Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU

By

Rana Raees Ahmed Khan

(Registration No: 00000326430)

Department of Cyber Security

Pakistan Navy Engineering College (PNEC)

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2023)

Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU

By

Rana Raees Ahmed Khan

(Registration No: 00000326430)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

Master of Science in
Cyber Security

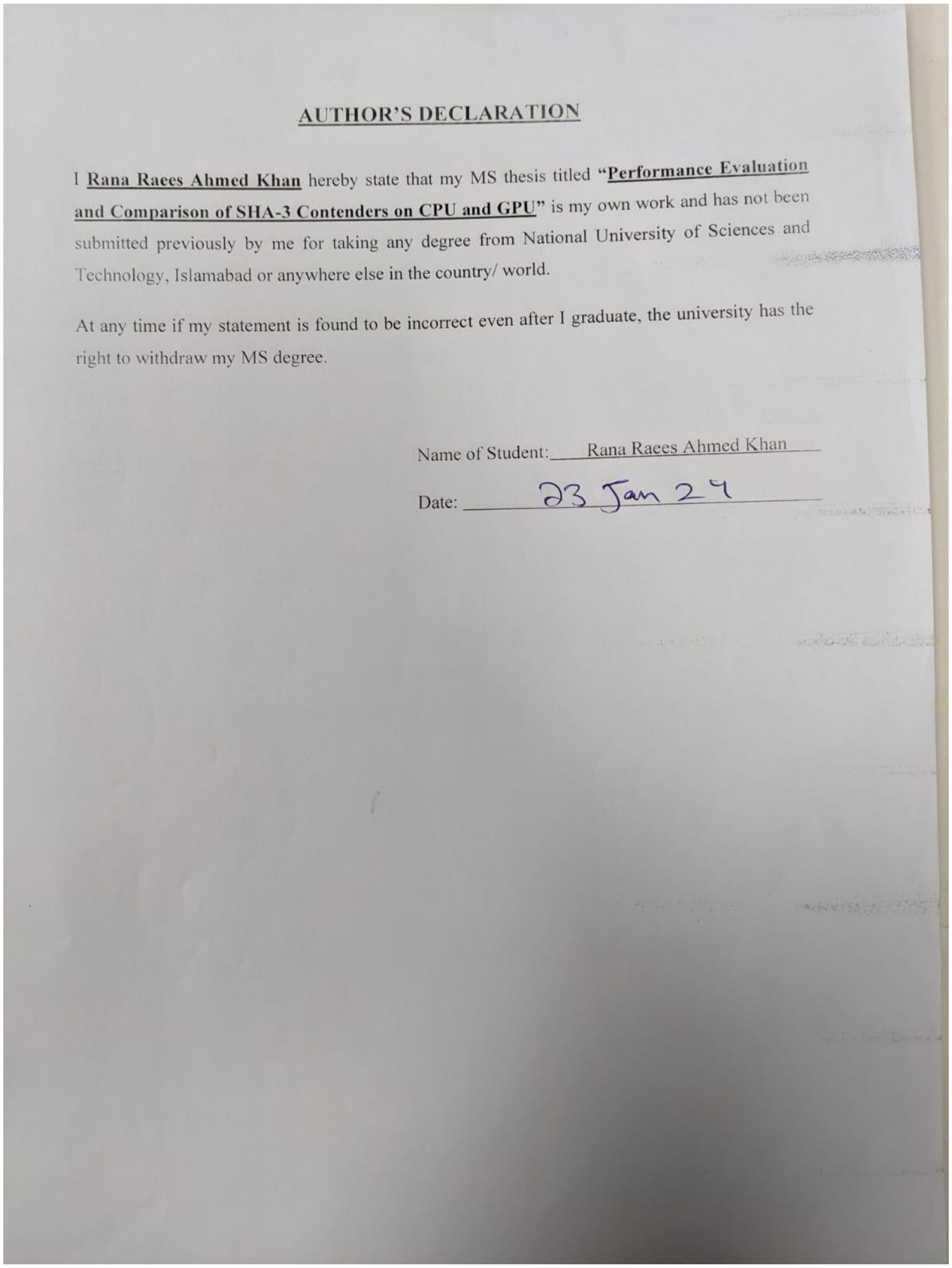
Supervisor: Cdre. Dr. Nazir Ahmed Malik,SI(M)

Pakistan Navy Engineering College (PNEC)

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2023)



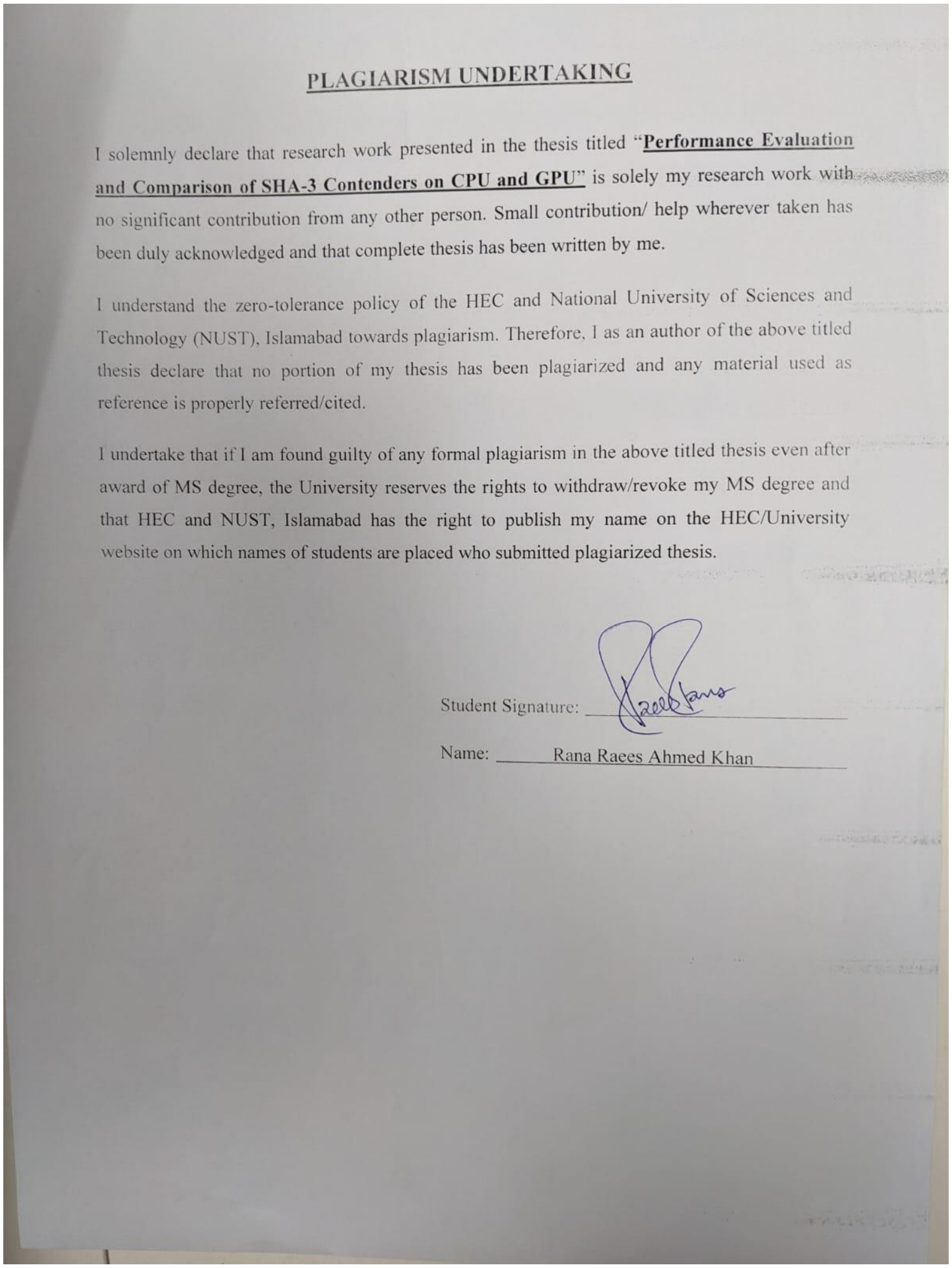
AUTHOR'S DECLARATION

I **Rana Raees Ahmed Khan** hereby state that my MS thesis titled **"Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU"** is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world.

At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Name of Student: Rana Raees Ahmed Khan

Date: 23 Jan 24



PLAGIARISM UNDERTAKING

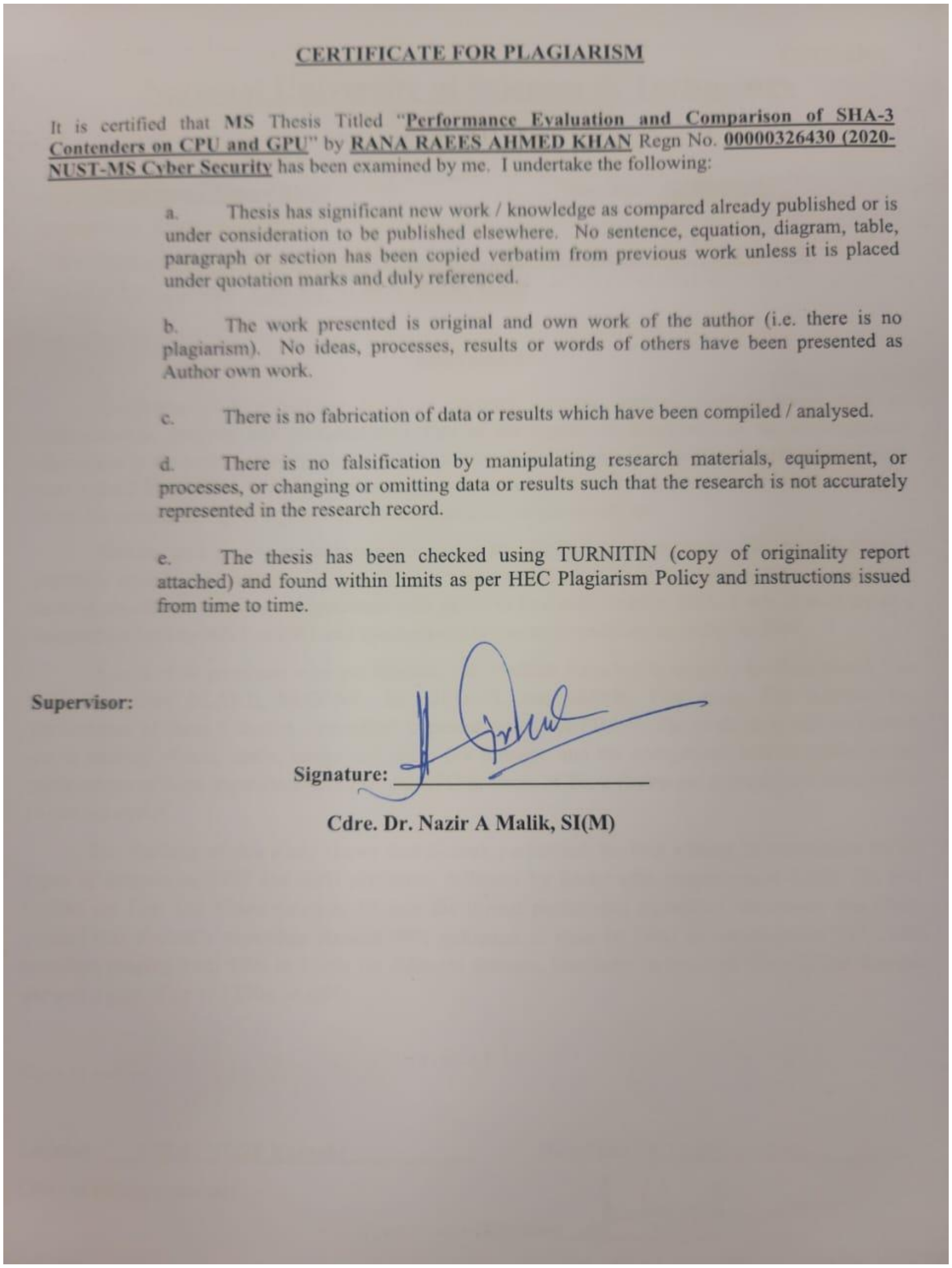
I solemnly declare that research work presented in the thesis titled "**Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU**" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

Student Signature: [illegible]

Name: Rana Raees Ahmed Khan



CERTIFICATE FOR PLAGIARISM

It is certified that **MS** Thesis Titled "**Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU**" by **RANA RAEES AHMED KHAN** Regn No. **00000326430 (2020-NUST-MS Cyber Security** has been examined by me. I undertake the following:

a. Thesis has significant new work / knowledge as compared already published or is under consideration to be published elsewhere. No sentence, equation, diagram, table, paragraph or section has been copied verbatim from previous work unless it is placed under quotation marks and duly referenced.

b. The work presented is original and own work of the author (i.e. there is no plagiarism). No ideas, processes, results or words of others have been presented as Author own work.

c. There is no fabrication of data or results which have been compiled / analysed.

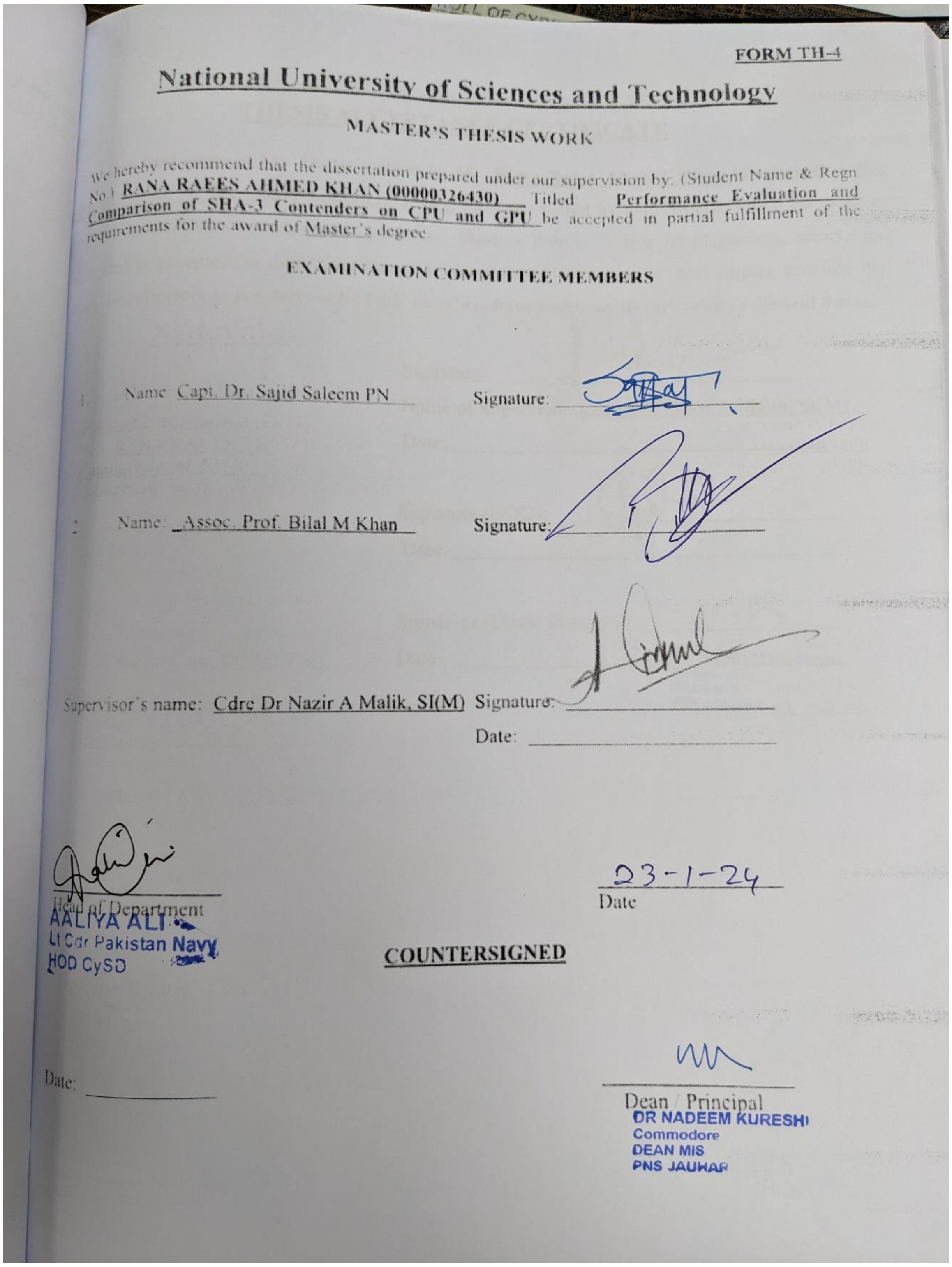
d. There is no falsification by manipulating research materials, equipment, or processes, or changing or omitting data or results such that the research is not accurately represented in the research record.

e. The thesis has been checked using TURNITIN (copy of originality report attached) and found within limits as per HEC Plagiarism Policy and instructions issued from time to time.

Supervisor:

Signature: [illegible]

Cdre. Dr. Nazir A Malik, SI(M)



FORM TH-4

National University of Sciences and Technology

MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: (Student Name & Regn No.) **RANA RAEES AHMED KHAN (00000326430)** Titled **Performance Evaluation and Comparison of SHA-3 Contenders on CPU and GPU** be accepted in partial fulfillment of the requirements for the award of Master's degree.

EXAMINATION COMMITTEE MEMBERS

1. Name: Capt. Dr. Sajid Saleem PN Signature: [illegible]

2. Name: Assoc. Prof. Bilal M Khan Signature: [illegible]

Supervisor's name: Cdre Dr Nazir A Malik, SI(M) Signature: [illegible]

Date: __________

[illegible]

Head of Department
AALIYA ALI
Lt Cdr Pakistan Navy
HOD CySD

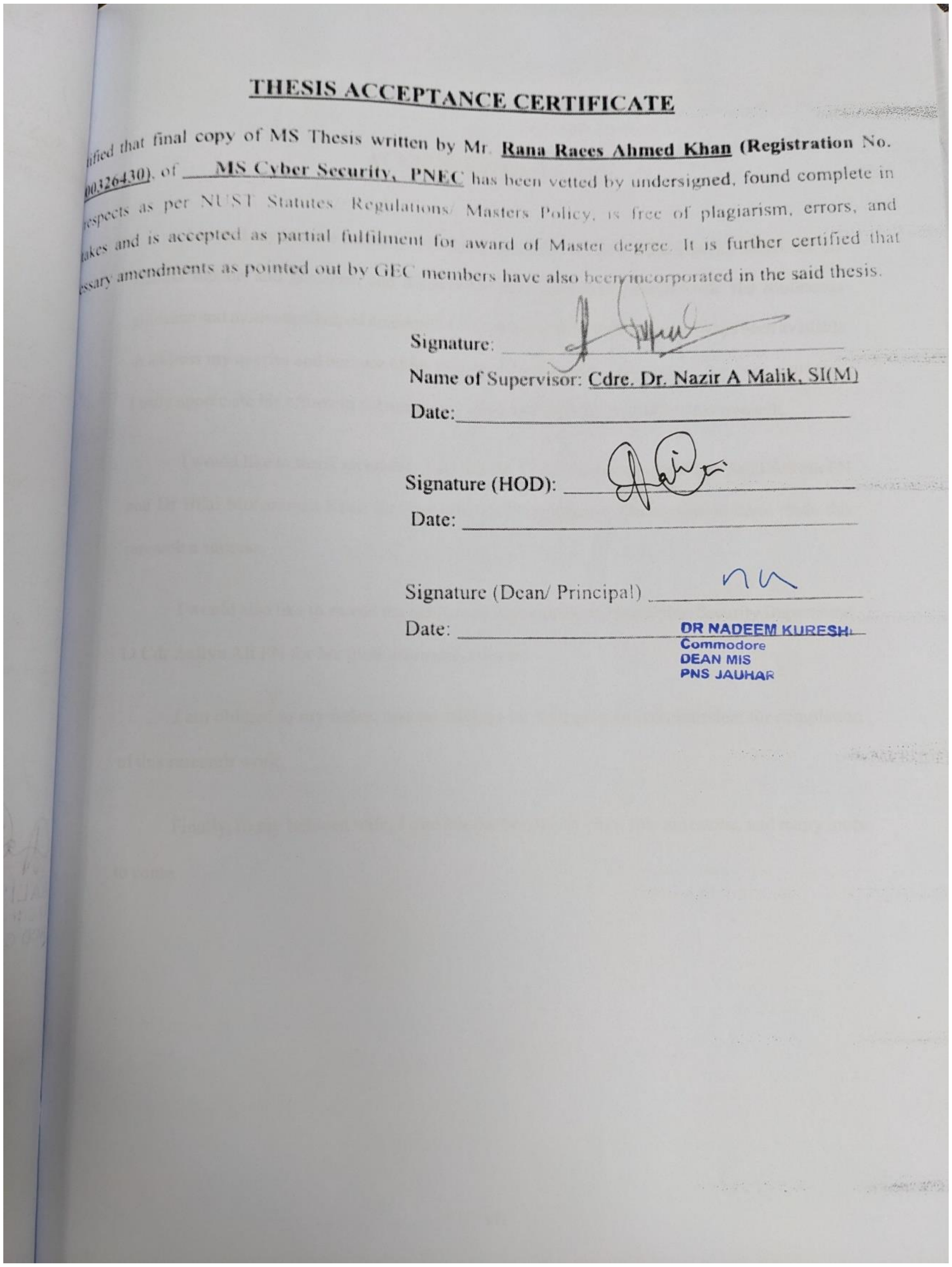
23-1-24
Date

COUNTERSIGNED

Date: __________

[illegible]

Dean / Principal
DR NADEEM KURESHI
Commodore
DEAN MIS
PNS JAUHAR



THESIS ACCEPTANCE CERTIFICATE

tified that final copy of MS Thesis written by Mr. **Rana Raees Ahmed Khan (Registration** No. **00326430)**, of **MS Cyber Security, PNEC** has been vetted by undersigned, found complete in espects as per NUST Statutes/ Regulations/ Masters Policy, is free of plagiarism, errors, and akes and is accepted as partial fulfilment for award of Master degree. It is further certified that essary amendments as pointed out by GEC members have also been incorporated in the said thesis.

Signature: [illegible]

Name of Supervisor: Cdre. Dr. Nazir A Malik, SI(M)

Date:

Signature (HOD): [illegible]

Date:

Signature (Dean/ Principal) [illegible]

Date:

DR NADEEM KURESHI
Commodore
DEAN MIS
PNS JAUHAR

ACKNOWLEDGEMENTS

I extend my humble thanks to my supervisor Cdre Dr Nazir Ahmed Malik SI(M), for his endless support and guidance, and his patience throughout the research work. His continuous guidance and motivation helped culminate the research work in time. He has always been available to address my queries and because of his dedicated participation, this research has been a success. I truly appreciate his efforts in enhancing my skills and capacity to conduct this research.

I would like to thank members of my Master Thesis committee, Capt. Dr. Sajid Saleem PN and Dr Bilal Muhammad Khan for their time and kind support. Their valuable input made this research a success.

I would also like to extend my thanks and appreciation to HoD Cyber Security Department, Lt Cdr Aaliya Ali PN for her motivation and support.

I am obliged to my father, and my siblings for their prayers and reminders for completion of this research work.

Finally, to my beloved wife, I owe her the best of my days, this milestone, and many more to come.

TABLE OF CONTENTS

LIST OF TABLES

Page No.

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

API	APPLICATION PROGRAMMING INTERFACE
CPU	CENTRAL PROCESSING UNIT
CUDA	COMPUTE UNIFIED DEVICE ARCHITECTURE
GPU	GRAPHICS PROCESSING UNIT
MD5	MESSAGE DIGEST METHOD 5
NIST	NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
SHA	SECURE HASHING ALGORITHM
$\|\|$	CONCATENATION OPERATOR
$\oplus$	XOR OPERATOR

ABSTRACT

Cryptography is a building block of security systems, used to address issues such as authentication, integrity and confidentiality. One of the significant disciplines of the cryptographic algorithms is the hashing family. Hashing is a technique which maps the arbitrary length of input data into a fixed length. It is widely used in modern information security systems such as authentication codes for messages, digital signatures, and authentication of passwords etc.

This research aims to evaluate one of the most popular hashing algorithms called SHA-3. SHA-3 primarily provides the integrity of data through hashing. SHA-3 was an upgrade to SHA-1 and MD5 hashing algorithms since these algorithms were prone to be cracked easily. SHA-3 was introduced in a competition held by NIST in 2007 and subsequently it was made publicly available in 2014.

A total of 64 proposals were put forward, out of which 5 made it through to the final round. The 5 finalists were BLAKE, KECCAK, JH, GROSTL and SKIEN. This study will evaluate the performance of these 5 finalist contenders' implementations of SHA-3. The evaluation will be carried out in hashing of text, audio, image and video file formats, and the comparison will be made on the performance of these algorithms on CPU and GPU in terms of Time (bytes per second) and Throughput (bytes per cycle).

The findings of this study shows that Keccak performed the best among its contenders on all types of datasets on CPU and GPU platforms, followed by Blake who outperformed Skein, JH, and Groestl on Text and Video datasets. JH was the lowest performing algorithm.Moreover, this study yielded that Keccak's algorithm showed 99% reduction in time on GPU as compared to CPU with speedups ranging from 420x to 1200x for different datasets. Similarly, in terms of Throughput, Keccak showed a gain of upto 1250x on GPU.

Keywords: SHA-3, NIST, CPU, GPU, Time, Throughput, Performance Evaluation

CHAPTER 1: INTRODUCTION

1.1 Background

Digital security is critically important in various fields including military, national defense, banks and business etc. Cryptography is a building block of security systems, used to address issues such as authentication, integrity and confidentiality. One of the significant disciplines of the cryptographic algorithms is the hashing family. Hashing is a technique which maps the arbitrary length of input data into a fixed length. It is widely used in modern information security systems such as authentication codes for messages, digital signatures, and authentication of passwords etc. The prior candidates of the Secure Hash Family (SHA) i.e., SHA–1 and SHA–2 were at high risk of getting cracked. In 2005, two researchers (Hongbo Yu and Wang) designed a collision attack which reduced the security level of the widely used SHA–1 and SHA–2 algorithms. Therefore, in November 2007, the National Institute of Standards and Technology (NIST) initiated a competition to develop a more robust and secure hashing algorithm, named as SHA-3. A total of 64 proposals were submitted, and five algorithms (Blake, Keccak, JH, Groestl and Skein) were selected for the final round. In October 2012, Keccak was announced as the winner of the competition due to its design, performance in both hardware and software and security provision. This study aims to evaluate the performance of final round candidates' algorithms on CPU and GPU. The evaluation will be carried out on Core i3-4005U 1.70GHz CPU and Nvidia 940 M GPU platforms. Criteria for evaluation will be CPU and GPU Time as bytes per second and CPU and GPU Throughput as bytes per cycle. These parameters will be evaluated using dataset comprising of text, image, video and audio files. Outcomes of this research will be useful for determining which algorithm out of the five finalists perform better on CPU or GPU environment and on what kind of dataset.

1.2 Research Objective

The main objective of this research is to evaluate the performance of the five finalist candidates (Keccak, BLAKE, Groestl, JH and Skein) of the SHA-3 competition in terms of speed on both CPU and GPU platforms. This research aims to compare the performance of these algorithms for audio, video, text and image files, and to determine which algorithm is best suited for a particular type of data. Additionally, this research aims to identify the best combination of parameters (CPU or GPU, type of dataset) for each algorithm to achieve the best performance.

1.3 Scope

This research focuses on the implementation and performance evaluation of the five finalist candidates of the SHA-3 competition on both CPU and GPU platforms. The study will be limited to the performance evaluation which will be measured in terms of speed as bytes per second and bytes per cycle. The research will be carried out using the Intel Core i3-4005U 1.70GHz CPU and Nvidia 940 M GPU platforms.

1.4 Methodology

Individual file from all datasets will be hashed separately on CPU and GPU environments. The hashing speed will be calculated as time and throughput on CPU and GPU. The results will be analyzed and compared to determine the best algorithm for a particular type of data and hardware setup.

1.5 Significance

The results of this research will provide a valuable contribution to the field of cryptography. This research will help in understanding how these algorithms perform on different architectures, this knowledge will aid in optimizing the algorithms for specific hardware, enhancing its efficiency and speeding up the hashing process. Further, GPUs are generally more efficient for parallel tasks, analyzing these five algorithms’ performance against CPU will help determining which hardware suits specific computational tasks better. In real-world application, outcomes of this research will guide organizations for better selection of hardware, for instance if a particular application heavily relies on fast hashing operations, this research will help in choosing the most suitable hardware and algorithm for optimal performance.

CHAPTER 2: LITERATURE REVIEW

In the past recent years, a number of attacks against many families of hash functions like SHA-1, SHA-2, MD etc. have been executed. Development of a more robust, efficient and attack-resistant hashing algorithm became critical.

To address the vulnerabilities in existing SHA algorithms, NIST formally announced the public competition of SHA-3 in November 2007 for submission of cryptographic Hash Algorithms and received 64 submissions. The acceptance further narrowed down to five candidates in the final round. These five finalists were Blake, Groestl, JH, Keccak and Skein. Subsequently, in 2012, Keccak was announced winner of the competition, and became the standard SHA-3 Algorithm.

2.1 Related Work

Since inception of SHA-3 algorithm, work has been done in testing the five finalist algorithms on the basis of performance of Hardware and Software implementation on different platforms to seek what algorithm works best on what platform. Some work has also been done on fine tuning the SHA-3 finalist algorithms to work more efficiently on certain systems. Prominent work in the said field is discussed in ensuing paragraphs.

Hassan et al. [1] describes an optimized implementation of Keccak algorithm using the CUDA programming model on NVIDIA GPUs. The research aimed to improve algorithm's performance by taking advantage of parallel processing capabilities of GPUs. The algorithm was implemented using CUDA libraries on NVIDIA GPU, and various parameters such as thread block size, and number of threads per block were optimized to achieve best performance. Authors further compared the performance of their GPU-based implementation with CPU-based implementation. They used various data sizes and measured the execution times of the algorithm both on CPU and GPU platforms. Results showed that GPU based implementation achieved significant speedup compared to that of CPU, with speed enhancement ranging from 3.3x to 17.8x for different data sizes.

Similar research was conducted by Wang et al. [2] based on similar parameters as [1]. Authors parallelized the Keccak algorithm on NVIDIA GPU and optimized parameters including thread block size, number of threads per block, and number of rounds used in the algorithm to achieve the best performance. Their experimentation yielded speedups ranging from 3.9x to 9.9x for different data sizes as compared to CPU based implementation of Keccak. Authors further studied the effect of above-mentioned parameters including the use of shared memory and found that further improvement in the performance of algorithm on GPU can be achieved.

Sideris et al. [3] used Nios-II processor to implement high throughput Keccak hash function. Author analysed Keccak algorithm to identify parts of the algorithm which could benefit from acceleration, and implemented a custom hardware accelerator on the Nios-II processor, which is a soft-core processor that can be programmed to perform custom tasks. The hardware accelerator was specifically designed to perform the most computationally intensive parts of the Keccak algorithm, while the rest of the algorithm was executed on the Nios-II processor itself. Authors used various data sizes and measured the execution times of the algorithms on both the hardware-accelerated and software-only platforms. The results showed that the hardware-accelerated implementation achieved significant boost in speeds as compared to the software-only implementations, with speed enhancement ranging from 15.7x to 21.8x for different data sizes. Authors further studied the effects of various parameters on the performance of the hardware accelerator, such as the clock frequency and number of parallel execution units used in the accelerator. They found that increasing the clock frequency from 50 MHz to 100 MHz improved the performance of the hardware accelerator by 1.5x for small input data sizes, and up to 1.9x for large input data sizes. However, increasing the clock frequency beyond 100 MHz did not lead to any significant improvement in performance.

Similarly, the results showed that increasing the number of execution units used in the hardware accelerator from 1 to 4 improved the performance of the hardware accelerator by 3.3x for small input data sizes, and up to 4.4x for large input data sizes. and the number of execution units could further improve the performance of the hardware accelerator. Authors also studied the effect of pipelining on the performance of the hardware accelerator. They found that pipelining the hardware accelerator improved the performance by up to 1.6x for small data sizes, and up to 1.8x for large data sizes.

Kuznetsov et al. [4] aimed to evaluate and compare the performance of several cryptographic hash functions that are commonly used in blockchain technology. The authors implemented the hash functions using the C programming language and conducted experiments to measure performance in terms of execution time, memory usage, and throughput. The hash functions that were evaluated included SHA-1, SHA-256, SHA-512, Keccak, and Blake2b-256. The authors used the OpenSSL library for implementation of SHA-1, SHA-256, SHA-512, while the Keccak and Blake2b-256 were implemented using optimized code available on GitHub. The experiments were conducted on an Intel Core i7-7500U CPU running at 2.7 GHz with 8GB of RAM. The authors evaluated the performance of the hash functions using different input sizes ranging from 100 bytes to 10 MB.

Experiments showed that the performance of the hash functions varied depending on the input size and the specific hash functions used. Overall, the authors found that Keccak and Blake2b-256 had the best performance in terms of execution time, memory usage and throughput for all input sizes tested. Specifically, Keccak had the best performance for small input sizes (less than 10 KB), while Blake2b-256 had the best performance for large input sizes (greater than 1 MB). Authors concluded that Keccak and Blake2b-256 are suitable hash functions for use in blockchain technology due to their good performance characteristics.

Jararweh et al. [5] evaluated and compared the hardware performance of the five SHA-3 candidate algorithms on a Xilinx Virtex 5 FPGA and evaluated their performance in terms of throughput, latency, and resource utilization. The authors used Verilog HDL to implement the SHA-3 candidate algorithms on the FPGA platform. They conducted experiments to measure the throughput and latency of each algorithm using various input sizes ranging from 8 bytes to 64 KB. The authors also evaluated the resource utilization of each algorithm in terms of the number of slices and flip-flops used on the FPGA. The results of the experiments showed that the performance of the SHA-3 candidate algorithms varied depending on the specific algorithms and the input size. Overall, Keccak had the highest throughput for all input sizes tested, while Skein had the lowest latency for all input sizes tested. In terms of resource utilization, Blake and Groestl had the lowest number of slices and flip-flops used on the FPGA, while Keccak had the highest number of slices and flip-flops used.

Hanser et al. [6] compared the performance of the five SHA-3 candidate algorithms in the Java programming language. The authors implemented each algorithm in Java and conducted experiments to measure their performance in terms of processing time and memory usage. The authors used the standard Java Development Kit (JDK) to implement the SHA-3 candidate algorithms. They then measured the processing time and memory usage of each algorithm using various input sizes ranging from 1 byte to 1 GB. The experiments were conducted on a computer with an Intel Core i7-2600K CPU and 16 GB of RAM. The experiments showed that the performance of the SHA-3 candidate algorithms varied depending on the specific algorithm and the input size. Overall, Keccak performed the best with fastest processing time for all input sizes tested, while Blake had the lowest memory usage for all input sizes tested. Overall, Keccak was found to be the most efficient algorithm, followed by Blake, Skein, Groestl, and JH.

Sobti et al. [7] evaluated the performance of three SHA-3 candidate algorithms (Groestl, JH, and Blake) on ARM Cortex – M3 processor. The authors implemented each algorithm on the ARM Cortex – M3 processor using the CodeSourcery toolchain and evaluated their performance in terms of execution time and code size. The experiments were conducted using a Keil MCBSTM32 evaluation board with and ARM Cortex – M3 processor clocked at 72 MHz. the authors used the CodeSourcery toolchain to compile the code for each algorithm and measured the execution time and code size for each algorithm. The results of experiments showed that the performance of the three SHA-3 candidate algorithms varied depending on the specific algorithm and input size. The Groestl algorithm had the fastest execution time for small input sizes, while the BLAKE algorithm had the fastest execution time for larger input sizes, the JH algorithm had the highest code size among the three algorithms.

Schmidt et. Al [8] proposed an efficient hardware accelerator for the SHA-3 hash function. The proposed accelerator is designed to be parameterized, allowing for flexibility in terms of the hash output size and the number of rounds used in the SHA-3 computation. The methodology used in the paper involved implementing the proposed accelerator on a field programmable gate array (FPGA) platform and evaluating its performance in terms of throughput and area efficiency. The authors also compared the performance of their accelerator with existing implementations of SHA-3, such as software implementations and other hardware accelerators. The results showed that the proposed accelerator outperformed other SHA-3 implementations in terms of throughput and area efficiency. For example, the proposed accelerator achieved a throughput of 9.9 Gbps for a 256-bit output hash, which is significantly higher than existing implementations. Further, the authors also performed a power analysis of the proposed accelerator and found that it consumes less power compared to other implementations.

This is an important factor for applications where power consumption is a critical concern, such as in mobile and embedded devices. The power analysis showed that the proposed accelerator consumed less power compared to other SHA-3 implementations. Results showed that their accelerator consumed 2.2 mW of power for a 256-bit hash output, which is significantly less than other SHA-3 implementations such as software implementations and other hardware accelerators. It is also noted that the power consumption of the accelerator depends on the size of the hash output and the number of rounds used in the computation. For example, increasing the number of rounds used in the computation result in a higher power consumption due to the increased complexity of the computation.

Singh et. Al [9] presents a hardware implementation of the SHA-3 Blake finalist algorithm on the ARM Cortex A8 processor. The authors used the ARM RealView Development Suite for compiling and running their code on the processor. The hardware implementation of the SHA-3 Blake finalist algorithms was optimized using pipeline techniques to increase the throughput and reduce the latency of the computation. The authors used the NEON SIMD instructions of the ARM Cortex A8 processor to accelerate the computation. Performance was evaluated in terms of throughput and latency, and compared to other SHA-3 implementations on similar platforms. The results showed that the proposed implementation achieved a higher throughput and lower latency compared to other SHA-3 implementations. For example, throughput of 124.56 Mbps was reported for a 512-bit hash output using the proposed implementation, which is significantly higher than the throughput achieved by other implementations. The latency of the computation was also reduced to 0.8 cycles per byte.

Lowden et al [10] presented an analysis of Keccak tree hashing on GPU architectures. The methodology used in the research involves implementing the Keccak tree hashing algorithm on CUDA-enabled GPUs, specifically the NVIDIA Tesla K20c, NVIDIA K40c, NVIDIA GTX 680, and AMD RADEON HD 7970. The implementation is done using the OpenCL framework, and has rate and power consumption of each GPU are measured and compared. The author also compares the performance of the GPU with a CPU on the same algorithm. The implementation involves parallelizing the tree traversal process by dividing the input data into smaller chunks and computing the hash of each chunk in parallel. The parallelization is achieved by dividing the input data into a number of equal-sized chunks and computing the hash of each chunk independently. The hash values of the chunks are then combined to compute the final hash value of the entire input data.

