Problem 4 - Solution: CUDA Password Cracker

Implementation and Design details:

The only differences between design and implementation in CUDA with respect to ISPC/Pthreads program in Project 2 are:

1. In Cuda, we have used shared memory to store the intermeidate words from different threads in a device. And each shared memory present within each device speeds up the processes as we won't have to load the data again and again from the main memory.
2. In ISPC/Pthreads, we had generated files for all the character length with all the combinations of the characters during the compilation. For example: 1.txt used to store combinations of single letter password with letters a-zA-z0-9. Similarly, 2.txt used to store aa,ab,..aA,aB...a0,a9,...za,zz,...zA,zB,..z0,z9,.....99 combinations of characters. Furthermore, 3.txt store aaa,aab.....999 and 4.txt store aaaa,aaab,....9999. Now, storing all the combinations of password of length 1-8 was not possible because of storage constraints as the file gets larger in size after 4 characters length and the program while compilation crashes due to insufficient storage. So while running the program, code used to generate different character passwords from the given hash and compare it with words from each of these files to get the correct password. In CUDA, we changed the approach where we are no longer generating these files and we are generating these combinations in runtime saving a lot of physical memory in the system.
3. In CUDA, due to above point where we are generating the combination of passwords during the runtime, the program supports length of password between 1 to 8 unlike in ISPC/Pthreads, where we were just able to do it for the length of 1 to 4 due to storage constraints resulted due to the program design.
4. In CUDA, we are creating some number of threads in different blocks and these blocks later constitute to form a grid altogether. These grids can be considered as part of a core and similarly we would have as many grids as the number of cores in the system. These designs make the core run faster because all threads within the block work together irrespective of threads present in other blocks. And all the blocks work together in a single grid. Thus, there is less synchronization required between threads as we just have to work within the block compared to Pthreads where all threads or tasks launched need to be synchronized.
5. In Pthreads/ISPC, the charset were a-z, A-Z, and 0-9 while in CUDA we have added support for "**#**" too and flexibility to add more characters is allowed in the code.

Steps to compile and execute the program:

NOTE:

1. We have implemented MD5 Hashing algorithm
2. We have added support for below ASCII Characters - 0 to 9, a to z and A to Z and #
3. Password lengths is from 1 to 8 characters length
4. We highly recommend to use MD5 hash as we have implemented MD5 algorithm to get the correct decoded password

STEPS:

1. Navigate to Problem 4 folder and run ***make***
2. This will start the compilation process and generate an executable file **md5_password_cracker_gpu**
3. Execute the program: **./md5_password_cracker_gpu** ***<md5 hash>***
4. NOTE: MD5 hash must be of length 32 for execution

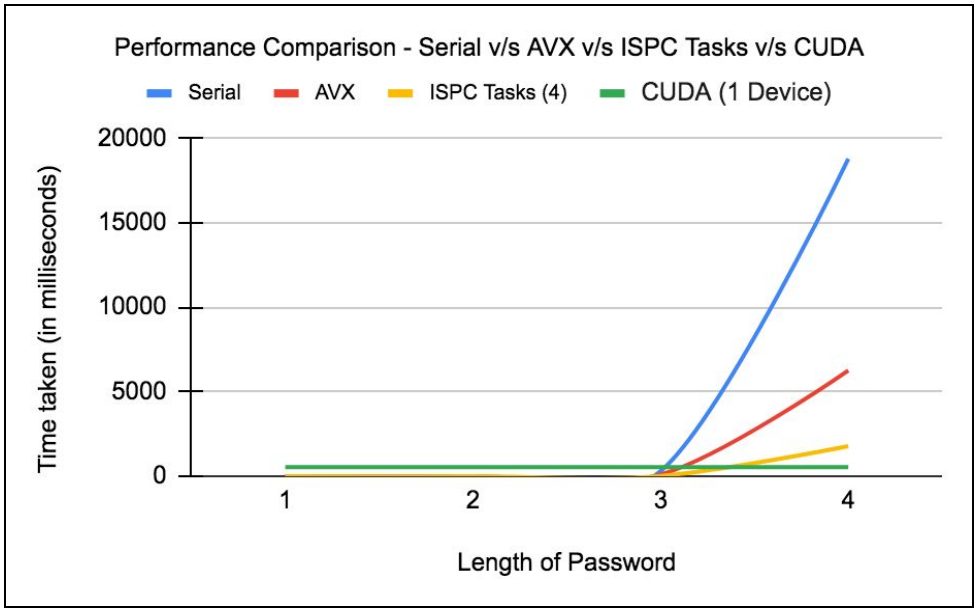
Analysis:

A. Performance Comparison of performance between CUDA and Pthreads:

1. We have compared CUDA with Pthreads/ISPC tasks, Serial and AVX Intrinsics runtime that was done in project 2.
2. The number of passwords tested in project 2 were four with length 1 to 4 and hence we have chosen the same passwords of the same length in CUDA too for comparison.
3. Below is the table which shows performance i.e. time taken to crack password for a given hash value for CUDA in Personal Laptop with respect to that of the methods implemented in Project 2 - Serial, AVX and ISPC Tasks:

Password	Length of Password	Serial (ms)	AVX (ms)	ISPC Tasks (4) (ms)	CUDA in Personal Laptop (ms)
Z	1	0	0	0	544.936
9Q	2	4	1	0	538.489
Ac4	3	305	101	29	546.72
zzzz	4	18784	6266	1790	541.156

4. Below is the graph created based on the above table for better visualization:



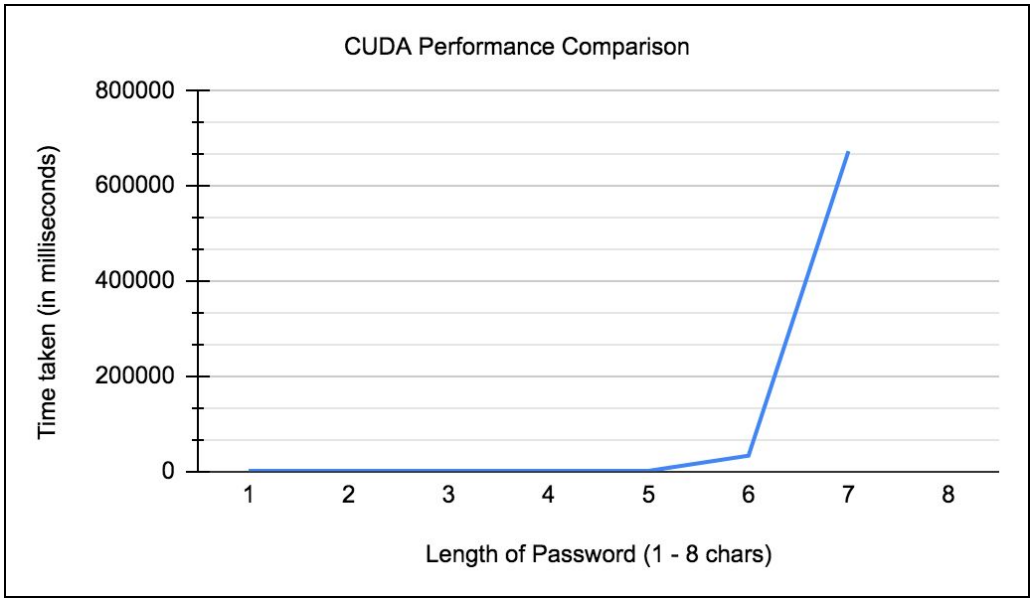
5. Based on the above results table and graph we are able to relate the performance difference between Serial, AVX, ISPC/Pthreads and CUDA parallel implementation for brute force password cracker problem.
    a. Ideally, the CUDA program must have lower time then all other implementations specially for lower length of passwords (length 1,2 and 3) but as per the table and graph we see that it takes longer time to crack the password. Two very main reasons which we think of are:
    1) CUDA program that we are testing is done on personal computers which have lower hardware resources compared to Project 2 implementations of AVX, ISPC/Pthreads and Serial as they were experimented in Orbit systems.
    2) Another reason is the implementation and design difference. During project 2, we had the constraint that we can only crack passwords between 1-4 charset because during compilation it stores all combinations of passwords between 1-4 in physical files which the program uses in runtime to crack the password. Using more than 4 i.e. 1-8 was not supported. We had to change that in CUDA and that is the reason why we implemented the logic in CUDA where at runtime these combinations of passwords will be generated of different lengths and will be compared with cracked hashes.
    b. Now, to defend the results and above point, we can see that performance of passwords starting length of 4 is lower in case of project 2 implementations while its higher in CUDA. CUDA takes less time to crack passwords of length 4 compared to other implementations.
    c. Lastly, the performance of CUDA could be enhanced to even greater extent if the laptop/machine has higher specifications in terms of GPU card, RAM and CPU. The personal laptop has less hardware resources and has only one GPU Device as seen in the later part of outputs, the performance doesn't seem to be that great.

