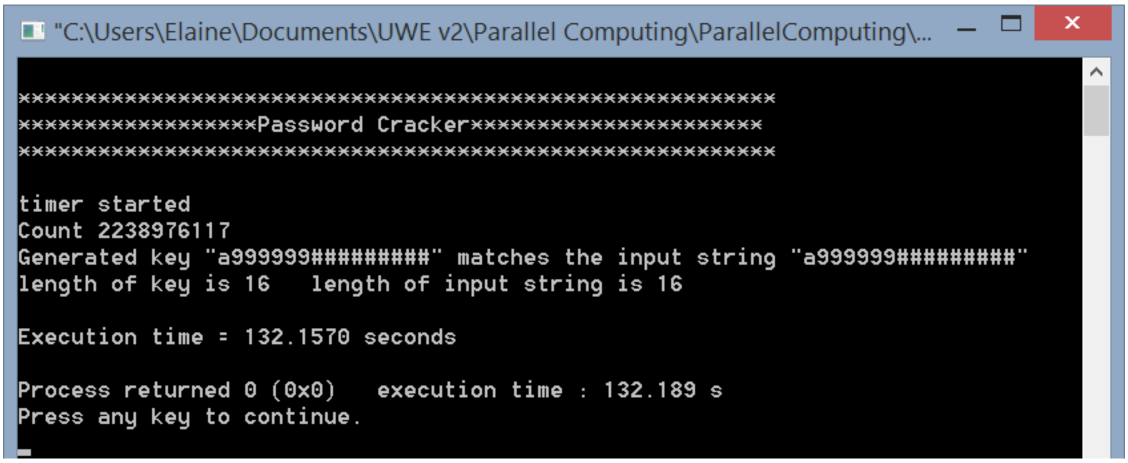


"C:\Users\Elaine\Documents\UWE v2\Parallel Computing\ParallelComputing\...

```
*********************************************************
******************Password Cracker**********************

*********************************************************

timer started
Count 2238976118
Generated key "b###############" matches the input string "b###############"
length of key is 16   length of input string is 16

Execution time = 130.3250 seconds

Process returned 0 (0x0)   execution time : 130.361 s
Press any key to continue.
```

Thus, it will take approx 130s to generate strings up to length 7 beginning with each letter of the alphabet. "a999999" should come directly before "b" and therefore take roughly the same length of time.

Check:

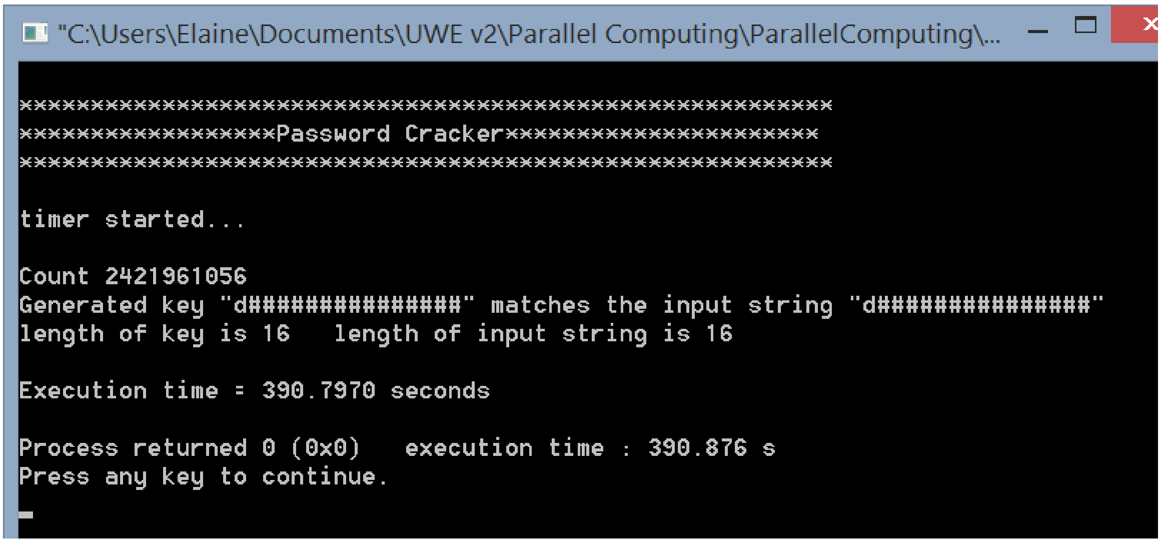


"C:\Users\Elaine\Documents\UWE v2\Parallel Computing\ParallelComputing\...

```
*********************************************************
******************Password Cracker**********************
*********************************************************

timer started
Count 2238976117
Generated key "a999999#########" matches the input string "a999999#########"
length of key is 16   length of input string is 16

Execution time = 132.1570 seconds

Process returned 0 (0x0)   execution time : 132.189 s
Press any key to continue.
```

It takes approx 130s to check through all possible combinations of length up to 7 starting with a particular letter. The last possible permutation, "9999999", should take 36 times this time, or approx an hour and quarter. Rather than running the program for this length of time, I'll check for "d", which comes immediately after all the passwords beginning with a, b and c have been generated, and therefore should take approx 3 x 130s. (390s)

Result as expected:


"C:\Users\Elaine\Documents\UWE v2\Parallel Computing\ParallelComputing\...

```
*********************************************************
******************Password Cracker**********************
*********************************************************

timer started...

Count 2421961056
Generated key "d###############" matches the input string "d###############"
length of key is 16   length of input string is 16

Execution time = 390.7970 seconds

Process returned 0 (0x0)   execution time : 390.876 s
Press any key to continue.
```

Consideration needs to be given to the size of the initial string generated from the search alphabet (length 36) as this will effect the time taken to generate strings for the whole search space and thus testing purposes.

For strlen upto 6:
36^1 + 36^2 + 36^3 + 36^4 + 36^5 + 35^6 = **2,238,976,116**

For strlen upto 7:
36^1 + 36^2 + 36^3 + 36^4 + 36^5 + 35^6 + 36^7 = **10,075,392,530**

The program could also be modified to take user input for the max password length

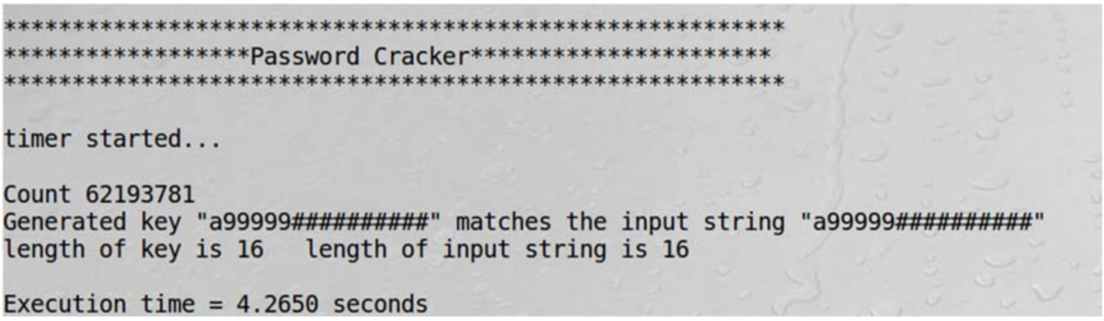
09/11/18 Session Time 3:25 hr Cumulative Time = 20:00 hr

Goals:

To test in Ubuntu environment (VM)
To add encryption code using OpenSSL.