The results of the study show that the AMD Radeon HD 7970 GPU has the highest hash rate, followed by the NVIDIA K20, and then the NVIDIA GTX 680/ however, the power consumption of the AMD Radeon HD 7970 is also the highest, while the NVIDIA GTX 680 has the lowest power consumption. The study also showed that increasing the tree depth in Keccak tree hashing result in a significant increase in hash rate, but also increases the power consumption of the GPUs.

Cayrel et al. [11] explores the implementation of the Keccak hash function family on GPUs, including NVIDIA GeForce 880 GTX, NVIDIA Tesla C1060, AND NVIDIA Tesla C2050. CUDA programming model is used to implement parallelization of the hash function on these architectures. The results showed that the GPU implementation outperformed the optimized CPU implementations, with a speedup of up to 32 times for large size dataset. By increasing the number of GPU threads, performance was improved up to a certain level, after which the performance plateaued.

Rao et al. [12] presented a high-speed implementation of SHA-3 on Virtex-5 and Virtex-6 FPGA. Methodology involves the use of an RTL level design and implementation of SHA-3 on FPGA using Xilinx ISE design suite. The implementation is done with a pipelined structure, which enhances the throughput and performance of the system. The results showed that the proposed implementation is significantly faster, achieving a maximum throughput of 20.6 Gbps on Virtex-6 and 14.8 Gbps on Virtex-5 FPGA. The study also shows that the proposed implementation is efficient in terms of FPGA resource utilization, with the utilization of 53% and 45% of the total resources on Virtex-5 and Virtex-6 respectively.

Lastly, Dat et al. [13] presents the implementation of the Keccak hash function on a CUDA enabled NVIDIA GTX 1080 GPU. The authors aimed to achieve high-performance hash computations that are efficient and scalable. The methodology involves using CUDA to parallelize the hash function and accelerate its computation time. The authors implemented four different versions of the Keccak hash functions with different block sizes (ranging from 512 bits to 1600 bits) to test the performance of the GPU implementation. They compared the results with the serial implementation of the same algorithm on a single CPU core. Results showed that implementation of Keccak on CUDA enabled GTX 1080 GPU resulted in significant enhancement in speed as compared to a CPU- based implementation. They reported that their GPU implementation achieved a throughput of 9.32 Gbps for 256-bit message length and 11.1 Gbps for 512-bit message length, while the CPU implementation achieved only 1.35 Gbps for 256-bit message length and 1.50 Gbps for 512-bit message length. The authors also compared the performance of their implementation with other existing GPU-based implementations of Keccak and found that their implementation outperformed all other implementations as they noted a 60% speed boost as compared to the fastest existing GPU-based implementation.

CHAPTER 3: METHODOLOGY

3.1 Introduction

This research aimed at studying algorithms of SHA-3 final round candidates in detail and implementing the said algorithms' code for hashing individual files from image, video, audio and text datasets and calculating performance metrics. Performance of each algorithm was evaluated by computing the time taken as bytes per second, and throughput as bytes per cycleby hashing each file on CPU and GPU respectively. The CPU used for this research is Intel Core i3-4005U clocked at 1.70 GHz, and GPU used is Nvidia 940 M.

3.2 Architectures of SHA-3 Finalists

Following are the architectures of five SHA-3 finalists (Blake, Jh, Groestl, Keccak, Skein).

3.2.1 Architecture of Blake

1. **Compression Function**: Blake uses a compression function to transform the input message into a fixed-length hash value. The compression function consists of several main phases:

   a. Initialization Phase: During this phase, the state of the compression function is initialized with a set of constants and the input message is divided into message blocks. The initialization phase can be represented by the following equation:

$$\begin{gathered}\mathbf{V[0,0], V[0,1], ..., V[0,15] = IV[0,0], IV[0,1], ..., IV[0,15]} \\ \mathbf{M[0], M[1], ..., M[n-1] = pad(M)}\end{gathered} \tag{1.1}$$

   Where V[i,j] represents the state of the compression function at position (i,j), IV[i,j] represents the i-th word of the j-th initialization vector, M[i] represents the i-th message block, and pad represents the message padding function.

b. Mixing Phase: During this phase, the state of the compression function is mixed using a set of mixing functions. The mixing phase can be represented by the following formula:

```
for i in range(0, r):
    V = Mix(V, i)
```
(1.2)

Where r represents the number of rounds and Mix represents a set of mixing functions.

c. Finalization Phase: During this phase, the final hash value is computed by XORing the state of the compression function with a set of finalization constants. The finalization phase can be represented by the following formula:

$$H = V[0,0] \oplus V[0,1] \oplus \ldots \oplus V[0,7] \oplus V[0,8] \oplus \ldots \oplus V[0,15] \quad (1.3)$$

Where H represents the final hash value.

2. **MixingFunctions**: Blake uses a set of mixing functions that operate on the state of the compression function during the mixing phase. The mixing functions consist of three main stages:

a. Substitution Layer: This stage applies a non-linear transformation to each word of the state.

b. Diffusion Layer: This stage applies a linear transformation to each word of the state.

c. Permutation Layer: This stage permutes the words of the state.

3. **FinalizationConstants**: Blake uses a set of finalization constants to compute the final hash value. The finalization constants consist of a set of pre-defined values that are XORed with the state of the compression function.

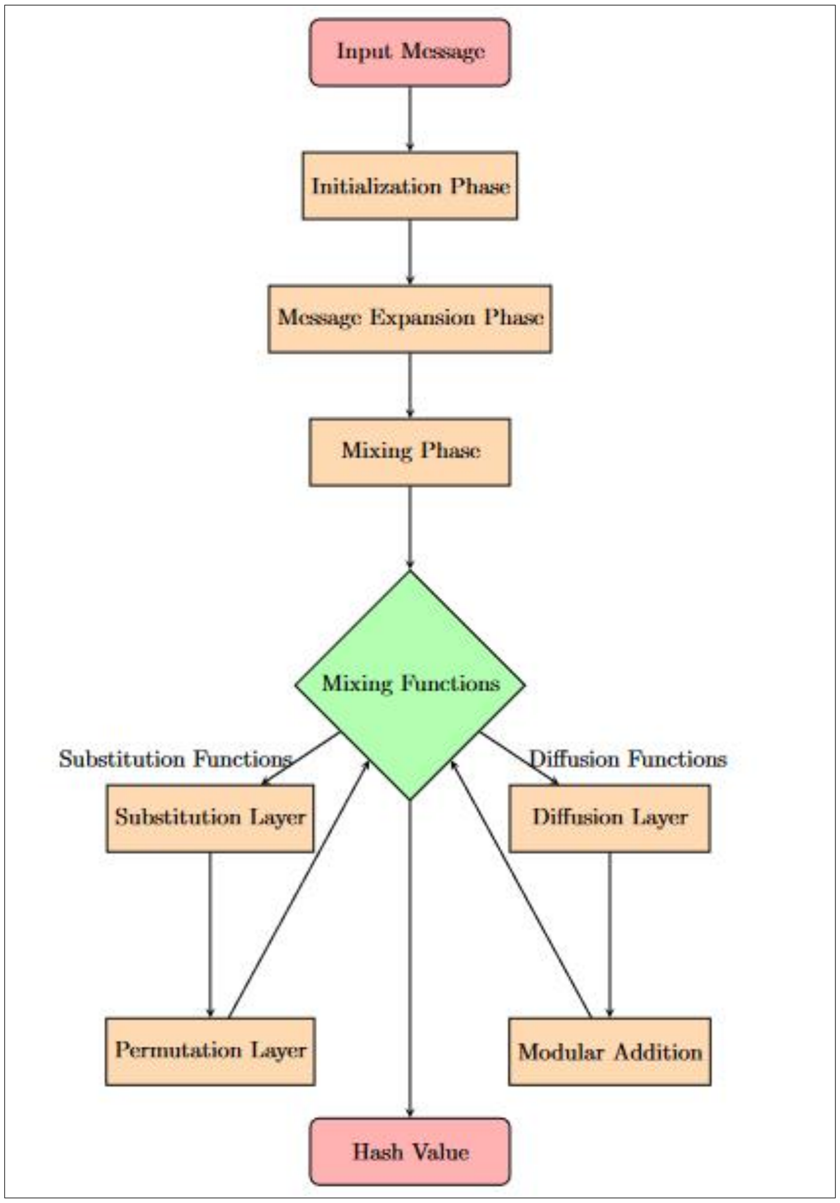


Figure 3.1: Architecture of Blake

3.2.2 Architecture of Groestl

The Groestl hash function uses a sponge construction, where the input is first padded to a multiple of the block size, and then processed through a sequence of permutations until the output hash value is obtained.The Groestl hash function consists of the following stages:

1. **Padding**: The input message is first padded to a multiple of the block size using the Merkle-Damgard padding scheme.

2. **Initialization**: The Groestl hash function uses a set of initial constants to initialize the internal state of the hash function. The initial constants consist of a set of pre-defined values that are XORed with the state of the compression function, which can be represented by following equation:

$$\mathbf{V[i,j] = PI[i,j]\ XOR\ C[i,j]} \tag{1.4}$$

   Where PI[i,j] represents the initial state of the hash function, and C[i,j] represents the initialization constants.

3. **Substitution**: The Groestl hash function uses a substitution layer to mix the input message with the internal state of the hash function. The substitution layer consists of two components: the s-box and the bit permutation.

4. **Mixing**: The Groestl hash function uses a mixing layer to further mix the internal state of the hash function. The mixing layer consists of two components: the linear diffusion layer and the non-linear diffusion layer.

5. **Squeezing**: Once the input message has been processed through the permutation and mixing layers, the output hash value is obtained by squeezing the internal state of the hash function.

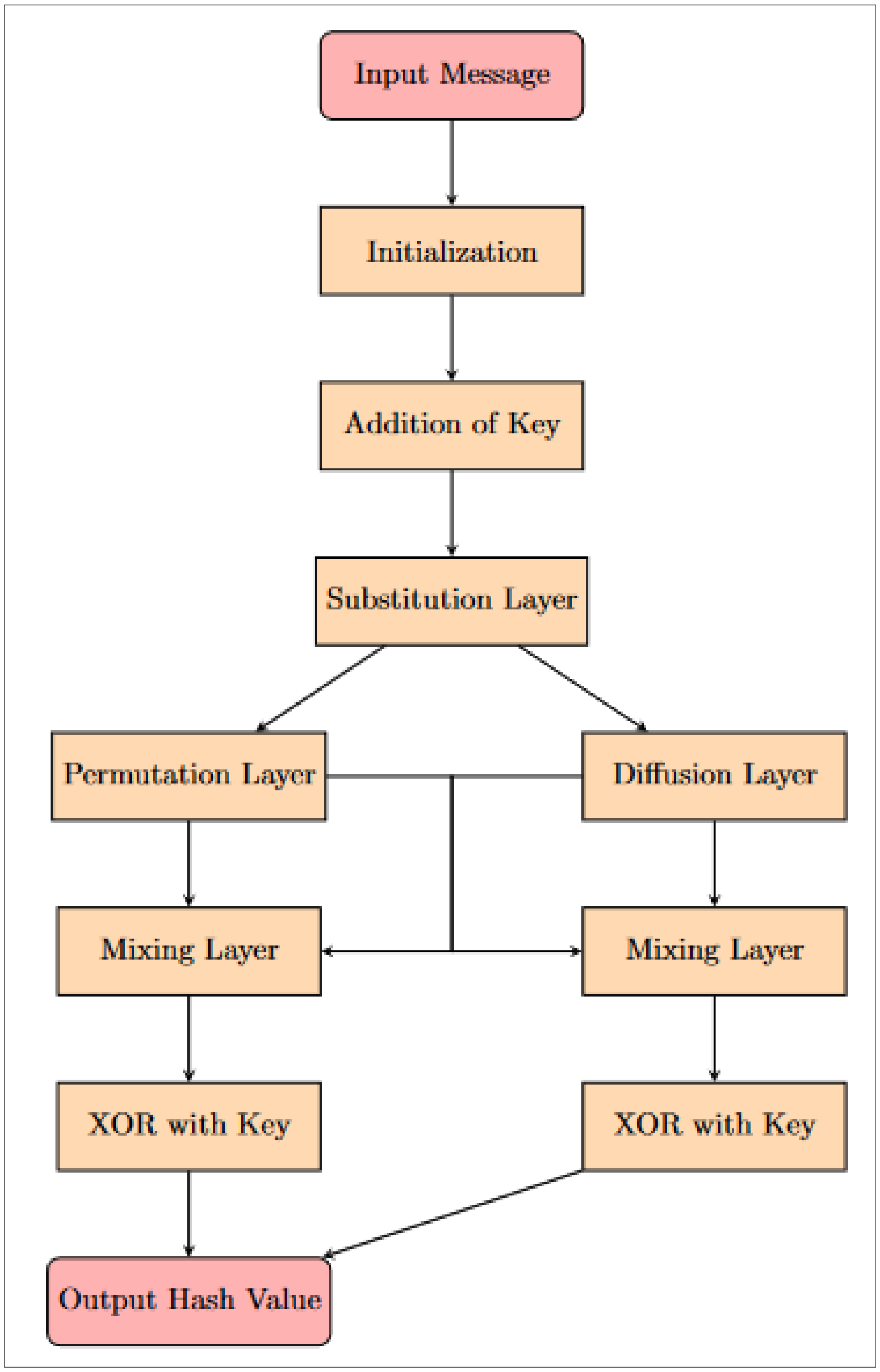


Figure 3.2: Architecture of Groestl

3.2.3 Architecture of JH

JH is a cryptographic hash function that operates on input blocks of up to 2^64 bits and produces hash values of variable length, up to a maximum of 512 bits. The architecture of JH consists of several main components:

1. **CompressionFunction**: JH uses a compression function to transform the input message into a fixed-length hash value. The compression function consists of three main phases:

   a. Initialization Phase: During this phase, the state of the compression function is initialized with a set of constants. The initialization phase can be represented by the following formula:

$$\mathbf{V[0,0], V[0,1], ..., V[0,15] = IV[0,0], IV[0,1], ..., IV[0,15]} \qquad (1.5)$$

   Where V[i,j] represents the state of the compression function at position (i,j), and IV[i,j] represents the i-th word of the j-th round constant.

   b. Message Expansion Phase: During this phase, the input message is expanded into a series of message blocks that are XORed with the state of the compression function, which can be represented by following equation:

$$\begin{gathered}\mathbf{for\ i\ in\ range(1, n+1):} \\ \mathbf{M[i-1] = M[i-1] \oplus V[i-1,0], V[i-1,1], ..., V[i-1,15]}\end{gathered} \qquad (1.6)$$

   Where M[i-1] represents the i-th message block and n represents the number of message blocks.

   c. Mixing Phase: During this phase, the state of the compression function is mixed using a set of mixing functions, which can be represented by following:

$$\begin{gathered}\mathbf{for\ i\ in\ range(1, r+1):} \\ \mathbf{V = Mix(V, i)}\end{gathered} \qquad (1.7)$$

   Where r represents the number of rounds and Mix represents a set of mixing functions.

2. **MixingFunctions**: JH uses a set of mixing functions that operate on the state of the compression function during the mixing phase. The mixing functions consist of four main stages:

   a. Substitution Layer: This stage applies a non-linear transformation to each word of the state, which can be represented by:

$$\mathbf{V[i,j] = Sub(V[i,j])} \tag{1.8}$$

   Where Sub represents the substitution function.

   b. Diffusion Layer: This stage applies a linear transformation to each word of the state.

   c. Permutation Layer: This stage permutes the words of the state.

   d. Modular Addition: This stage applies a modular addition operation to each word of the state.

3. **RoundConstants**: JH uses a set of round constants that are used to expand the input message and to XOR with the state of the compression function during the mixing phase.

In summary, the JH architecture uses a compression function to transform the input message into a fixed-length hash value. The compression function consists of an initialization phase, a message expansion phase, and a mixing phase that uses a set of mixing functions. The mixing functions consist of four main stages (substitution, diffusion, permutation, and modular addition) that operate on the state of the compression function.

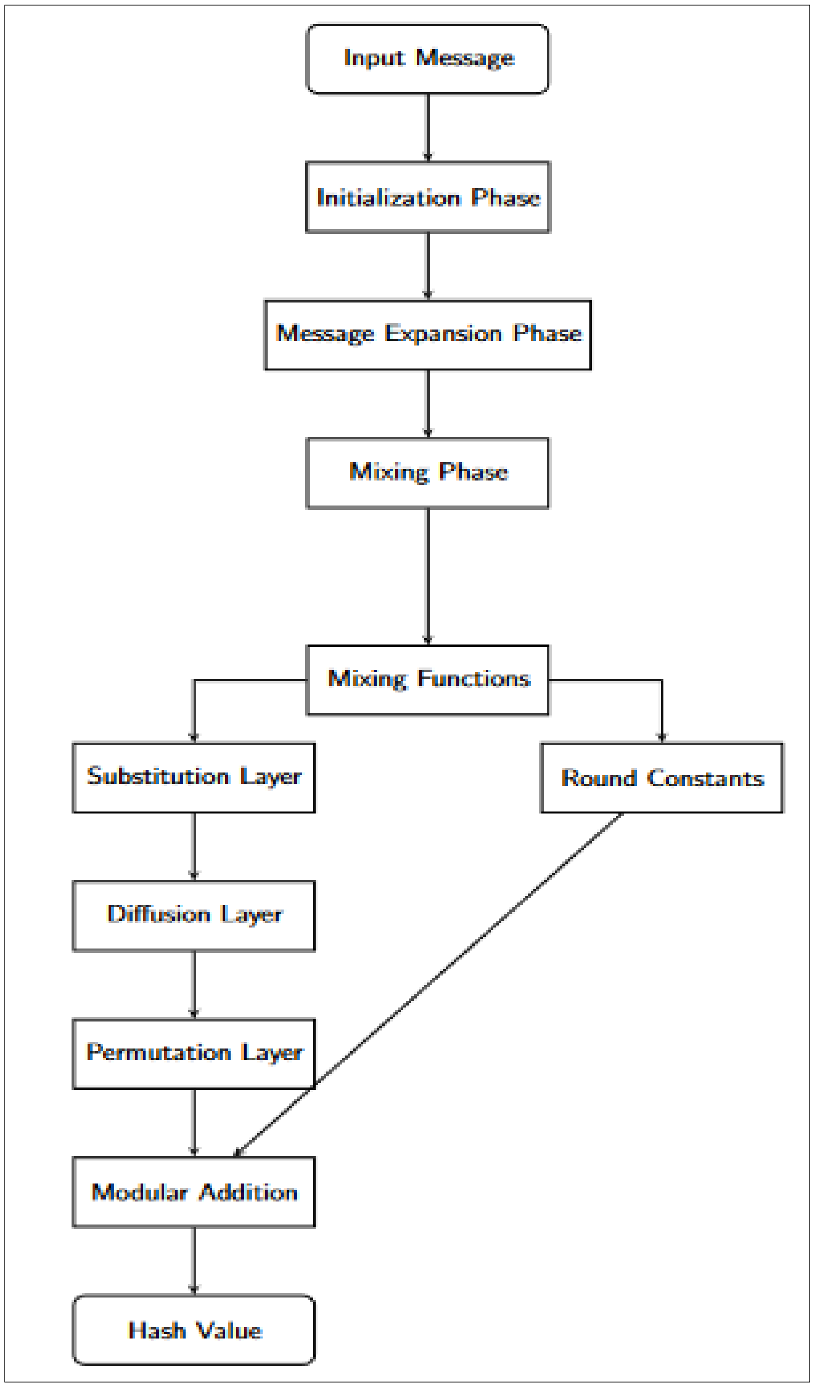


Figure 3.3: Architecture of JH

3.2.4 Architecture of Keccak

The Keccak architecture consists of several main components:

1. **SpongeConstruction**: Keccak uses a sponge construction to transform the input message into a fixed-length hash value. The sponge construction consists of two main phases:

   a. Absorb Phase: During this phase, the input message is divided into a series of blocks and XORed with the state of the sponge. The sponge then applies a permutation to the state to produce a new state, which is used for the next block of the input message, which can be represented by the following equation:

$$\mathbf{S[i,j] = S[i,j] \oplus M[x,y]} \tag{1.9}$$

   Where S[i,j] represents the state of the sponge at position (i,j), M[x,y] represents the message block at position (x,y), and ⊕ represents the bitwise XOR operation.

   b. Squeeze Phase: During this phase, the sponge repeatedly applies the permutation to the state and outputs a portion of the state as the hash value. The squeeze phase continues until the desired length of the hash value is reached, which can be represented by the following equation:

$$\mathbf{Z = Z \,||\, S[0,0] \,||\, S[1,0] \,||\, ... \,||\, S[n-1,0]} \tag{2.0}$$

   Where Z represents the output hash value and || represents concatenation.

2. **PermutationFunction**: Keccak uses a permutation function that is the core of the sponge construction. The permutation function operates on a 1600-bit state and consists of five main stages:

   a. Theta: This stage applies a linear transformation to each column of the state. The Theta stage can be represented by the following formula:

$$\begin{gathered}\mathbf{C[x] = S[x,0] \oplus S[x,1] \oplus ... \oplus S[x,4]} \\ \mathbf{D[x] = C[x\text{-}1] \oplus Rotate(C[x+1],1)} \\ \mathbf{S[x,y] = S[x,y] \oplus D[x]}\end{gathered} \tag{2.1}$$

Where C[x] represents the parity of the column x of the state, D[x] represents the difference between the parity of the neighbouring columns, and Rotate(C[x+1],1) represents the circular rotation of the parity of the column x+1 by one bit.

b. Rho: This stage rotates each lane of the state by a fixed amount.

S[i, j] = Rotate (S [i, j], R [i, j, t]) (2.2)

Where R[i,j,t] represents the rotation offset of the lane (i,j) at round t.

c. Pi: This stage permutes the lanes of the state. The Pi stage can be represented by the following formula:

```
for i in range(5):                                        (2.3)
    for j in range(5):
        S[i,j] = S[Pi(i,j),j]
```

Where Pi(i,j) represents the permutation index of the lane (i,j).

d. Chi: This stage applies a non-linear transformation to each row of the state. The Chi stage can be represented by the following formula:

```
for i in range(5):                                        (2.4)
    T = [S[i,j] for j in range(5)]
        for j in range(5):
        S[i,j] = T[j] ⊕ ((~T[(j+1)%5]) & T[(j+2)%5])
```

Where T represents the row of the state and & and ~ represent the bitwise AND and NOT operations, respectively.

e. Iota: This stage XORs a round constant with a specific lane of the state. The Iota stage can be represented by the following formula:

S[0,0] = S[0,0] ⊕ RC[t] (2.5)

Where RC[t] represents the round constant for round t.

3. **RoundConstants**: Keccak uses a set of round constants that are XORed with a specific lane of the state during the Iota stage. The round constants are derived from the binary expansion of the square root of a prime number. The round constants can be represented by the following formula:

$$\mathbf{RC[t] = r(t) * 2\text{^}(j(t)-1)} \tag{2.6}$$

Where r(t) represents the t-th element of the sequence {1, 2, 4, 8, 16, 32, 64, 128, 27, 54}, and j(t) represents the smallest integer such that 2^j(t) > r(t).

In summary, the Keccak architecture uses a sponge construction to transform the input message into a fixed-length hash value. The sponge construction consists of an absorb phase and a squeeze phase, and is based on a permutation function that operates on a 1600-bit state. The permutation function consists of five main stages (Theta, Rho, Pi, Chi, and Iota) that operate on the state and a set of round constants that are XORed with a specific lane of the state during the Iota stage.

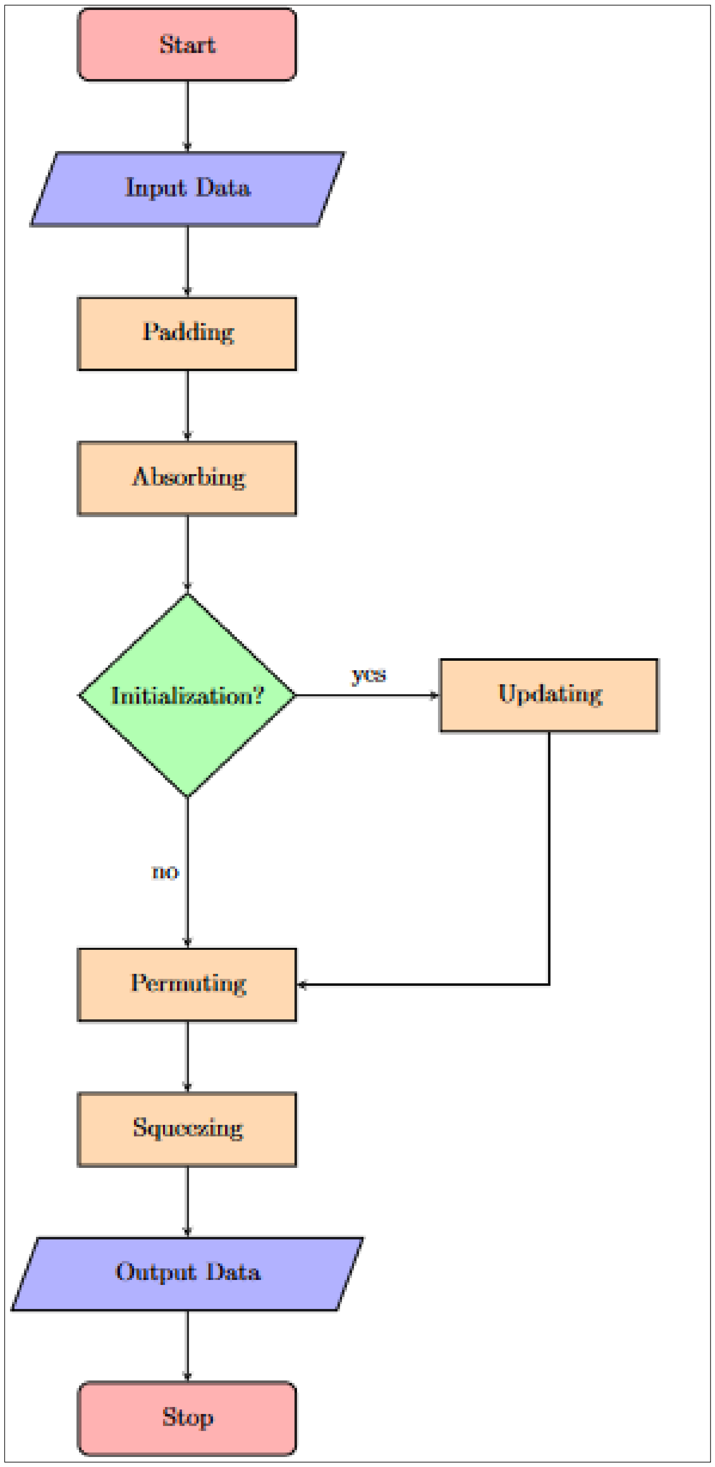


Figure 3.4: Architecture of Keccak

3.2.5 Architecture of Skein

The Skein hash function uses a unique approach called the Threefish block cipher, which is a tweakable block cipher that uses a key, a plaintext, and a tweak as input to produce a ciphertext.The Skein hash function consists of the following stages:

1. **Initialization**: The Skein hash function uses a set of initial constants to initialize the internal state of the hash function. The initial constants consist of a set of pre-defined values that are XORed with the state of the compression function. The initialization constants can be represented by the following formula:

$$\mathbf{V[i,j] = PI[i,j]\ XOR\ C[i,j]} \tag{2.7}$$

   Where PI[i,j] represents the initial state of the hash function, and C[i,j] represents the initialization constants.

2. **KeySchedule**: The key schedule is used to expand the secret key into a set of round keys that are used in the encryption process. The key schedule for Skein uses a unique approach called the Threefish block cipher, which generates round keys by repeatedly encrypting a fixed input using the secret key.

3. **Tweak**: The tweak is a unique input that is used to modify the behavior of the Threefish block cipher. The tweak is a 128-bit input that is XORed with the plaintext and round keys at each round of the encryption process.

4. **Encryption**: The Skein hash function uses the Threefish block cipher to encrypt the input message. The encryption process consists of multiple rounds, where the plaintext and round keys are mixed using a set of pre-defined mixing functions. The encryption process can be represented by the following formula:

$$\mathbf{C[i] = E(K[i], T, M[i]\ XOR\ C[i-1])} \tag{2.8}$$

   Where E(K, T, M) represents the encryption of the message M using the key K and the tweak T, and C[i-1] represents the ciphertext from the previous round.

5. **Finalization**: The finalization stage is used to generate the final hash value by squeezing the internal state of the hash function. The output hash value is computed by

applying a series of finalization functions to the internal state of the hash function.The finalization functions can be represented by the following formula:

$$\mathbf{H = F(V)} \tag{2.9}$$

Where F is a set of finalization functions, and V is the internal state of the hash function.

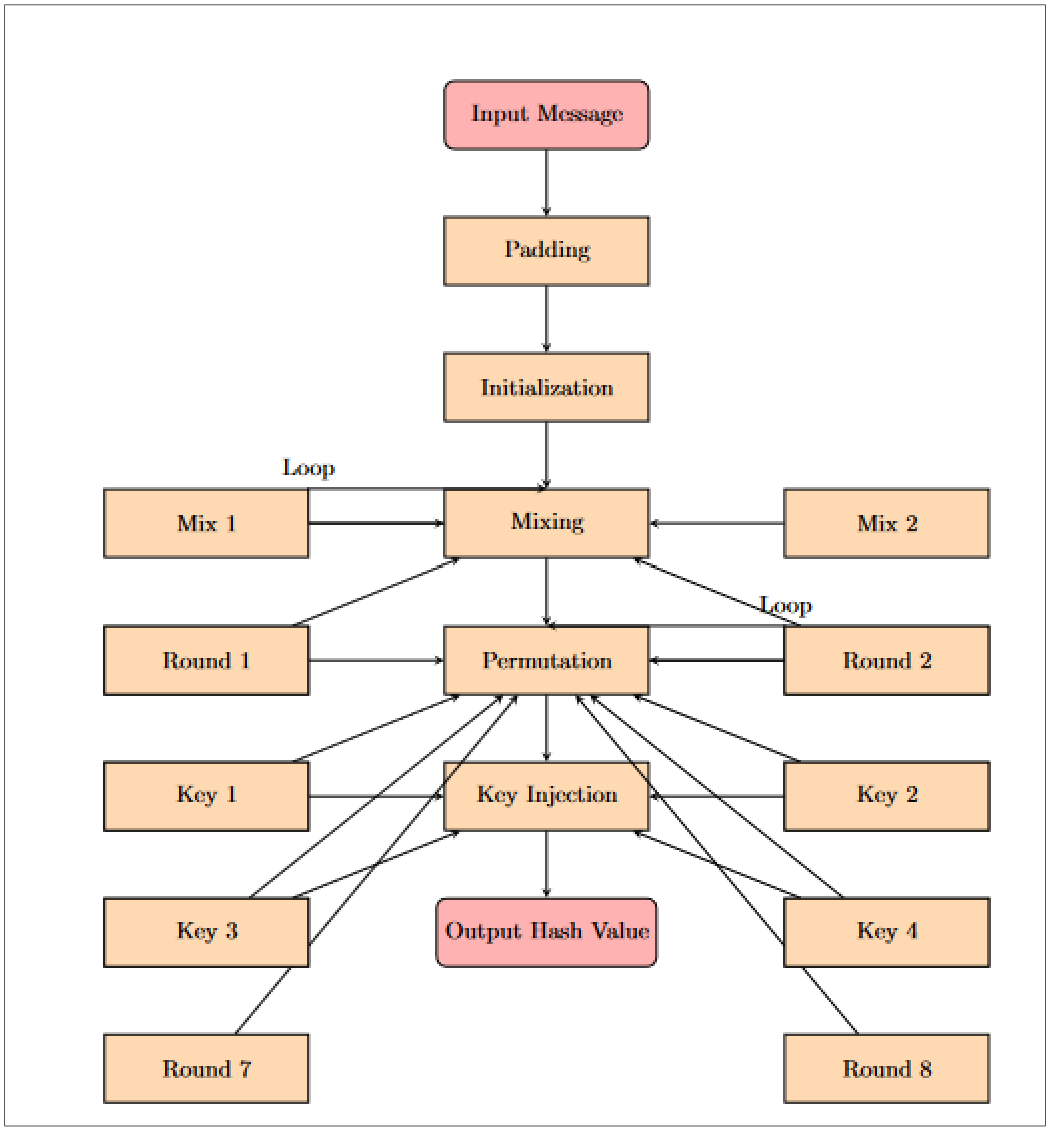


Figure 3.5: Architecture of Skein

3.3Software Specification

Algorithms mentioned in section 3.2 were coded on C# programming language. The IDE used for execution of the code is Visual Studio 2019.

3.4Hardware Specification

Intel Core i3-4005U 1.70 GHz CPU with 8GB DDR3 RAM and Windows 10 OS was used in this study. For evaluation on GPU, Nvidia 940 M GPU with CUDA Library V10.1 was used.

3.5Datasets

Following datasets were used in this study. The dataset included different sizes of Text, Audio, Video and Image files as mentioned below against each.

Table 3.1: Datasets

Format	Text (.txt)	Audio (.wav)	Video (.avi)	Image (.jpg)
Size (KB)	94	384	99	80
	319	457	141	107
	538	719	240	145
	963	1110	769	769
	977		960	960
	1163			

3.6 Methodology for CPU

Individual files from each dataset were uploaded into the system, and were hashed using Keccak, Blake, JH, Groestl and Skein hash functions. The system calculated the hash value and measured the performance based on CPU Throughput in Bytes per Cycle and CPU Time in Bytes per second. Figure 3.6 illustrates the working of the system for the CPU.

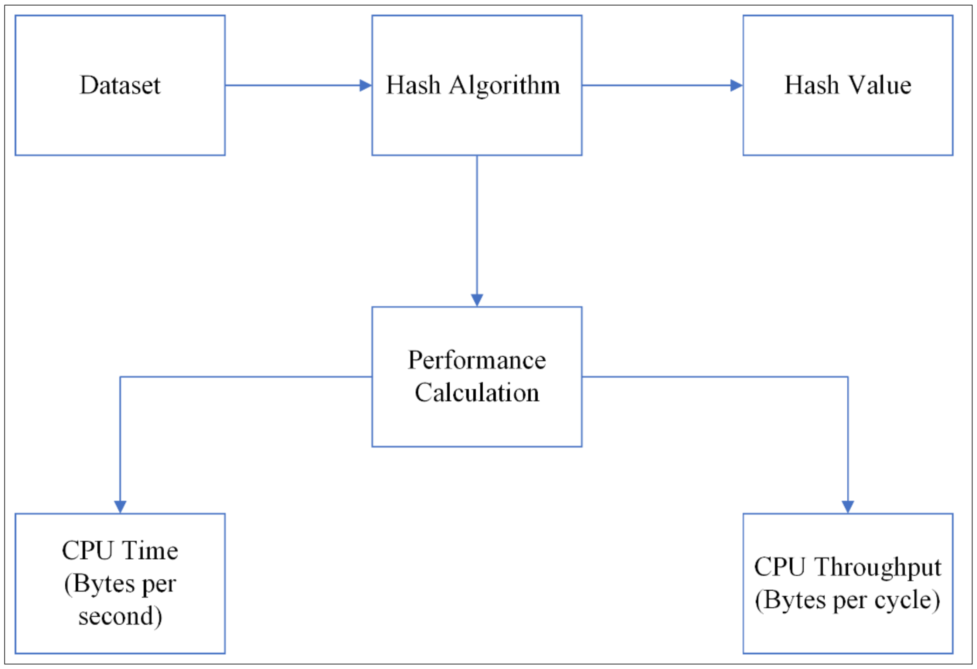


Figure 3.6: Methodology for CPU Evaluation

3.7 Methodology for GPU

Similar to the CPU implementation, files from all datasets were uploaded individually into the code, hashing process is divided into multiple threads using CUDA, which are processed by GPU simultaneously. Consequently, Hash value is obtained and performance in terms of GPU Time and Throughput is obtained. Figure 3.7 illustrates the working of the system for the GPU.

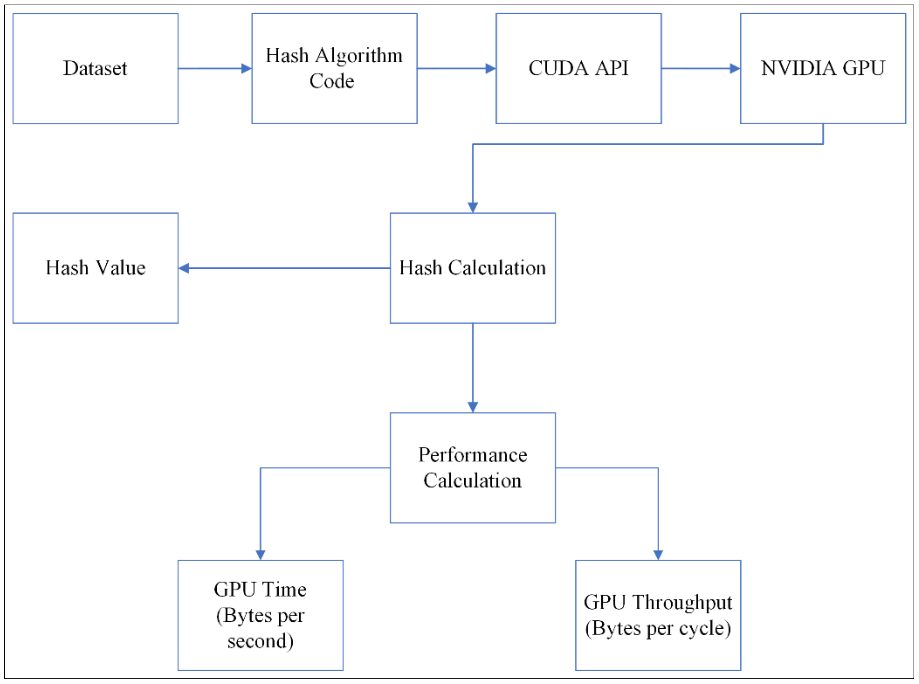


Figure 3.7: Methodology for GPU Evaluation

3.8 Summary

This chapter provides a detailed explanation of the methodology used in the experiment. Algorithms of each finalist were studied in detail and code was developed for individual functions of each algorithm mentioned in section 3.2. Code was then tested on an Intel Core i3-4005U 1.70GHz CPU and Nvidia 940 M GPU. Datasets comprising of variable sized Text, Audio, Video, and Image files were hashed using the code and performance was evaluated as Time (Bytes per second) and Throughput (Bytes per cycle).

CHAPTER 4: EXPERIMENTS AND RESULTS

This chapter covers the experimentation results of performance evaluation of SHA-3 final round candidates on image, video, audio and text datasets. The results of the experiment are presented in tabular and graphical forms in terms of CPU and GPU time (bytes per second) and throughput (bytes per cycle).

