Impact of CUDA Accelerated Computing On The Viability of Current Security Protocols

Krishna Raju

November 12th 2024

Abstract

The rapid advancement of GPU computing, particularly through CUDA acceleration, has transformed the landscape of cryptographic security, posing new challenges to traditional security protocols. This study investigates the impact of CUDA-accelerated brute-force techniques on password hashing algorithms, examining how accelerated hashing influences the viability of established security standards. By leveraging GPU parallelism, the project demonstrates a significant increase in hash computation speed, enabling the rapid testing of millions of password permutations. The project analyzes the implications for common security protocols, especially those reliant on hashing algorithms like MD5 and SHA-256, and discusses potential vulnerabilities that emerge under accelerated brute-force conditions. Although MD5 is known to be vulnerable, it is still commonly used in legacy systems across a variety of domains. This research highlights the need for robust cryptographic practices in light of growing computational power, advocating for enhanced algorithmic defenses and adaptive security measures that can withstand GPU-accelerated attacks.

Procedures

Script Development and Implementation were performed in CUDA C++ and Standard C++. The CUDA C++ version was optimized to harness the parallel processing capabilities of the GPU, specifically configuring it to utilize 256 threads for concurrent execution. This GPU-based approach relied on direct GPU memory allocation and management to handle the data-intensive nature of cryptographic algorithms, aiming to reduce processing times and enhance efficiency by exploiting the large number of simultaneous operations that the GPU can handle. Meanwhile, the C++ implementation served as a baseline, allowing for direct comparisons of execution speed and resource utilization between traditional CPU processing and GPU-accelerated computing.

The algorithms scripted varied in input data sets and string lengths: For comprehensive testing, The input parameters across both types of computation were varied to observe their effect on performance. The input data sets consisted of distinct character ranges, including numerical characters (0–9), uppercase alphabetic characters (A–Z), and lowercase alphabetic characters (a–z). Each configuration was tested with permutation lengths ranging from 4 to 7 characters, providing a diverse set of data conditions. This allowed observing performance impacts based not only on computational complexity but also on memory demands and handling efficiency as the input space expanded. The different configurations are summarized in table 1 below. Each permutation configuration underwent MD5 or SHA-256 hashing to mimic real-world cryptographic loads.

Table 1 Configurations of Algorithms

Algorithm	10-6	10-7	26-4	26-5	52-4
Character Set	0-9	0-9	A-Z	A-Z	a-z, A-Z
Size of Character Set	10	10	26	26	52
Length	6	7	4	5	4
Permutations	10^6	10^7	26^4	26^5	52^4
Number of Permutations	1000000	10000000	456976	11881376	7311616

All tests were conducted on a T4 GPU instance, which provided an optimal balance of compute and memory bandwidth. The T4 instance's architecture enabled efficient GPU memory utilization, making it suitable for high-throughput operations essential in cryptographic testing. For each script and input configuration, we measured total execution time, GPU memory usage, and hashing throughput. These metrics were then compared to assess the degree of acceleration achieved through CUDA and to evaluate the feasibility of using GPU-based solutions in cryptographic applications. The results were analyzed to identify patterns of efficiency gains across different character sets and string lengths, with a focus on the scalability and resilience of the CUDA-accelerated approach when applied to varying cryptographic workloads.

To ensure accuracy and reliability, potential confounding factors were minimized, including variations in CPU/GPU load, memory latency fluctuations, and other sources of potential errors. Each experiment was conducted multiple times, facilitating the observation of consistent performance trends and the identification of any anomalies. This repetition provided a more reliable dataset, allowing for an analysis that accurately reflects the computational efficiency and trends inherent to the tested configurations.

Data and Observations

The following data sets were gathered from the conducted experiments. Table 2 presents data collected from standard CPU-based computations, illustrating the performance metrics and trends observed when utilizing CPU resources exclusively. In contrast, Table 3 provides the corresponding data for computations accelerated by CUDA, showcasing the effects of parallel processing on a GPU.

Preliminary observations suggest that, on average, the algorithms executed on the CUDA platform exhibit a performance improvement of approximately 10 times when compared to those run on the standard CPU-based implementation. Additionally, the results indicate that the computational time for inputs of length 10^7 appears to be consistently 10 times greater than that for inputs of length 10^6. This observation supports the hypothesis of a linear relationship between the length of permutations and the corresponding computation times. Such a relationship is crucial as it will enable the prediction of computation times for larger input sets, providing valuable insights into the scalability, efficiency and viability of the algorithms as the permutation size increases.