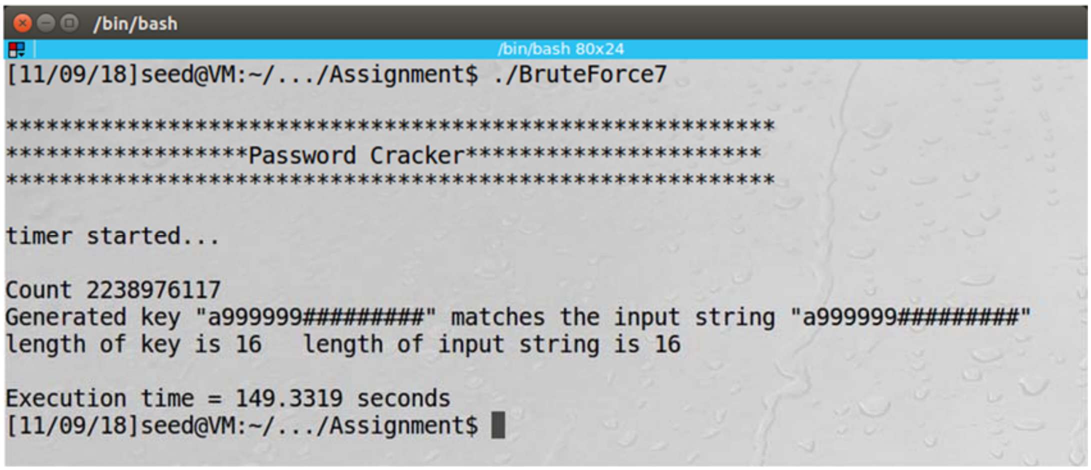
Ubuntu testing.

6 char string at end of all passwords beginning with "a":



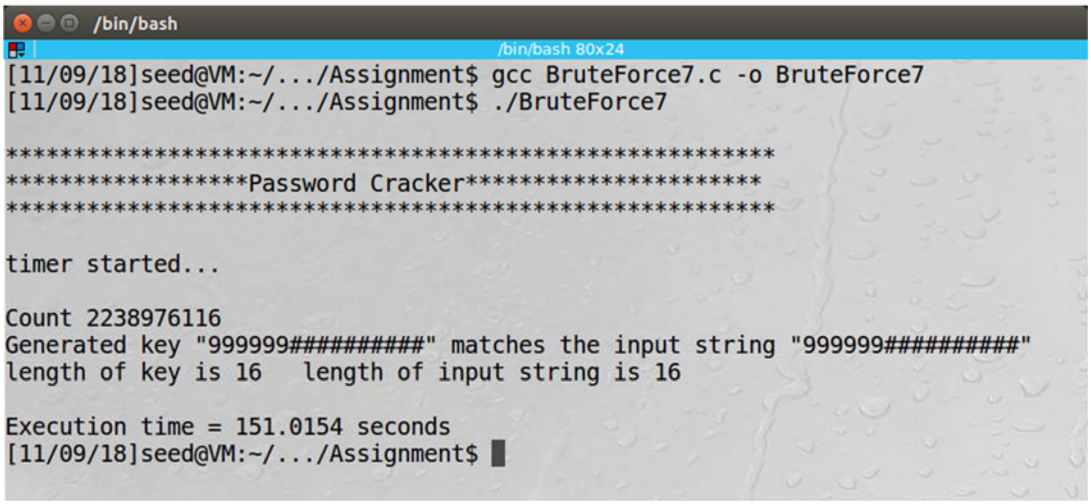
```
*********************************************************
******************Password Cracker**********************
*********************************************************

timer started...

Count 62193781
Generated key "a99999##########" matches the input string "a99999##########"
length of key is 16   length of input string is 16

Execution time = 4.2650 seconds
```

7 char password, (end of all passwords beginning with "a"):



```
[11/09/18]seed@VM:~/.../Assignment$ ./BruteForce7

**********************************************************
******************Password Cracker***********************
**********************************************************

timer started...

Count 2238976117
Generated key "a999999#########" matches the input string "a999999#########"
length of key is 16   length of input string is 16

Execution time = 149.3319 seconds
[11/09/18]seed@VM:~/.../Assignment$ █
```

6 char password (at the end of the search space):



```
[11/09/18]seed@VM:~/.../Assignment$ gcc BruteForce7.c -o BruteForce7
[11/09/18]seed@VM:~/.../Assignment$ ./BruteForce7

**********************************************************
******************Password Cracker***********************
**********************************************************

timer started...

Count 2238976116
Generated key "999999##########" matches the input string "999999##########"
length of key is 16   length of input string is 16

Execution time = 151.0154 seconds
[11/09/18]seed@VM:~/.../Assignment$ █
```

The timings take a little longer in the Ubuntu VM environment, than in the native Windows environment.

Adding Encryption

A final decision was made to use an initial generated string of length 6, partway through the search space, "pa55wd", as this takes approximately a minute to generate.

Using both the command line and an existing C program to check, I encrypted the message "This is a secret message." using AES cbc 128, with IV (in hex) "aabbccddeeff00998877665544332211", and key "\x68\x65\x6c\x6c\x6f\x38\x23\x23\x23\x23\x23\x23\x23\x23\x23\x23" (hex notation of "pa55wd##########"). The resultant ciphertext was:

"5f4429bbed0cbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535"

This string will be hardcoded into the program and then used to match ciphertexts produced after encryption with the generated keys.

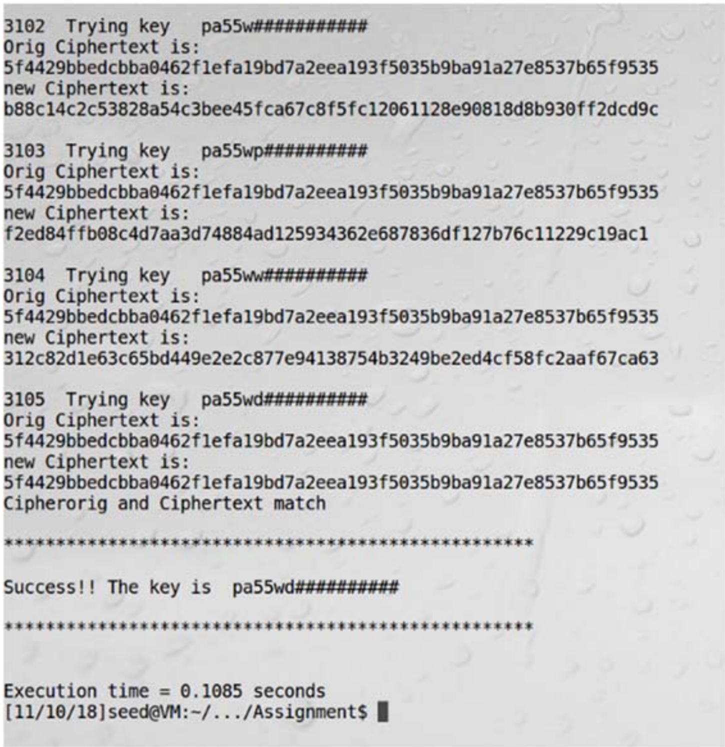
10/11/18 Session Time 5:25 hr Cumulative Time = 25:25 hr

Goals:

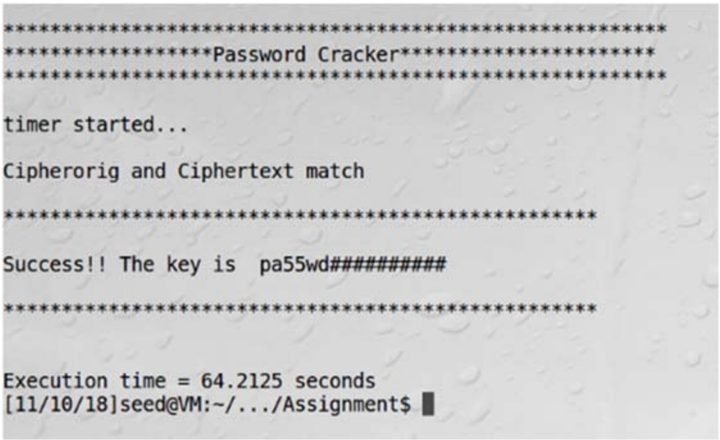
To add encryption function and test matching the generated ciphertexts with the hardcoded known ciphertext.