Based on the results, it was found that Keccak outperformed the other algorithms in terms of CPU Time and GPU Time for all data types. However, for video and text datasets, Blake performed better than Skein, JH, and Groestl.The performance of JH was the lowest among all candidates. Further, the experimentation also indicated thatKeccak's algorithm showed 95% to 99% reduction in time on GPU as compared to CPU with speedups ranging from 420x to 1200x for different datasets, with similar gains of up to 1250x in throughput.

These results have important implications for organizations, including military, national defence, banks, and businesses, as they can use this information to select the most suitable algorithm for their specific data format and hardware architecture. The findings also provide a foundation for future research to explore the performance of other SHA-3 candidates on different hardware architectures.

4.1 Experimental Results for Groestl

4.1.1 Text Dataset

Table 4.1.1 shows experimental results of Groestl for Text Dataset. The highest CPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 14.6 bytes per sec and Lowest CPU Time (Bytes per Second) is observed for the file size of 538 kb i.e., 13.06 bytes per sec. The highest CPU Throughput (Bytes per Cycle) is found for file size of 94 kb i.e., 101.12 bytes per cycle and Lowest CPU Throughput (Bytes per Cycle) is observed for the file size of 977 kb i.e., 100.5 bytes per cycle.

Similarly, highest GPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 0.00247 bytes per sec and Lowest GPU Time is found for the file size of 977 kb i.e., 0.0084 bytes per sec. Highest GPU Throughput (Bytes per Cycle) is found for the file size of 1.1 Mb i.e., 172187.8136 bytes per cycle and lowest GPU Throughput is observed on the file size of 94 kb i.e., 59747.3421 bytes per cycle.

Test results for Groestl on CPU and GPU for Text File Dataset is appended below.

Table 4.1.1: Test results for Groestl on text files

Text File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
94 Kb	14.6	101.12	0.0247	59747.3421
319 Kb	14.45	100.63	0.0105	138128.2034
538 Kb	13.06	100.52	0.0095	137977.2136
963 Kb	14.59	100.58	0.0085	172119.3628
977 Kb	14.3	100.5	0.0084	171982.4614
1.1 Mb	14.57	100.62	0.0085	172187.8136

Following are graphical representations of results of Groestl on Text dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Text data set.

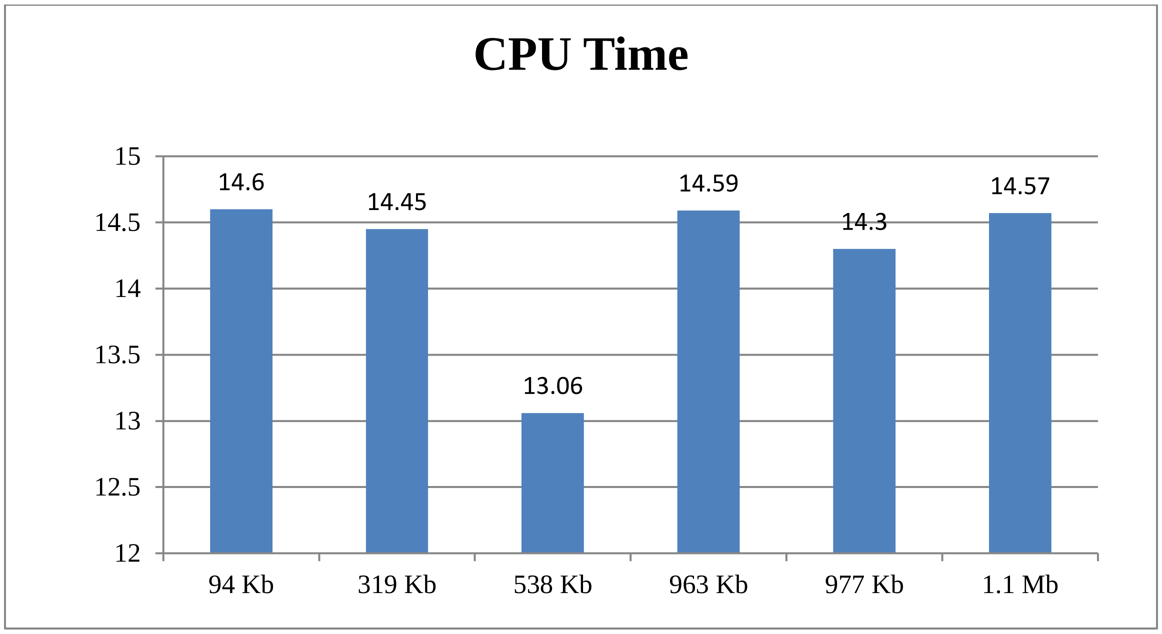


Figure 4.1.1.1: CPU Time for Groestl for Text Files

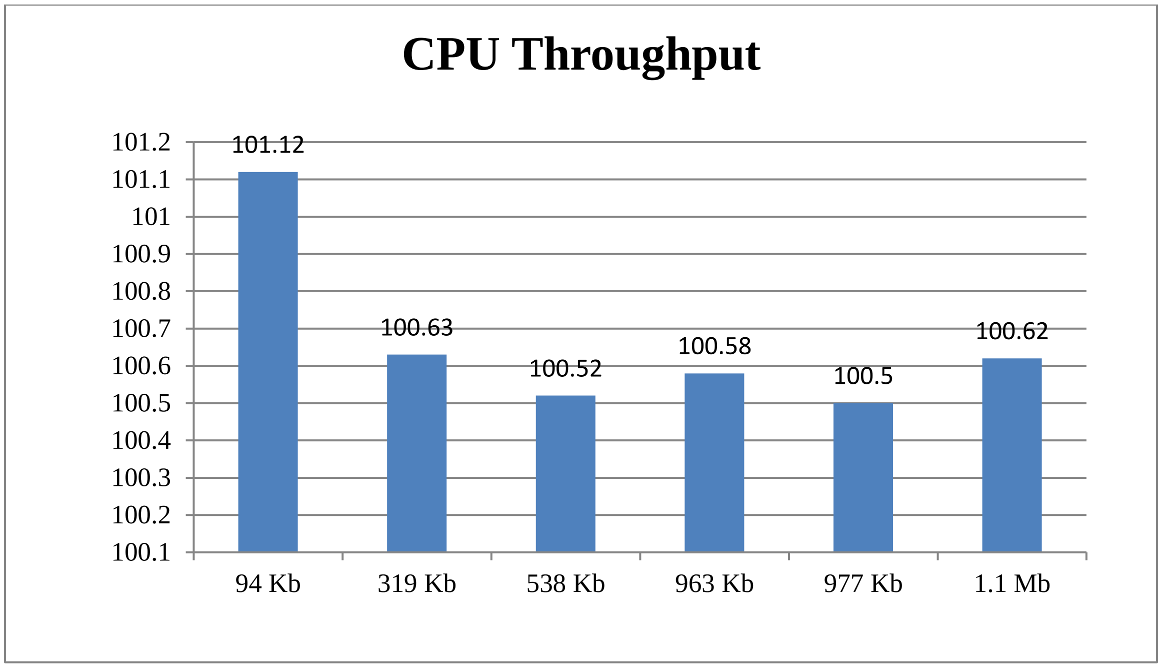


Figure 4.1.1.2: CPU Throughput for Groestl for Text Files

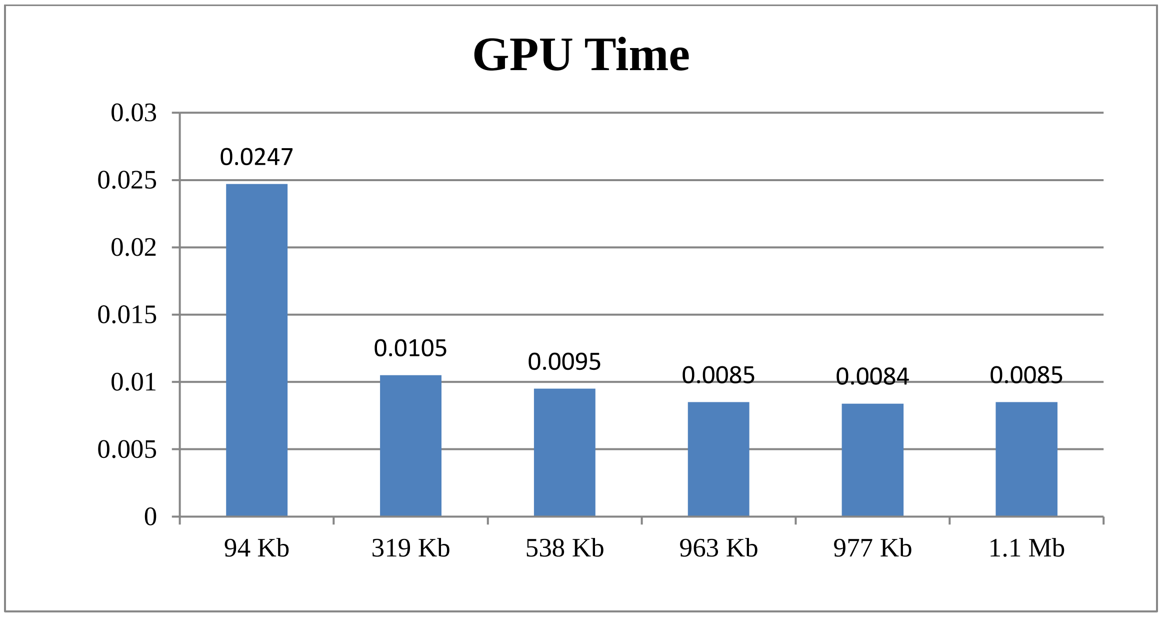


Figure 4.1.1.3: GPU Time for Groestl for Text Files

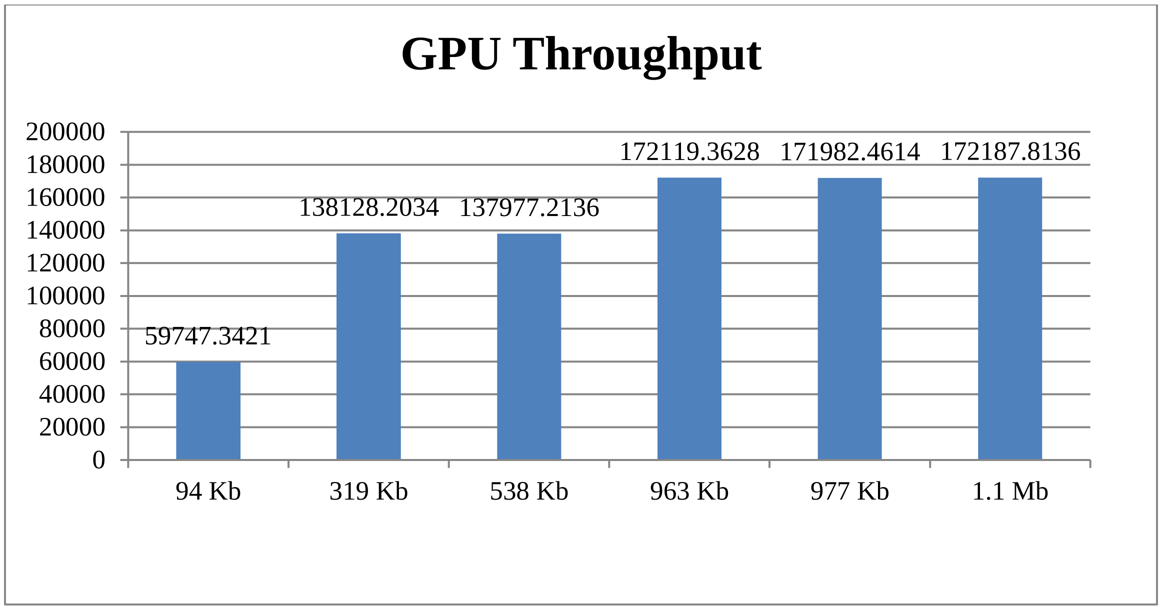


Figure 4.1.1.4: GPU Throughput for Groestl for Text Files

4.1.2 Audio Dataset

The following table 4.1.2 shows experimental results of Groestl for Audio Dataset. The highest CPU Time is found for the file size of 1.11 Mb i.e., 14.59 bytes per sec, and lowest CPU Time was observed on file of size 719 Kb i.e., 14.23 bytes per sec. Whereas, highest CPU Throughput (Bytes per Cycle) was found for the file size of 1.11 Mb i.e., 101.20 bytes per cycle, and lowest CPU Throughput was observed for the file size of 384 kb i.e., 101.13 bytes per cycle.

Similarly, highest GPU Time was found for the file size of 457 kb i.e., 0.0427 bytes per sec and the lowest GPU Time is for the file size of 1.11 mb i.e., 0.0085 bytes per sec. Whereas, highest GPU Throughput was found for the file size of 1.11 mb i.e., 173180.3492 bytes per cycle and lowest GPU Throughput was observed for the file size of 384 kb i.e., 34282.2758 bytes per cycle.

Table 4.1.2: Test results for Groestl on audio files

Audio File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
384 Kb	14.25	101.13	0.042	34282.2758
457 Kb	14.46	101.18	0.0427	34299.2254
719 Kb	14.23	101.19	0.042	34302.6153
1.1 Mb	14.59	101.2	0.0085	173180.3492

Following are graphical representations of results of Groestl on Audio dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Audio dataset.

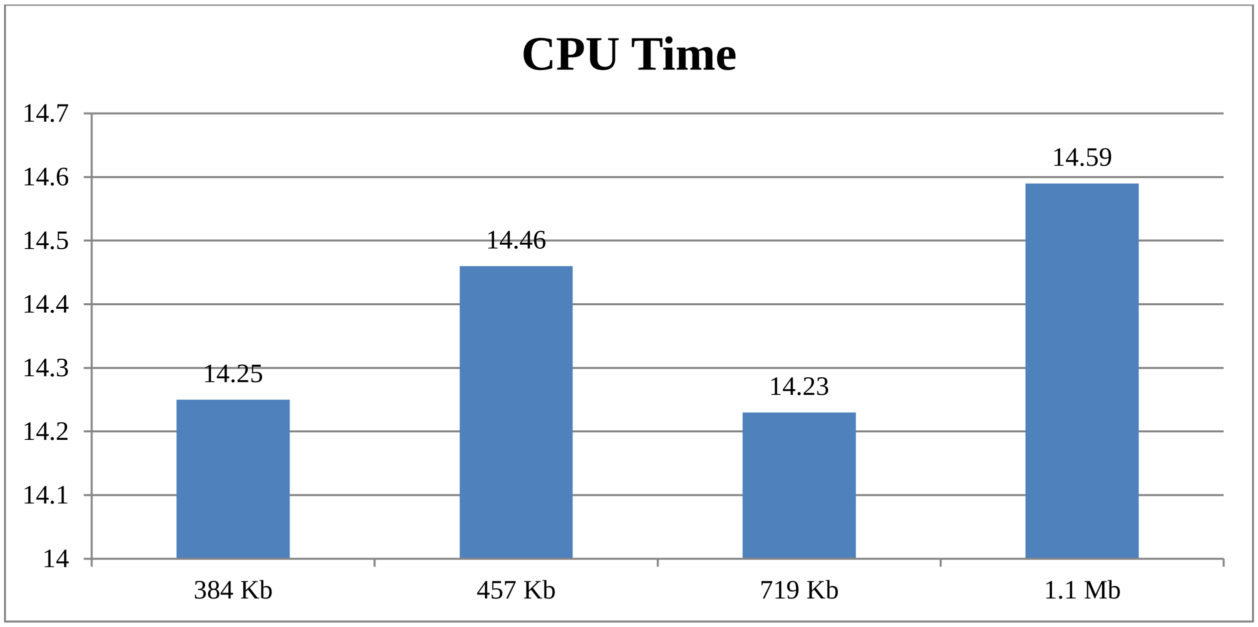


Figure 4.1.2.1: CPU Time for Groestl for Audio Files

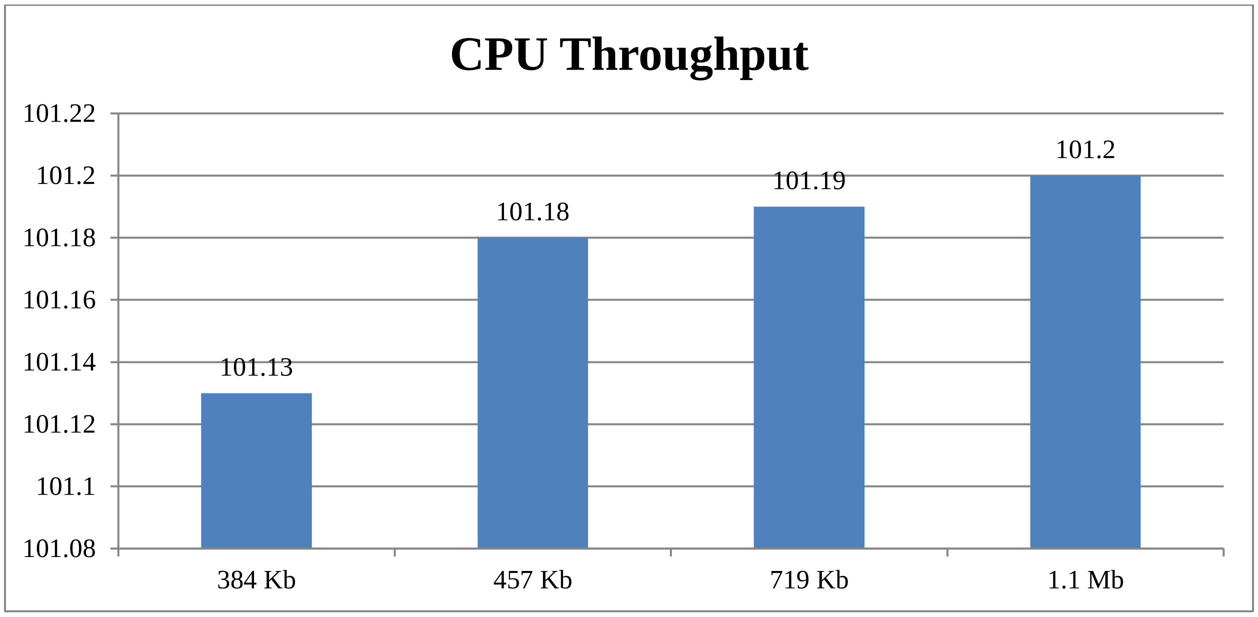


Figure 4.1.2.2: CPU Throughput for Groestl for Audio Files

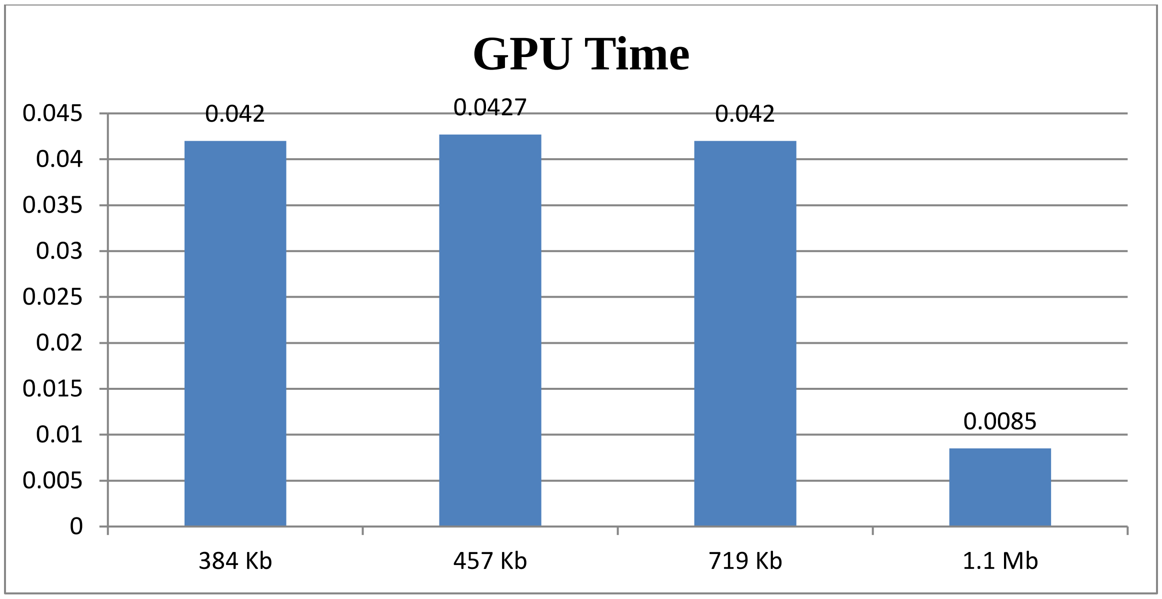


Figure 4.1.2.3: GPU Time for Groestl for Audio Files

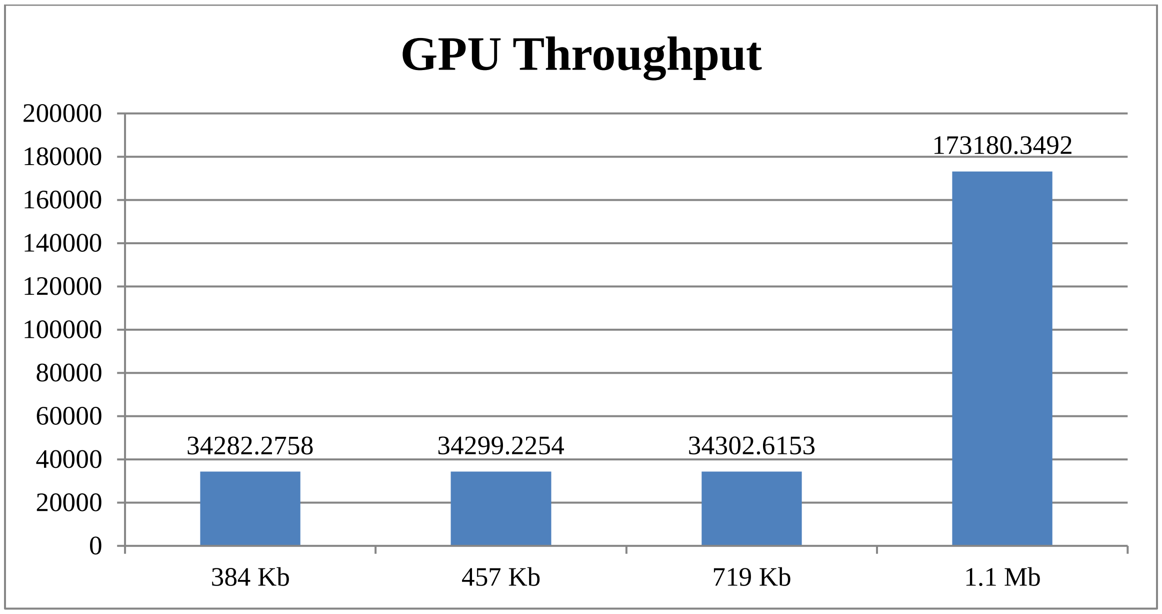


Figure 4.1.2.4: GPU Throughput for Groestl for Audio Files

4.1.3 Video Dataset

Table 4.1.3 shows results for Groestl on Video Dataset. Highest CPU time was recorded for the file size of 960 kb i.e., 6.11 bytes per sec and lowest was recorded for file size of 99 kb i.e., 4.96 bytes per sec. Whereas, highest CPU Throughput was recorded for the file size of 99 kb i.e., 218.75 bytes per cycle and lowest was recorded for file size of 141 kb i.e., 217.93 bytes per cycle.

Similarly, highest GPU Time was recorded for the file size of 141 kb i.e., 0.0091 bytes per sec and lowest was recorded for file size 769 kb i.e., 0.0035 bytes per sec. Whereas, highest GPU Throughput was recorded for file size of 960 kb i.e., 373604.0893 bytes per cycle and lowest was recorded for file size of 99 kb i.e., 129249.714 bytes per cycle.

Table 4.1.3: Test results for Groestl on Video files

Video File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
99 kb	4.96	218.75	0.0084	129249.714
141 kb	5.39	217.93	0.0091	128765.2122
240 kb	5.33	218.39	0.0058	202345.5885
769 kb	5.96	218.3	0.0035	373569.8639
960 kb	6.11	218.32	0.0036	373604.0893

Following are graphical representations of results of Groestl on video dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Video data set.

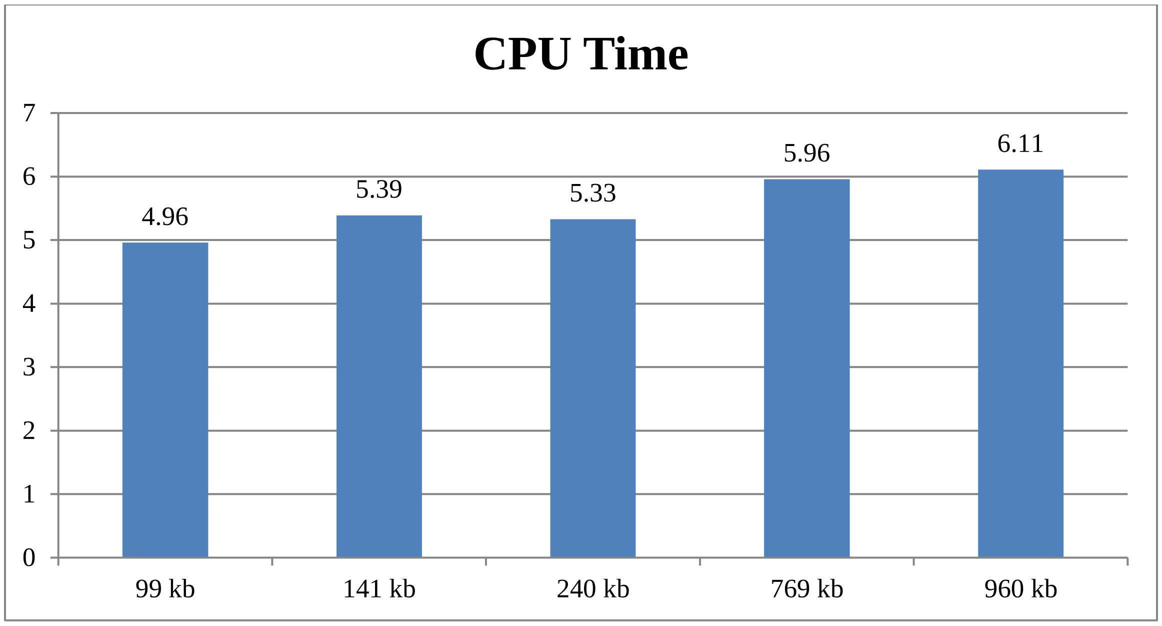


Figure 4.1.3.1: CPU Time for Groestl for Video Files

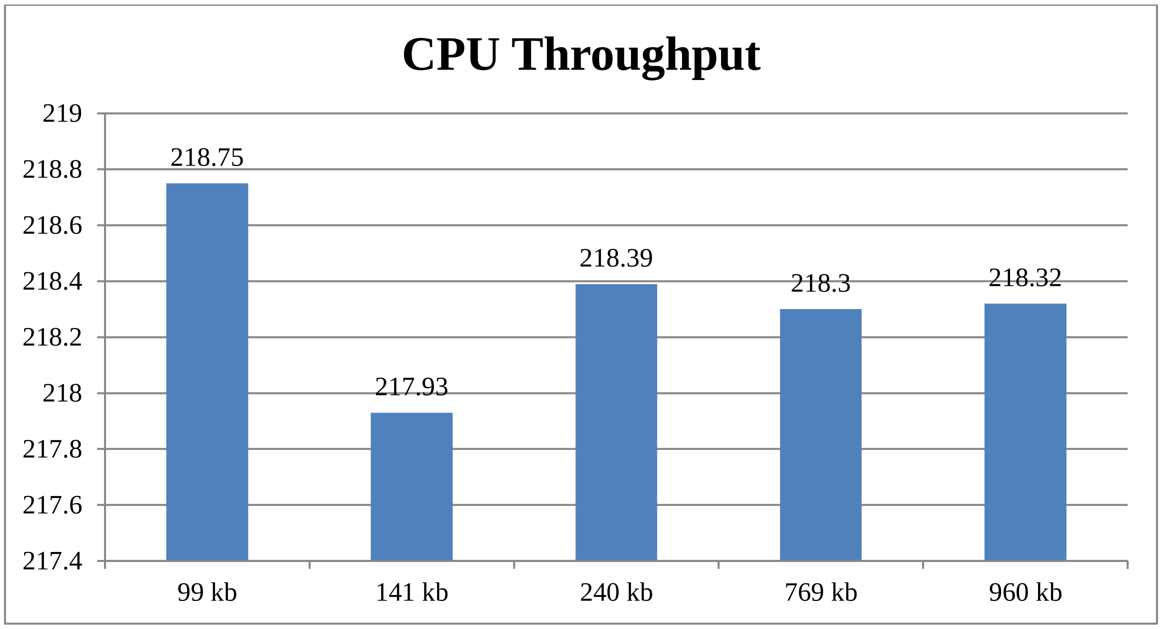


Figure 4.1.3.2: CPU Throughput for Groestl for Video Files

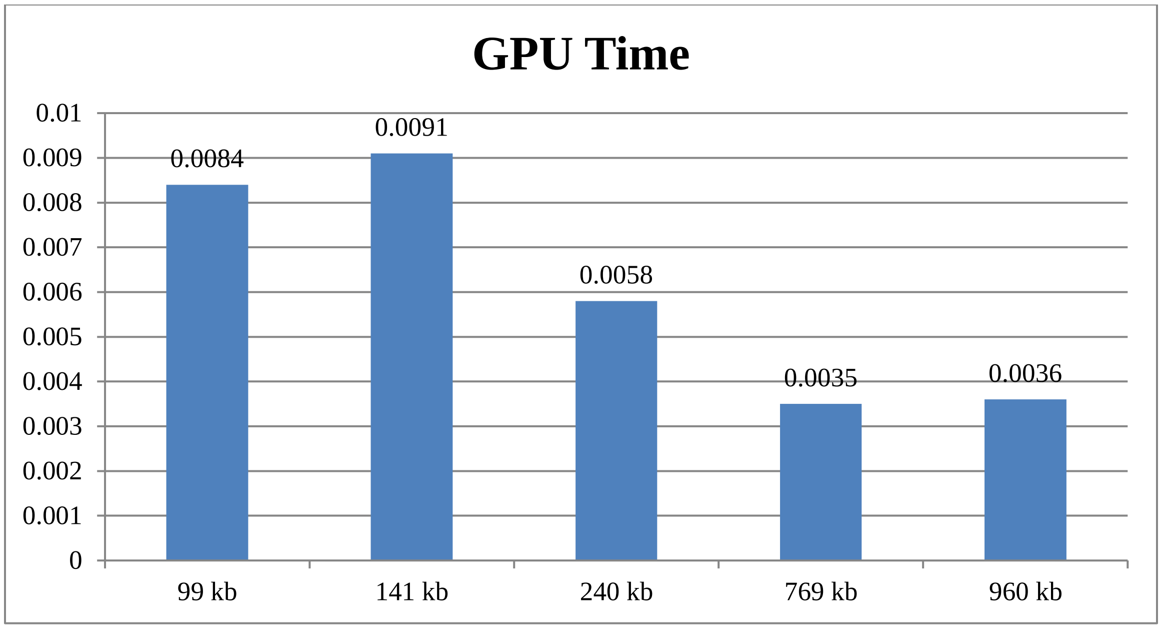


Figure 4.1.3.3: GPU Time for Groestl for Video Files

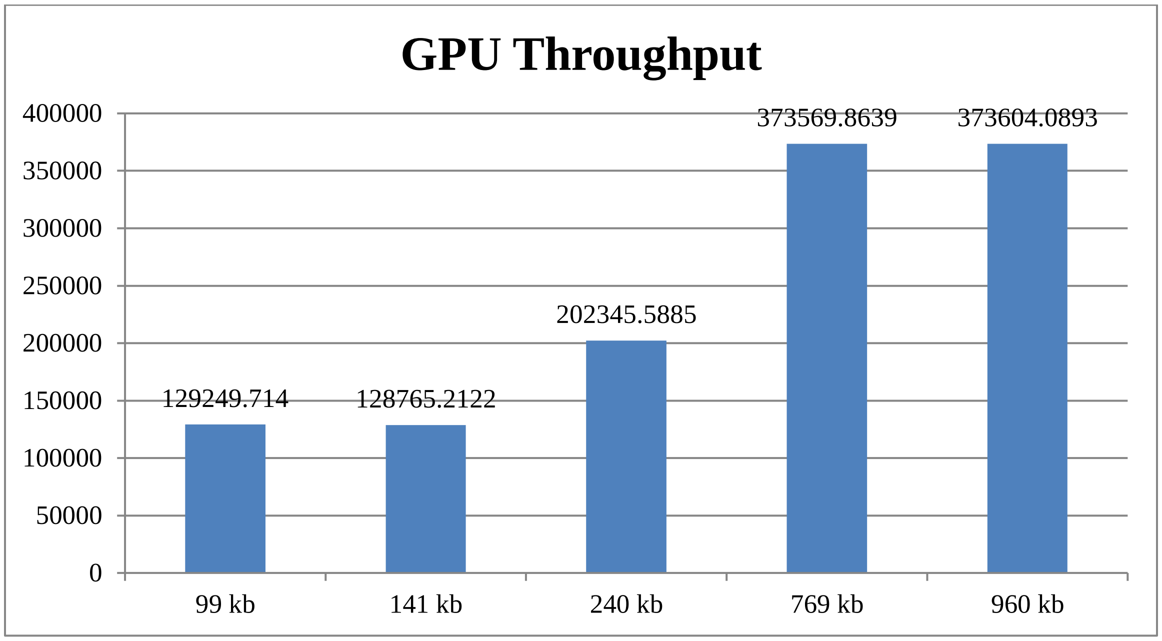


Figure 4.1.3.4: GPU Throughput for Groestl for Video Files

4.1.4 Image Dataset

Table 4.1.4depicts results of Groestl on Image Dataset. Highest CPU Time was observed on file size of 960 kb i.e., 6.35 bytes per sec, and lowest CPU Time was observed on file size of 107 kb i.e., 5.92 bytes per sec. Highest CPU Throughput was observed on file size of 960 kb i.e., 231.32 bytes per cycle and lowest CPU Throughput was observed on file size of 80 kb i.e., 217.82 bytes per cycle.

Similarly, highest GPU Time was observed on file size of 80 kb i.e 0.0103 bytes per sec and lowest was observed on file size of 145 kb i.e., 0.0065 bytes per sec. Highest GPU Throughput was recorded on 960 kb file i.e., 375745.6589 bytes per cycle and lowest was recorded on file size of 80 kb i.e., 128700.2181 bytes per cycle.

Table 4.1.4: Test results for Groestl on Image files

Image File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
80 kb	6.09	217.82	0.0103	128700.2181
107 kb	5.92	221.55	0.01	130904.1103
145 kb	6.01	222.41	0.0065	202364.1192
769 kb	6.22	223.3	0.0069	374545.5231
960 kb	6.35	231.32	0.0074	375745.6589

Following are graphical representations of results of Groestl on Image dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Image data set.

