Fall 2022 ME/CS/ECE759 Final Project Report
University of Wisconsin-Madison

MD5 Hash Calculator

Muhammad Salman Munaf

December 14, 2022

Abstract

The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. MD5 can be used as a checksum to verify data integrity against unintentional corruption. Historically it was widely used as a cryptographic hash function; however it has been found to suffer from extensive vulnerabilities. We will explore one such vulnerability today by cracking the MD5 hash through bruteforce using the power of GPUs. In this project, we will implement a serial approach and a parallel approach using CUDA. We will compare the performance of both of these implementations. The motivation behind this comparison is to understand that with increasing power of GPUs it is getting easier to crack the cryptographic algorithms. In addition, we will explore how much difference it makes to keep a longer password or a password containing a bigger character set.

Link to Final Project `git` repo: https://git.doit.wisc.edu/MUNAF/repo759/-/tree/main/FinalProject759

Contents

1. Problem statement 4
2. Solution description 4
3. Overview of results. Demonstration of your project 4
4. Deliverables: 4
5. Conclusions and Future Work 4
References 4

1. General information

In this short section, please provide only the following information, in bulleted form (four bullets) and in this order:

1. CS Department
2. Current status: MS student
3. I am not interested in releasing my code as open-source code.

2. Problem statement

I took CS 642 - Introduction to Information Security last semester. In this course, we covered various topics such as cryptographic primitives, security protocols and system security. We learned about different hash algorithms and how they are getting easier to crack because of the increase in power of GPUs. I want to take this opportunity to explore one particular type of attack - MD5 Hash Attack which can be run in a reasonable amount of time on a GPU, even on relatively inexpensive consumer-grade hardware and determine how easy it is to crack user passwords. This project will allow me to implement a complicated algorithm on a GPU and iterate over it to optimize its performance.

3. Solution description

My implementation of MD5 Hash Attack goes through all the possible combinations of passwords and calculates the MD5 Hash until it finds the target password. My solution consists of two components: the MD5 hash calculator and a password generator. Firstly, I implemented the algorithm to calculate the MD5 Hash (*getMd5Hash()*) on a CPU using the pseudocode provided in the RFC document. In order to implement the attack, I implemented a function (*next()*) to generate the next permutation of the password from a given character set. These two components run until the target password is found.

The primary focus of this project is to achieve efficient data parallelism by leveraging features of the GPU architecture. In the serial implementation all the password combinations were handled by a single CPU thread. For the CUDA implementation, I divided the password combinations among GPU threads so that the hashes can be computed in parallel and once that is found a global flag is set. The CUDA implementation supports multiple devices and varying block and thread counts.

Lastly, I have gathered results to compare the performance of above implementations and create plots of my results. I also performed analysis on how much time it takes to crack passwords of different lengths and varying strengths and if that makes a difference.

4. Overview of results. Demonstration of your project

Firstly, I compared the performance of serial and CUDA implementation. The primary focus of this project was to evaluate the performance gain achieved by leveraging the power of GPUs and data parallelism. The figure below shows the time taken for both implementations on different lengths of password.