The search alphabet was reduced to only contain the letters of the known key, i.e. “p,a,w,d,5”. I had issues with not calling the encryption function from within the genKeys function, but this was resolved by making the variables global. There was also a problem, in that the program worked by finding the correct key, but the ciphertext printed on screen from using that key, was not the same as the one obtained by a hex dump of the ciphertext array. This issue was because the array had not been declared correctly, i.e. a char, rather than an unsigned char. After resolving these issues, I increased the size of the alphabet to contain “a -z” and “0 – 9”.

Various positions of the initial character of the key (“p”) within the search alphabet were tested and user input for this position added to the code, by selection of different alphabets.



```
3102  Trying key   pa55w###########
Orig Ciphertext is:
5f4429bbedcbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535
new Ciphertext is:
b88c14c2c53828a54c3bee45fca67c8f5fc12061128e90818d8b930ff2dcd9c

3103  Trying key   pa55wp##########
Orig Ciphertext is:
5f4429bbedcbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535
new Ciphertext is:
f2ed84ffb08c4d7aa3d74884ad125934362e687836df127b76c11229c19ac1

3104  Trying key   pa55ww##########
Orig Ciphertext is:
5f4429bbedcbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535
new Ciphertext is:
312c82d1e63c65bd449e2e2c877e94138754b3249be2ed4cf58fc2aaf67ca63

3105  Trying key   pa55wd##########
Orig Ciphertext is:
5f4429bbedcbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535
new Ciphertext is:
5f4429bbedcbba0462f1efa19bd7a2eea193f5035b9ba91a27e8537b65f9535
Cipherorig and Ciphertext match

***************************************************

Success!! The key is  pa55wd##########

***************************************************


Execution time = 0.1085 seconds
[11/10/18]seed@VM:~/.../Assignment$ 
```



```
**********************************************************
******************Password Cracker**********************
**********************************************************

timer started...

Cipherorig and Ciphertext match

***************************************************

Success!! The key is  pa55wd##########

***************************************************


Execution time = 64.2125 seconds
[11/10/18]seed@VM:~/.../Assignment$ 
```

Printing each generated ciphertext, and key attempt was removed.

“p” was in position 4

<u>11/11/18 Session Time 4:15 hr Cumulative Time = 29:40 hr</u>

Goal

To add user input.

Issues arose because the stringlength variable for the size of the search alphabet was being passed too early before the specific search alphabet had been selected by user input. Further tests were conducted.

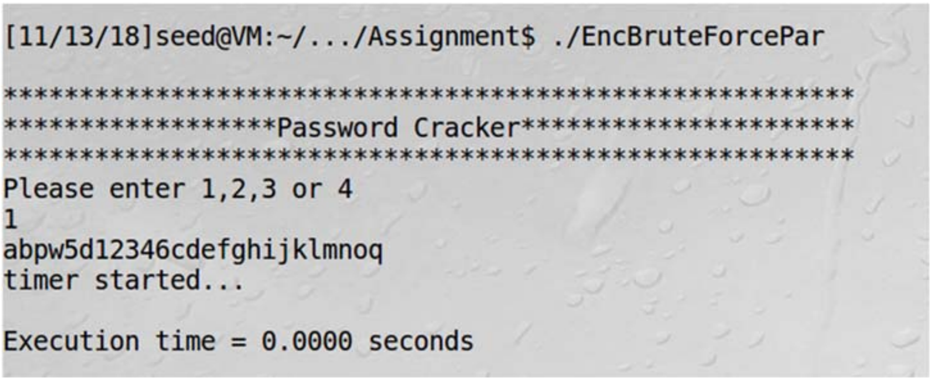
<u>13/11/18 Session Time 4:30 hr Cumulative Time = 34:10 hr</u>

Goal

To Parallelize code using OpenMP

Program has obvious place before some “for loops” where to place OMP pragma for statement, to automatically distribute the task among different threads.

No success:



```
[11/13/18]seed@VM:~/.../Assignment$ ./EncBruteForcePar

*********************************************************
******************Password Cracker***********************
*********************************************************
Please enter 1,2,3 or 4
1
abpw5d12346cdefghijklmnoq
timer started...

Execution time = 0.0000 seconds
```

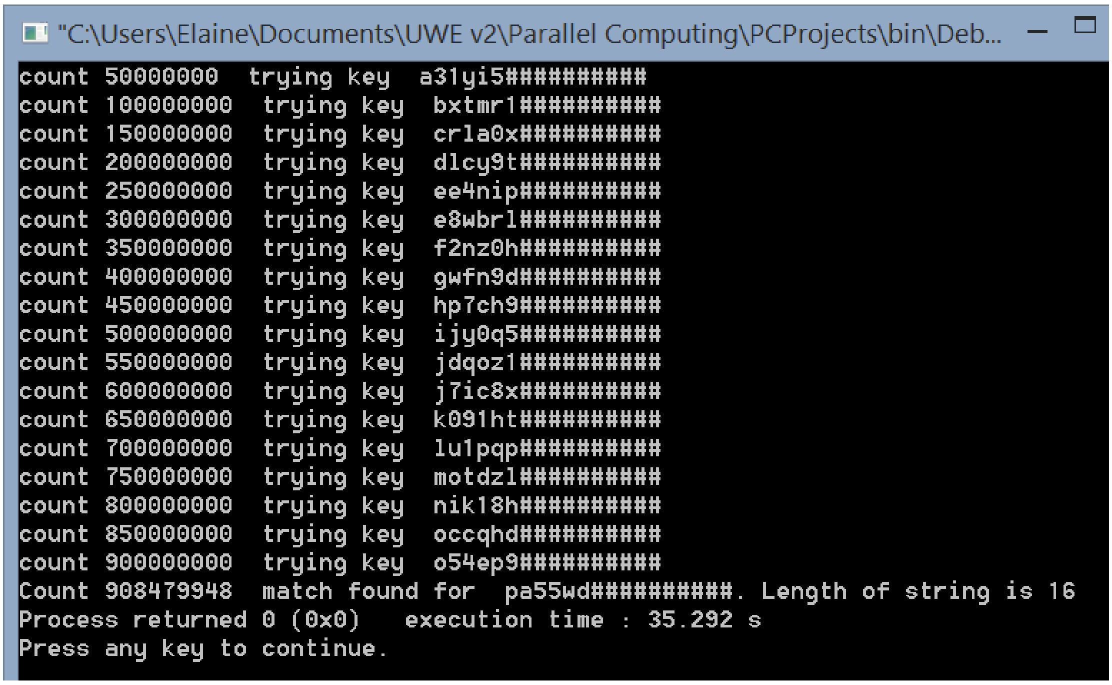
I decided to change the basis of the serial program; currently it contains a recursive function, and to develop a serial program that contains nested for loops instead. This would be more straightforward to parallelize.