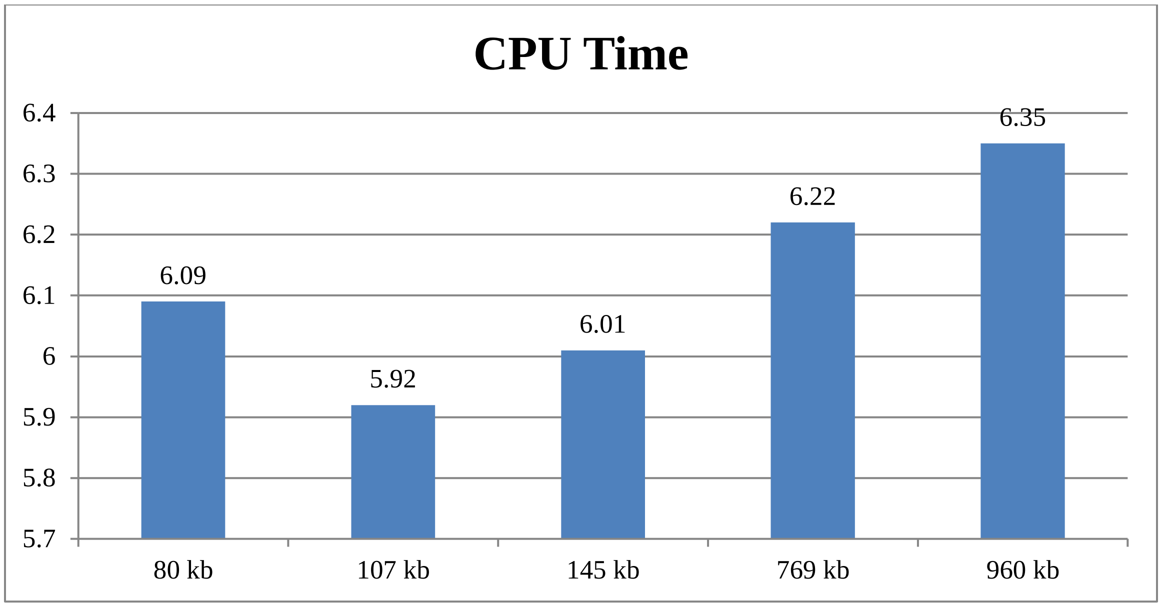


Figure 4.1.4.1: CPU Time for Groestl for Image files

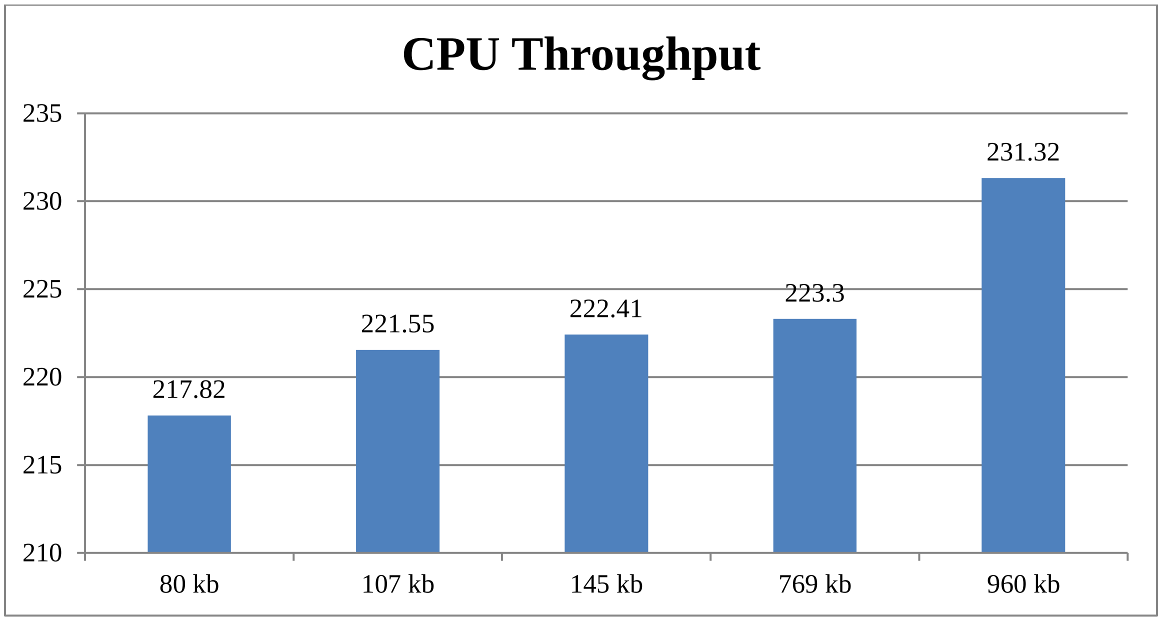


Figure 4.1.4.2: CPU Throughput for Groestl for Image files

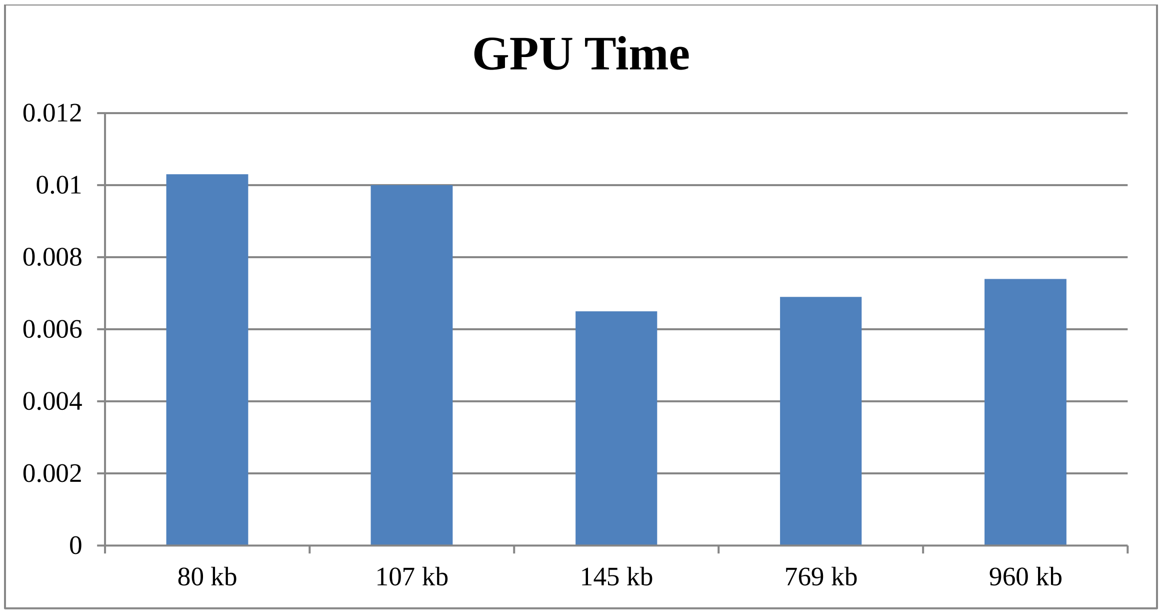


Figure 4.1.4.3: GPU Time for Groestl for Image files

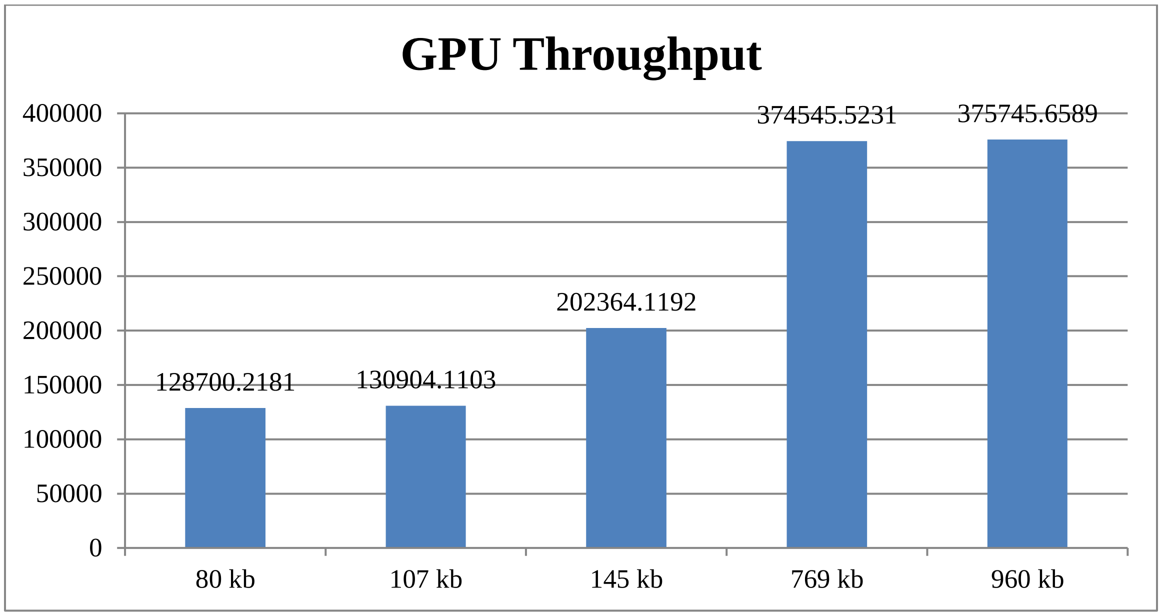


Figure 4.1.4.4: GPU Throughput for Groestl for Image files

4.2 Experimental Results for JH

4.2.1 Text Dataset

Table 4.2.1 shows experimental results of JH for Text Dataset. The highest CPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 9.9 bytes per sec and lowest CPU Time (Bytes per Second) is observed for the file size of 963 kb i.e., 9.49 bytes per sec. The highest CPU Throughput (Bytes per Cycle) is found for file size of 94 kb i.e., 72.87 bytes per cycle and Lowest CPU Throughput (Bytes per Cycle) is observed for the file size of 963 kb i.e., 72.63 bytes per cycle.

Similarly, highest GPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 0.0168 bytes per sec and Lowest GPU Time is found for the file size of 963 kb i.e., 0.0055 bytes per sec. Highest GPU Throughput (Bytes per Cycle) is found for the file size of 1.1 mb i.e., 124323.64 bytes per cycle and lowest GPU Throughput is observed on the file size of 94 kb i.e., 43055.6647 bytes per cycle.Test results for JH on CPU and GPU for Text File Dataset is appended below

Table 4.2.1: Test results for JH on Text files

Text File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
94 kb	9.9	72.87	0.0168	43055.6647
319 kb	9.57	72.64	0.007	99708.1655
538 kb	9.53	72.66	0.0069	99735.6182
963 kb	9.49	72.63	0.0055	124289.4146
977 kb	9.73	72.64	0.0059	124306.5273
1.1 mb	9.8	72.65	0.0061	124323.64

Following are graphical representations of results of JH on Text dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Text data set.

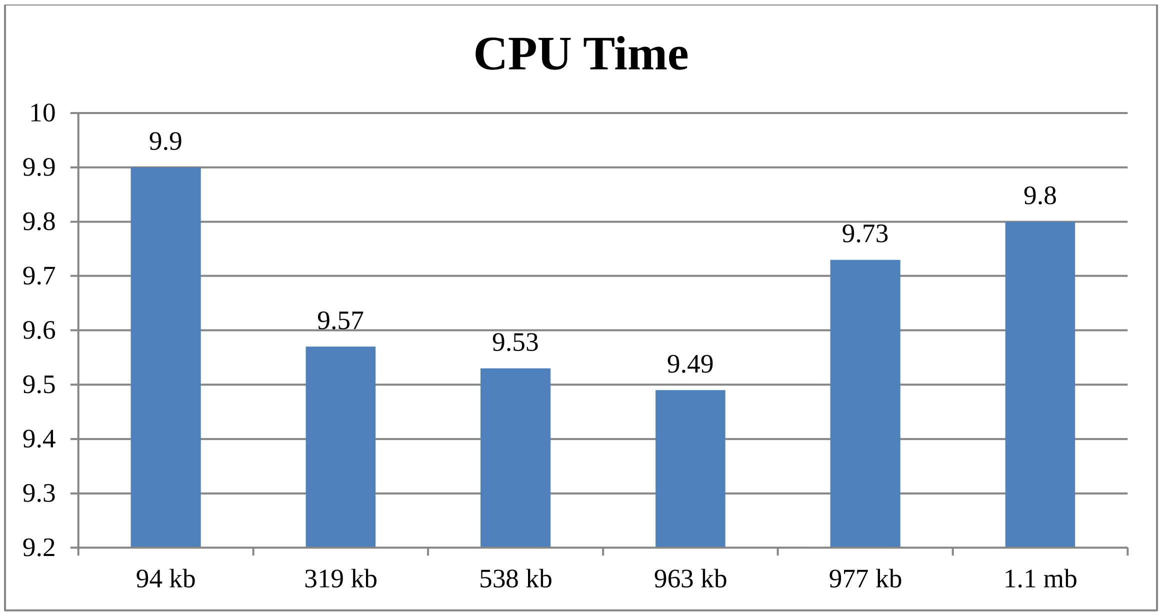


Figure 4.2.1.1: CPU Time for JH for Text Files

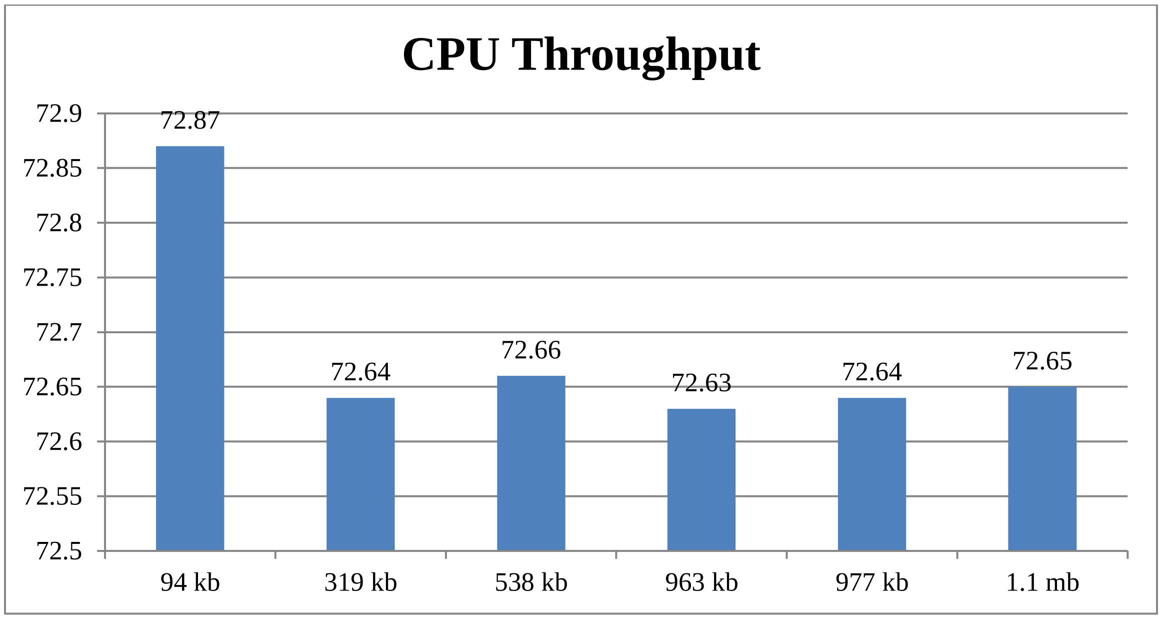


Figure 4.2.1.2: CPU Throughput for JH for Text files

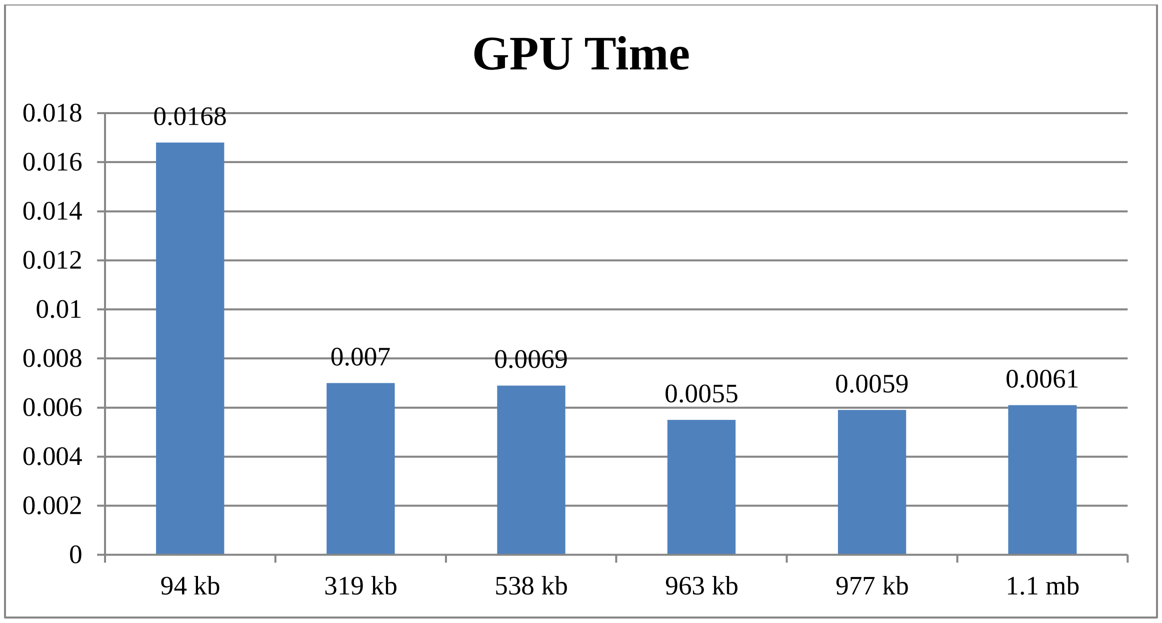


Figure 4.2.1.3: GPU Time for JH for Text files

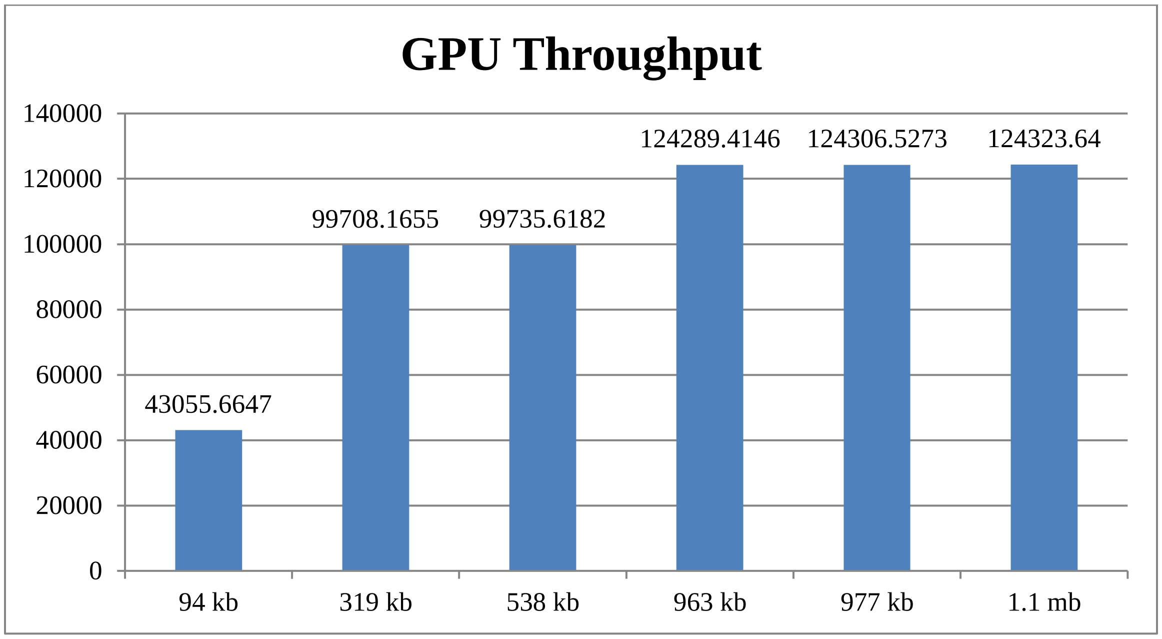


Figure 4.2.1.4: GPU Throughput for JH for Text files

4.2.2 Audio Dataset

The following table 4.2.2 shows experimental results of JH for Audio Dataset. The highest CPU Time is found for the file size of 719 kb i.e., 9.82 bytes per sec, and lowest CPU Time was observed on file of size 457 kb i.e., 9.74 kb. Whereas, highest CPU Throughput (Bytes per Cycle) was found for the file size of 457 kb i.e., 72.89 bytes per cycle, and lowest CPU Throughput was observed for the file size of 384 kb i.e., 72.83 bytes per cycle.

Similarly, highest GPU Time was found for the file size of 719 kb i.e., 0.029 bytes per sec and the lowest GPU Time is for the file size of 1.1 mb i.e., 0.0057 bytes per sec. Whereas, highest GPU Throughput was found for the file size of 1.1 mb i.e., 124700.119 bytes per cycle and lowest GPU Throughput was observed for the file size of 384 kb i.e., 24688.78 bytes per cycle.

Table 4.2.2: Test results for JH on Audio files

Audio File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
384 kb	9.81	72.83	0.0289	24688.78
457 kb	9.74	72.89	0.0287	24709.1376
719 kb	9.82	72.87	0.029	24702.3577
1.1 mb	9.8	72.87	0.0057	124700.119

Following are graphical representations of results of JH on Audio dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Audio data set.

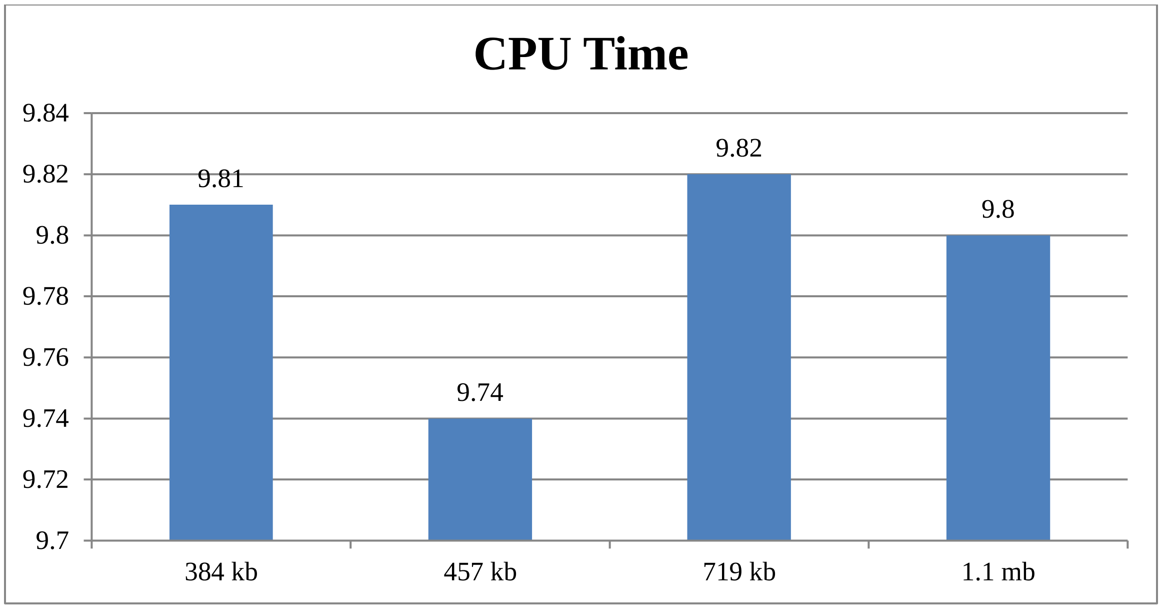


Figure 4.2.2.1: CPU Time for JH for Audio files

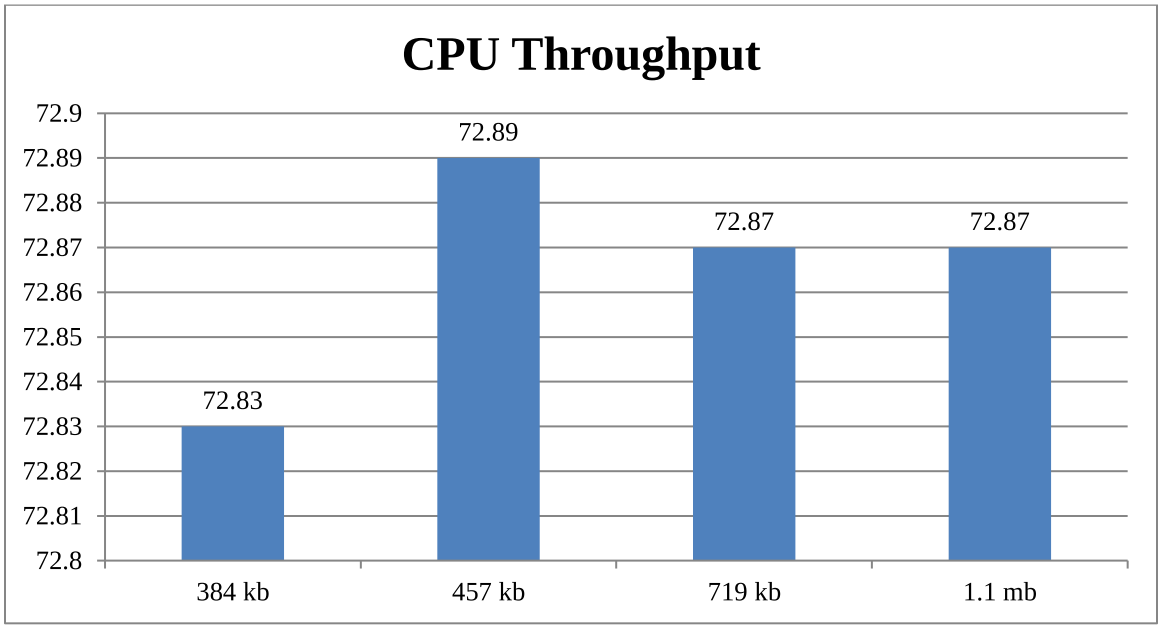


Figure 4.2.2.2: CPU Throughput for JH for Audio files

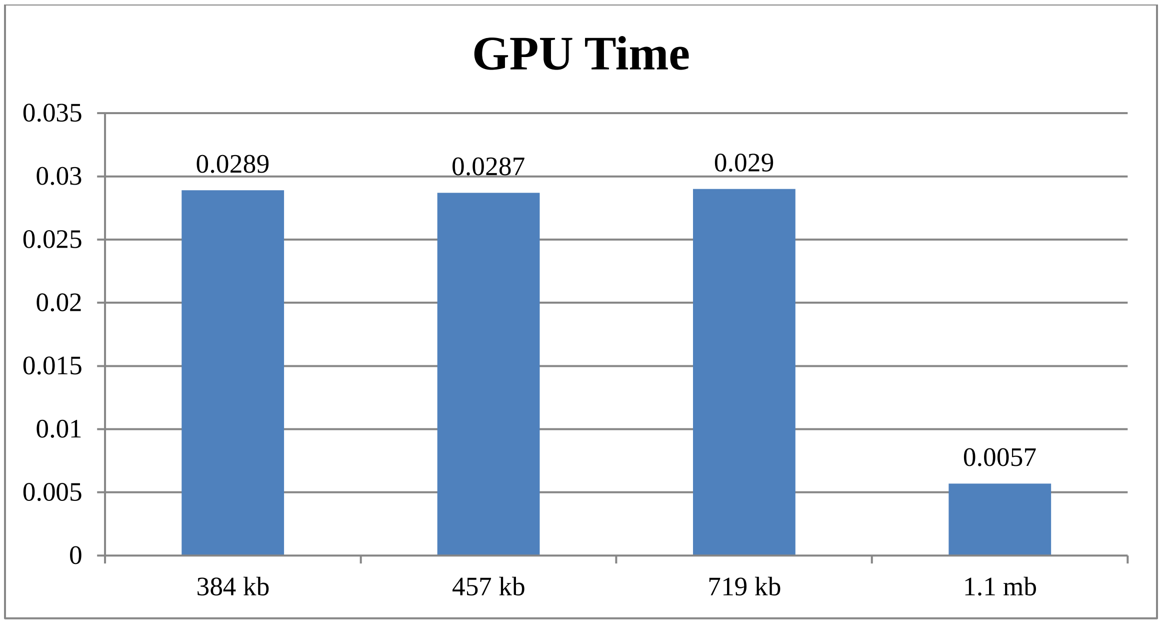


Figure 4.2.2.3: GPU Time for JH for Audio files

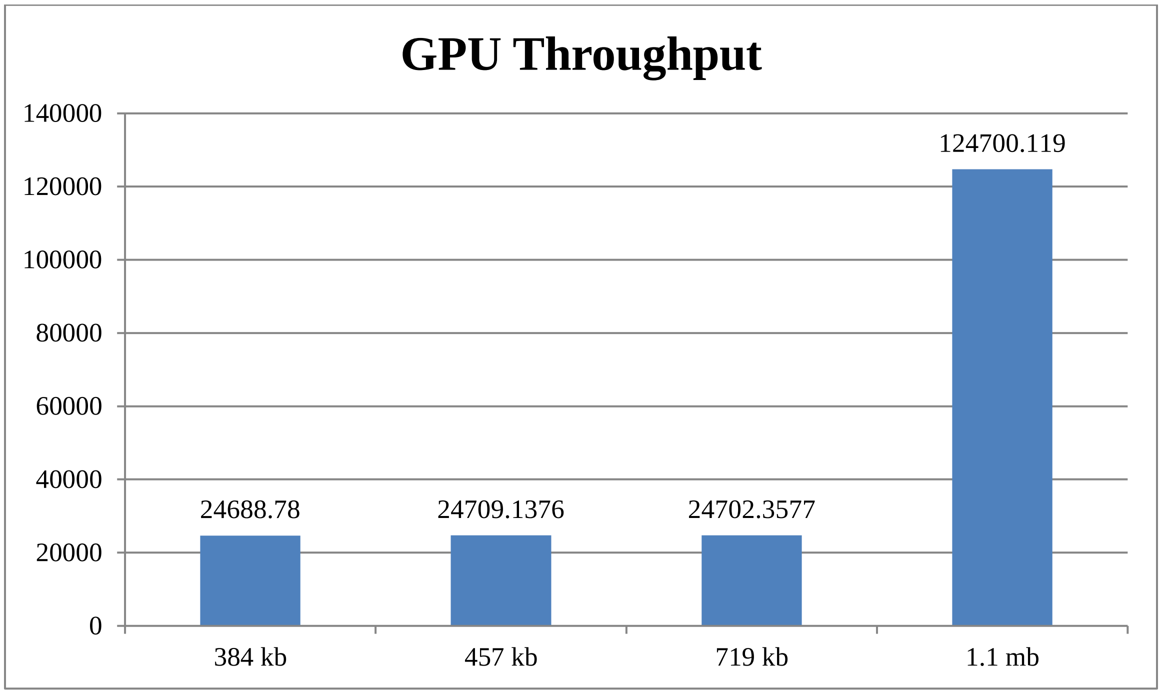


Figure 4.2.2.4: GPU Throughput for JH for Audio files

4.2.3 Video Dataset

Table 4.2.3 shows results for JH on Video Dataset. Highest CPU time was recorded for the file size of 960 kb i.e., 4.24 bytes per sec and lowest was recorded for file size of 99 kb i.e., 3.47 bytes per sec. Whereas, highest CPU Throughput was recorded for the file size of 99 kb i.e., 155.92 bytes per cycle and lowest was recorded for file size of 141 kb i.e., 154.99 bytes per cycle.

Similarly, highest GPU Time was recorded for the file size of 141 kb i.e., 0.0063 bytes per sec and lowest was recorded for file size 769 kb i.e., 0.0024 bytes per sec. Whereas, highest GPU Throughput was recorded for file size of 769 kb i.e., 265503.2725 bytes per cycle and lowest was recorded for file size of 141 kb i.e., 91576.7459 bytes per cycle.

Table 4.2.3: Test results for JH on Video files

Video File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
99 kb	3.47	155.92	0.0059	92126.2418
141 kb	3.7	154.99	0.0063	91576.7459
240 kb	3.88	155.9	0.0042	144446.528
769 kb	4.17	155.15	0.0024	265503.2725
960 kb	4.24	155.14	0.0025	265486.1598

Following are graphical representations of results of JH on Video dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Video data set.

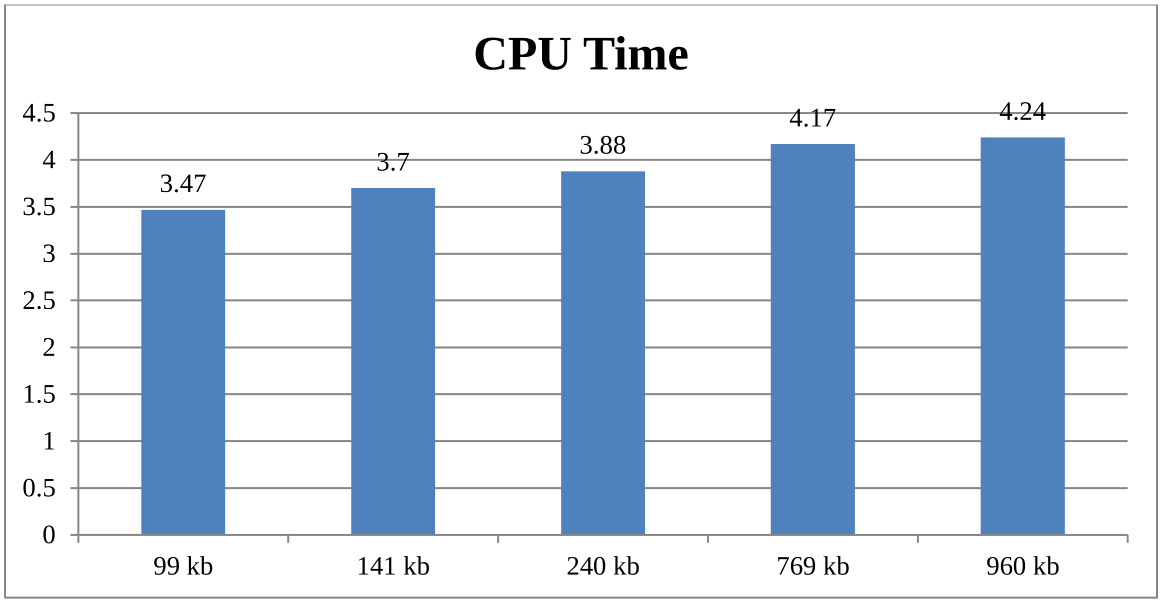


Figure 4.2.3.1: CPU Time for JH for Video files

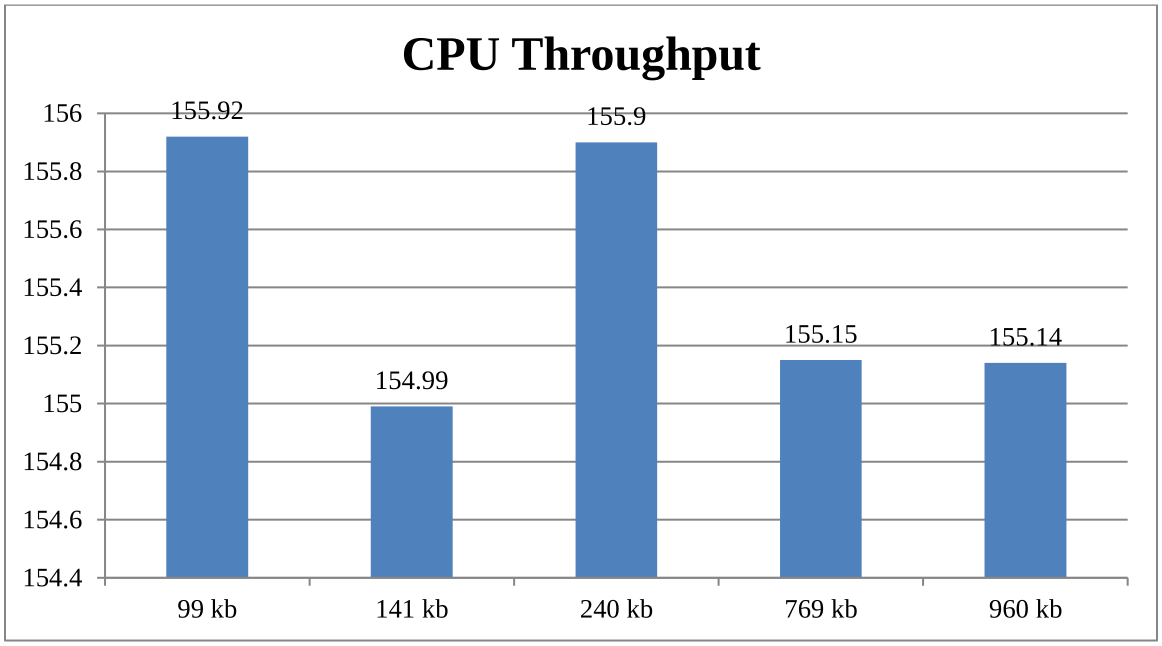


Figure 4.2.3.2: CPU Throughput for JH for Video files

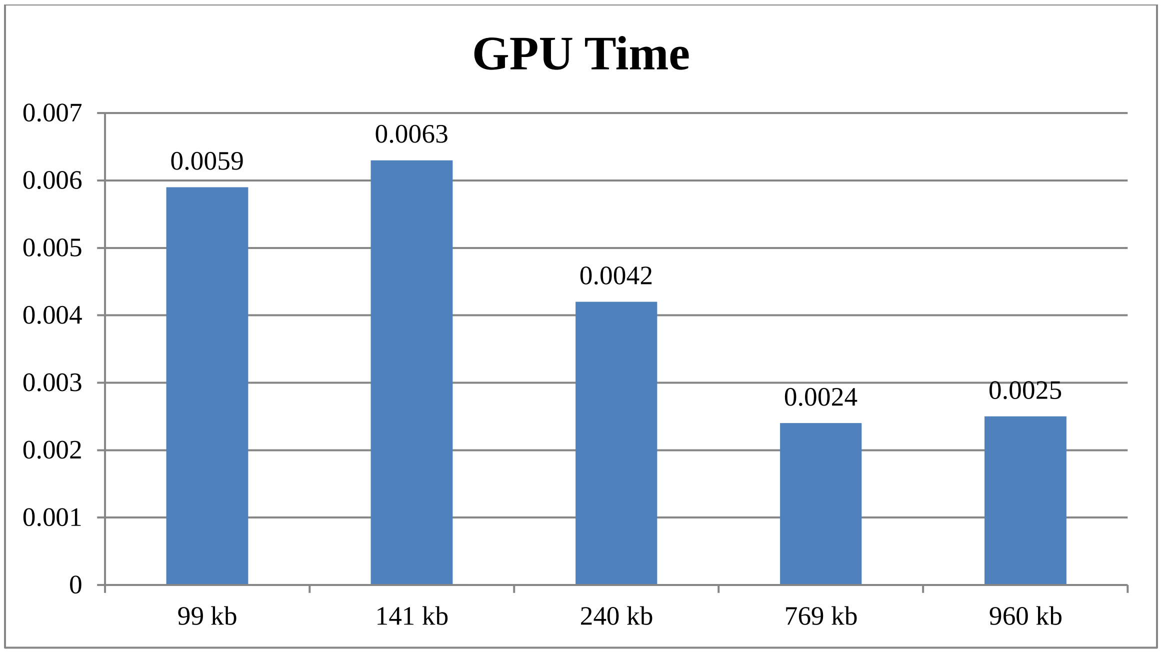


Figure 4.2.3.3: GPU Time for JH for Video files

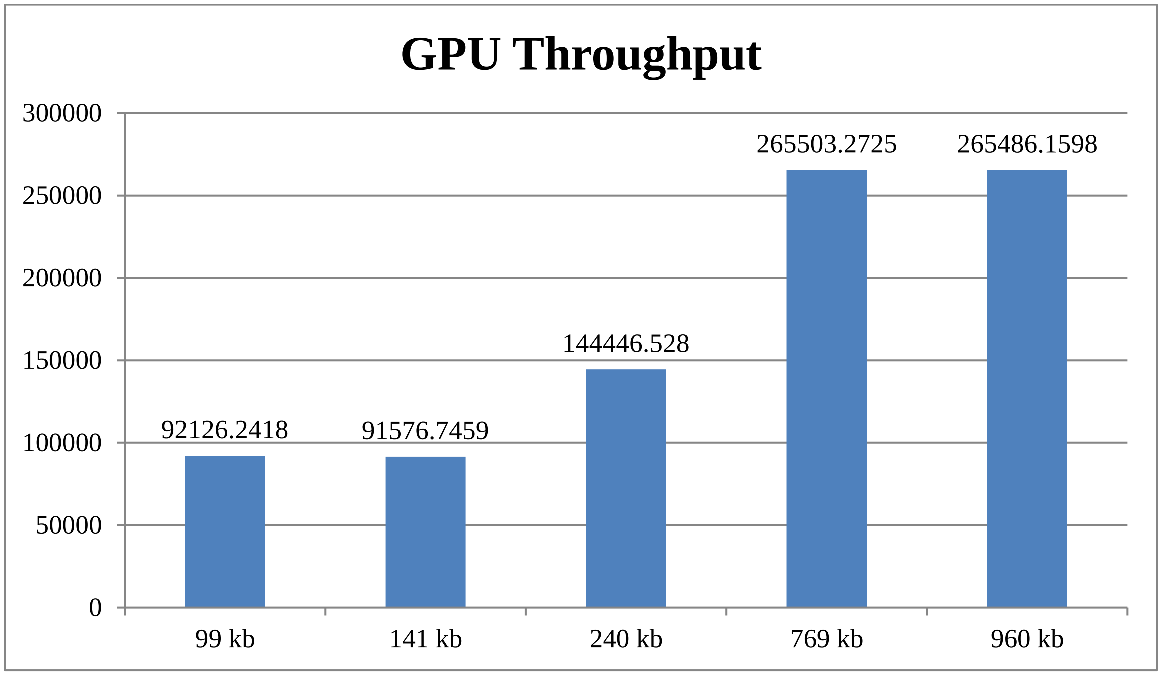


Figure 4.2.3.4: GPU Throughput for JH for Audio files

4.2.4 Image Dataset

Table 4.2.4depicts results of JH on Image Dataset. Highest CPU Time was observed on file size of 960 kb i.e., 4.3 bytes per sec, and lowest CPU Time was observed on file size of 80 kb i.e., 4.15 bytes per sec. Highest CPU Throughput was observed on file size of 769 kb i.e., 155.74 bytes per cycle and lowest CPU Throughput was observed on file size of 80 kb i.e., 155.09 bytes per cycle.

