**Determining the Suitability of Hashing Algorithms for Password Storage Using GPU Acceleration.**

**by**

**Ross Gray**

Project Aims

The aim of this project is to determine which hashing algorithms are suitable for password storage by comparing the speed that hashes can be calculated on a CPU against the speed they can be calculated on a GPU. All GPU programming will be done in CUDA C. Throughout this project each hashing algorithm being tested will have to be implemented in regular C or C++, then it will have to be implemented in CUDA C. The two will run for a set time and the number of hashes per second will be measured. The most suitable algorithms will be ones that the GPU does not give an advantage in calculating.

Project Objectives

The objectives of this project are to implement multiple hashing algorithms in both C++ and CUDA C. Each hashing algorithm will first be implemented in normal CPU code. It will then need thorough testing to ensure that it has been implemented correctly. GPUs boast many more cores than a CPU, however each core is slower on its own, compared with a single CPU core. Programs designed to run on a GPU must be written with this in mind, and each implementation must be refactored and optimized to run on a GPU. The code also must be written in CUDA C. CUDA C is like C and C++, but it introduces new functionality to interface with the GPU. A program will have to be written that runs each of the implementations for a set amount of time and measures the number of hashes completed per second. It will then have to populate a spreadsheet with the results of the test. Once the results are obtained the data will have to be analyzed to determine which hashing algorithms are suitable for storing passwords.

Background Research and Related Word

This section will outline the key background research I have undertaken to complete this project. It is important to understand the real-world security risks that are posed by storing passwords in an insecure manner, and the methods attackers will use to overcome any security measures taken. Therefore, we will explore the concept of storing passwords as hashes and why that is necessary and, also the kinds of attacks that bad actors will use to expose user data and sensitive information. It is also important that we understand how GPUs work and how writing programs designed to execute on a GPU differ from programs designed for CPUs. We will look at the architecture of CPUs and GPUs to understand the differences (and similarities) between them. We will also look at the CUDA C language and how it is used to write general purpose programs for Nvidia GPUs. The hashing algorithms will also have to be detailed here to understand how they work and how they can be optimized to run on a GPU.

Password Storage

If service providers want to store user’s password information, it must be stored as a hash rather than plain text. If the passwords are stored in plain text, anyone with access or who gains access to the server can extract any user’s login information and access their account. This is obviously a serious security risk as it means that a breach of the server storing the passwords would compromise every user’s account. A way to solve this problem is by storing passwords as a hash. A hash function takes data as an input and computes a hash from that data. This is usually a number represented in hexadecimal. The output of the hashing function will always be the same for the same input. Meaning that if you were to input the same password twice the hash would be the same. However, changing a single character in the password would produce a radically different output. This means that a hash for a user's password can be stored and when they enter their password again when logging in the hashes can be compared. If the hashes are identical then we know that the correct password was entered without ever having to know what the user’s password is. Storing passwords as hashes ensures that they are encrypted in a way that is not readable and cannot be reversed back to plain text. This is ideal for password storage as it means that the information is hidden and cannot be decrypted. If we were to use a regular encryption standard that used keys to encrypt passwords, anyone with access to the keys could decrypt those passwords, in turn compromising the sensitive user data. This is why hash functions are so beneficial for storing passwords.

Dictionary Attacks

A dictionary attack is when an attacker uses a dictionary of common passwords or passwords that have been discovered in previous attacks.

A vulnerability of many hash functions is that the hashes can be computed very quickly. Using GPU programming can make it possible to compute thousands of hashes per second. This means that if an attacker managed to steal hashed passwords, they would not be able to read them directly however, if they had a password dictionary, they would be able to compute the hashes of those passwords in the dictionary and compare them to the hashed passwords stolen from the server. If an attacker can compute hashes much quicker then they will be able to compromise more users' passwords in much less time. Therefore, meaning that more user’s information will be compromised. This is why hash functions need to be designed with this in mind to ensure that the algorithm is purposefully slow to make these kinds of attacks impossible. The ideal hashing algorithm is the one that is not computed any faster by a GPU and cannot be optimized for them.