15/11/18 Session Time 5:00 hr Cumulative Time = 39:10 hr

Goal

To develop serial program using nested for loops.

Having learnt from previous mistakes for the serial program, this was relatively straightforward to implement. Average result for matching the key (pa55wd##########) with strings generated from the alphabet “abcdefghijklmnopqrstuvwxyz0123456789” is approx. 35 sec.



"C:\Users\Elaine\Documents\UWE v2\Parallel Computing\PCProjects\bin\Deb...

```
count 50000000  trying key  a31yi5##########
count 100000000  trying key  bxtmr1##########
count 150000000  trying key  crla0x##########
count 200000000  trying key  dlcy9t##########
count 250000000  trying key  ee4nip##########
count 300000000  trying key  e8wbrl##########
count 350000000  trying key  f2nz0h##########
count 400000000  trying key  gwfn9d##########
count 450000000  trying key  hp7ch9##########
count 500000000  trying key  ijy0q5##########
count 550000000  trying key  jdqoz1##########
count 600000000  trying key  j7ic8x##########
count 650000000  trying key  k091ht##########
count 700000000  trying key  lu1pqp##########
count 750000000  trying key  motdzl##########
count 800000000  trying key  nik18h##########
count 850000000  trying key  occqhd##########
count 900000000  trying key  o54ep9##########
Count 908479948  match found for  pa55wd##########. Length of string is 16
Process returned 0 (0x0)   execution time : 35.292 s
Press any key to continue.
```

The program was also amended to only print every 50,000.000 attempts (using modulus function). This had negligible effect on the overall execution time. Also, this serial version only generates strings of exactly length 6, whereas the previous version generated all string lengths up to a user input maximum. This slightly reduces the search time as fewer keys are being generated, but within the same ballpark. Total no. of permutations is now **2,176,782,336**.

Encryption functionality was added and tested. Very little difference between recursive version and the nested for loop version.

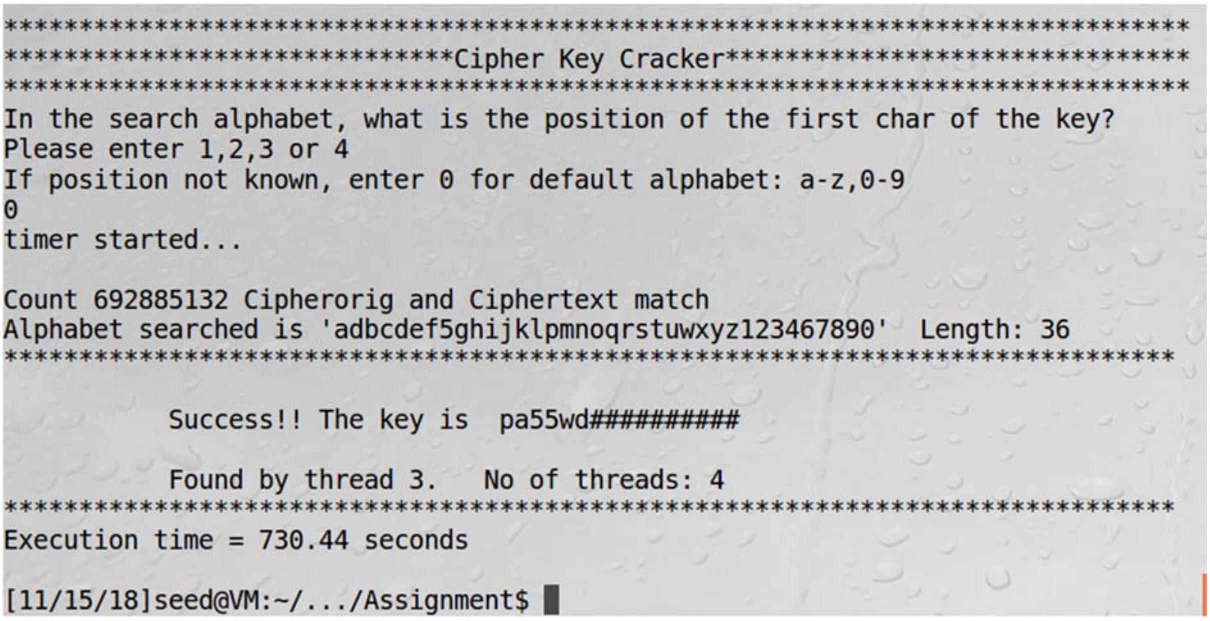
19/11/18 Session Time 4:20 hr Cumulative Time = 43:30 hr

Goal

To implement OMP Parallelisation

Decided to parallelize by dividing the outer loop between the threads. The outer loop sets the first character of the generated string, so the first thread will test strings starting with “a”, the second thread will test strings starting with “b” (using a chunksize of 1), etc. Consideration was also given to which variables needed to be private and which were shared.

Initial testing for 4 threads (where “p” is in the first 4 positions in a reduced search alphabet) produced very quick results. When tested on normal alphabet with p in positon 16, program timed it at 730s to find a match which was 4x the manual timing of the program 184s). It appears that each the total time is the cumulative sum of all the thread timings (eg 4 threads = 4 x actual time). “omp_get_wtime()” was implemented instead.

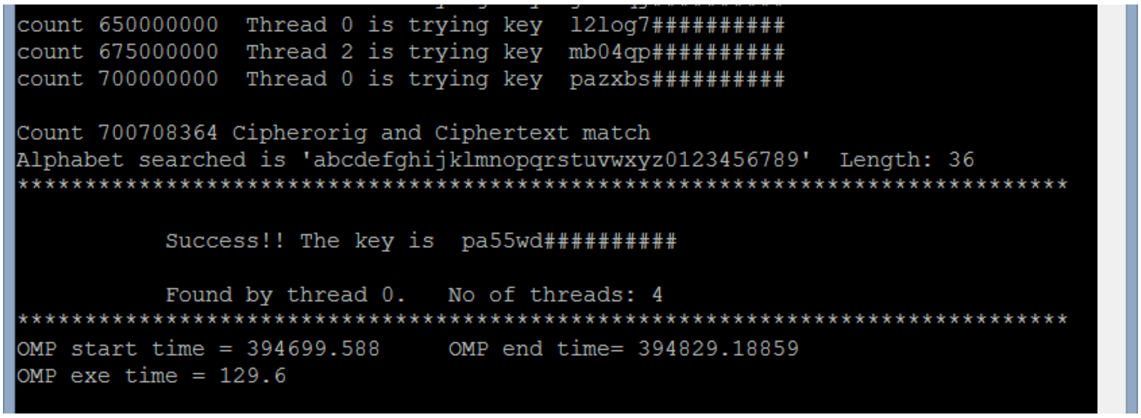


```
*******************************************************************************
******************************Cipher Key Cracker*******************************
*******************************************************************************
In the search alphabet, what is the position of the first char of the key?
Please enter 1,2,3 or 4
If position not known, enter 0 for default alphabet: a-z,0-9
0
timer started...

Count 692885132 Cipherorig and Ciphertext match
Alphabet searched is 'adbcdef5ghijklpmnoqrstuwxyz123467890'  Length: 36
******************************************************************************

           Success!! The key is  pa55wd##########

           Found by thread 3.   No of threads: 4
******************************************************************************
Execution time = 730.44 seconds

[11/15/18]seed@VM:~/.../Assignment$ 
```