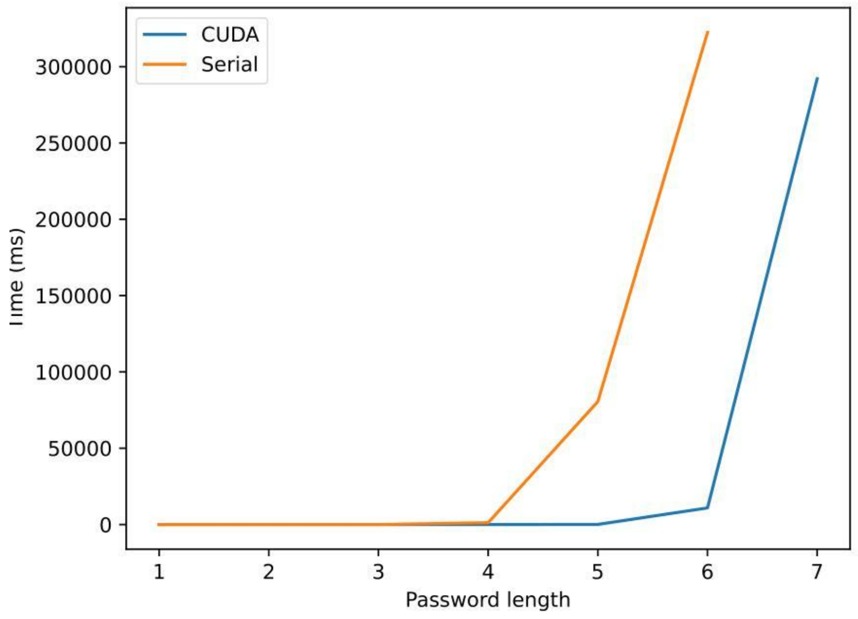

Figure 1

As the plot shows above that time increases exponentially as the length of password increases. This is probably due to the fact that the number of permutations also increases exponentially and the code has to try all of them. Moreover, the plot also shows that after the length of password exceeds 4 characters the CUDA implementation is faster as it divides the combinations among GPU threads and works on them in parallel.

Since, the above plot does not clearly show the performance difference when there are few characters in the password. I generated another plot of Log(Time) vs Length of Password to observe the behavior.

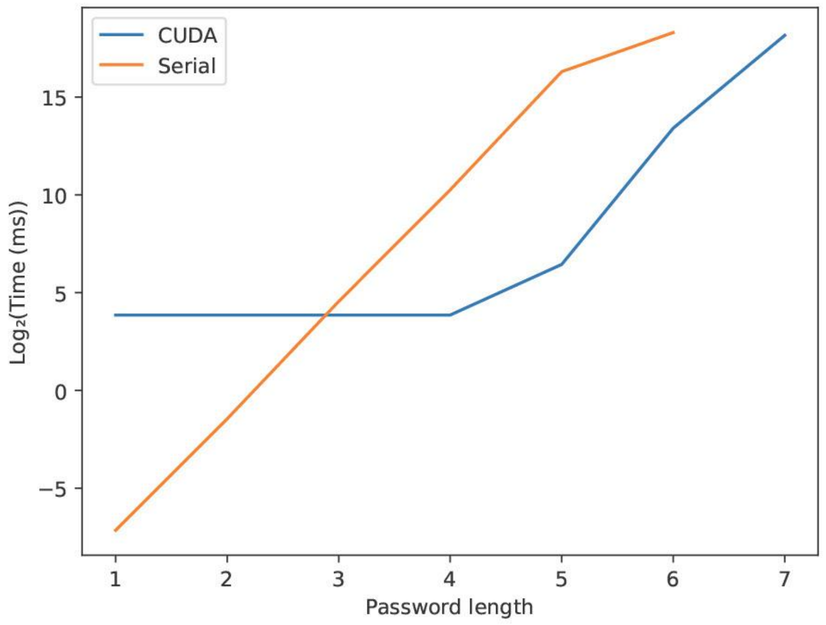

Figure 2

Figure 2 shows that the serial implementation takes less time when the password contains less than 4 characters. I believe this is due to the overhead of transferring data to the GPU and getting it back. The time it takes to transfer and get data exceeds the overall computation time. The above figure shows that the CUDA implementation performance for passwords less than 4 characters is fairly constant which could probably be due to this overhead.

Lastly, the above two plots clearly state that the length of password clearly makes it difficult to crack a password.

In order to analyze how the strength of password affects the computation time I ran the CUDA implementation with different character sets.