Table 1 Raw Data of Normal Algorithms Performance

Normal (Computation time in seconds)										
Algorithm	10-7		10-6		26-4		26-5		52-4	
Trial	MD5	SHA256	MD5	SHA256	MD5	SHA256	MD5	SHA256	MD5	SHA256
1	2.030719	8.747745	0.217881	0.794411	0.110608	0.362045	2.450795	10.135272	1.500773	6.831545
2	2.079702	8.131271	0.211507	0.822802	0.112743	0.374193	2.426005	10.344523	1.510392	7.333354
3	2.029987	8.723923	0.201036	0.964187	0.093843	0.384352	2.451691	10.121747	1.508211	6.858592
4	2.075113	8.380196	0.205947	0.813184	0.101792	0.359020	2.438926	10.083395	1.497971	7.565162
5	2.029202	7.824038	0.220668	0.800620	0.092891	0.367088	2.448372	10.110850	1.604270	7.898963
6	2.022104	8.910541	0.206855	0.840403	0.101487	0.367706	2.460856	9.665218	1.515187	7.854459
7	2.044322	8.775747	0.201929	0.798095	0.092586	0.363805	2.446184	10.080237	1.494598	7.886374
8	2.033281	8.739930	0.204787	0.809747	0.098049	0.375240	2.458534	10.257182	1.499793	7.875987
9	2.025747	8.765068	0.205534	0.785152	0.094199	0.361771	2.473972	10.168247	1.518581	7.146385
10	2.097039	8.749754	0.201261	0.824376	0.096308	0.388784	2.445681	10.153502	1.498967	6.855123

Table 2 Raw Data of CUDA Algorithms Performance

CUDA (Computation time in seconds)										
Algorithm	10-7		10-6		26-4		26-5		52-4	
Trial	MD5	SHA256	MD5	SHA256	MD5	SHA256	MD5	SHA256	MD5	SHA256
1	0.546776	0.662812	0.057134	0.070008	0.025638	0.031813	0.591312	0.777581	0.381155	0.495041
2	0.540917	0.677472	0.054344	0.069798	0.025599	0.033066	0.581955	0.745372	0.401571	0.507168
3	0.552970	0.670836	0.056382	0.074083	0.025391	0.031996	0.588601	0.751276	0.389380	0.496115
4	0.549814	0.675704	0.056946	0.067638	0.025613	0.031909	0.596206	0.749899	0.386489	0.478326
5	0.547445	0.673051	0.055267	0.070587	0.025623	0.033821	0.579707	0.756512	0.384446	0.486333
6	0.553837	0.668185	0.055711	0.068186	0.025644	0.032650	0.595387	0.750701	0.393748	0.485792
7	0.536108	0.690527	0.054243	0.069100	0.025457	0.032550	0.583115	0.758283	0.398135	0.488143
8	0.532966	0.662328	0.053870	0.071332	0.025470	0.032804	0.587986	0.779627	0.386009	0.500406
9	0.527330	0.676477	0.054366	0.068041	0.026024	0.044554	0.590229	0.758468	0.393713	0.480588
10	0.548013	0.766835	0.054623	0.069919	0.026991	0.032177	0.592551	0.758583	0.397873	0.468204

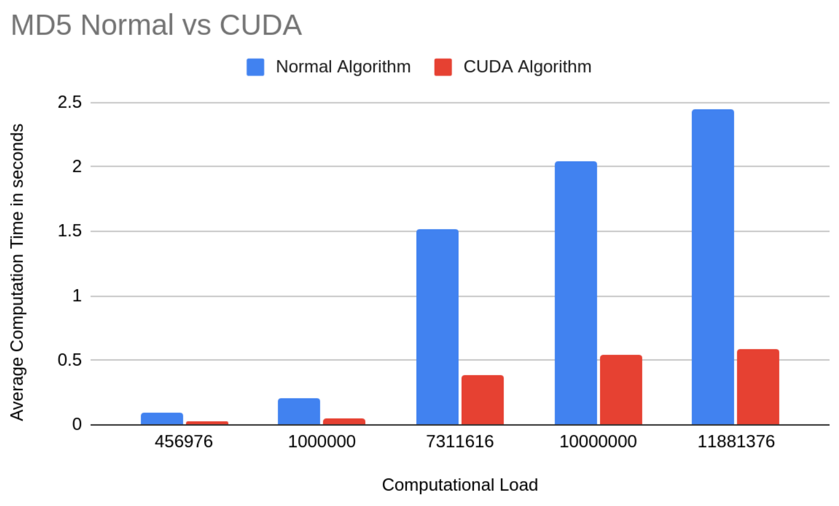
Results

The results demonstrate that, on average, the CUDA-accelerated algorithms perform significantly faster than their standard CPU-based counterparts. Specifically, as illustrated in Graphs 1 and 2, both the MD5 and SHA-256 hashing algorithms benefit substantially from CUDA-based computation compared to traditional computation.