23/11/18 Session Time 4:05 hr Cumulative Time = 47:35 hr

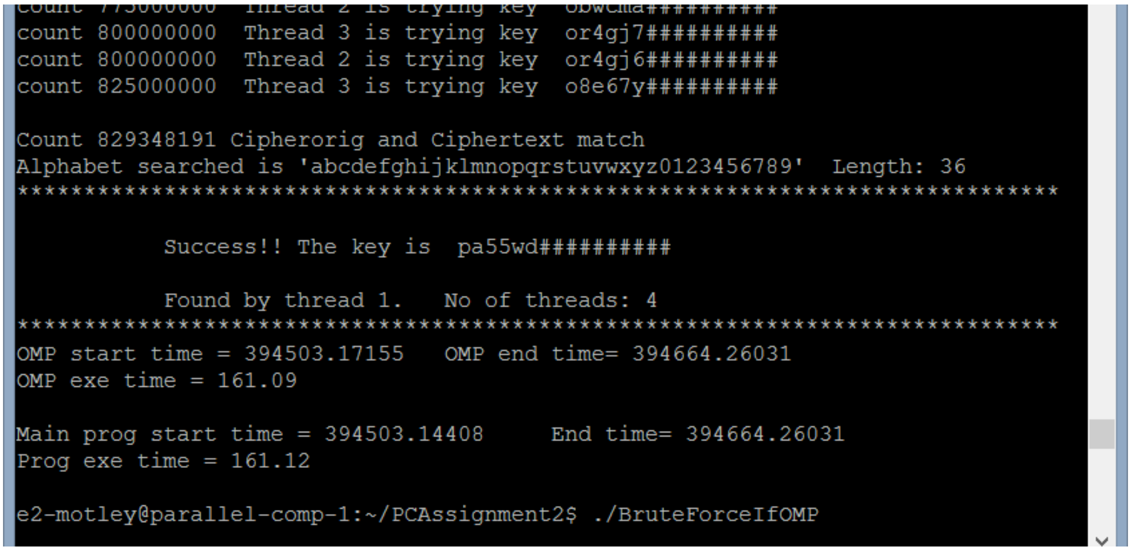
Time spent testing the OMP version. Instead of parallelizing the outer loop, I have amended program so that the inner loop is parallelized. This produces slightly inconsistent results and is not as fast as parallelizing the outer loop. Also experimented with using the "collapse" OMP command which will collapse 2 of the loops and parallelize them at the same time. Again, this did not speed up the process.

Outer loop parallelization: approx. 129s, using standard alphabet order



```
count 650000000  Thread 0 is trying key  l2log7##########
count 675000000  Thread 2 is trying key  mb04qp##########
count 700000000  Thread 0 is trying key  pazxbs##########

Count 700708364 Cipherorig and Ciphertext match
Alphabet searched is 'abcdefghijklmnopqrstuvwxyz0123456789'  Length: 36
*****************************************************************************

          Success!! The key is  pa55wd##########

          Found by thread 0.   No of threads: 4
*****************************************************************************
OMP start time = 394699.588     OMP end time= 394829.18859
OMP exe time = 129.6
```

Inner loop parallelization. Standard alphabet order. Approx 161s



```
count 800000000  Thread 3 is trying key  or4gj7##########
count 800000000  Thread 2 is trying key  or4gj6##########
count 825000000  Thread 3 is trying key  o8e67y##########

Count 829348191 Cipherorig and Ciphertext match
Alphabet searched is 'abcdefghijklmnopqrstuvwxyz0123456789'  Length: 36
*****************************************************************************

          Success!! The key is  pa55wd##########

          Found by thread 1.   No of threads: 4
*****************************************************************************
OMP start time = 394503.17155   OMP end time= 394664.26031
OMP exe time = 161.09

Main prog start time = 394503.14408     End time= 394664.26031
Prog exe time = 161.12

e2-motley@parallel-comp-1:~/PCAssignment2$ ./BruteForceIfOMP
```

26/11/18 Session Time 4:10 hr Cumulative Time = 51:45 hr

Goal

To research parallelization using MPI.

No coding completed. Research only.

28/11/18 Session Time 3:00 hr Cumulative Time = 54:45 hr

Goal

To learn how to use GitHub and to connect to the cluster.

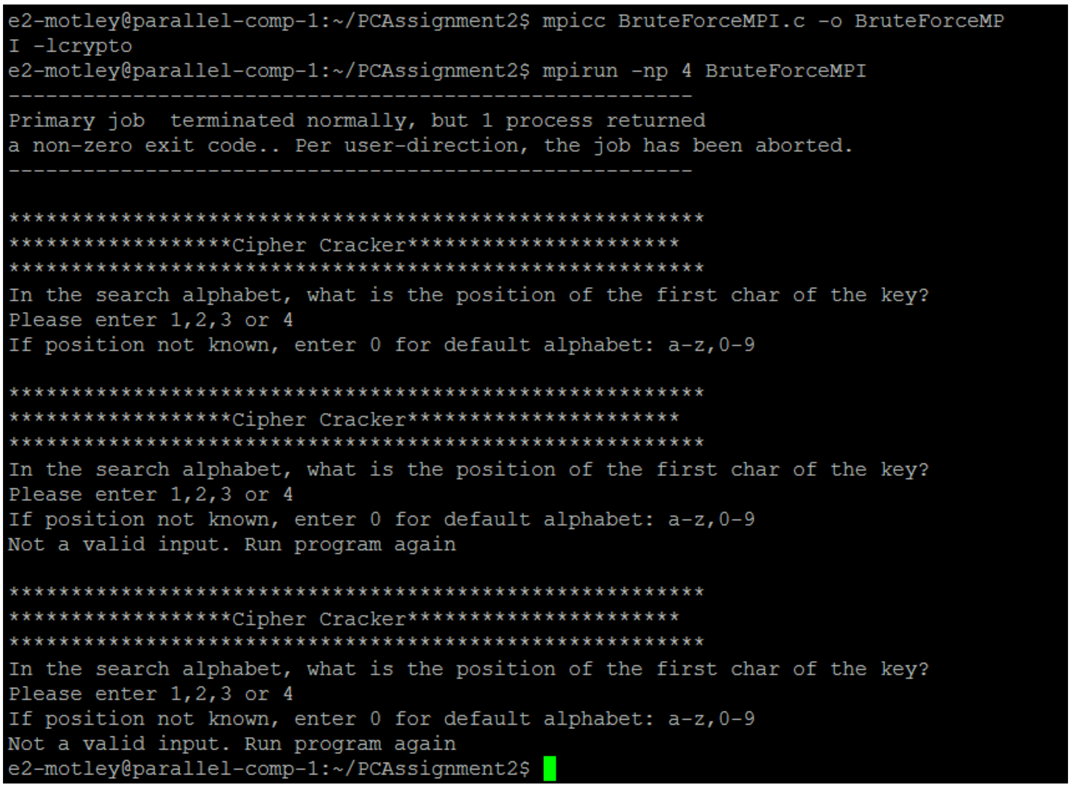
Serial and OpenMP versions successfully ran on the cluster.

29/11/18 Session Time 4:15 hr Cumulative Time = 59:00 hr

Goal

To implement MPI parallelization.

