GPU Programming - Final Project - Report
Unsalted MD5 Password cracking and using GPU

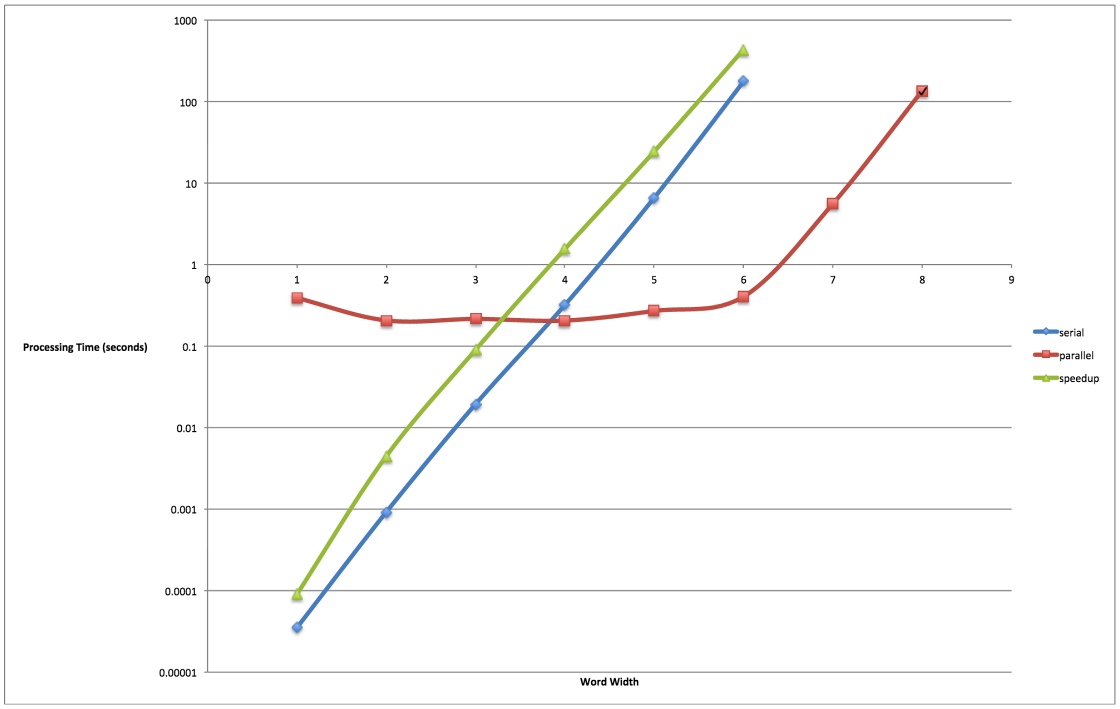
Team: Michael Peterson, Kevin Chumbley, Abdullah Almutairi