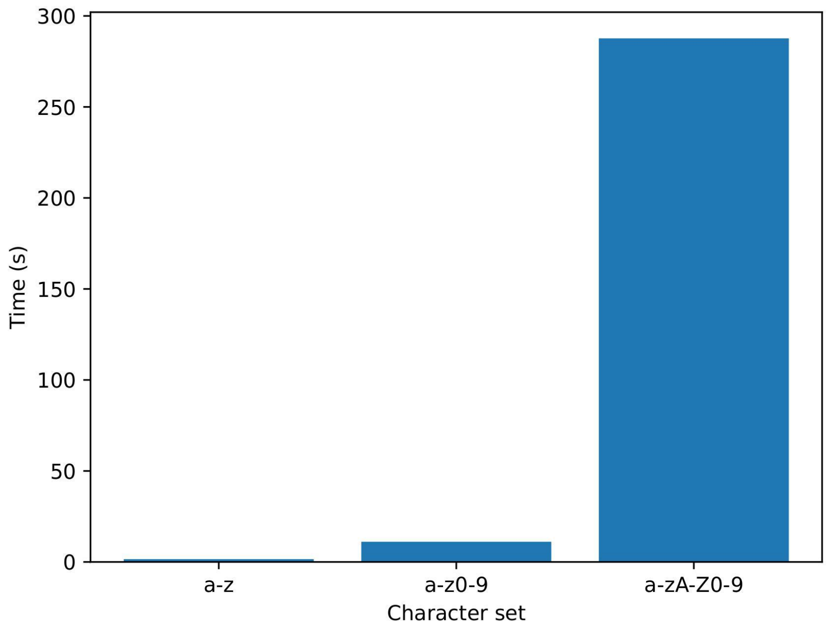

Figure 3

Figure 3 highlights the difference in performance with varying character sets. For this plot, I kept all the other variables like length of password, threads per block, devices etc same. One can observe that the time it takes for password with character set a-zA-Z0-9 is exponentially greater than only character set containing lowercase alphabets (a-z). This shows that increasing the strength of password is one of the best ways to make it tougher to crack.

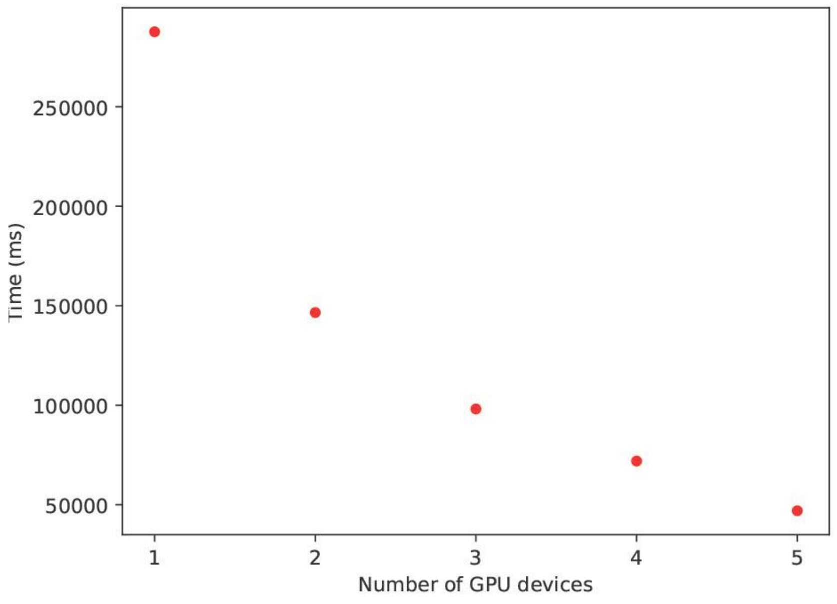

Figure 4

Figure 4 shows how the performance varies as we increase the number of GPUs. The results are expected and they show that more GPUs lead to higher performance as the computations can be divided among GPUs. However, the plot also shows that the performance gain decreases as we add more GPUs which can be due to the fact that less amount of computation gets divided and synchronization overhead also increases.