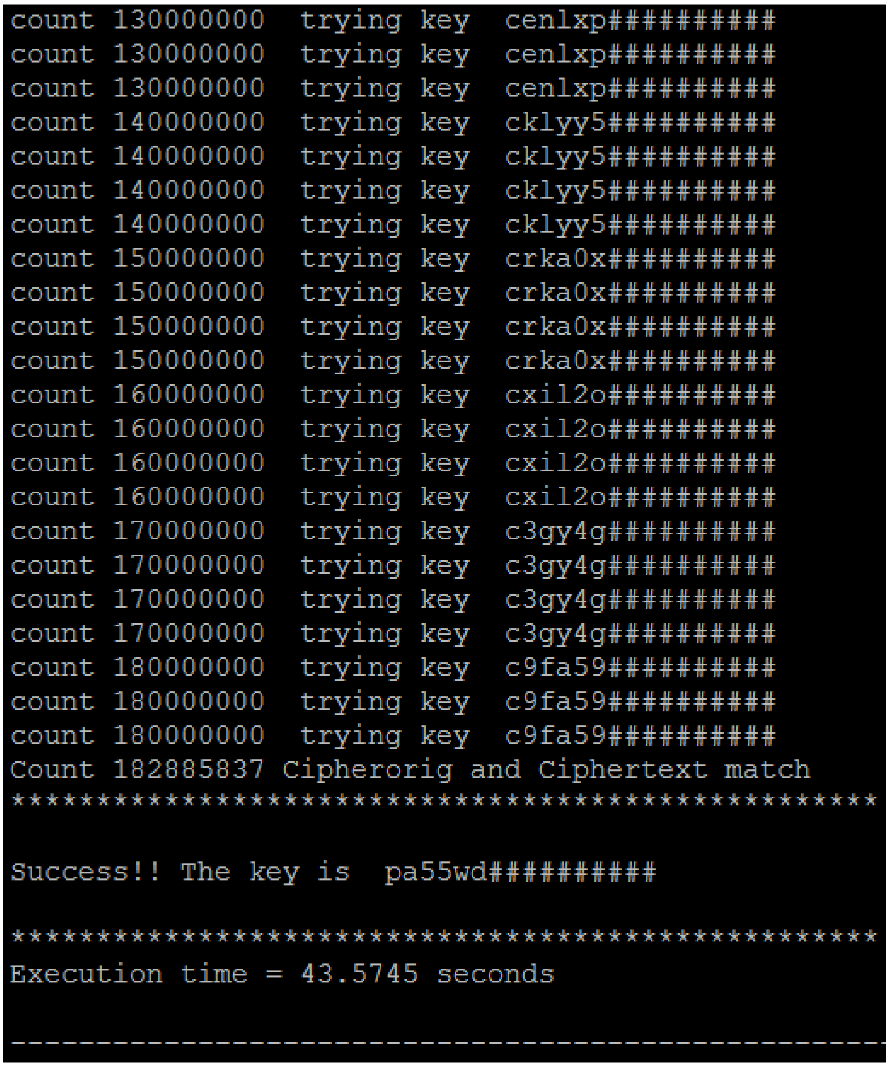
An MPI wrapper was placed round the whole program, but not successful, as the requests for user input for the alphabets printed 3 times, and then program exited as no valid user input was made.



```
e2-motley@parallel-comp-1:~/PCAssignment2$ mpicc BruteForceMPI.c -o BruteForceMP
I -lcrypto
e2-motley@parallel-comp-1:~/PCAssignment2$ mpirun -np 4 BruteForceMPI
-------------------------------------------------------
Primary job  terminated normally, but 1 process returned
a non-zero exit code.. Per user-direction, the job has been aborted.
-------------------------------------------------------

********************************************************
******************Cipher Cracker**********************
********************************************************
In the search alphabet, what is the position of the first char of the key?
Please enter 1,2,3 or 4
If position not known, enter 0 for default alphabet: a-z,0-9

********************************************************
******************Cipher Cracker**********************
********************************************************
In the search alphabet, what is the position of the first char of the key?
Please enter 1,2,3 or 4
If position not known, enter 0 for default alphabet: a-z,0-9
Not a valid input. Run program again

********************************************************
******************Cipher Cracker**********************
********************************************************
In the search alphabet, what is the position of the first char of the key?
Please enter 1,2,3 or 4
If position not known, enter 0 for default alphabet: a-z,0-9
Not a valid input. Run program again
e2-motley@parallel-comp-1:~/PCAssignment2$ 
```

I removed user input and use fixed alphabet to check that basic program works with MPI.

The program executed successfully, but the parallelization not successful as each process generated and tested all keys.



```
count 130000000  trying key  cenlxp##########
count 130000000  trying key  cenlxp##########
count 130000000  trying key  cenlxp##########
count 140000000  trying key  cklyy5##########
count 140000000  trying key  cklyy5##########
count 140000000  trying key  cklyy5##########
count 140000000  trying key  cklyy5##########
count 150000000  trying key  crka0x##########
count 150000000  trying key  crka0x##########
count 150000000  trying key  crka0x##########
count 150000000  trying key  crka0x##########
count 160000000  trying key  cxil2o##########
count 160000000  trying key  cxil2o##########
count 160000000  trying key  cxil2o##########
count 160000000  trying key  cxil2o##########
count 170000000  trying key  c3gy4g##########
count 170000000  trying key  c3gy4g##########
count 170000000  trying key  c3gy4g##########
count 170000000  trying key  c3gy4g##########
count 180000000  trying key  c9fa59##########
count 180000000  trying key  c9fa59##########
count 180000000  trying key  c9fa59##########
Count 182885837 Cipherorig and Ciphertext match
***************************************************

Success!! The key is  pa55wd##########

***************************************************
Execution time = 43.5745 seconds

---------------------------------------------------
```

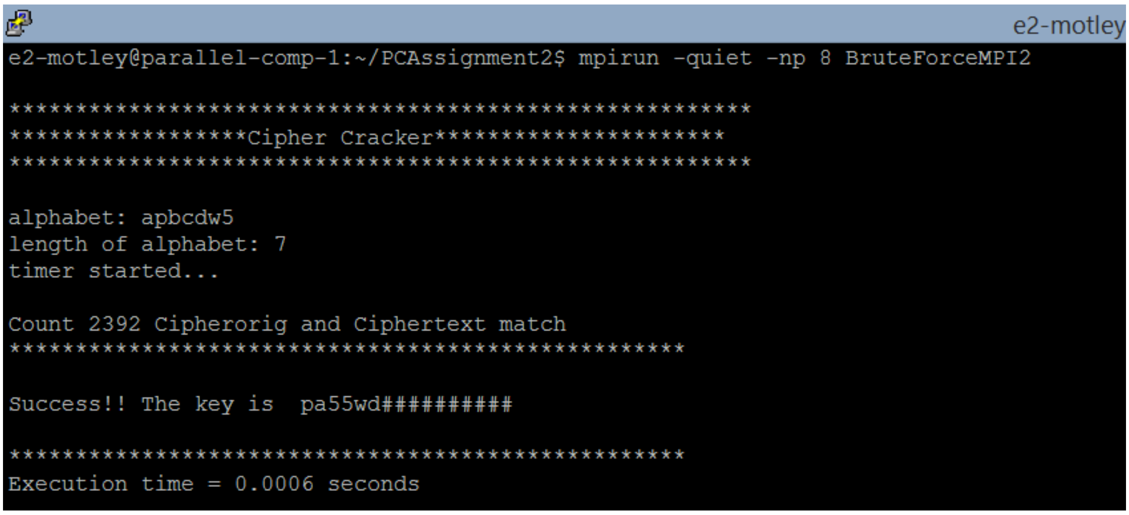
The search space needs to be divided between the number of processes. There are **2,176,782,336** possible permutations.