Similarly, highest GPU Time was observed on file size of 107 kb i.e 0.0071 bytes per sec and lowest was observed on file size of 769 kb i.e., 0.0024 bytes per sec. Highest GPU Throughput was recorded on 960 kb file i.e., 267545.6963 bytes per cycle and lowest was recorded on file size of 80 kb i.e., 91635.8315 bytes per cycle.

Table 4.2.4: Test results for JH on Image files

Image File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
80 kb	4.15	155.09	0.007	91635.8315
107 kb	4.17	155.13	0.0071	91659.4657
145 kb	4.17	155.12	0.0045	143723.8321
769 kb	4.16	155.74	0.0024	266412.2125
960 kb	4.3	155.8	0.0027	267545.6963

Following are graphical representations of results of JH on Image dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Image data set.

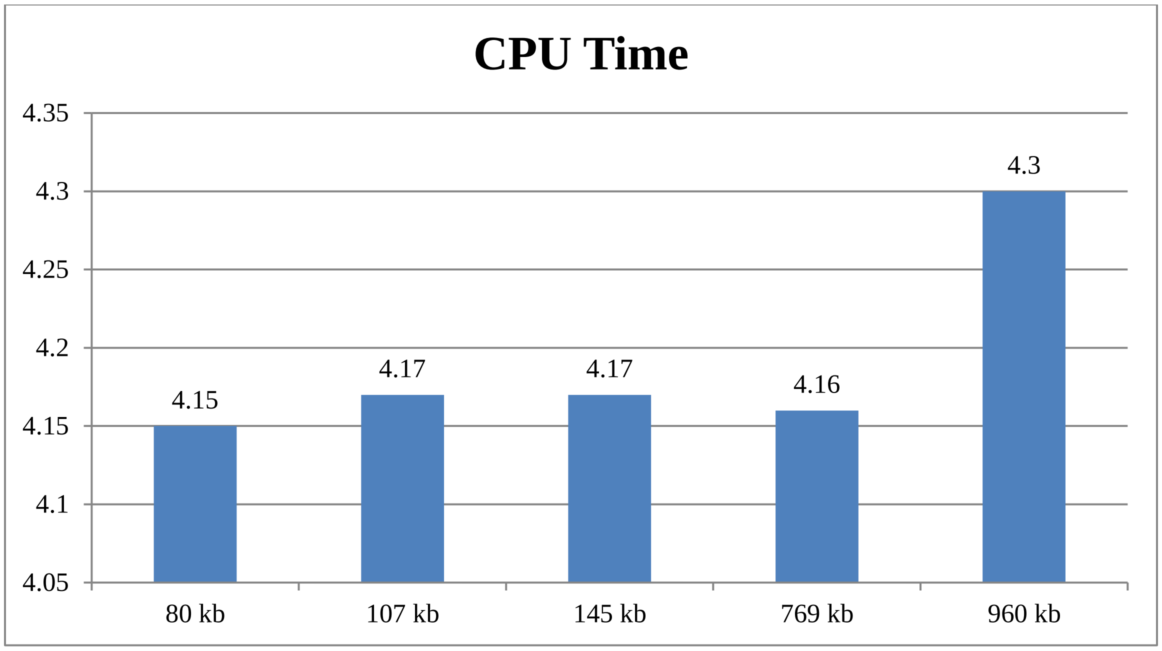


Figure 4.2.4.1: CPU Time for JH for Image files

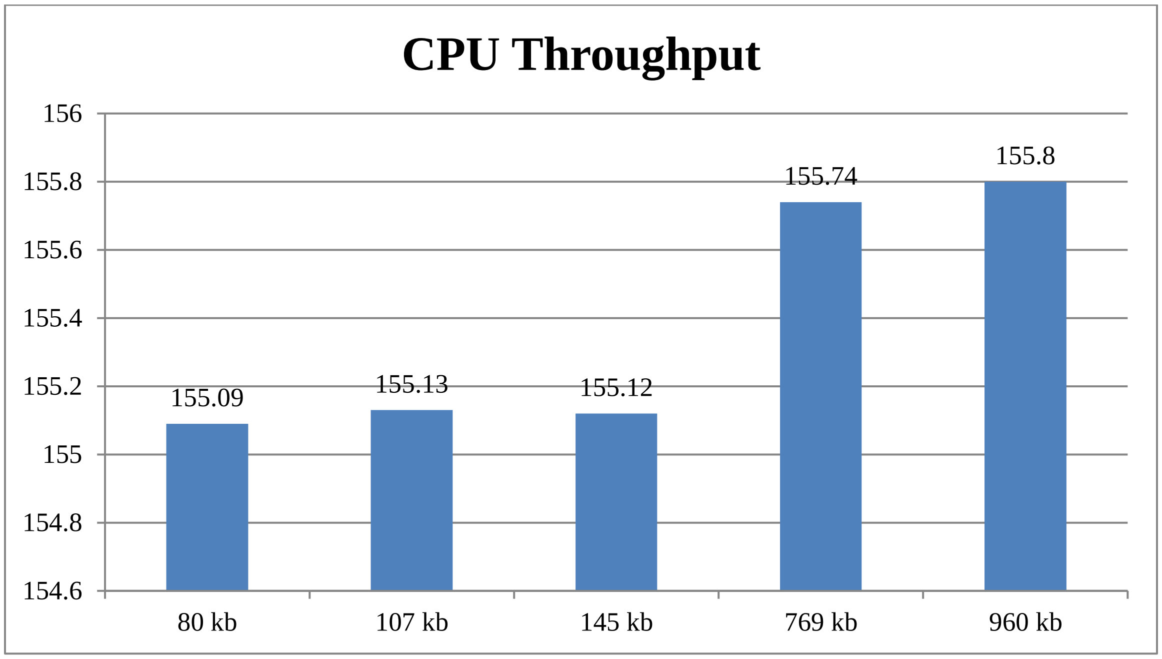


Figure 4.2.4.2: CPU Throughput for JH for Image files

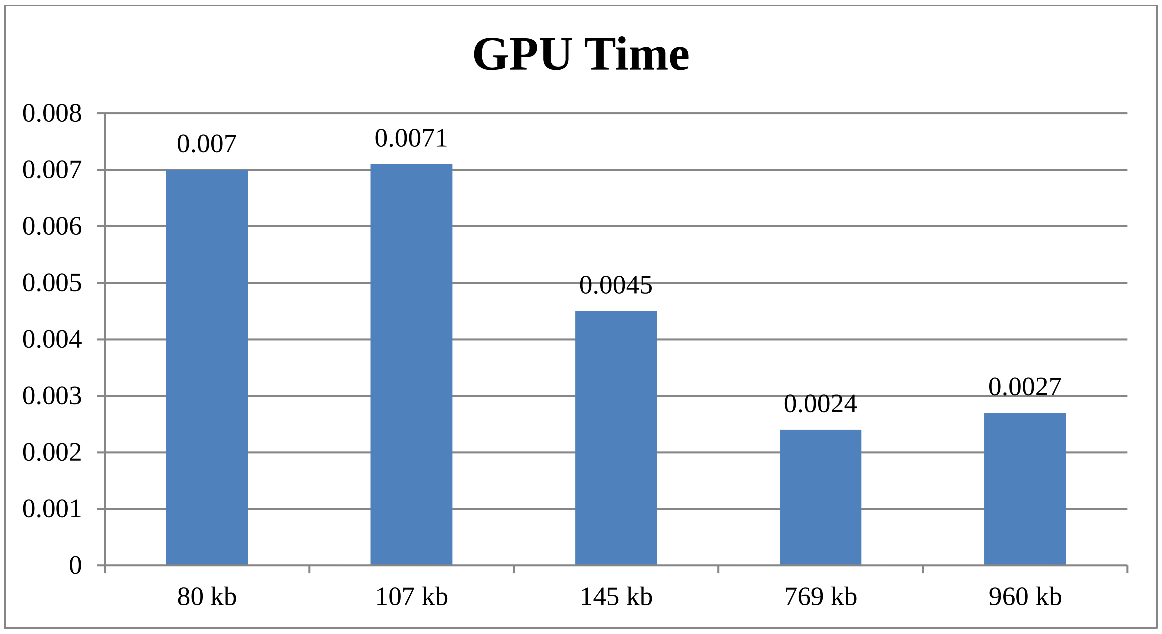


Figure 4.2.4.3: GPU Time for JH for Imagefiles

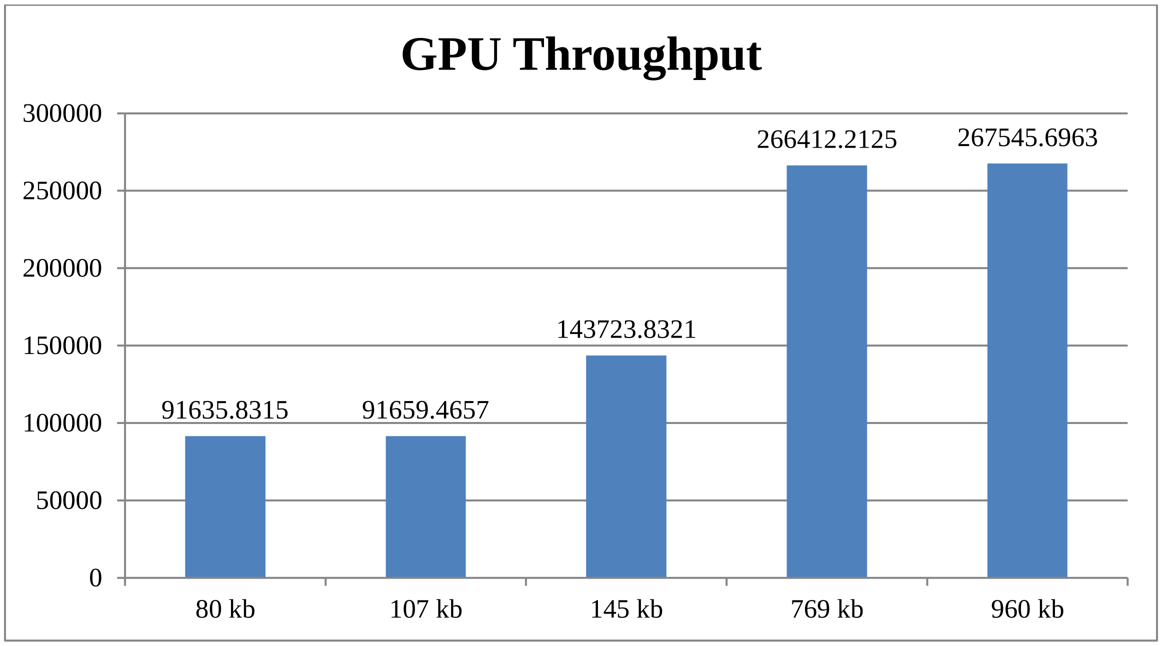


Figure 4.2.4.4: GPU Throughput for JH for Image files

4.3 Experimental Results for Keccak

4.3.1 Text Dataset

Table 4.3.1 shows experimental results of Keccak for Text Dataset. The highest CPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 20.47 bytes per sec and Lowest CPU Time (Bytes per Second) is observed for the file size of 319 kb i.e., 19.63 bytes per sec. The highest CPU Throughput (Bytes per Cycle) is found for file size of 1.1 mb i.e., 153.65 bytes per cycle and Lowest CPU Throughput (Bytes per Cycle) is observed 152.77 bytes per cycle for file size of 538 kb.

Similarly, highest GPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 0.0346 bytes per sec and Lowest GPU Time is found for the file size of 963 kb i.e., 0.0118 bytes per sec. Highest GPU Throughput (Bytes per Cycle) is 262328.63 bytes per cycle for file size of 1.1 mb and lowest GPU Throughput is observed on the file size of 94 kb i.e., 90377.3086.

Test results for Keccak on CPU and GPU for Text File Dataset is appended below

Table 4.3.1: Test results for Keccak on Text files

Text File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
94 kb	20.47	152.96	0.0346	90377.3086
319 kb	19.63	152.81	0.0143	209752.2683
538 kb	19.66	152.77	0.0143	209697.363
963 kb	20.26	153.21	0.0118	261430.454
977 kb	20.45	152.98	0.012	261289.7
1.1 mb	20.32	153.65	0.0119	262328.63

Following are graphical representations of results of Keccak on Text dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Text data set.

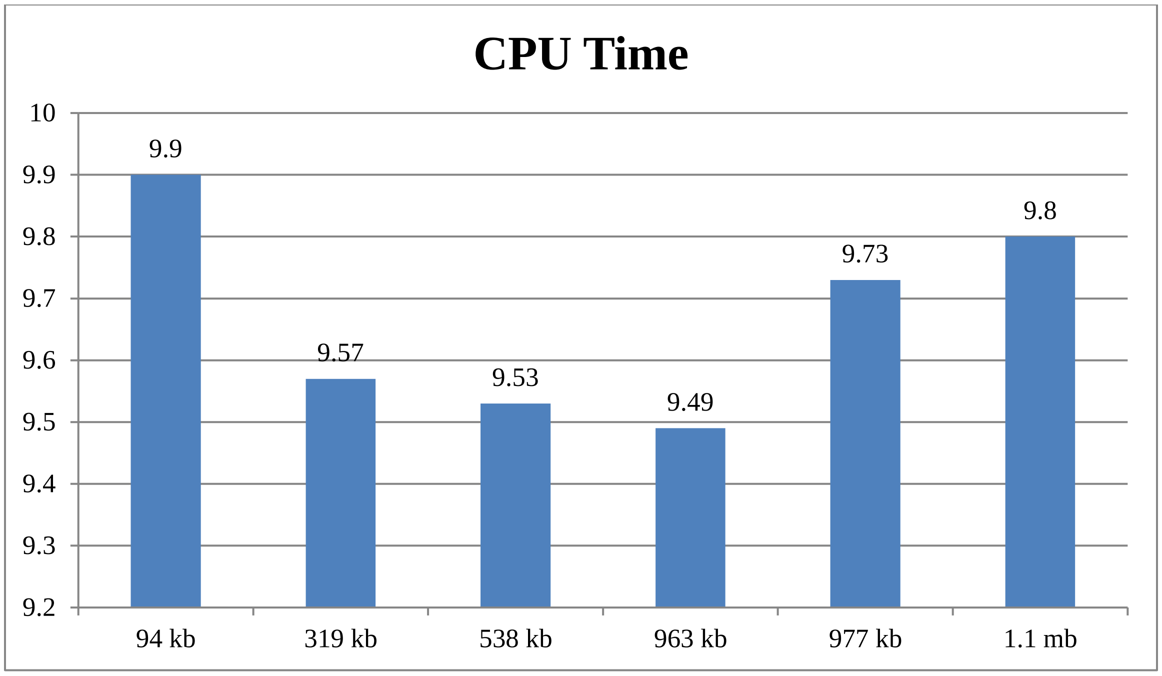


Figure 4.3.1.1: CPU Time for Keccak for Text files

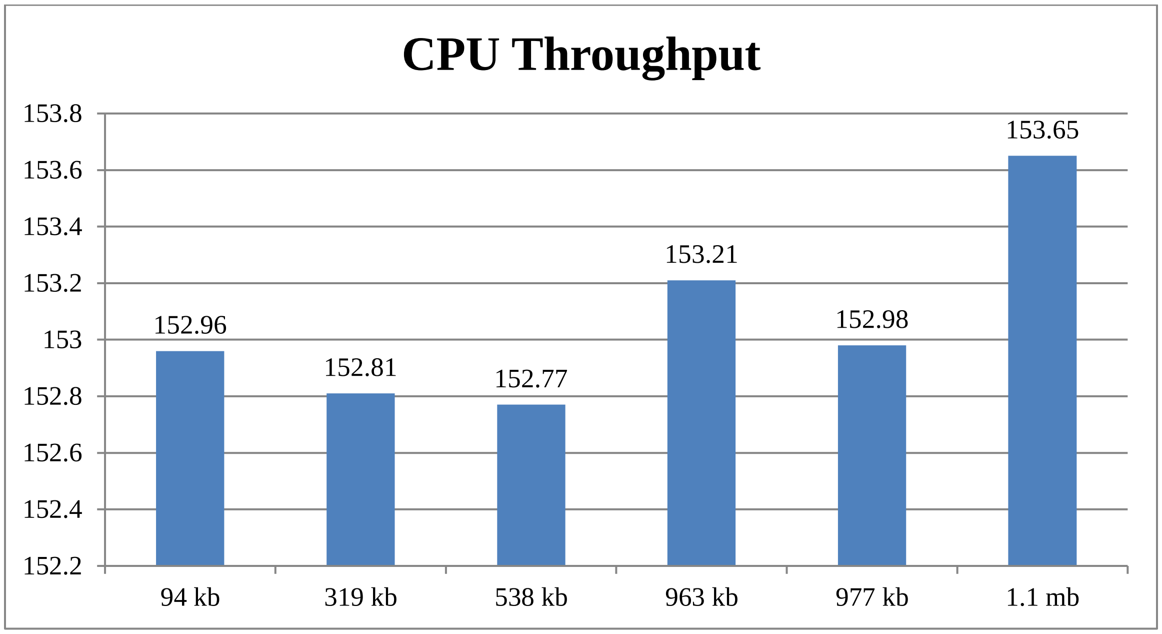


Figure 4.3.1.2: CPU Throughput for Keccak for Text files

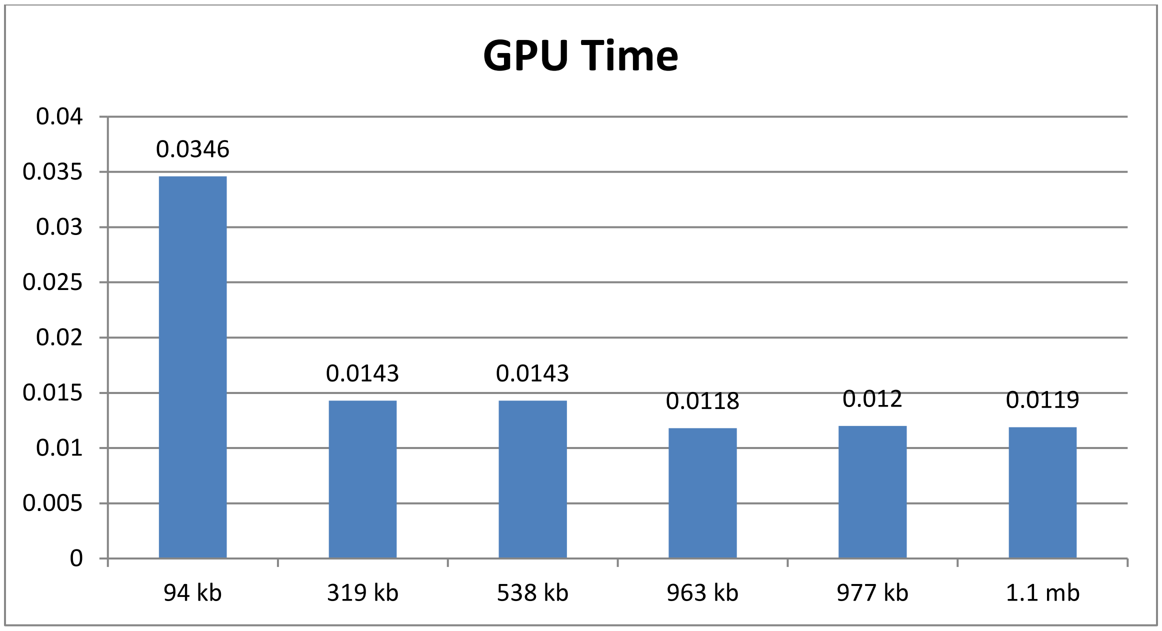


Figure 4.3.1.3: GPU Time for Keccak for Text files

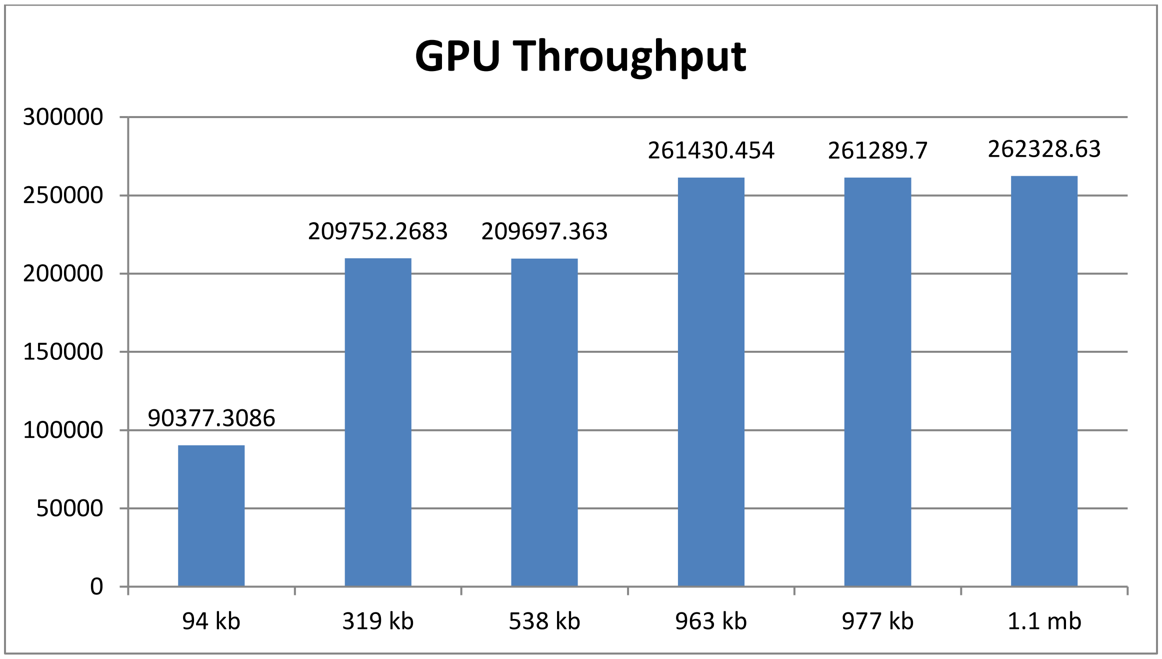


Figure 4.3.1.4: GPU Throughput for Keccak for Text files

4.3.2 Audio Dataset

Table 4.3.2 shows experimental results of Keccak for Audio Dataset. The highest CPU Time is found for the file size of 457 kb i.e., 20.97 bytes per sec, and lowest CPU Time was observed on file of size 719 kb i.e., 18.89 bytes per sec. Whereas, highest CPU Throughput (Bytes per Cycle) was found for the file size of 1.11 mb i.e., 152.96 bytes per cycle, and lowest CPU Throughput was observed for the file size of 384 kb i.e., 152.85 bytes per cycle.

Similarly, highest GPU Time was found for the file size of 457 kb i.e., 0.0619 bytes per sec and the lowest GPU Time is for the file size of 1.11 mb i.e., 0.0122 bytes per sec. Whereas, highest GPU Throughput was found for the file size of 1.11 mb i.e., 261755.595 bytes per cycle and lowest GPU Throughput was observed for the file size of 384 kb i.e., 51814.9496 bytes per cycle.

Table 4.3.2: Test results for Keccak on Audio files

Audio File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
384 kb	19.1	152.85	0.0563	51814.9496
457 kb	20.97	152.95	0.0619	51848.8488
719 kb	18.89	152.95	0.0557	51848.8488
1.1 mb	20.91	152.96	0.0122	261755.595

Following are graphical representations of results of Keccak on Audio dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Audio data set.

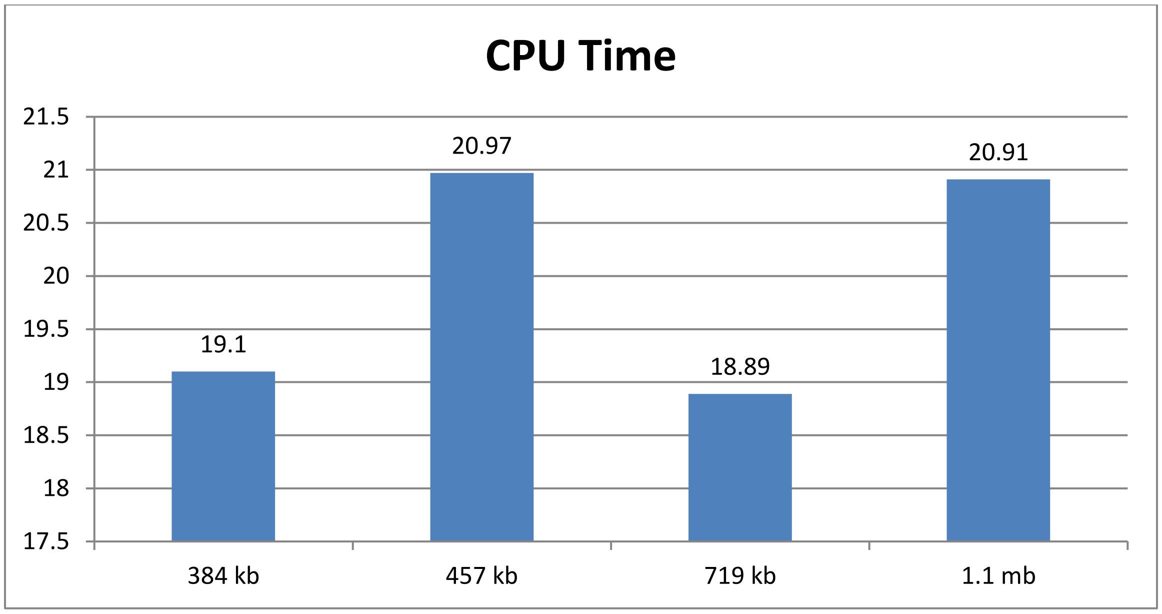


Figure 4.3.2.1: CPU Time for Keccak for Audio files

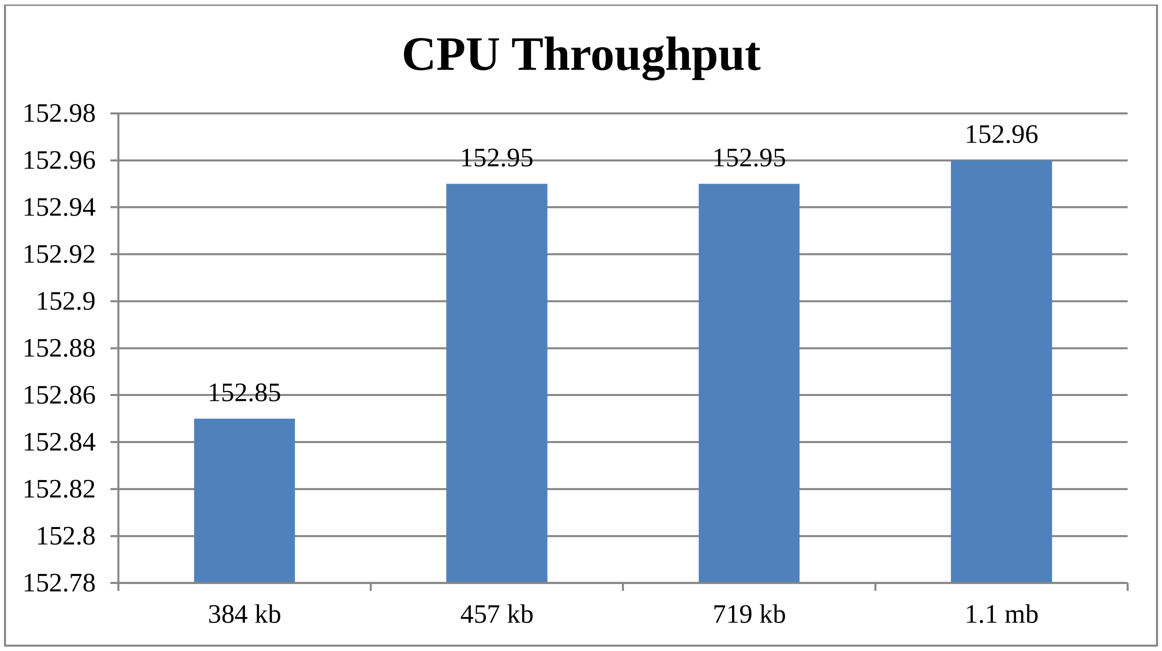


Figure 4.3.2.2: CPU Throughput for Keccak for Audio files

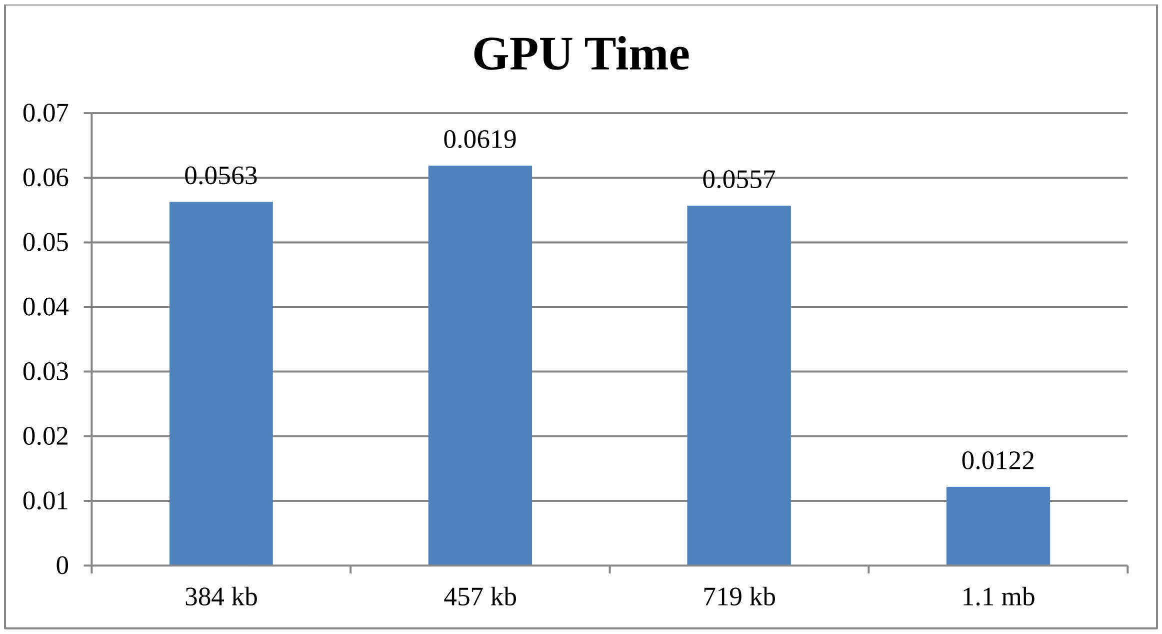


Figure 4.3.2.3: GPU Time for Keccak for Audio files

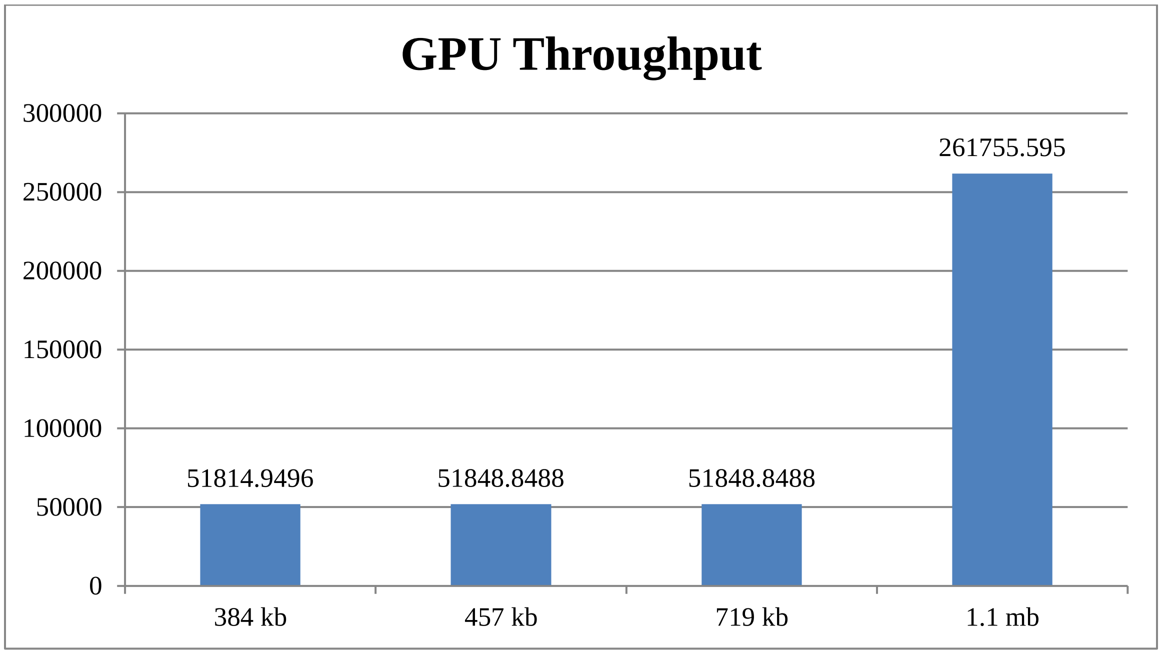


Figure 4.3.2.4: GPU Throughput for Keccak for Audio files

4.3.3 Video Dataset

Table 4.3.3 shows results for Keccak on Video Dataset. Highest CPU time was recorded for the file size of 960 kb i.e., 8.88 bytes per sec and lowest was recorded for file size of 99 kb i.e., 7.57 bytes per sec. Whereas, highest CPU Throughput was recorded for the file size of 99 kb i.e., 330.74 bytes per cycle and lowest was recorded for file size of 960 kb i.e., 327.34 bytes per cycle.

Similarly, highest GPU Time was recorded for the file size of 141 kb i.e., 0.0132 bytes per sec and lowest was recorded for file size 769 kb i.e., 0.0051 bytes per sec. Whereas, highest GPU Throughput was recorded for file size of 769 kb i.e., 565385.9245 bytes per cycle and lowest was recorded for file size of 141 kb i.e., 195378.2991 bytes per cycle.

Table 4.3.3: Test results for Keccak on Video files

Video File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
99 kb	7.57	330.74	0.0128	195419.659
141 kb	7.79	330.67	0.0132	195378.2991
240 kb	8.61	329.5	0.0093	305292.6939
769 kb	8.72	330.39	0.0051	565385.9245
960 kb	8.88	327.34	0.0052	560166.5563

Following are graphical representations of results of Keccak on Video dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Video data set.