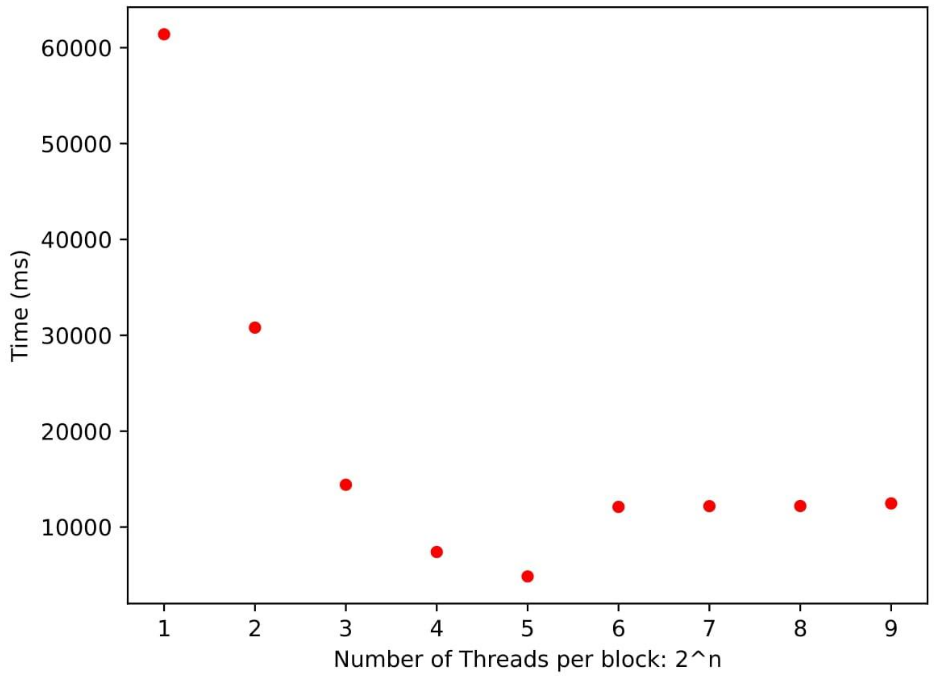

Figure 5

Lastly, I experimented with a varying number of threads per block in order to observe how that affects the performance. Figure 5 shows that the performance increases till 32 threads then it stays constant for a higher number of threads. This might be due to less amount of data to parallelize or higher contention between threads for memory or higher overhead to synchronize the threads. This can be explored further by observing the performance with varying password length so that we have varying workload size as well.

5. Deliverables:

I have presented the code along with this report for the project. My project repository contains three folders. The serial folder contains the code and results for the CPU implementation and the cuda folder contains the code and results for the GPU(CUDA) implementation. The plots folder contains all the plots that are present in this report. In order to run the GPU implementation, you will have to load the following module: nvidia/cuda/11.6.0.lua. The code is tested on Euler machines and the sbatch files provide sufficient information on how to compile and run the code. I have provided the Makefile for both implementations to compile the code. However, I will reiterate in this report as well on how to run both the implementations.

In order to run the CPU implementation, you will launch it using the following command and parameters:

```
./md5_cpu <md5 hash of target password> <target password length>
```

Moreover, you can run the GPU implementation using the following command and parameters:

```
./md5_gpu <md5 hash of target password> <threads per block>
```

6. Conclusions and Future Work

In this project, we implemented a serial approach and a parallel approach using CUDA. It compares the performance of both of these implementations and demonstrates the increasing power of GPUs. In addition, we explored how much difference it makes to keep a longer password or a password containing a bigger character set.

In future, I would also like to implement this attack using OpenMP framework in order to observe the performance gain by using multiple CPU threads and try to find the ceiling of that performance. This will show how the GPU breaks the performance barrier of the CPU. This project only explores the bruteforce method of attack in this project, I would like to explore how the performance differs in case of a dictionary attack or when the attacker has a salt (hint) for the target password. Most of my code will be reused and repurposed for those kinds of attacks.

The course was extremely important for this project as it provided the fundamental knowledge and tools to successfully complete this project. The project utilized the CUDA framework to parallelize the MD5 Hash calculation using GPUs. Moreover, it required me to debug the CUDA kernel using cuda-gdb and cuda-memcheck and profile the performance of the implementation. Lastly, I used the knowledge gained from this course to reason about the difference in performance of various implementations throughout this project.

References

[1] https://en.wikipedia.org/wiki/MD5
[2] https://www.ietf.org/rfc/rfc1321.txt