Initial attempts failed. I spent some time investigating the use of the “SCATTER” function. Again, this proved not to be successful, as unsure of the parameters to use in the function call.

I then decided on a simpler approach – to parallelize based on the outer “for loop” of the key generation. Instead of looping from the first to the last element of the alphabet array, I split this up so that the iterations were divided among the number of processes:

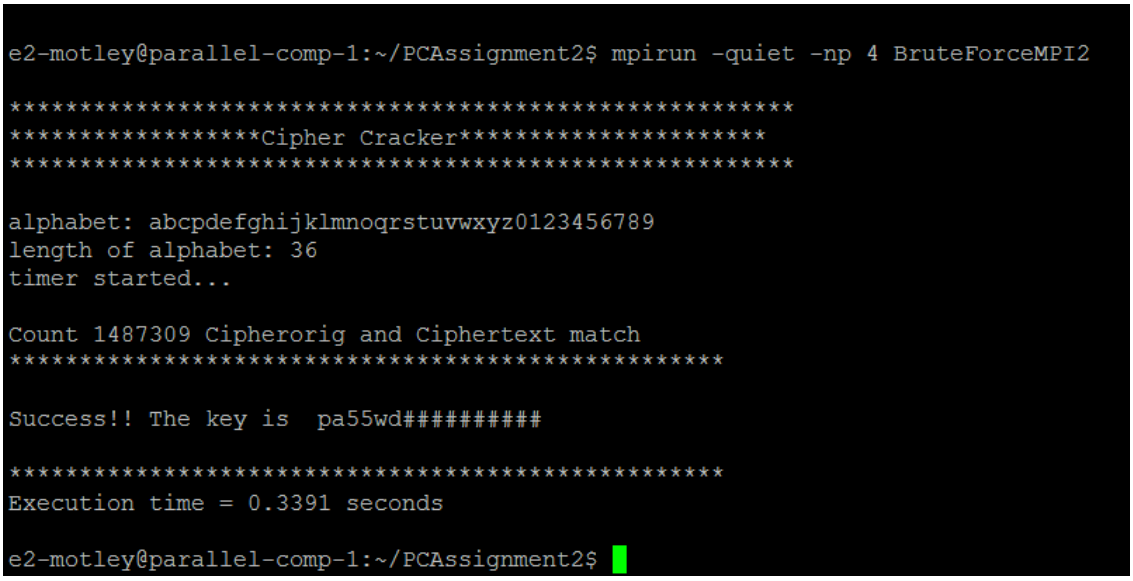
```
for (i = id; i < s; i = i+procs)        (where s = length of the alphabet array)
```

To test I amended the alphabet array so that it only contained 7 characters.



```
e2-motley@parallel-comp-1:~/PCAssignment2$ mpirun -quiet -np 8 BruteForceMPI2

********************************************************
******************Cipher Cracker**********************
********************************************************

alphabet: apbcdw5
length of alphabet: 7
timer started...

Count 2392 Cipherorig and Ciphertext match
***************************************************

Success!! The key is  pa55wd##########

***************************************************
Execution time = 0.0006 seconds
```

This was successful so the size of the search alphabet was increased 36 and positioned the first character of the key to position 4. Tested using 4 processors and successfully found the key in approx. 0.3 s



```
e2-motley@parallel-comp-1:~/PCAssignment2$ mpirun -quiet -np 4 BruteForceMPI2

********************************************************
******************Cipher Cracker**********************
********************************************************

alphabet: abcpdefghijklmnoqrstuvwxyz0123456789
length of alphabet: 36
timer started...

Count 1487309 Cipherorig and Ciphertext match
***************************************************

Success!! The key is  pa55wd##########

***************************************************
Execution time = 0.3391 seconds

e2-motley@parallel-comp-1:~/PCAssignment2$
```

Having the bare bones of the program working, the next stage is to add user input for the OpenMP version for the number of threads. Once all three versions of the program (serial, OpenMP, and MPI) have consistent features actual timings can be collated and analysed.

01/12/18 Session Time 6:30 hr Cumulative Time = 65:30 hr

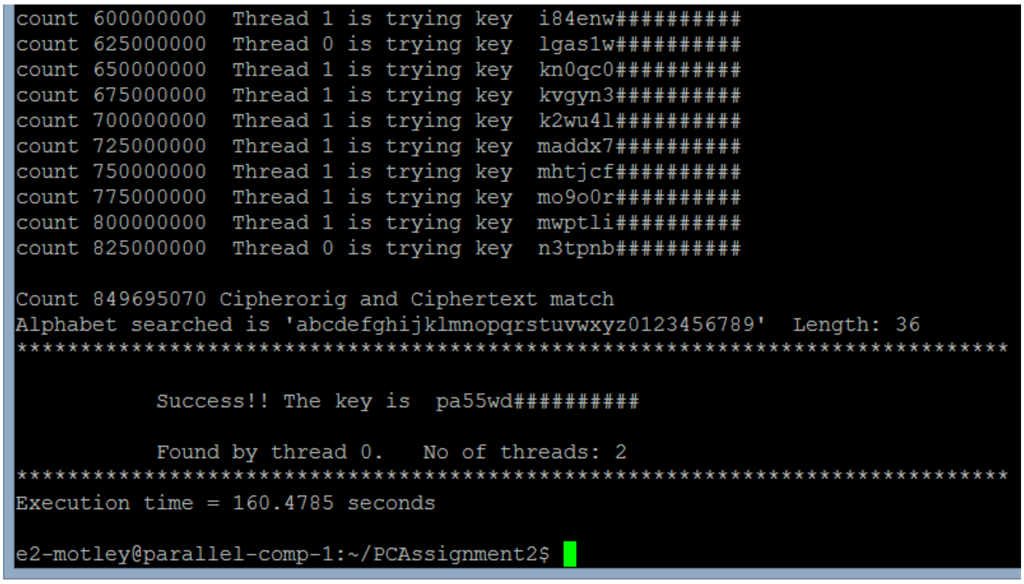
Goal

To ensure consistent features on all 3 versions (including accurate timings)

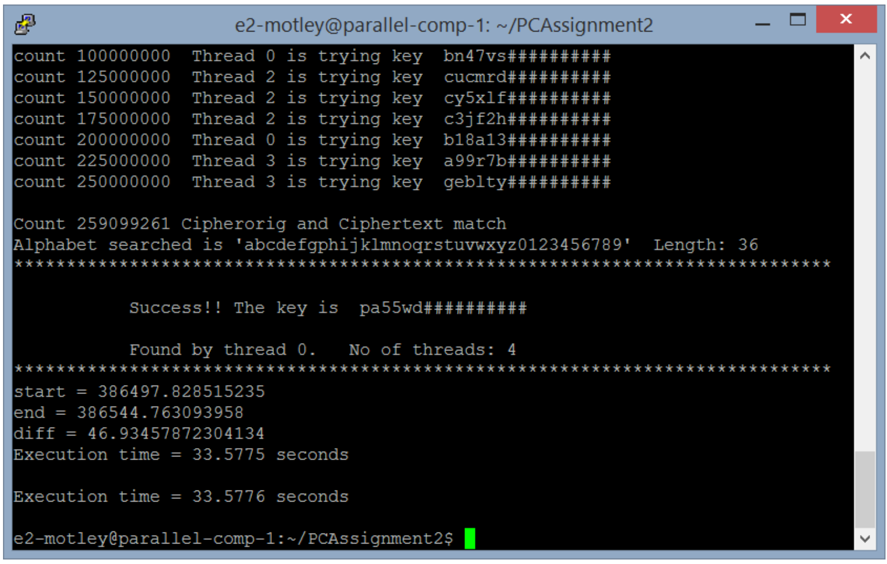
To take user input for no. of threads on OpenMP