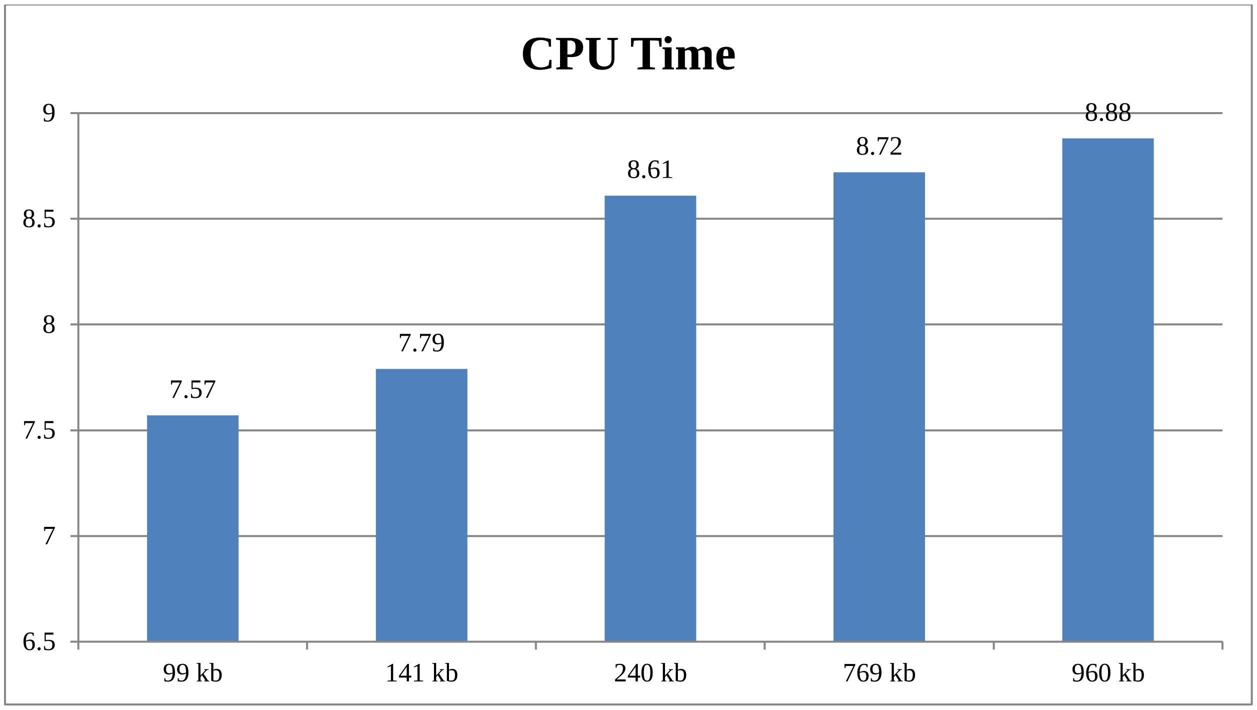


Figure 4.3.3.1: CPU Time for Keccak for Video files

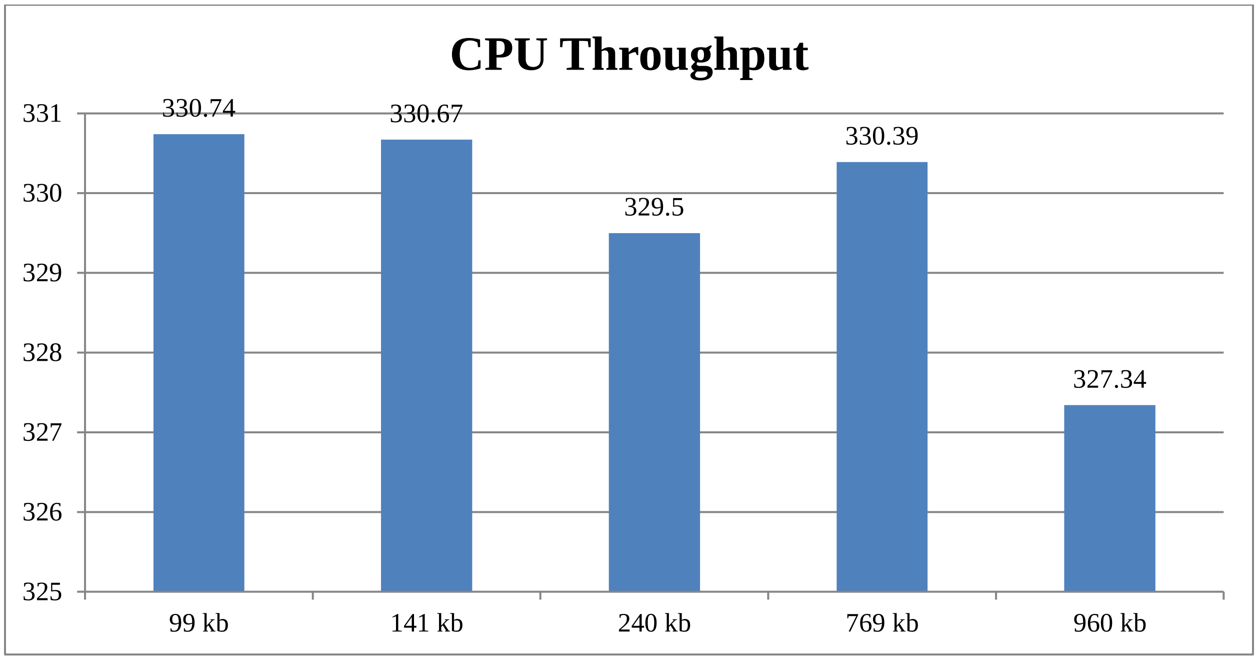


Figure 4.3.3.2: CPU Throughput for Keccak for Video files

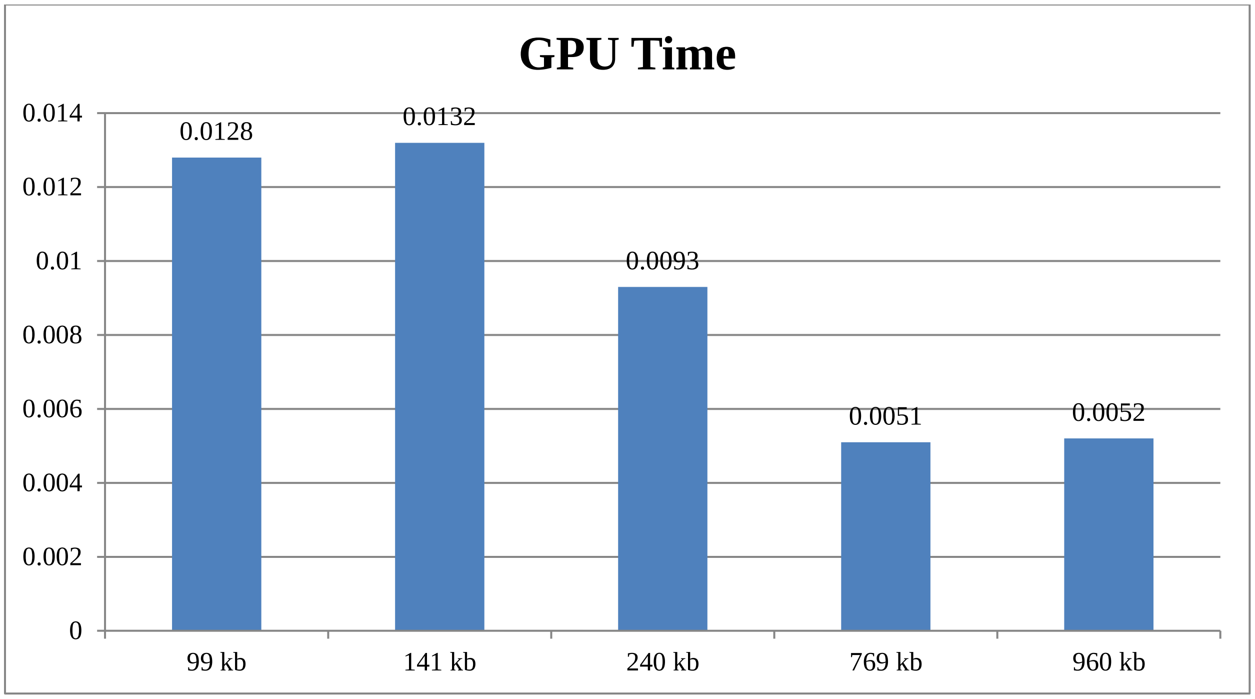


Figure 4.3.3.3: GPU Time for Keccak for Video files

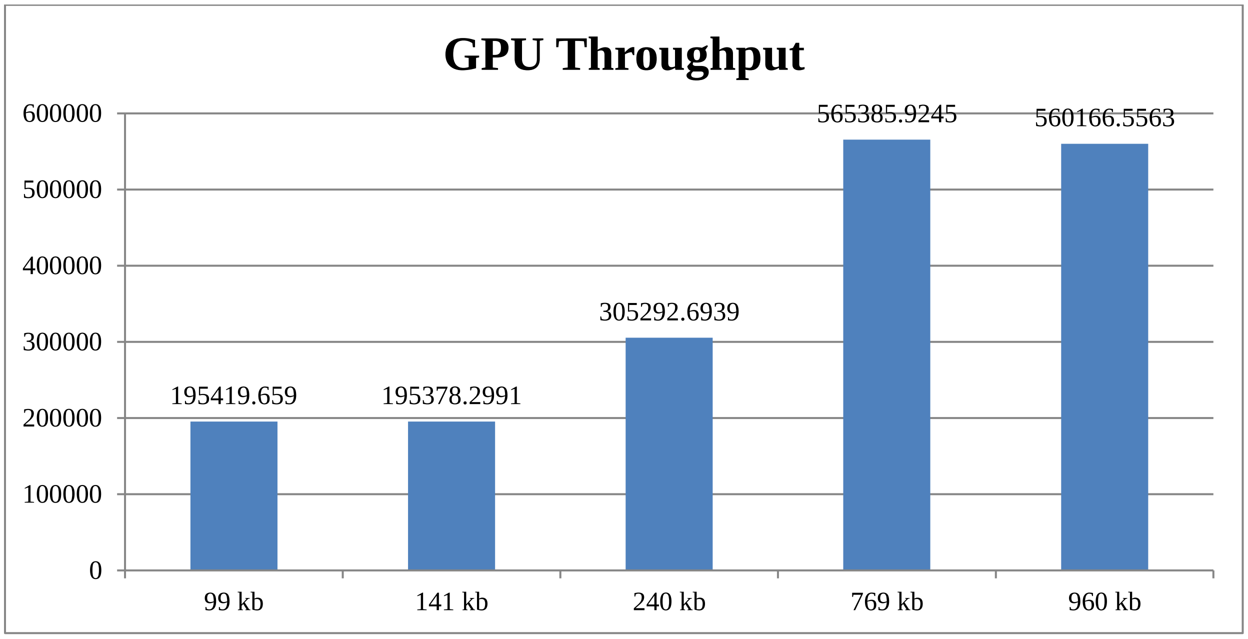


Figure 4.3.3.4: GPU Throughput for Keccak for Video files

4.3.4 Image Dataset

Table 4.3.4depicts results of Keccak on Image Dataset. Highest CPU Time was observed on file size of 960 kb i.e., 8.94 bytes per sec, and lowest CPU Time was observed on file size of 145 kb i.e., 8.67 bytes per sec. Highest CPU Throughput was observed on file size of 769 kb i.e., 331.82 bytes per cycle and lowest CPU Throughput was observed on file size of 145 kb i.e., 326.9 bytes per cycle.

Similarly, highest GPU Time was observed on file size of 80 kb i.e 0.0148 bytes per sec and lowest was observed on file size of 960 kb i.e., 0.0064 bytes per sec. Highest GPU Throughput was recorded on 769 kb file i.e., 553546.985 bytes per cycle and lowest was recorded on file size of 80 kb i.e., 195401.9333 bytes per cycle.

Table 4.3.4: Test results for Keccak on Image files

Image File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
80 kb	8.76	330.71	0.0148	195401.9333
107 kb	8.7	331.4	0.0147	195809.6238
145 kb	8.67	326.9	0.0094	302883.7075
769 kb	8.75	331.82	0.0074	553546.985
960 kb	8.94	324.74	0.0064	525398.6324

Following are graphical representations of results of Keccak on Image dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Image data set.

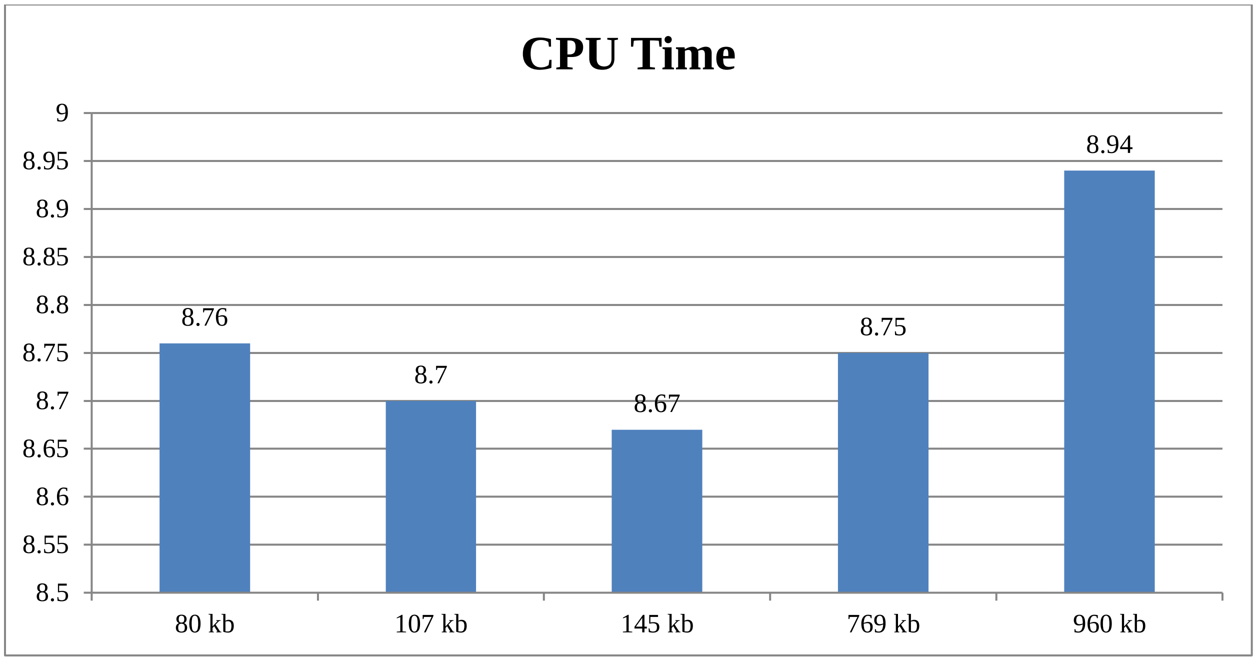


Figure 4.3.4.1: CPU Time for Keccak for Image files

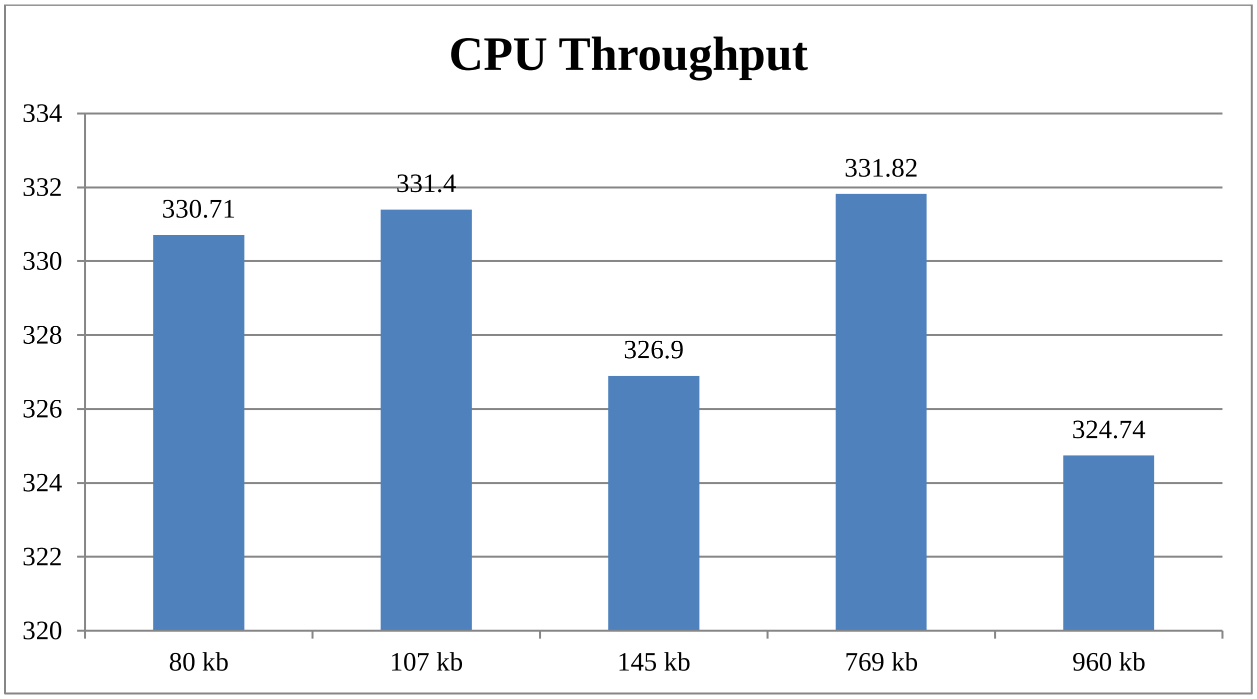


Figure 4.3.4.2: CPU Throughput for Keccak for Image files

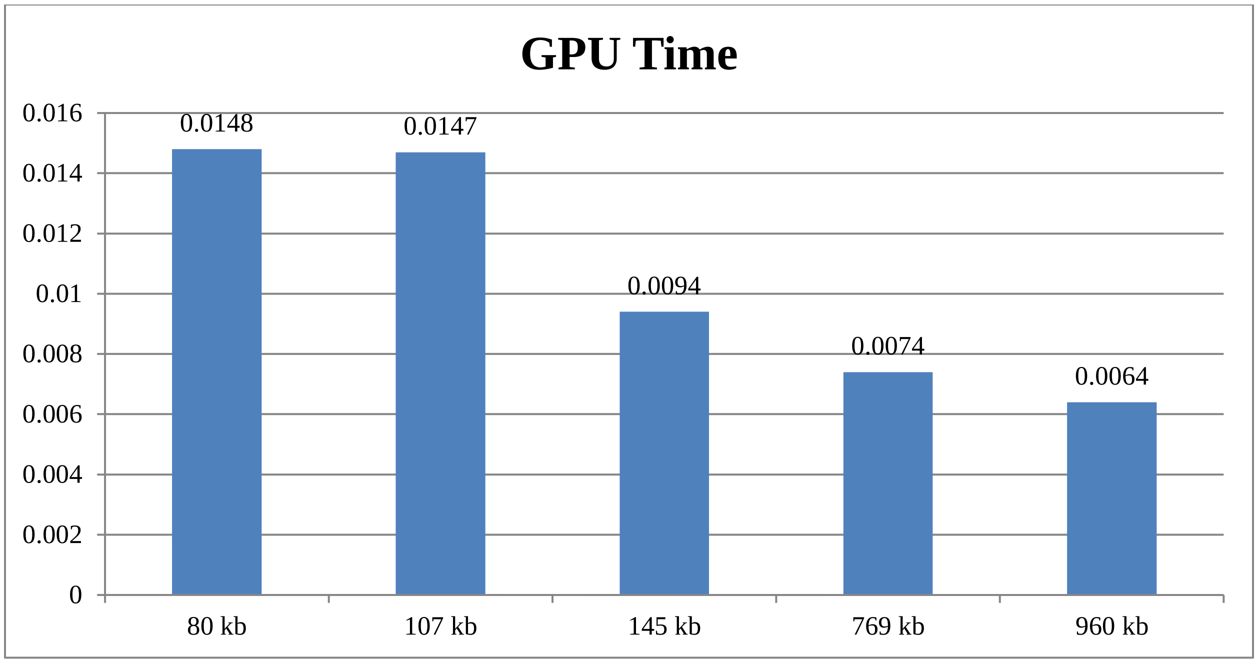


Figure 4.3.4.3: GPU Time for Keccak for Image files

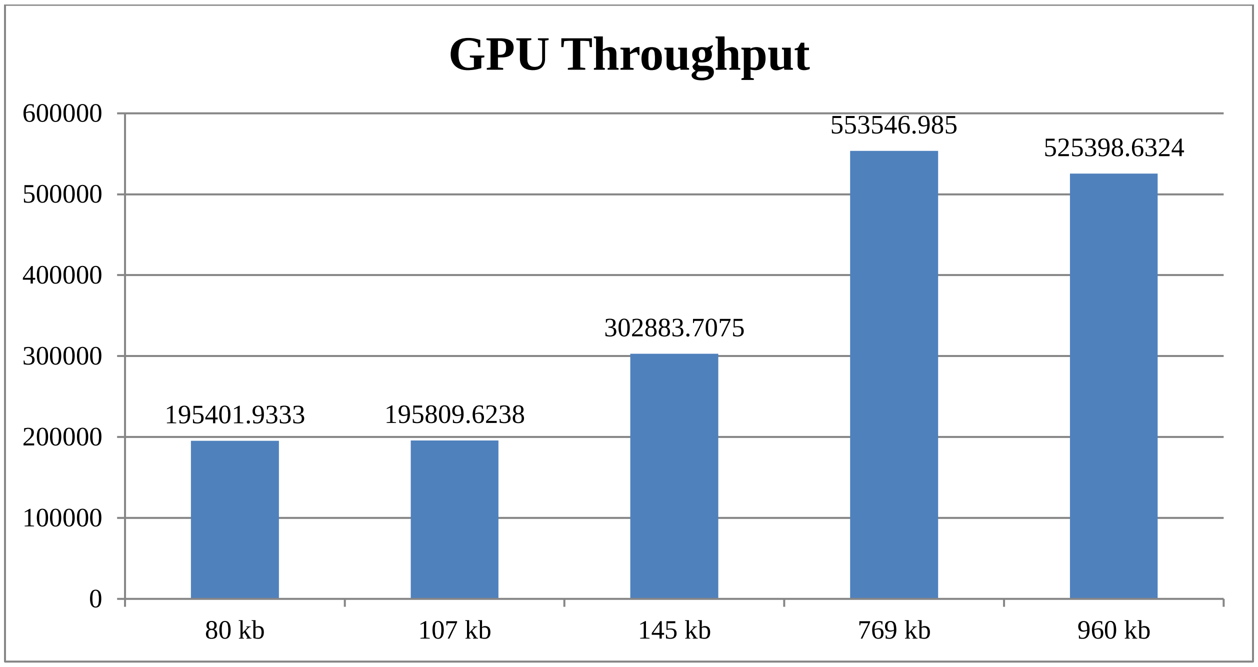


Figure 4.3.4.4: GPU Throughput for Keccak for Image files

4.4 Experimental Results for Skein

4.4.1 Text Dataset

Table 4.4.1 shows experimental results of Skeinfor Text Dataset. The highest CPU Time (Bytes per Second) is found for the file size of 977 kb i.e., 20.54 bytes per sec and Lowest CPU Time (Bytes per Second) is observed for the file size of 319 kb i.e., 19.45 bytes per sec. The highest CPU Throughput (Bytes per Cycle) is found for file size of 963 kb i.e., 76.52 bytes per cycle and Lowest CPU Throughput (Bytes per Cycle) is observed for the file size of 538 kb i.e. 73.25 bytes per cycle.

Similarly, highest GPU Time (Bytes per Second) is found for the file size of 94 kb i.e., 0.0337 bytes per sec and Lowest GPU Time is found for the file size of 963 kb i.e., 0.0152 bytes per sec. Highest GPU Throughput (Bytes per Cycle) is 125648.987 bytes per cycle for file size of 1.1 mb and lowest GPU Throughput is observed on the file size of 94 kb i.e., 45182.7457 bytes per cycle.Test results for Skein on CPU and GPU for Text File Dataset is appended below.

Table 4.4.1: Test results for Skein on Text files

Text File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
94 kb	19.92	76.47	0.0337	45182.7457
319 kb	19.45	75.41	0.0321	99777.445
538 kb	19.56	73.25	0.0252	98754.254
963 kb	20.35	76.52	0.0152	121245.2642
977 kb	20.54	74.85	0.0198	123548.594
1.1 mb	20.5	75.75	0.0185	125648.987

Following are graphical representations of results of Skein on Text dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Text data set.

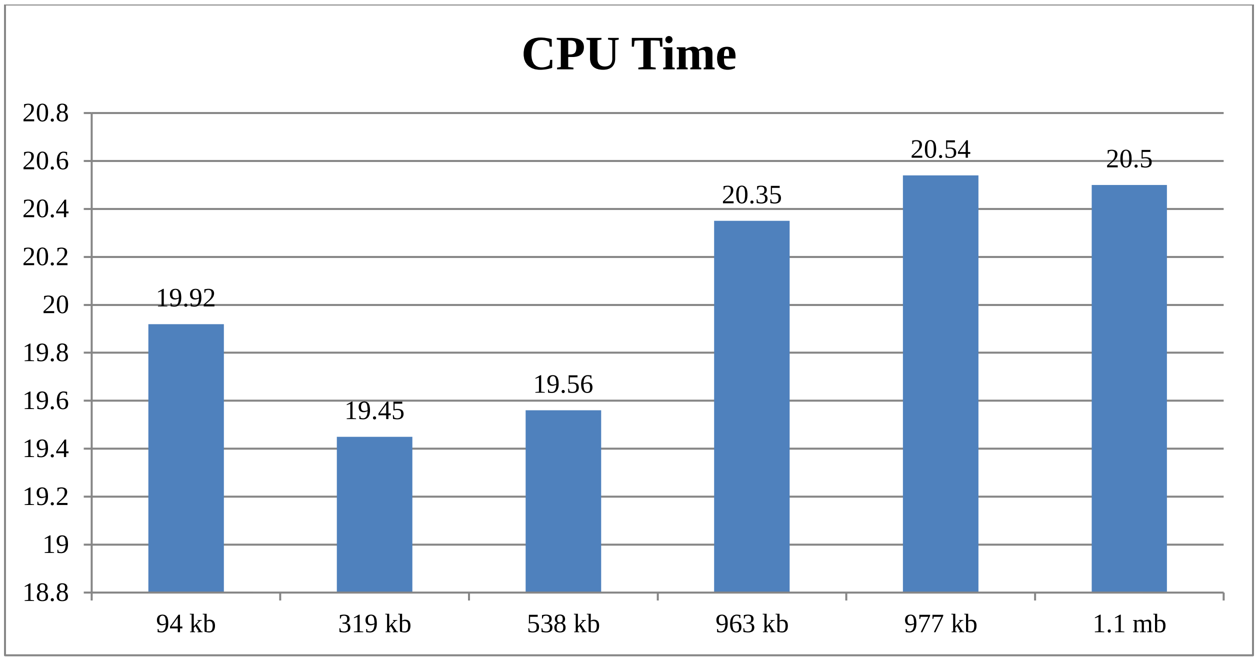


Figure 4.4.1.1: CPU Time for Skein for Text files

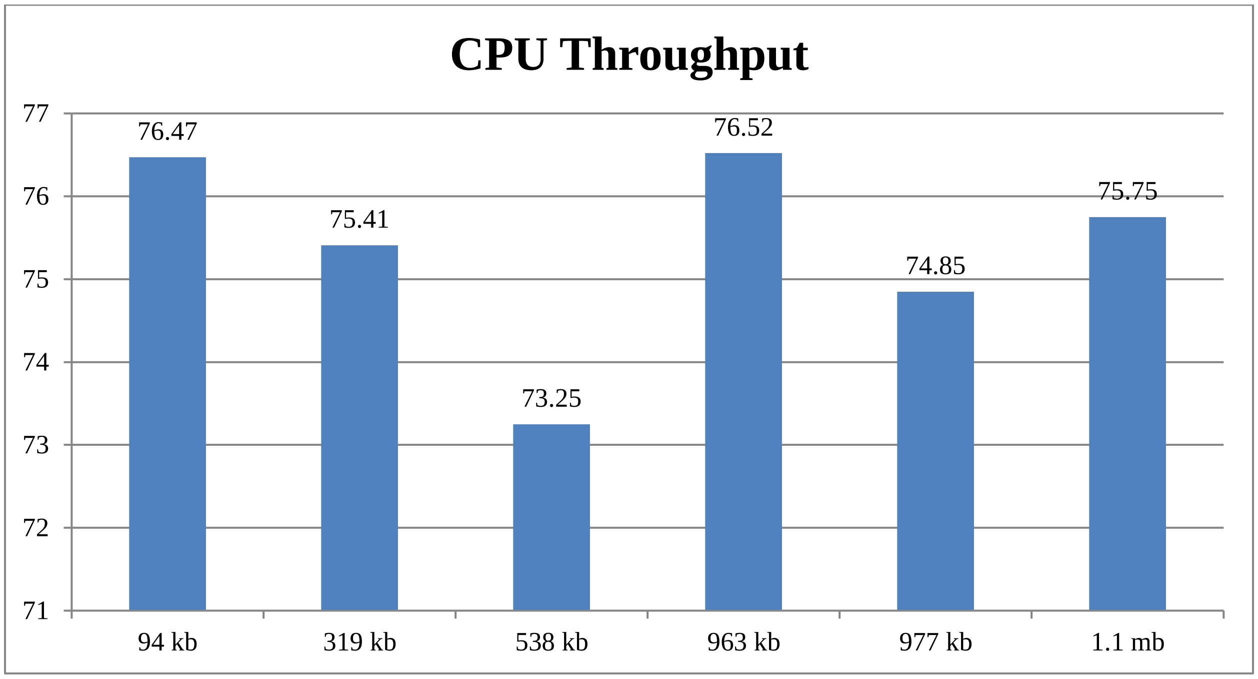


Figure 4.4.1.2: CPU Throughput for Skein for Text files

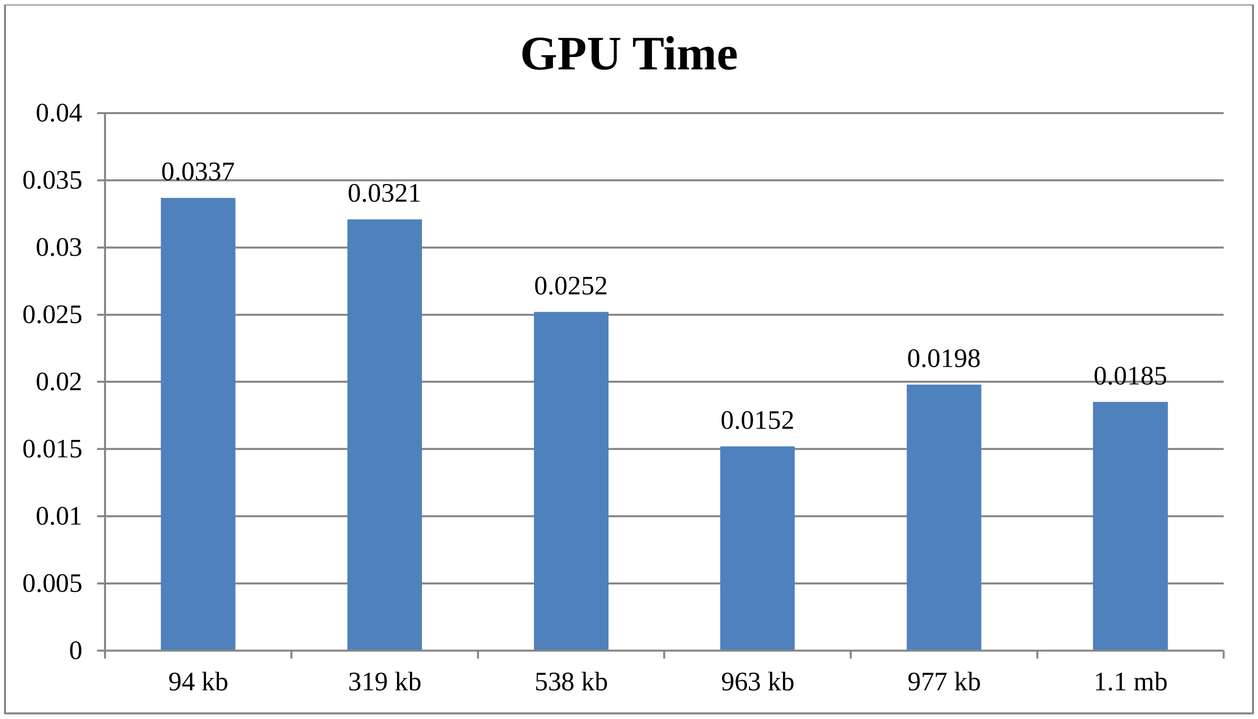


Figure 4.4.1.3: GPU Time for Skein for Text files

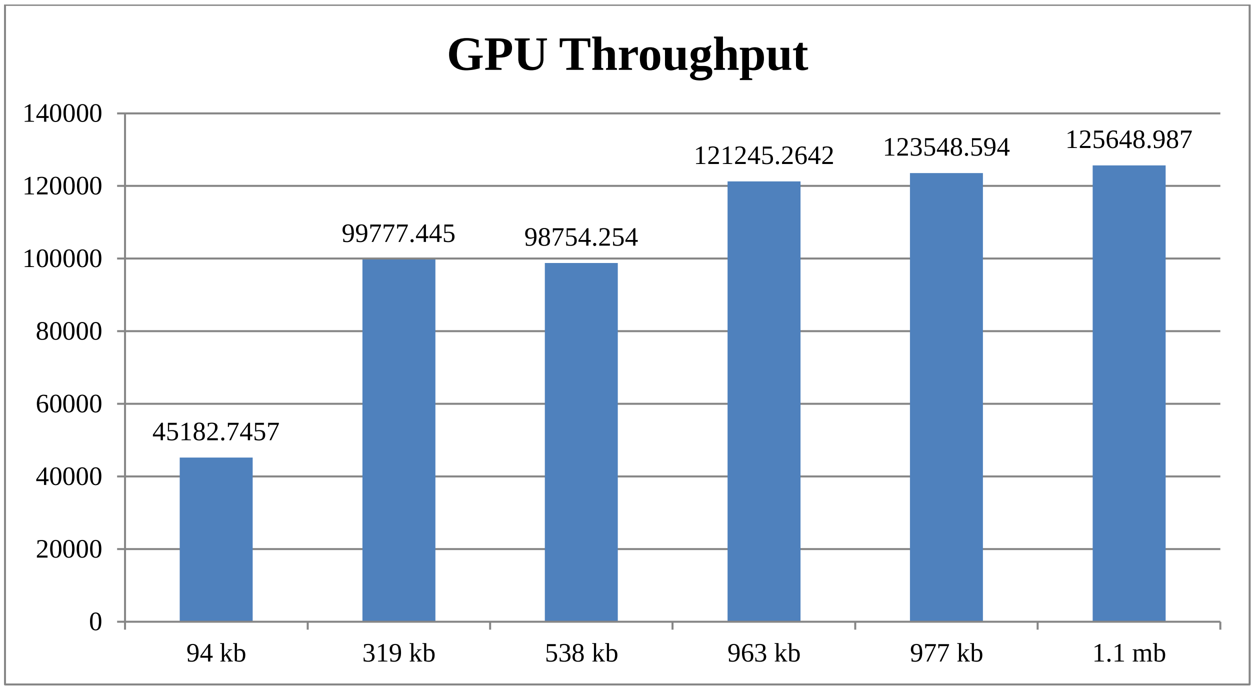


Figure 4.4.1.4: GPU Throughput for Skein for Text files

4.4.3 Audio Dataset

The following table 4.4.2 shows experimental results of Skeinfor Audio Dataset. The highest CPU Time is found for the file size of 1.1 Mb i.e., 20.05 bytes per sec, and lowest CPU Time was observed on file of size 384 kb i.e., 19.83 bytes per sec. Whereas, highest CPU Throughput (Bytes per Cycle) was found for the file size of 719 kb i.e., 76.56 bytes per cycle, and lowest CPU Throughput was observed for the file size of 38.4 kb i.e., 76.42 bytes per cycle.

Similarly, highest GPU Time was found for the file size of 457 kb i.e., 0.0591 bytes per sec and the lowest GPU Time is for the file size of 1.11 mb i.e., 0.0117 bytes per sec. Whereas, highest GPU Throughput was found for the file size of 1.11 mb i.e., 130843.5721 bytes per cycle and lowest GPU Throughput was observed for the file size of 384 kb i.e., 25905.7798 bytes per cycle.

Table 4.4.2: Test results for Skein on Audio files

Audio File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
384 kb	19.83	76.42	0.0585	25905.7798
457 kb	20.02	76.46	0.0591	25919.3395
719 kb	19.92	76.56	0.0588	25919.3395
1.1 mb	20.05	76.49	0.0117	130843.5721

Following are graphical representations of results of Skein on Audio dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates’ performances for Audio data set.