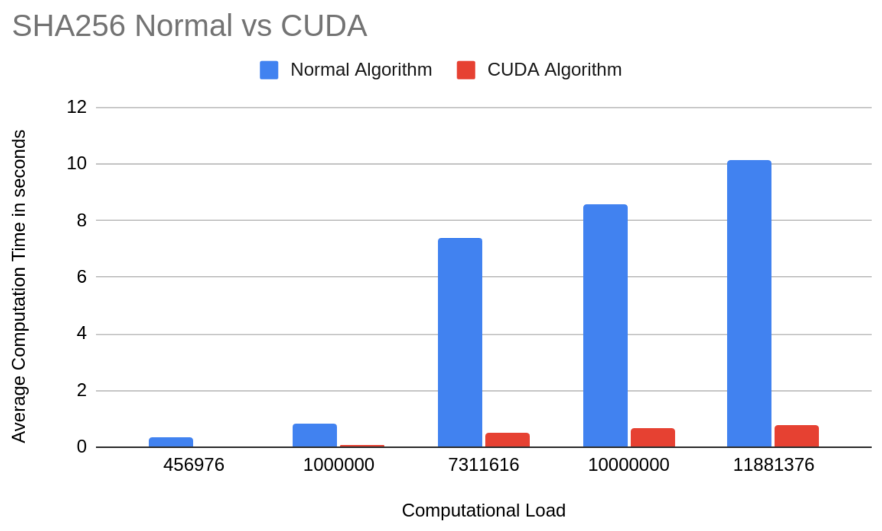
The computational efficiency gained from CUDA is particularly notable with SHA-256, which is inherently more computationally intensive than MD5. Consequently, the performance improvement achieved with CUDA for SHA-256 is more pronounced than that observed with conventional CPU processing.

Additionally, across various configurations, the data shown in Graph 3 indicates that the CUDA-based implementation achieves a speed increase by approximately a factor of 10. This substantial improvement underscores the value of GPU acceleration for complex hashing operations and highlights the efficiency of parallel processing for cryptographic computations.

Graph 1



Graph 2



Graph 3

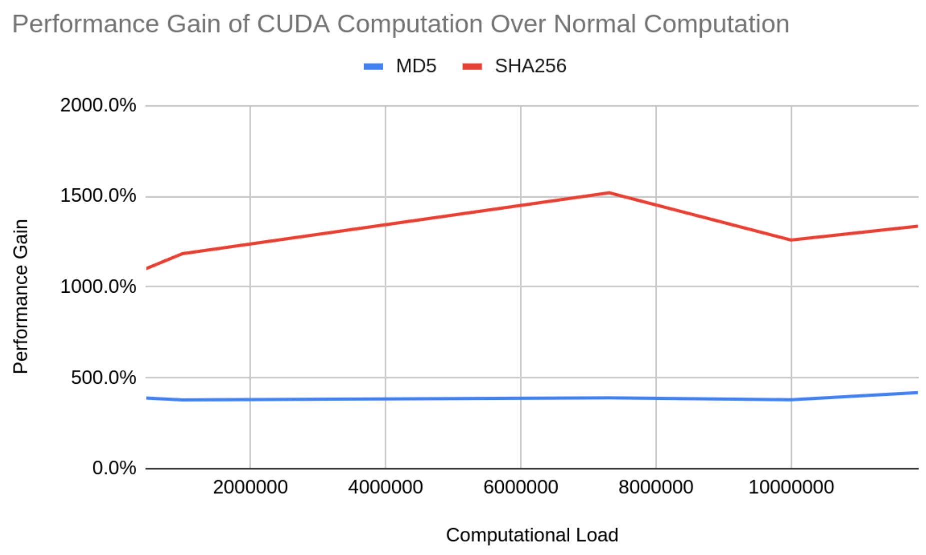


Table 3 Performance Gains Observed

Average Performance Gain Using CUDA	832.7%
Average Gain for MD5	388.4%
Average Gain for SHA256	1277.1%

Conclusion

This project underscores the transformative potential of CUDA-accelerated computation for cryptographic algorithms, particularly in high-performance contexts where speed is critical. By leveraging the parallel processing power of GPUs, the CUDA implementations of MD5 and SHA-256 demonstrated substantial performance gains compared to traditional CPU-based approaches. Notably, SHA-256, being more computationally intensive, saw a marked improvement in efficiency, highlighting the suitability of CUDA for handling complex cryptographic workloads.

In reality, most users choose stronger passwords involving a larger character set and longer length. This increases the computational load required. However, adversaries rarely perform a full bruteforce of every possible permutation of a password; instead they rely on password lists consisting of millions of entries of commonly used passwords. In this experiment the CUDA SHA256 algorithm was able to compute almost 12 million hashes in only 0.75 seconds. Therefore, it can be reasonably concluded that adversaries can bruteforce passwords and gain access to vulnerable accounts in a relatively short time. There are 2 ways to counter this threat, users should either create complex passwords that are not commonly used or systems should use hashing algorithms that are more computationally intensive and secure such as SHA-3.

In conclusion, as the demand for faster and more efficient data processing grows, CUDA-accelerated computation presents a promising pathway to meeting these needs, providing significant advantages in both performance and scalability for cryptographic operations.