User input for 6 different alphabet orders was added to all programs. User input for the no. of threads was also successfully implemented (take user input variable at end of initial pragma statement):

#pragma omp parallel ……… num_threads(2)



```
count 600000000  Thread 1 is trying key  i84enw##########
count 625000000  Thread 0 is trying key  lgas1w##########
count 650000000  Thread 1 is trying key  kn0qc0##########
count 675000000  Thread 1 is trying key  kvgyn3##########
count 700000000  Thread 1 is trying key  k2wu41##########
count 725000000  Thread 1 is trying key  maddx7##########
count 750000000  Thread 1 is trying key  mhtjcf##########
count 775000000  Thread 1 is trying key  mo9o0r##########
count 800000000  Thread 1 is trying key  mwptli##########
count 825000000  Thread 0 is trying key  n3tpnb##########

Count 849695070 Cipherorig and Ciphertext match
Alphabet searched is 'abcdefghijklmnopqrstuvwxyz0123456789'  Length: 36
******************************************************************************

          Success!! The key is  pa55wd##########

          Found by thread 0.   No of threads: 2
******************************************************************************
Execution time = 160.4785 seconds

e2-motley@parallel-comp-1:~/PCAssignment2$
```



```
count 100000000  Thread 0 is trying key  bn47vs##########
count 125000000  Thread 2 is trying key  cucmrd##########
count 150000000  Thread 2 is trying key  cy5xlf##########
count 175000000  Thread 2 is trying key  c3jf2h##########
count 200000000  Thread 0 is trying key  b18a13##########
count 225000000  Thread 3 is trying key  a99r7b##########
count 250000000  Thread 3 is trying key  geblty##########

Count 259099261 Cipherorig and Ciphertext match
Alphabet searched is 'abcdefgphijklmnoqrstuvwxyz0123456789'  Length: 36
******************************************************************************

          Success!! The key is  pa55wd##########

          Found by thread 0.   No of threads: 4
******************************************************************************
start = 386497.828515235
end = 386544.763093958
diff = 46.93457872304134
Execution time = 33.5775 seconds

Execution time = 33.5776 seconds

e2-motley@parallel-comp-1:~/PCAssignment2$
```

e

Screenshot showing different timings from `time()` and `omp_get_wtime()`

For the MPI version it was not possible to take user input for the no. of processors to use. This is a parameter that can only be specified at runtime.

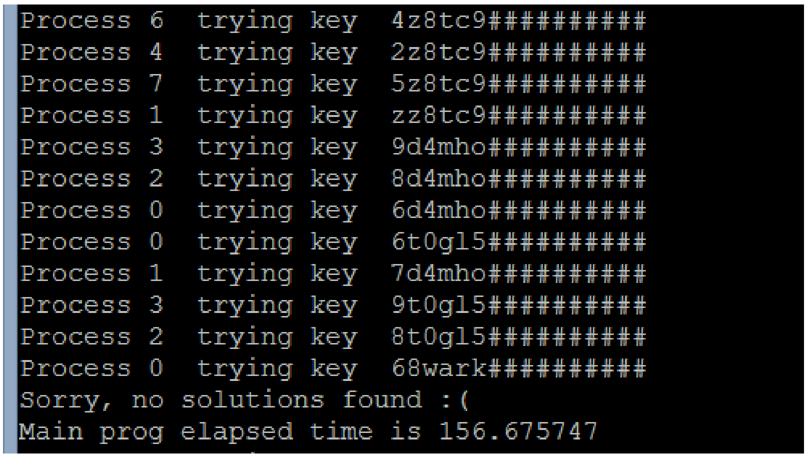
02/12/18 Session Time 13:15 hr Cumulative Time = 78:45 hr

Goals:

To add all necessary search alphabets for testing
To check all program versions have accurate timings.
To formally test the programs.

I added another 2 search alphabet versions to all programs, one where “p” will be the last element, and will thus take the max length of time to find, and one where “p” is not included, so a no success search.

For the MPI version, I had to ensure that only one process took user input for choice of alphabet and then broadcast this to the remaining processes, and used MPI_Wtime() for execution timings.



```
Process 6  trying key  4z8tc9##########
Process 4  trying key  2z8tc9##########
Process 7  trying key  5z8tc9##########
Process 1  trying key  zz8tc9##########
Process 3  trying key  9d4mho##########
Process 2  trying key  8d4mho##########
Process 0  trying key  6d4mho##########
Process 0  trying key  6t0gl5##########
Process 1  trying key  7d4mho##########
Process 3  trying key  9t0gl5##########
Process 2  trying key  8t0gl5##########
Process 0  trying key  68wark##########
Sorry, no solutions found :(
Main prog elapsed time is 156.675747
```

Formal tests can now run.

03/12/18 Session Time 6:15 hr Cumulative Time = 85:00 hr

Goals:

To complete test runs

To compile and analyse results.

05/12/18 Session Time 9:30 hr Cumulative Time = 94:30 hr

Goals:

To complete Log Book and Results.

Create make file

7 TIME LOG

Day	Date	Start Time	End Time	Breaks	Working Time	Cumulative Time
Sun	04/11/2018	10:00	16:15	00:45	**05:30**	05:30
Mon	05/11/2018	10:25	16:30	00:50	**05:15**	10:45
Tue	06/11/2018				00:00	10:45
Wed	07/11/2018	12:10	13:45	00:00	**01:35**	12:20
Thu	08/11/2018	11:15	16:00	00:30	**04:15**	16:35
Fri	09/11/2018	14:20	17:45	00:00	**03:25**	20:00
Sat	10/11/2018	10:20	16:30	00:45	**05:25**	25:25
Sun	11/11/2018	09:45	14:00	00:00	**04:15**	29:40
Mon	12/11/2018				00:00	29:40
Tue	13/11/2018	09:45	15:15	01:00	**04:30**	34:10
Wed	14/11/2018				00:00	34:10
Thu	15/11/2018	10:30	16:15	00:45	**05:00**	39:10
Fri	16/11/2018				00:00	39:10
Sat	17/11/2018				00:00	39:10
Sun	18/11/2018				00:00	39:10
Mon	19/11/2018	10:15	15:20	00:45	**04:20**	43:30
Tue	20/11/2018				00:00	43:30
Wed	21/11/2018				00:00	43:30
Thu	22/11/2018				00:00	43:30
Fri	23/11/2018	10:45	15:25	00:35	**04:05**	47:35
Sat	24/11/2018				00:00	47:35
Sun	25/11/2018				00:00	47:35
Mon	26/11/2018	10:15	15:10	00:45	**04:10**	51:45
Tue	27/11/2018				00:00	51:45
Wed	28/11/2018	12:00	15:00	00:00	**03:00**	54:45
Thu	29/11/2018	11:15	15:30	00:00	**04:15**	59:00
Fri	30/11/2018				00:00	59:00
Sat	01/12/2018	09:45	17:30	01:15	**06:30**	65:30
Sun	02/12/2018	09:44	23:59	01:00	**13:15**	78:45
Mon	03/12/2018	09:45	17:00	01:00	**06:15**	85:00
Tue	04/12/2018				00:00	85:00
Wed	05/12/2018	10:00	22:00	02:00	**10:00**	95:00
Thu	06/12/2018				00:00	95:00

8 REFERENCES