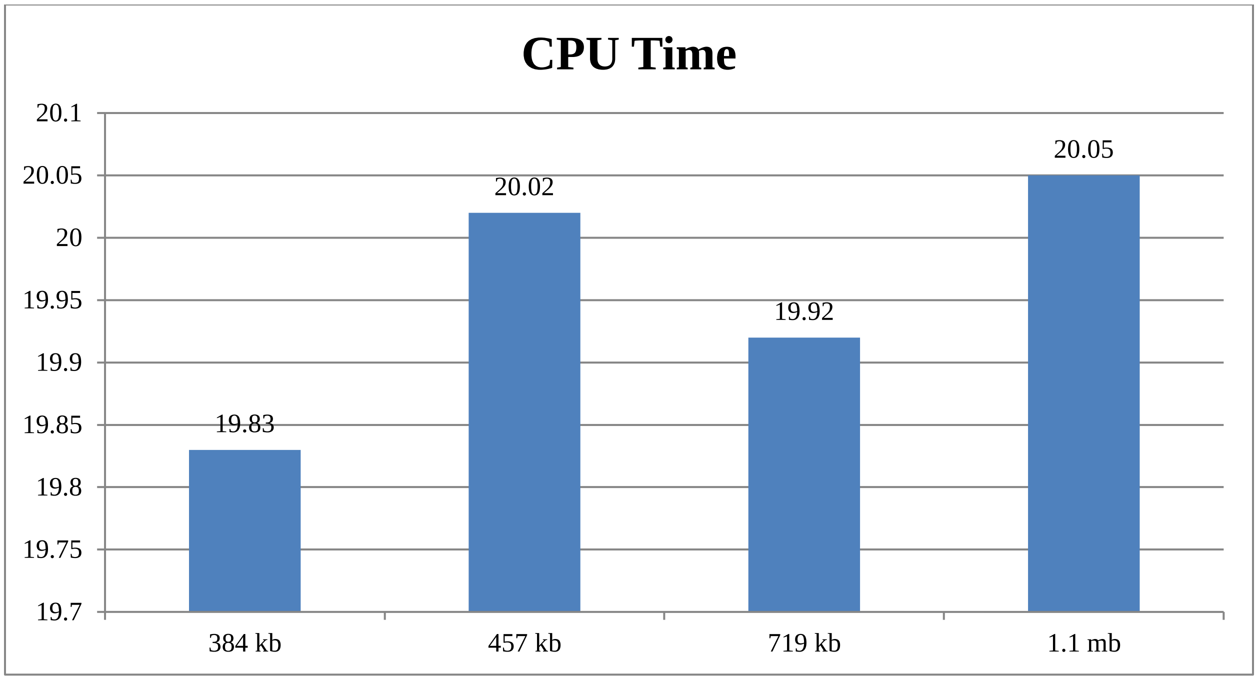


Figure 4.4.2.1: CPU Time for Skein for Audio files

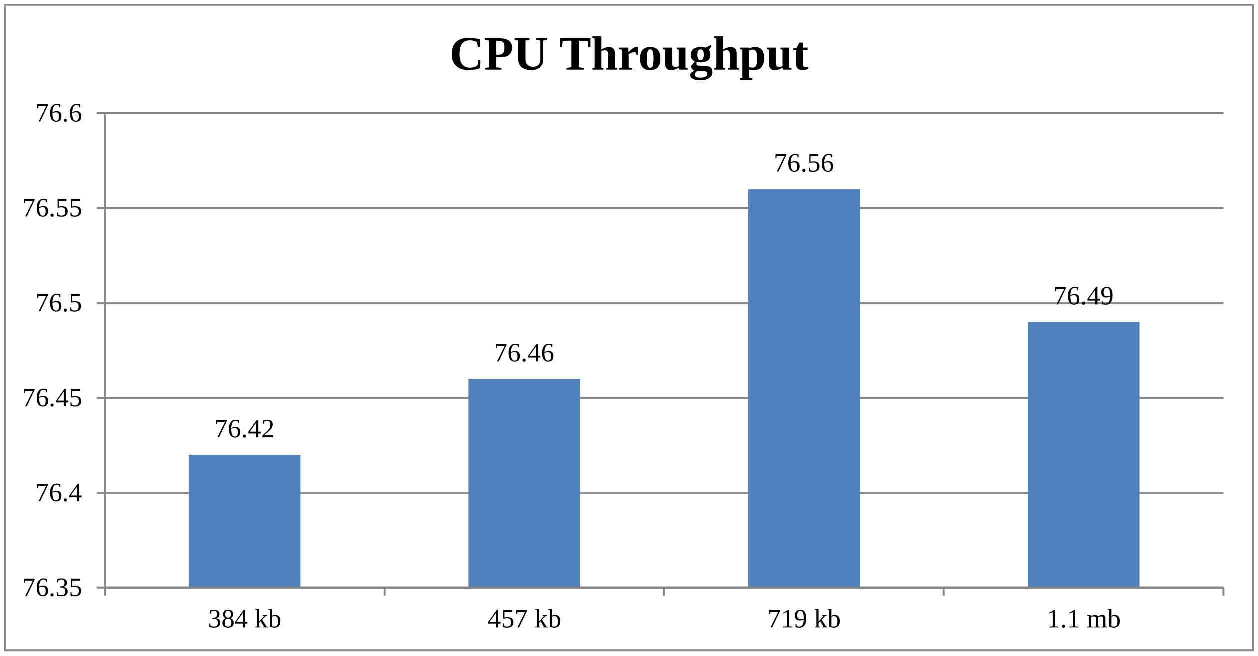


Figure 4.4.2.2: CPU Throughput for Skein for Audio files

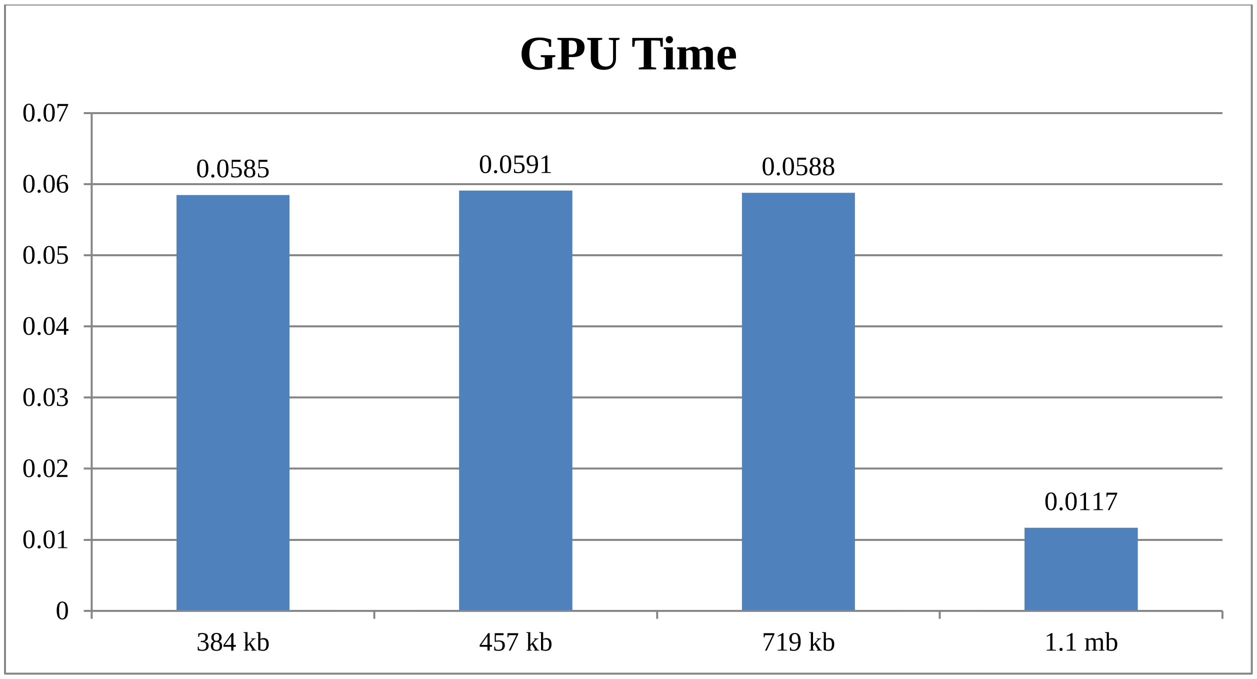


Figure 4.4.2.3: GPU Time for Skein for Audio files

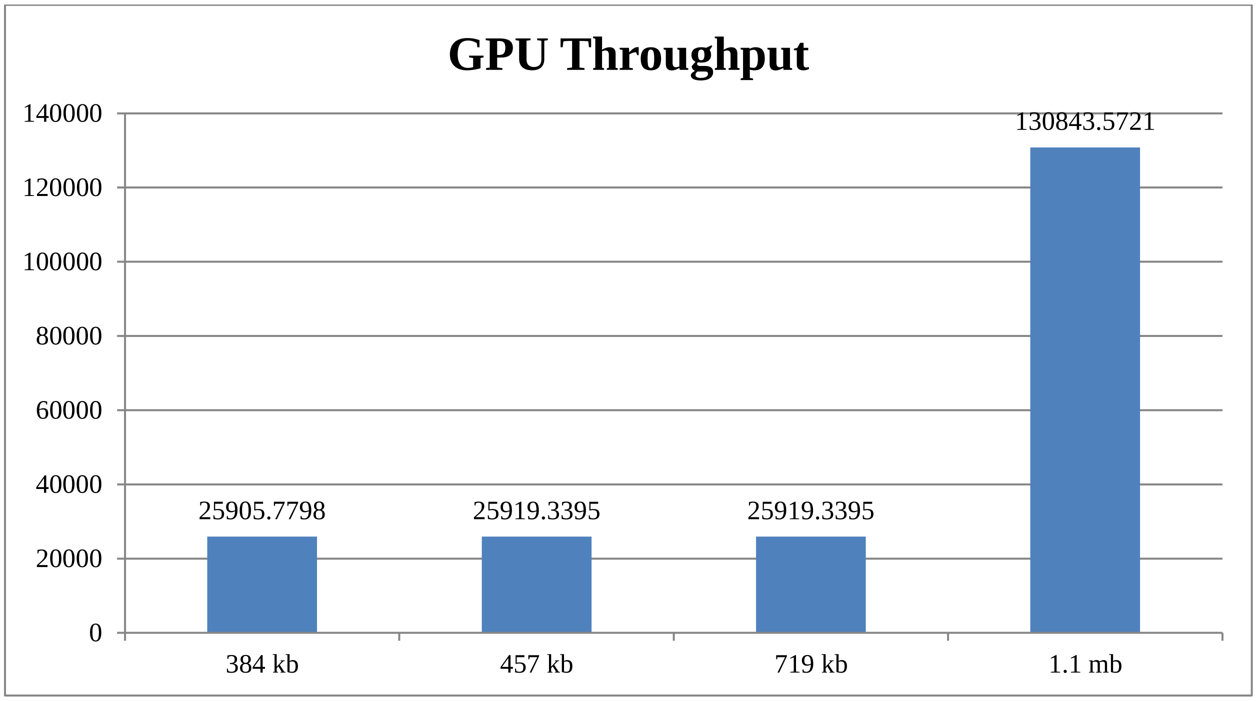


Figure 4.4.2.4: GPU Throughput for Skein for Audio files

4.4.3 Video Dataset

Table 4.4.3 shows results for Skeinon Video Dataset. Highest CPU time was recorded for the file size of 960 kb i.e., 8.65 bytes per sec and lowest was recorded for file size of 99 kb i.e., 7.02 bytes per sec. Whereas, highest CPU Throughput was recorded for the file size of 240 kb i.e., 164.22 bytes per cycle and lowest was recorded for file size of 769 kb i.e., 163.03 bytes per cycle.

Similarly, highest GPU Time was recorded for the file size of 141 kb i.e., 0.0127 bytes per sec and lowest was recorded for file size 769 kb i.e., 0.0049 bytes per sec. Whereas, highest GPU Throughput was recorded for file size of 960 kb i.e., 279347.4328 bytes per cycle and lowest was recorded for file size of 99 kb i.e., 96404.0381 bytes per cycle.

Table 4.4.3: Test results for Skein on Video files

Video File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
99 kb	7.02	163.16	0.0119	96404.0381
141 kb	7.49	163.17	0.0127	96409.9467
240 kb	8.33	164.22	0.009	152155.2843
769 kb	8.41	163.03	0.0049	278988.0665
960 kb	8.65	163.24	0.0051	279347.4328

Following are graphical representations of results of Skein on Video dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates’ performances for Video data set.

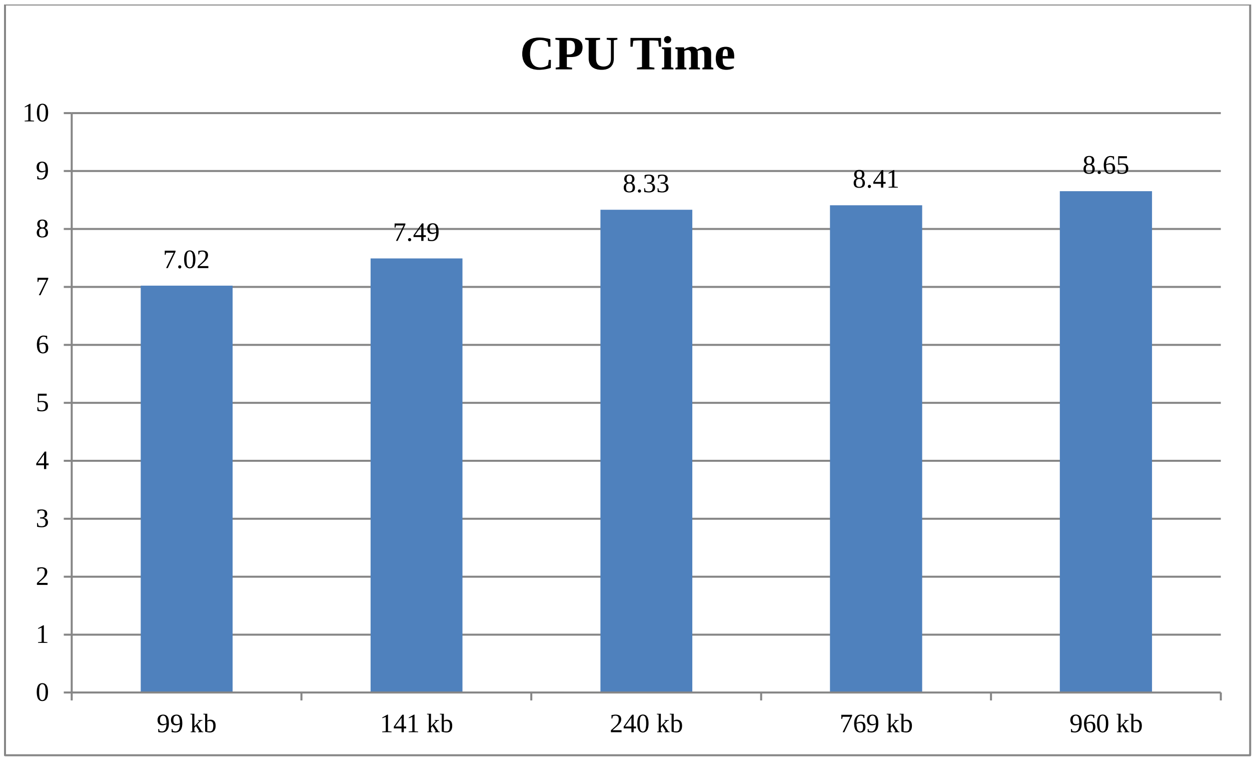


Figure 4.4.3.1: CPU Time for Skein for Video files

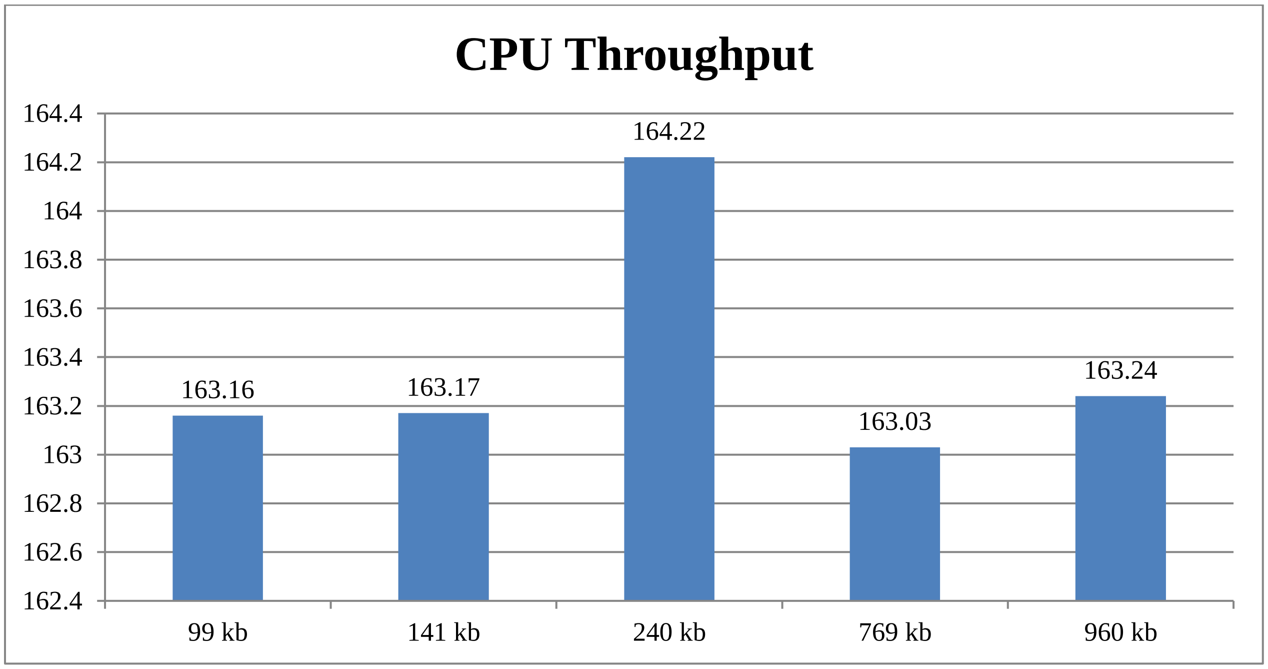


Figure 4.4.3.2: CPU Throughput for Skein for Video files

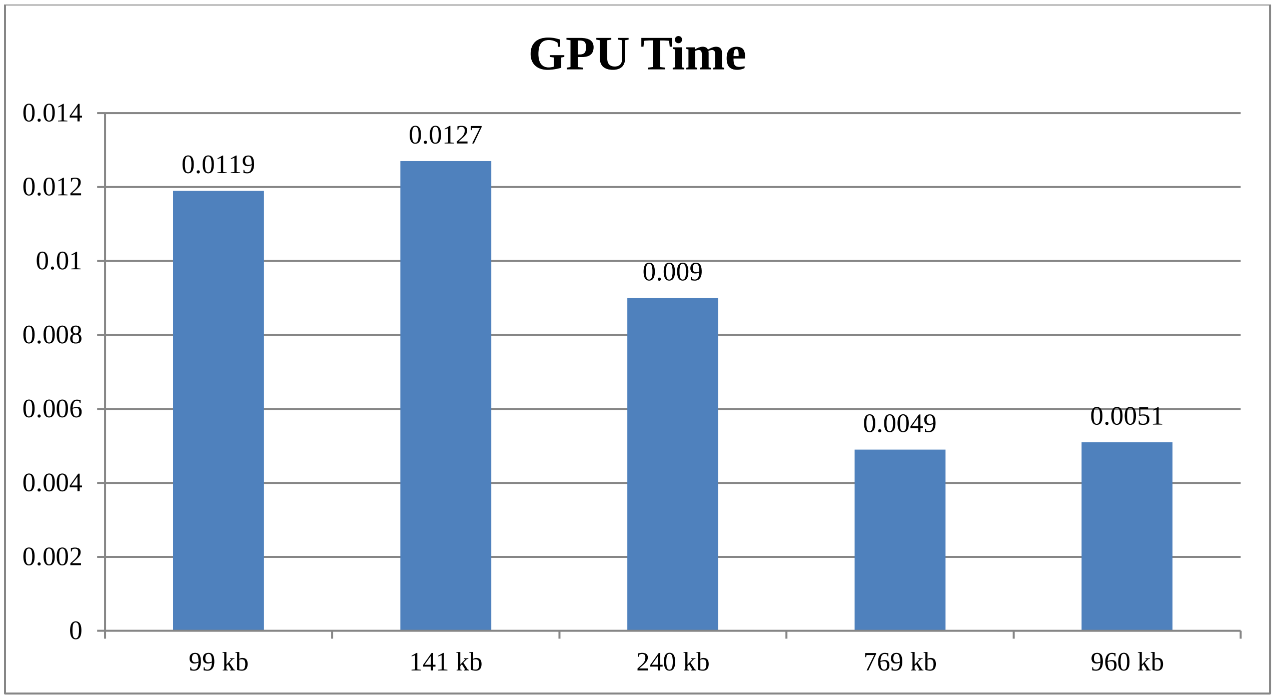


Figure 4.4.3.3: GPU Time for Skein for Video files

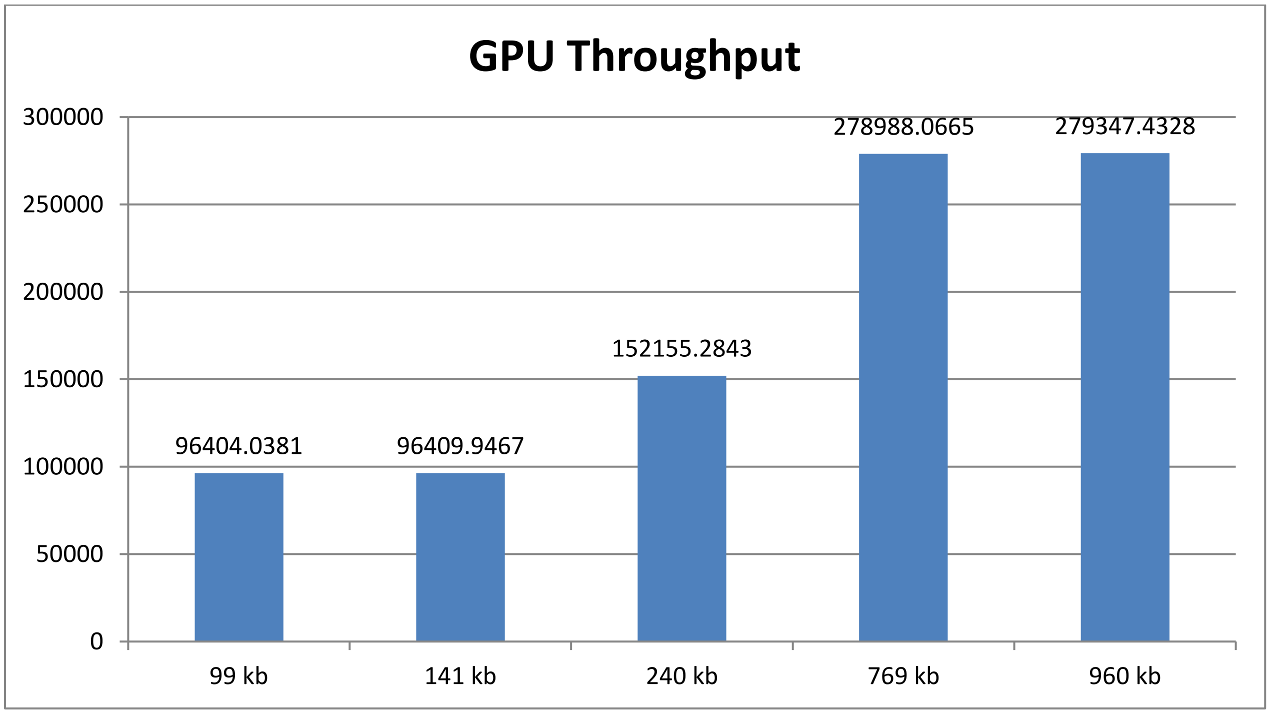


Figure 4.4.3.4: GPU Throughput for Skein for Video files

4.4.4 *Image Dataset*

Table 4.4.4depicts results of Skeinon Image Dataset. Highest CPU Time was observed on file size of 960 kb i.e., 8.52 bytes per second, and lowest CPU Time was observed on file size of 145 kb i.e., 7.32 bytes per second. Highest CPU Throughput was observed on file size of 145 kb i.e., 163.51 bytes per cycle and lowest CPU Throughput was observed on file size of 769 kb i.e., 163.03 bytes per cycle.

Similarly, highest GPU Time was observed on file size of 80 kb i.e 0.0132 bytes per sec and lowest was observed on file size of 769 kb i.e., 0.0049 bytes per sec. Highest GPU Throughput was recorded on 960 kb file i.e., 278988.0665 bytes per cycle and lowest was recorded on file size of 107 kb i.e., 96398.1295 bytes per cycle.

Table 4.4.4: Test results for Skein on Image files

Image File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
80 kb	7.81	163.37	0.0132	96528.1178
107 kb	7.37	163.15	0.0125	96398.1295
145 kb	7.32	163.51	0.0079	151497.4457
769 kb	8.41	163.03	0.0049	278988.0665
960 kb	8.52	163.24	0.0055	279347.4328

Following are graphical representations of results of Skein on Image dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Image data set.

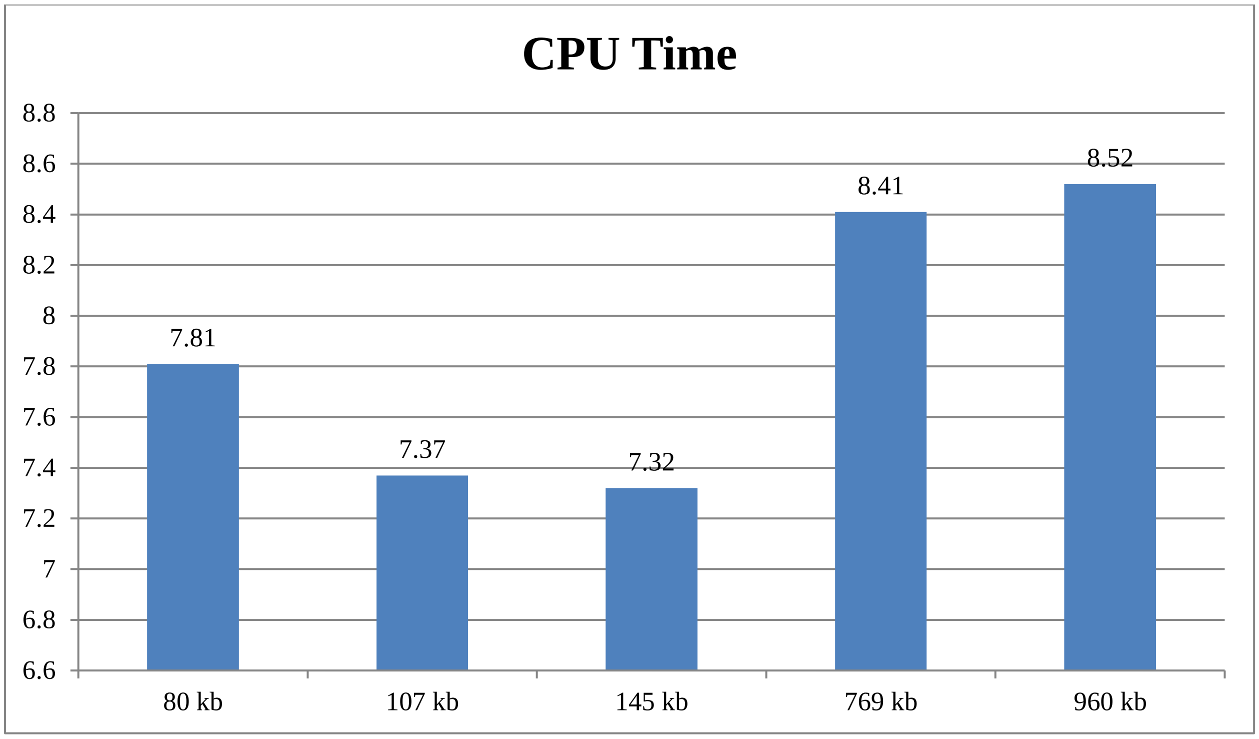


Figure 4.4.4.1: CPU Time for Skein for Image files

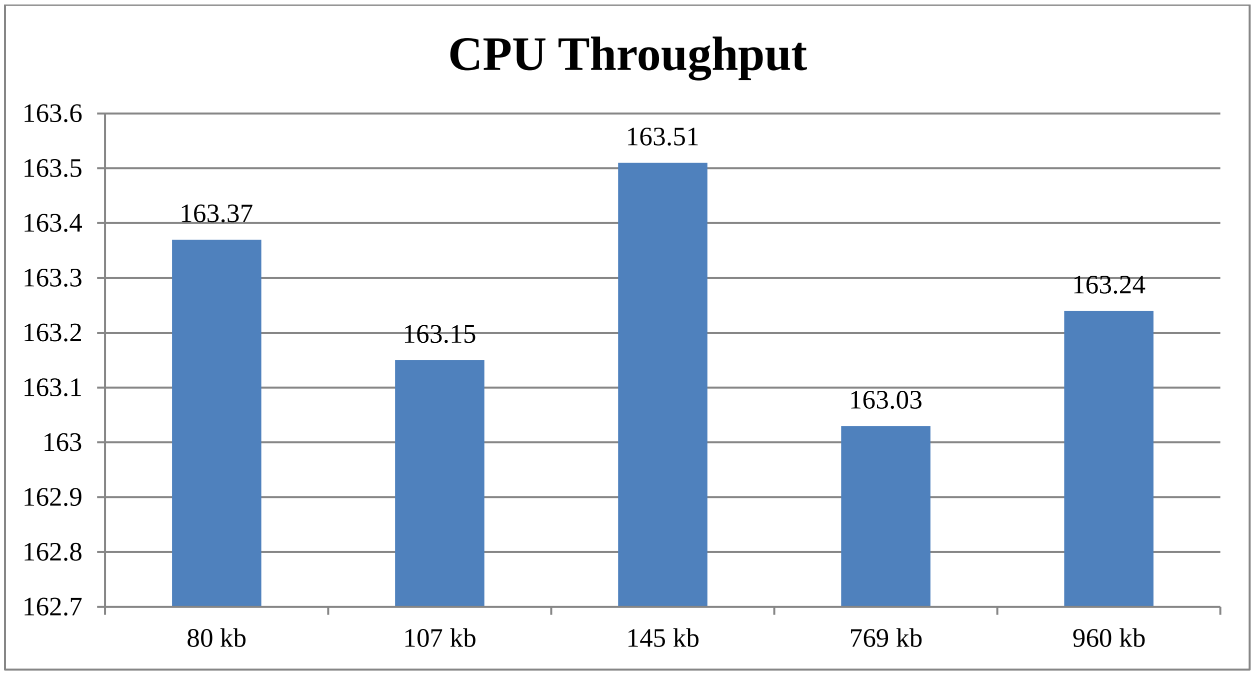


Figure 4.4.4.2: CPU Throughput for Skein for Image files

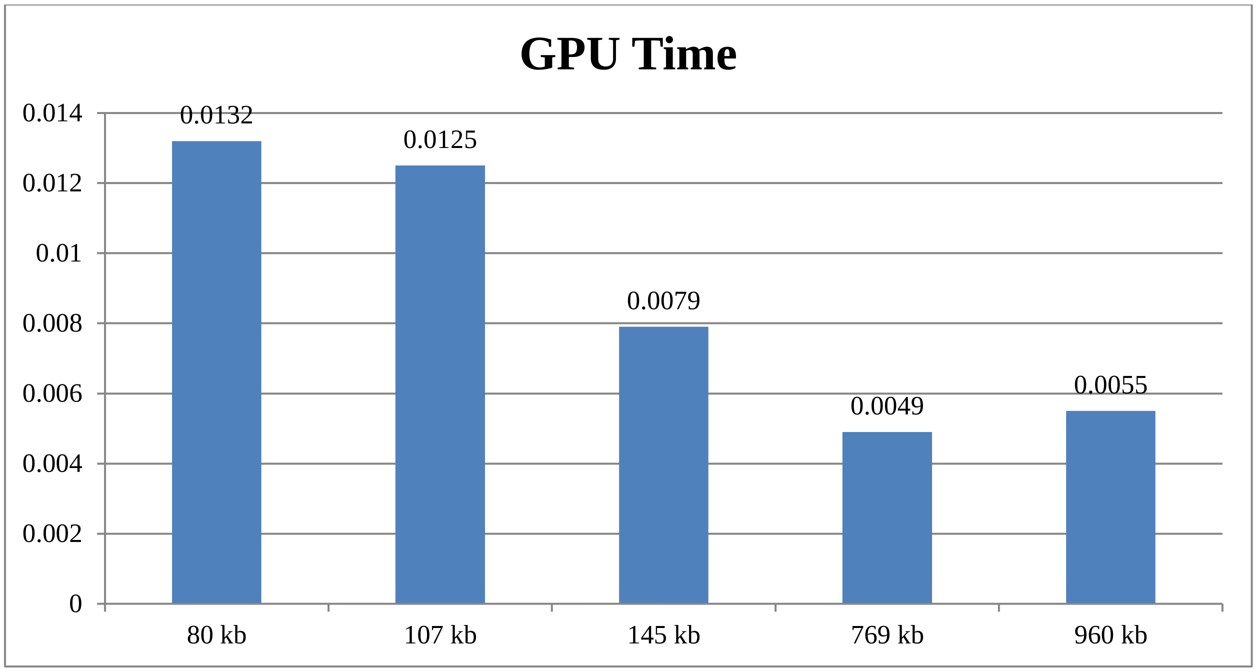


Figure 4.4.4.3: GPU Time for Skein for Image files

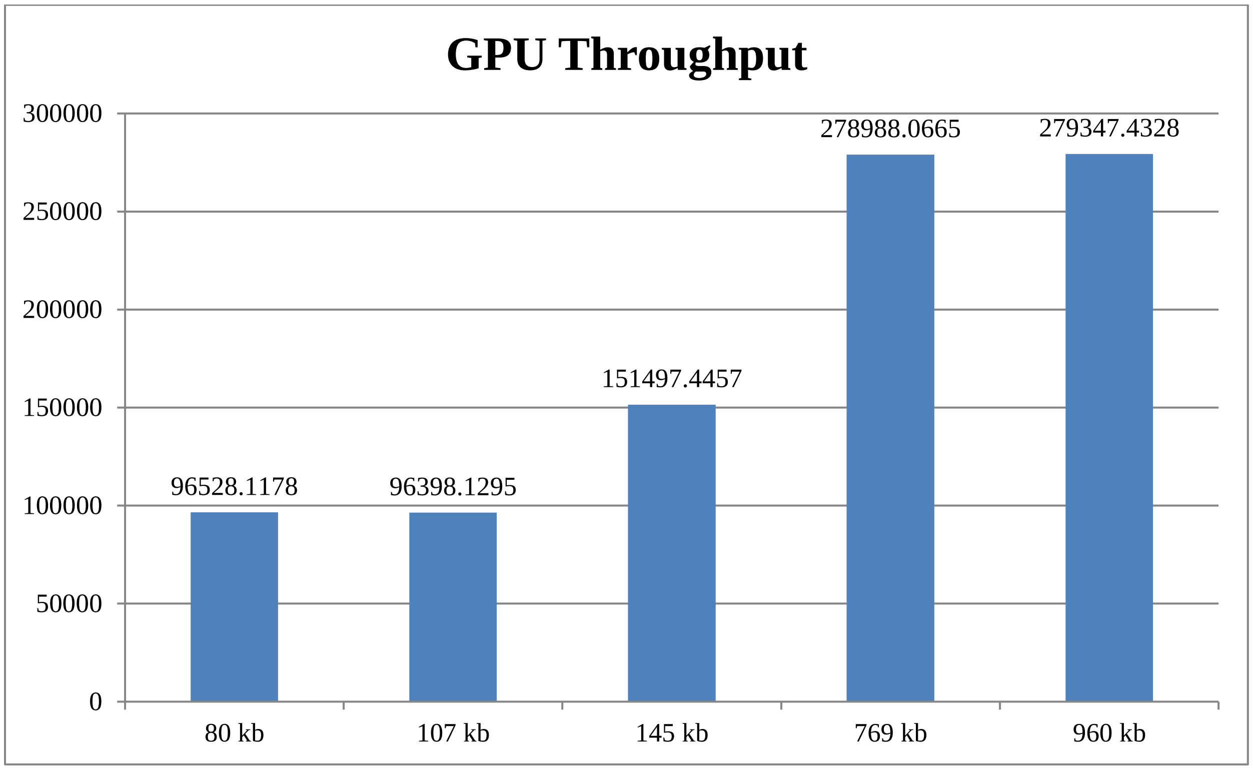


Figure 4.4.4.4: GPU Throughput for Skein for Image files

4.5 Experimental Results for Blake

4.5.1 Text Dataset

Table 4.5.1 shows experimental results of Blakefor Text Dataset. The highest CPU Time is found for the file size of 1.1 mb i.e., 20.19 bytes per sec and Lowest CPU Time (Bytes per Second) is observed for the file size of 94 kb i.e., 19.26 bytes per sec. The highest CPU Throughput (Bytes per Cycle) is found for file size of 94 kb i.e., 79.24 bytes per cycle and Lowest CPU Throughput (Bytes per Cycle) is observed 72.63 bytes per cycle for file size of 963 kb.

Similarly, highest GPU Time (Bytes per Second) is found for the file size of 963 kb i.e., 0.0411 bytes per sec and Lowest GPU Time is found for the file size of 1.1 mb i.e., 0.0215 bytes per sec. Highest GPU Throughput (Bytes per Cycle) is 124323.64 bytes per cycle for file size of 1.1 mb and lowest GPU Throughput is observed on the file size of 94 kb i.e., 46819.4164 bytes per cycle.Test results for BLAKE on CPU and GPU for Text File Dataset is appended below.

Table 4.5.1: Test results for Blake on Text files

Text File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
94 kb	19.26	79.24	0.0326	46819.4164
319 kb	19.56	72.64	0.0345	99708.1655
538 kb	19.52	72.66	0.0347	99735.6182
963 kb	19.89	72.63	0.0411	124289.4146
977 kb	20.14	72.64	0.0314	124306.5273
1.1 mb	20.19	72.65	0.0215	124323.64

Following are graphical representations of results of Blake on Text dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates’ performances for Text data set.