B. <u>Performance Comparison of performance of CUDA with Password Lengths between 1 to 8 characters:</u>
Performance in Personal Laptop:

1. We performed program execution in personal laptop where we had below specifications of the machine: **NVIDIA GPU card**: NVIDIA GeForce GTX 1650, **RAM**: 8 GB , **Processor**: Intel i9
2. Below is the table which shows time taken to crack password between length 1-8:

Password	Length of Password	Time Taken (ms)
Z	1	544.936
9Q	2	538.489
Ac4	3	546.72
zzzz	4	541.156
Yc2ac	5	544.264
a2B4c8	6	32581.1
rutgers	7	673076
PdcF2020	8	NAN

3. Below is the representation in the form of line chart which shows performance of password cracker program in CUDA for passwords of length 1-8:



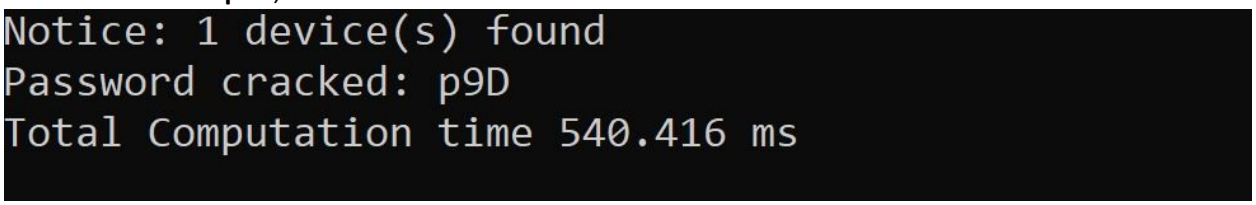
4. Based on the above results table and graph we are able to relate the performance difference between different character lengths password:
    a. Firstly, the performance of CUDA could be enhanced to even greater extent if the laptop/machine has higher specifications in terms of GPU card, RAM and CPU. The personal laptop has less hardware resources and has only **one** GPU Device as seen in the later part of outputs, the performance doesn't seem to be that great.
    b. As seen in the table and graph above, the performance is almost the same for passwords of length 1-5 as they are of shorter length. However, as the length increases

we can see that the performance degrades i.e. takes longer time to finish the execution. This is mainly because as discussed earlier the design and implementation is such that it generates all the combinations of passwords of different length at **RUNTIME** to compare and find the cracked password. Hence, for lower length time taken is almost the same but as the length increases from 5 to 8, the time taken increases.

c. Now, as we can see that there is no value associated with password of length 8 (i.e. NAN) because when we executed the program for length of 8, it took forever to complete. This is again because it generated all combinations of passwords of length 8 for the 63 characters set i.e. a-z,A-Z,0-9 and #. Also, since this is a personal laptop with lower hardware resources and less GPU capability, it takes a large amount of time to finish the program.

C. <u>Other results based on the Project 3 Questions in Sakai:</u>

1. We have been asked to run the code for two 3 character length passwords and two 4 character length passwords. Below are the related hashes, passwords and output screenshots of the same:
2. Password: **p9D**, HASH: **80b7ac9c28f8755da21a2ea0cd96c60c**



3. Password: **29z** , HASH: **85dde44d7e811346f1d8128abd96cea4**

4. Password: **p9D4**, HASH: **c2f7bbc0747ffe27df22dbcb716adaec**

5. Password: **sM18**, HASH: **24e92e5d0c16fb175c6006b2af14bdfa**

6. Password: **Z2t#**, HASH: **cd323c239e5cb0e17b0b2a6204c60854**

```
Notice: 1 device(s) found
Password cracked: Z2t#
Total Computation time 537.298 ms
```

7. In the last screenshot, we can see the output for password with char # as well which was **NOT** supported in Project 2.
8. We were not able to execute the program for **bv37qi#f** because the characters length is 8 and the program takes forever to complete because of limited hardware.