Running the Program

```
mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/Turn In$ ./project
./project
usage: ./project targetWord [-s][-v]
       -s: perform serial Version (if omitted, the paralell version will be used)
       -v: Be verbose (if omitted only the time cost of the implementation will be printed


mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/ParallelVersion$ make test
make test
makefile:14: warning: overriding commands for target `test'
makefile:5: warning: ignoring old commands for target `test'
echo  > output.txt
echo "length,serial,parallel" > output.txt
echo "1,`./md5Gpu z -s`,`./md5Gpu z`" >> output.txt
echo "1,`./md5Gpu z -s`,`./md5Gpu z`" >> output.txt
echo "2,`./md5Gpu zz -s`,`./md5Gpu zz`" >> output.txt
echo "2,`./md5Gpu zz -s`,`./md5Gpu zz`" >> output.txt
echo "3,`./md5Gpu zzz -s`,`./md5Gpu zzz`" >> output.txt
echo "3,`./md5Gpu zzz -s`,`./md5Gpu zzz`" >> output.txt
echo "4,`./md5Gpu zzzz -s`,`./md5Gpu zzzz`" >> output.txt
echo "4,`./md5Gpu zzzz -s`,`./md5Gpu zzzz`" >> output.txt
echo "5,`./md5Gpu zzzzz -s`,`./md5Gpu zzzzz`" >> output.txt
echo "5,`./md5Gpu zzzzz -s`,`./md5Gpu zzzzz`" >> output.txt
echo "6,`./md5Gpu zzzzzz -s`,`./md5Gpu zzzzzz`" >> output.txt
echo "6,`./md5Gpu zzzzzz -s`,`./md5Gpu zzzzzz`" >> output.txt
echo "7,-----,`./md5Gpu zzzzzzz`" >> output.txt
echo "7,-----,`./md5Gpu zzzzzzz`" >> output.txt
echo "8,-----,`./md5Gpu zzzzzzzz`" >> output.txt
echo "8,-----,`./md5Gpu zzzzzzzz`" >> output.txt
mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/ParallelVersion$ ls
ls
errout.txt  makefile  md5Gpu        newKernel   README.txt  test.cu   test.txt   timing.h
main.cu     md5.cu    md5kernel.cu  output.txt  runTest.sh  test.php  timing.cu
mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/ParallelVersion$ cat output.txt
cat output.txt
length,serial,parallel
1,0.000035,0.411324
1,0.000035,0.359664
2,0.000920,0.190275
2,0.000917,0.222556
3,0.017903,0.223310
3,0.020938,0.207766
4,0.338724,0.205371
4,0.300411,0.205790
5,6.604192,0.270746
5,6.596024,0.269526
6,176.664046,0.404638
6,176.516597,0.404078
7,-----,5.593492
7,-----,5.570958
8,-----,135.579340
8,-----,135.861491

mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/Turn In$ ./project abcde -s -v
./project abcde -s -v
hash for abcde: 0xd9b456ab3a71402b99f85acc86b7d485
performing serial brute force md5 password hash cracking...
---------- Serial Version ---------------
FOUND: abcde
Time Cost: 1.362548
mpeterson10@cscd-gpu01:~/github/CSCD439-Project-cuda-md5-Fork/Turn In$ ./project abcde -v
./project abcde -v
hash for abcde: 0xd9b456ab3a71402b99f85acc86b7d485
performing parallel brute force md5 password hash cracking...
---------- Parallel Version ---------------


FOUND: abcde
Time Cost: 0.276208
```

Results

We focused our study on the worst case for our application. This occurs when the password we want find is the last combination we search. The worst case is when the word is a sequence of the lowercase letter “z”.



We found at most 500 times speedup over the serial version when the input was 6 characters long. Beyond 6 characters the serial version is too slow to observe and record for the scope of this project.

Introduction

The purpose of this project was to understand an efficient method for a brute force search of passwords encrypted with MD5 hash function on a GPU. We want to show the performance of a GPU along side of a CPU. Also, we wanted to improve where the original code creator left off.

Research

We searched for a reference to use in our project. We found an older project on Github that worked after re-configuring to the new capabilities of our server. We initially thought that the reference we found had some room for improvement. As it turns out, it was very challenging to implement any changes and analyze the results. Because it was very hard to understand the program since there is so much going on. After we attempted rewrite the code in a more elegant way we found a flaw with our formulation of the kernel.

Method

We wrote a clean and fast serial version to make a comparison against. We created a command line interface for our program and a bash script to run our batch of test case. We compare the Serial and Parallel runtimes and find the speed up. Unfortunately, the script will not work on the GPU server due to a specific issue with the version of bash. We tested on Linux and OSX.

Once we had a stable program we again looked for optimizations. We experimented with various parameters to find a more efficient kernel configuration and batch size to increase the speed. We also reduced the scope of the problem by increasing the constraints on the input. Instead of searching all printable ASCII characters we search for only lower case characters. We also removed the salt of the hash to further reduce the problem size. We made these simplifications for the sake of time and the project’s success.

To simulate password cracking we first hash the word and send the hash into the password cracker kernel. The host function launches many kernels, each of which is responsible for a batch of the permutations of the possible passwords. Within each launch the threads execute a loop to hash the current permutation and check if the current permutation hash matches the password hash. If found we record the password into the output char* called correctPass. When the host copies the array back from the GPU if we find that the array has non-null characters, we are done. Then we record the time.

Future Work

If we want to extend this research we would expand the problem to include a full ASCII character set as well as re-implement the salts on both ends of the hash. We could also run the program in serial or parallel and tailor its execution strategy with the current platform in mind. Also it would be nice to implement a version on an AWS EC2 GPU cluster.

References

1. https://defuse.ca/gpucrack.htm
2. http://opensource.apple.com/source/xnu/xnu-1228.5.18/libkern/crypto/md5.c
3. http://stackoverflow.com/questions/33467876/iterative-solution-to-permutation-calculation-in-c