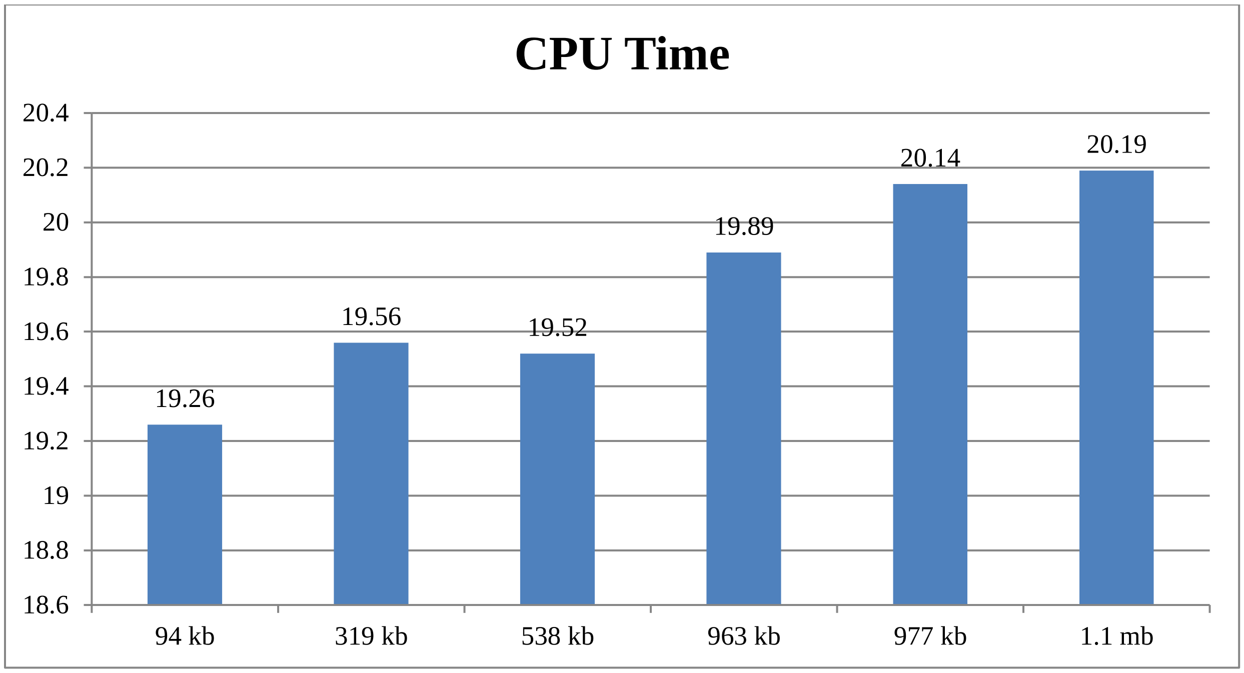


Figure 4.5.1.1: CPU Time for Blake for Text files

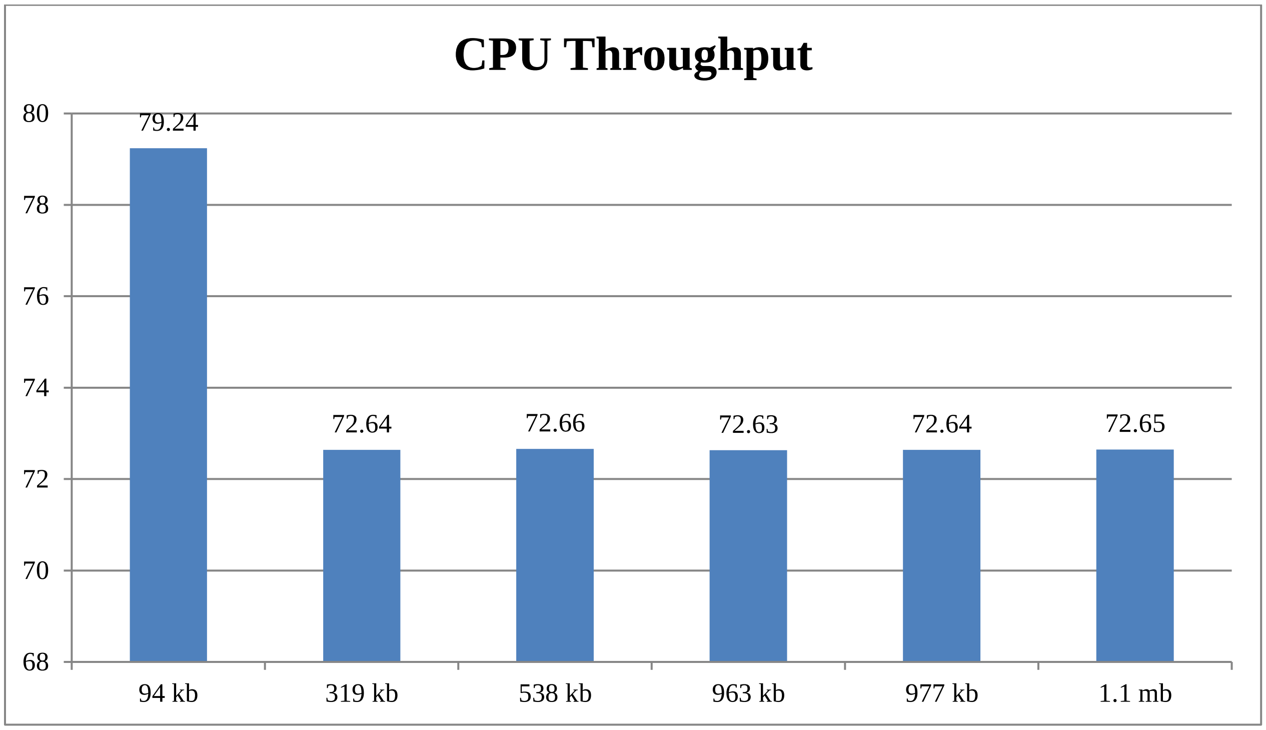


Figure 4.5.1.2: CPU Throughput for Blake for Text files

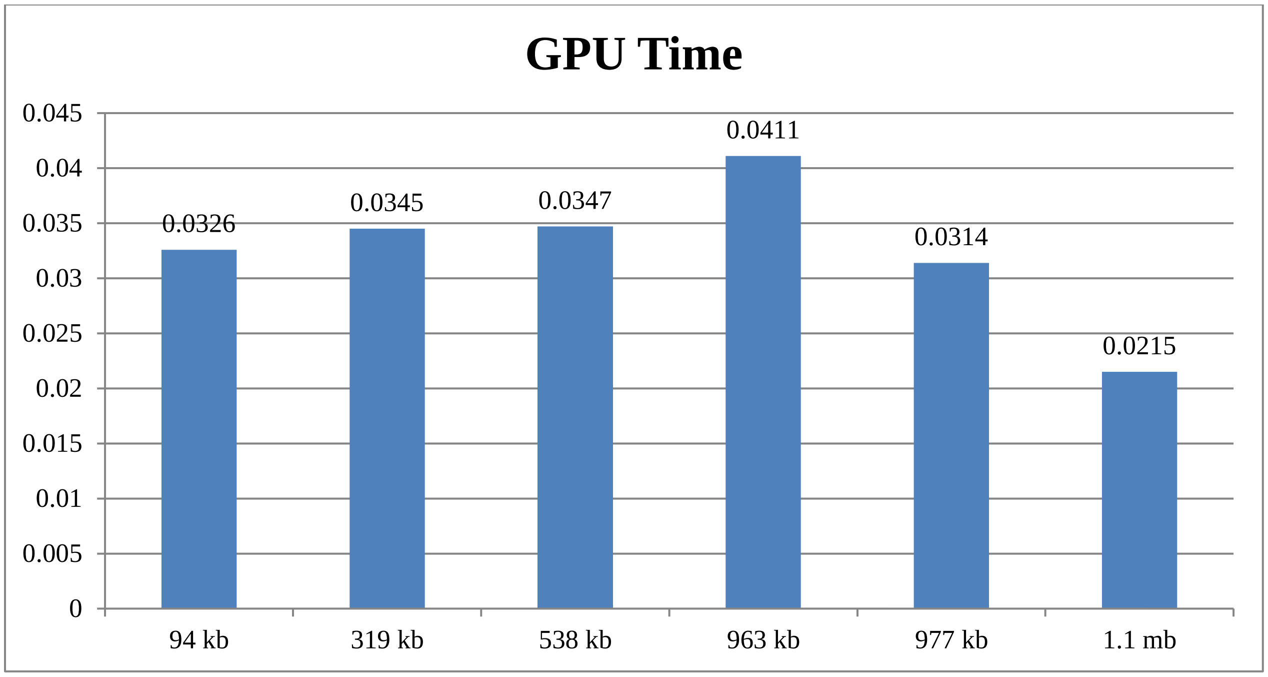


Figure 4.5.1.3: GPU Time for Blake for Text files

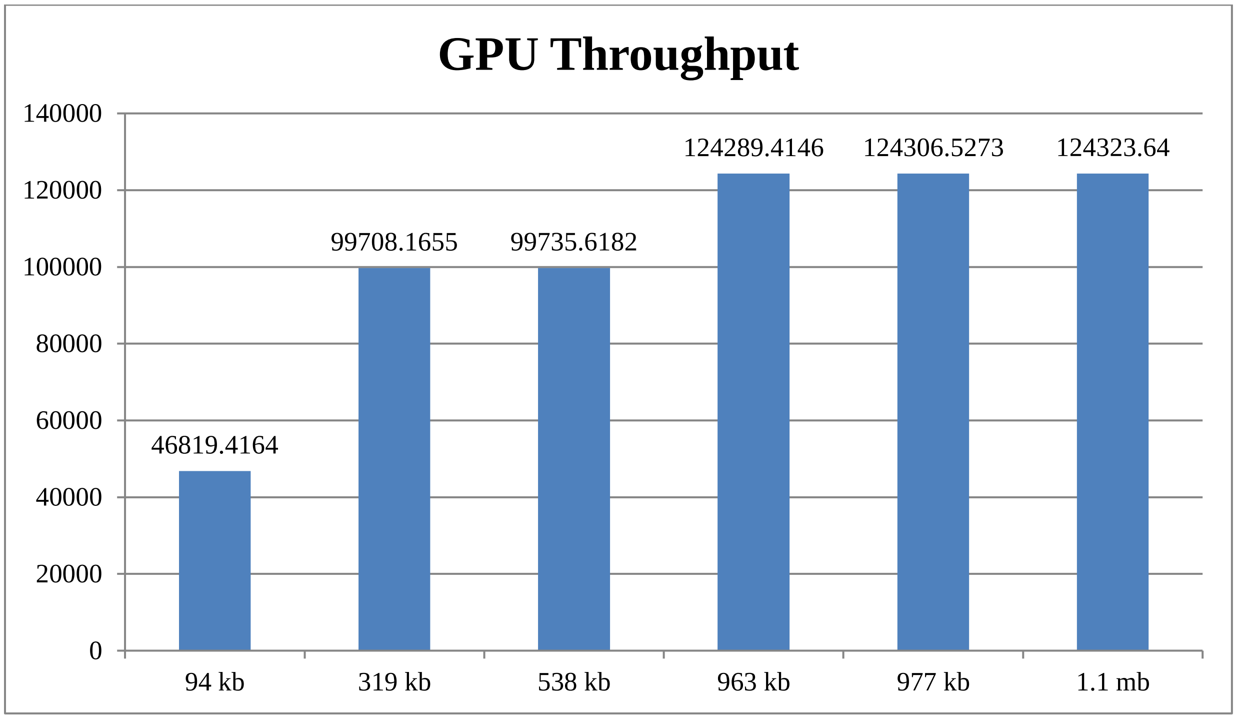


Figure 4.5.1.4: GPU Throughput for Blake for Text files

4.5.2 Audio Dataset

The following table 4.5.2 shows experimental results of Blakefor Audio Dataset. The highest CPU Time is found for the file size of 1.1 mb i.e., 19.6 bytes per sec, and lowest CPU Time was observed on file of size 384 kb i.e., 18.65 bytes per sec. Whereas, highest CPU Throughput (Bytes per Cycle) was found for the file size of 1.11 mb i.e., 80.01 bytes per cycle, and lowest CPU Throughput was observed for the file size of 384 kb i.e., 79.28 bytes per cycle.

Similarly, highest GPU Time was found for the file size of 1.1 mb i.e., 0.0584 bytes per sec and the lowest GPU Time is for the file size of 384 kb i.e., 0.0055 bytes per sec. Whereas, highest GPU Throughput was found for the file size of 1.11 mb i.e., 26945.0955 bytes per cycle and lowest GPU Throughput was observed for the file size of 384 kb i.e., 26875.2974 bytes per cycle.

Table 4.5.2: Test results for Blake on Audio files

Audio File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
384 kb	18.65	79.28	0.055	26875.2974
457 kb	19.57	79.47	0.0577	26939.7059
719 kb	19.45	79.48	0.0574	26943.0958
1.1 mb	19.6	80.01	0.0584	26945.0955

Following are graphical representations of results of Blake on Audio dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Audio data set.

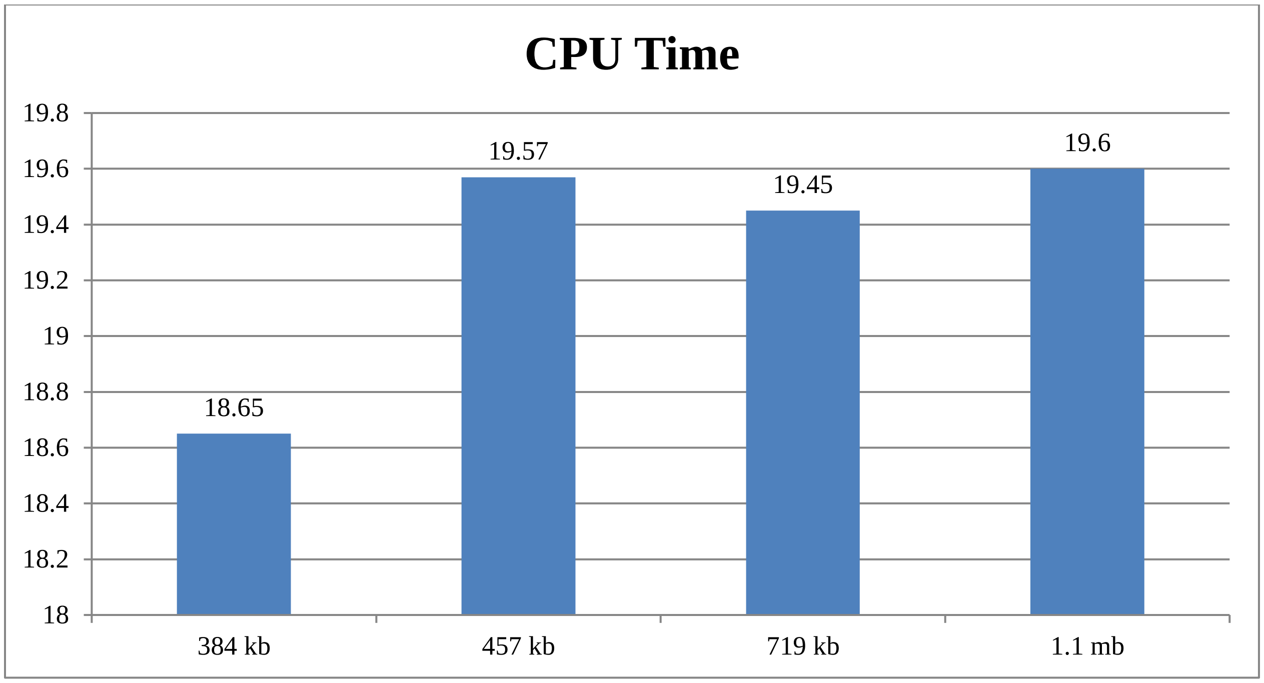


Figure 4.5.2.1: CPU Time for Blake for Audio files

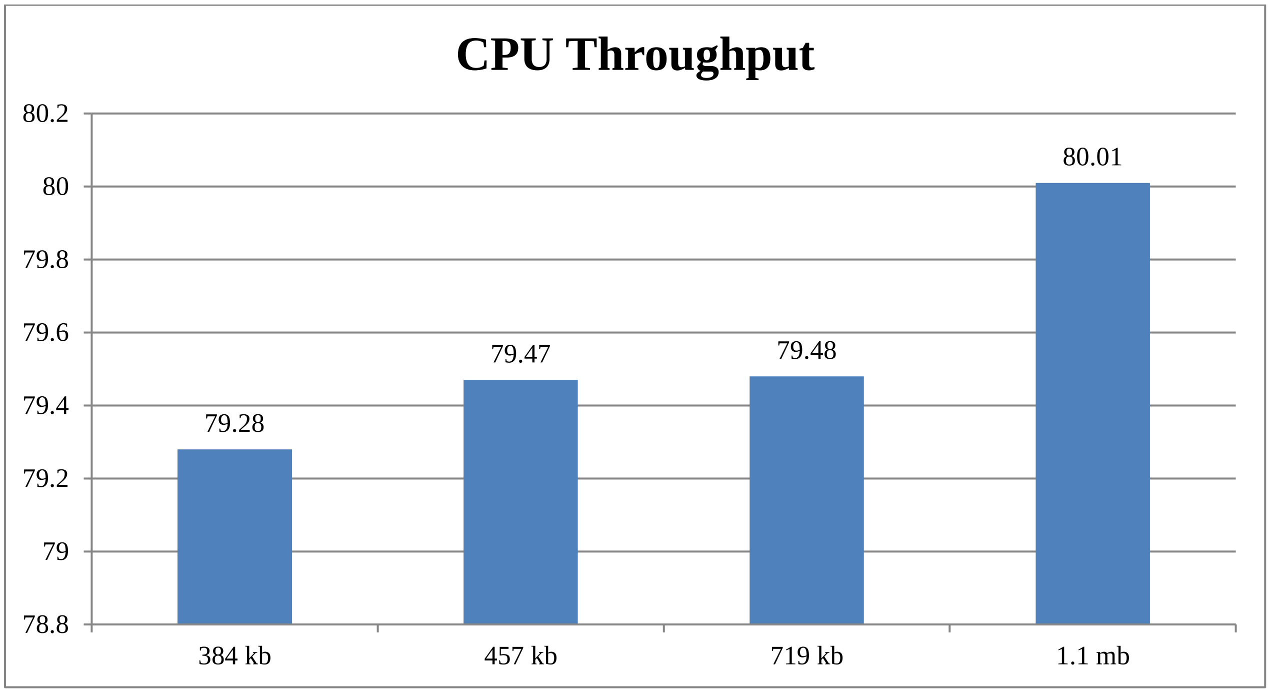


Figure 4.5.2.2: CPU Throughput for Blake for Audio files

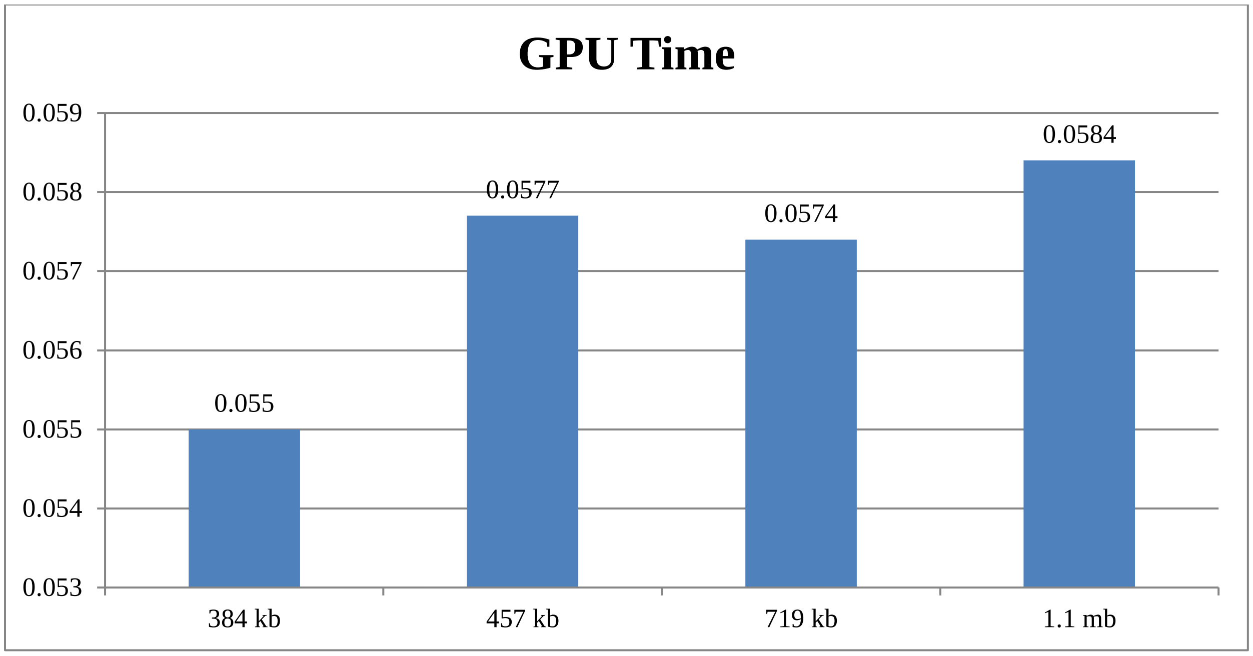


Figure 4.5.2.3: GPU Time for Blake for Audio files

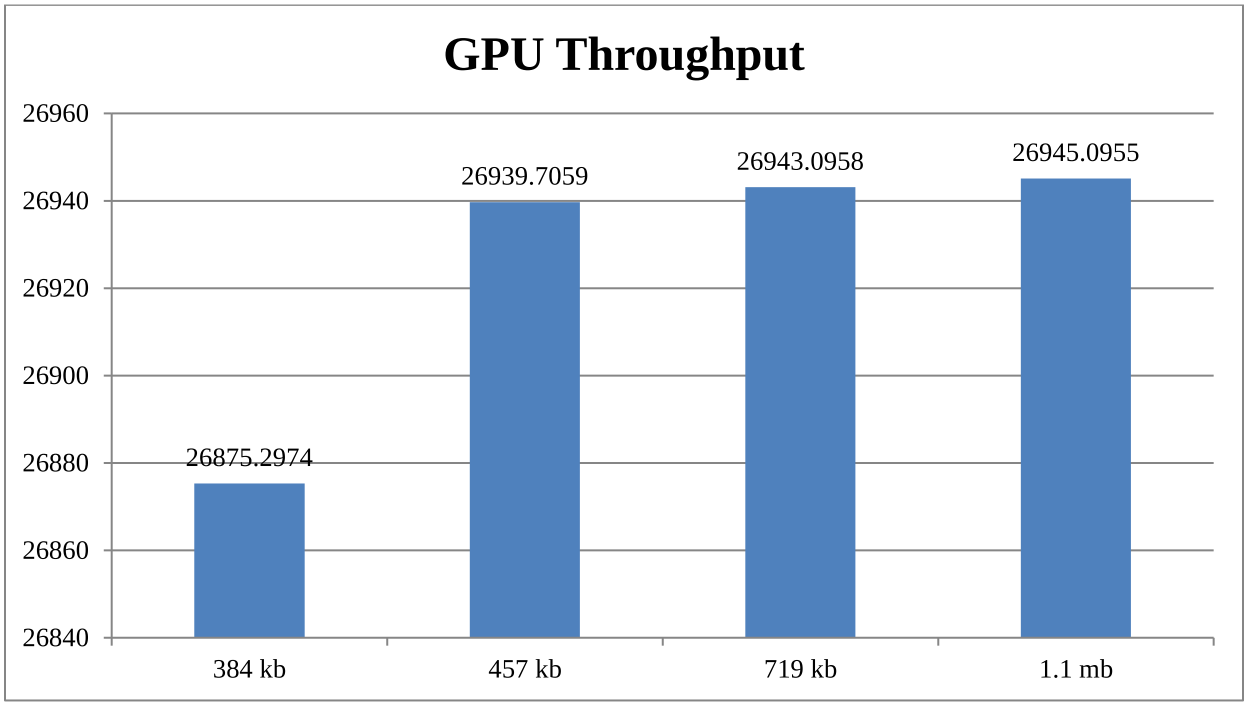


Figure 4.5.2.4: GPU Throughput for Blake for Audio files

4.5.3 Video Dataset

Table 4.5.3 shows results for Blakeon Video Dataset. Highest CPU time was recorded for the file size of 960 kb i.e., 8.54 bytes per sec and lowest was recorded for file size of 99 kb i.e., 7.37 bytes per sec. Whereas, highest CPU Throughput was recorded for the file size of 141 kb i.e., 174.81 bytes per cycle and lowest was recorded for file size of 960 kb i.e., 169.58 bytes per cycle.

Similarly, highest GPU Time was recorded for the file size of 141 kb i.e., 0.0131 bytes per sec and lowest was recorded for file size 769 kb i.e., 0.0049 bytes per sec. Whereas, highest GPU Throughput was recorded for file size of 769 kb i.e., 294149.9034 bytes per cycle and lowest was recorded for file size of 99 kb i.e., 100427.7663 bytes per cycle.

Table 4.5.3: Test results for Blake on Video files

Video File	CPU Time	CPU Throughput	GPU Time	GPU Throughput
99 kb	7.37	169.97	0.0125	100427.7663
141 kb	7.75	174.81	0.0131	103287.5086
240 kb	7.76	169.58	0.0084	157121.5024
769 kb	8.33	171.89	0.0049	294149.9034
960 kb	8.54	169.56	0.005	290162.6483

Following are graphical representations of results of Blake on Video dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Video data set.

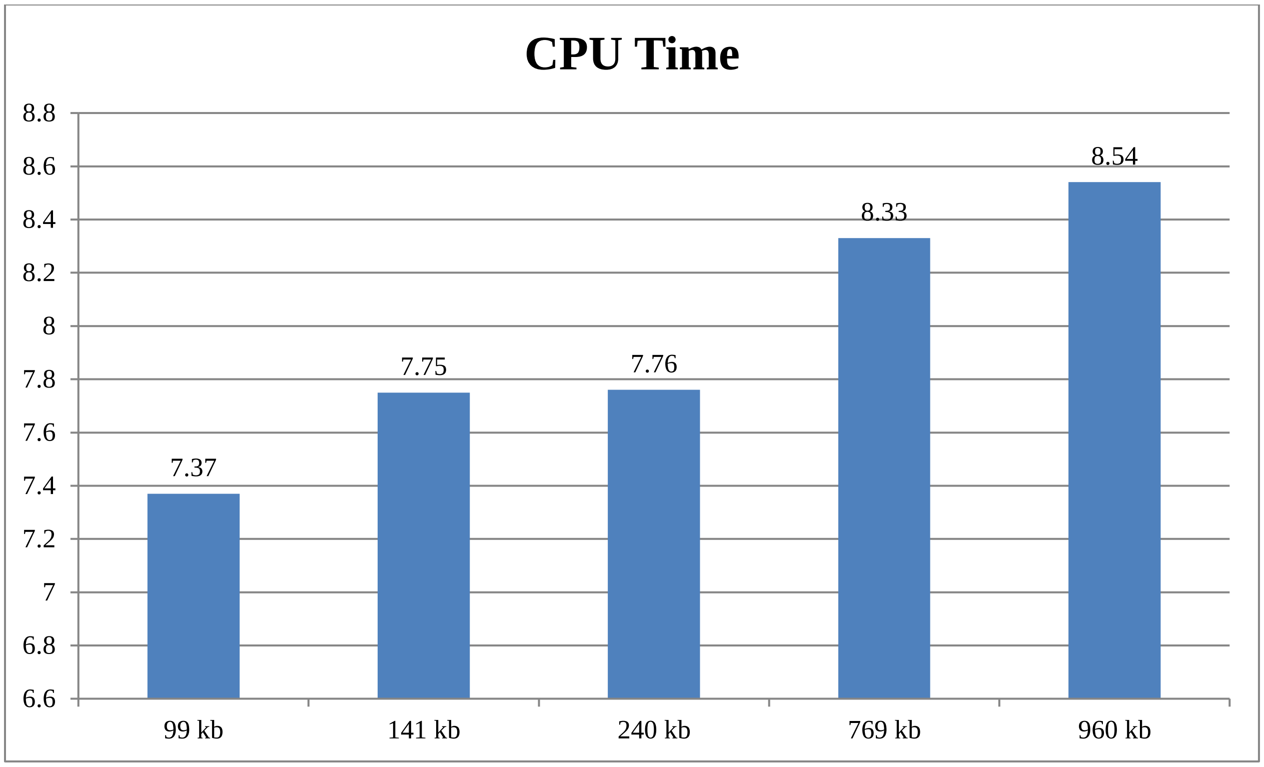


Figure 4.5.3.1: CPU Time for Blake for Video files

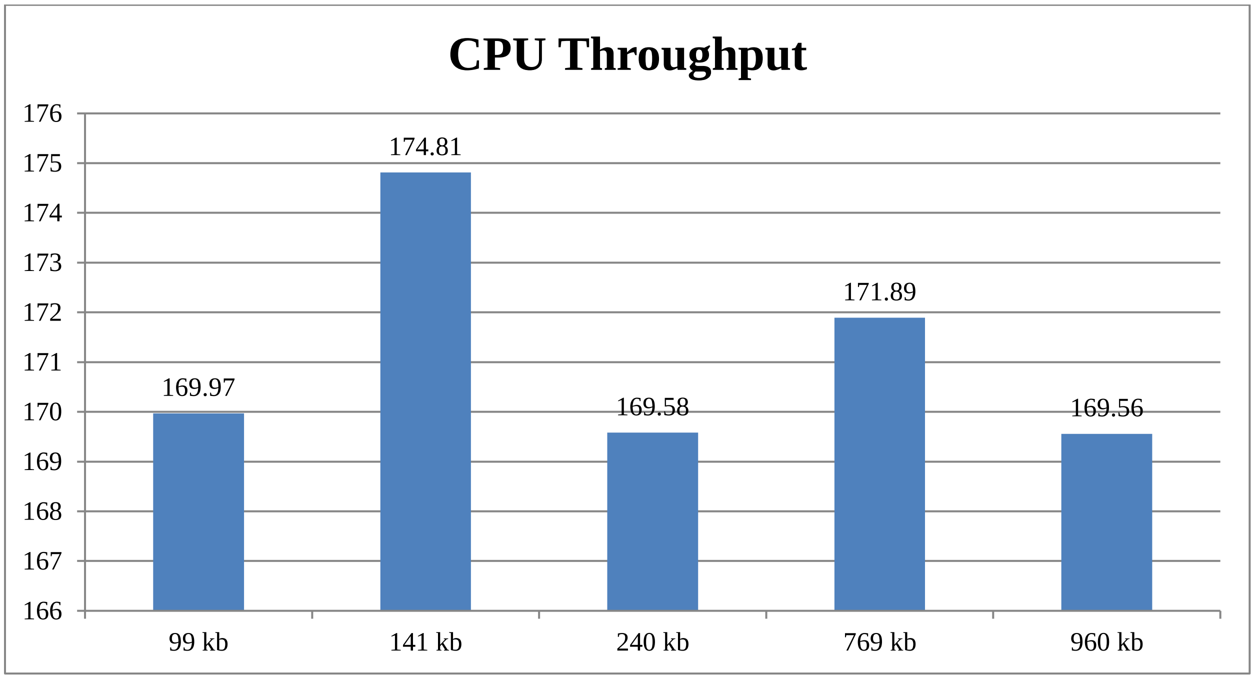


Figure 4.5.3.2: CPU Throughput for Blake for Video files

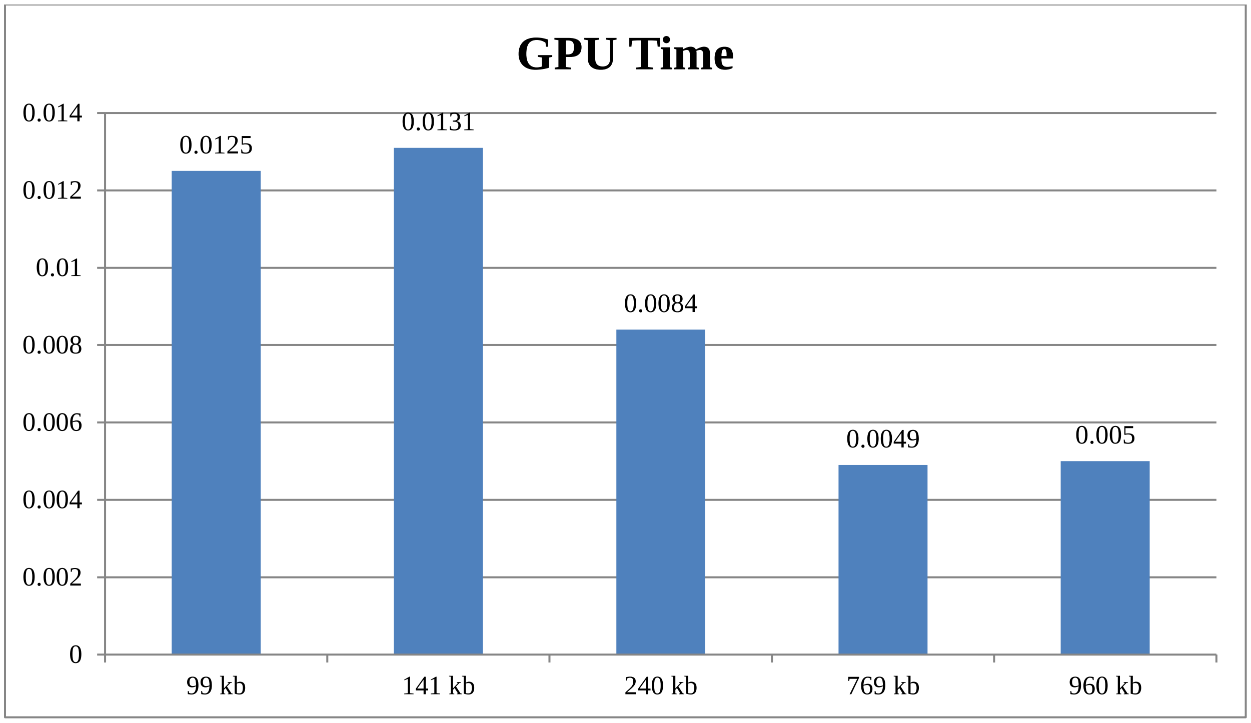


Figure 4.5.3.3: GPU Time for Blake for Video files

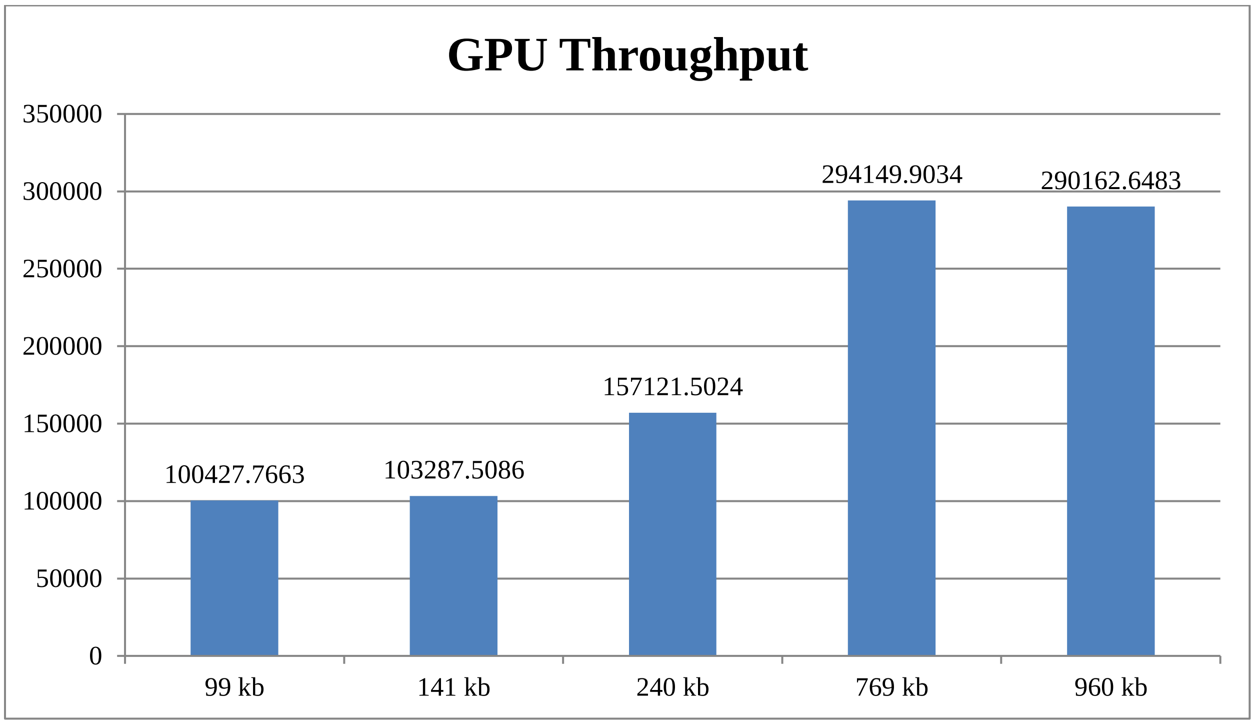


Figure 4.5.3.4: GPU Throughput for Blake for Video files

4.5.4 Image Dataset

Table 4.5.4depicts results of Blakeon Image Dataset. Highest CPU Time was observed on file size of 960 kb bytes i.e., 8.67 bytes per sec, and lowest CPU Time was observed on file size of 145 kb i.e., 8.41 bytes per sec. Highest CPU Throughput was observed on file size of 80 kb i.e., 204.42 bytes per cycle and lowest CPU Throughput was observed on file size of 145 kb bytes i.e., 168.95 bytes per cycle.

Similarly, highest GPU Time was observed on file size of 80 kb i.e 0.0144 bytes per sec and lowest was observed on file size of 769 kb i.e., 0.0051 bytes per sec. Highest GPU Throughput was recorded on 769 kb file i.e., 295874.6555 bytes per cycle and lowest was recorded on file size of 107 kb i.e., 101195.8789 bytes per cycle.

Table 4.5.4: Test results for Blake on Image files

Image File	**CPU Time**	**CPU Throughput**	**GPU Time**	**GPU Throughput**
80 kb	8.52	204.42	0.0144	120782.7499
107 kb	8.45	171.27	0.0143	101195.8789
145 kb	8.41	168.95	0.0091	156537.7864
769 kb	8.43	172.15	0.0051	295874.6555
960 kb	8.67	171.55	0.0053	294658.7411

Following are graphical representations of results of Blake on Image dataset depicting performance based on CPU Time (Bytes per Second), CPU Throughput (Bytes per Cycle), GPU Time (Bytes per Second) and GPU Throughput (Bytes per Cycle). Same can be compared with other SHA-3 candidates' performances for Image data set.

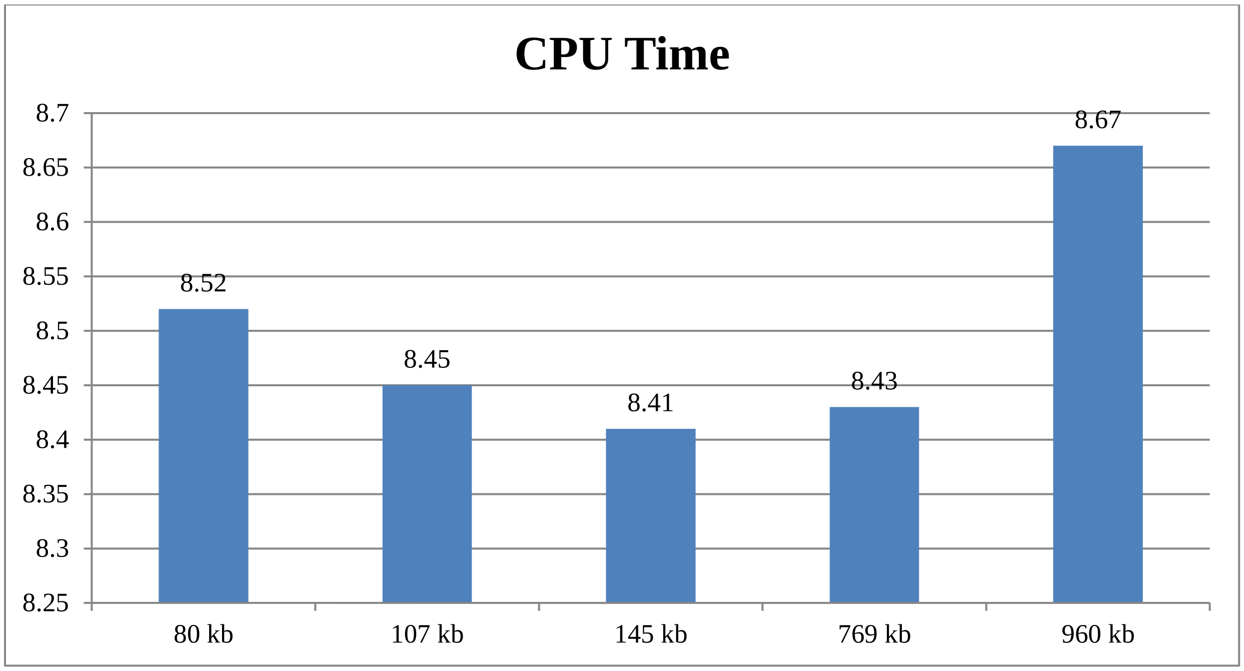


Figure 4.5.4.1: CPU Time for Blake for Image files

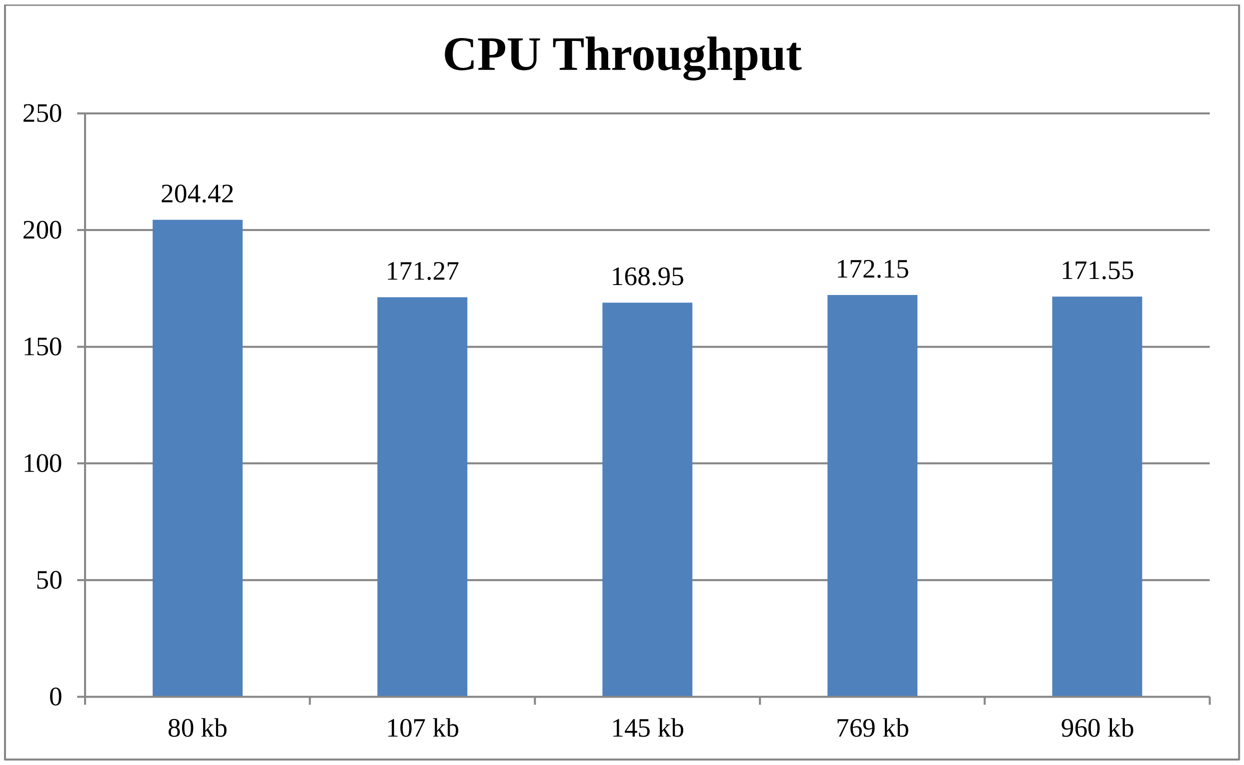


Figure 4.5.4.2: CPU Throughput for Blake for Image files

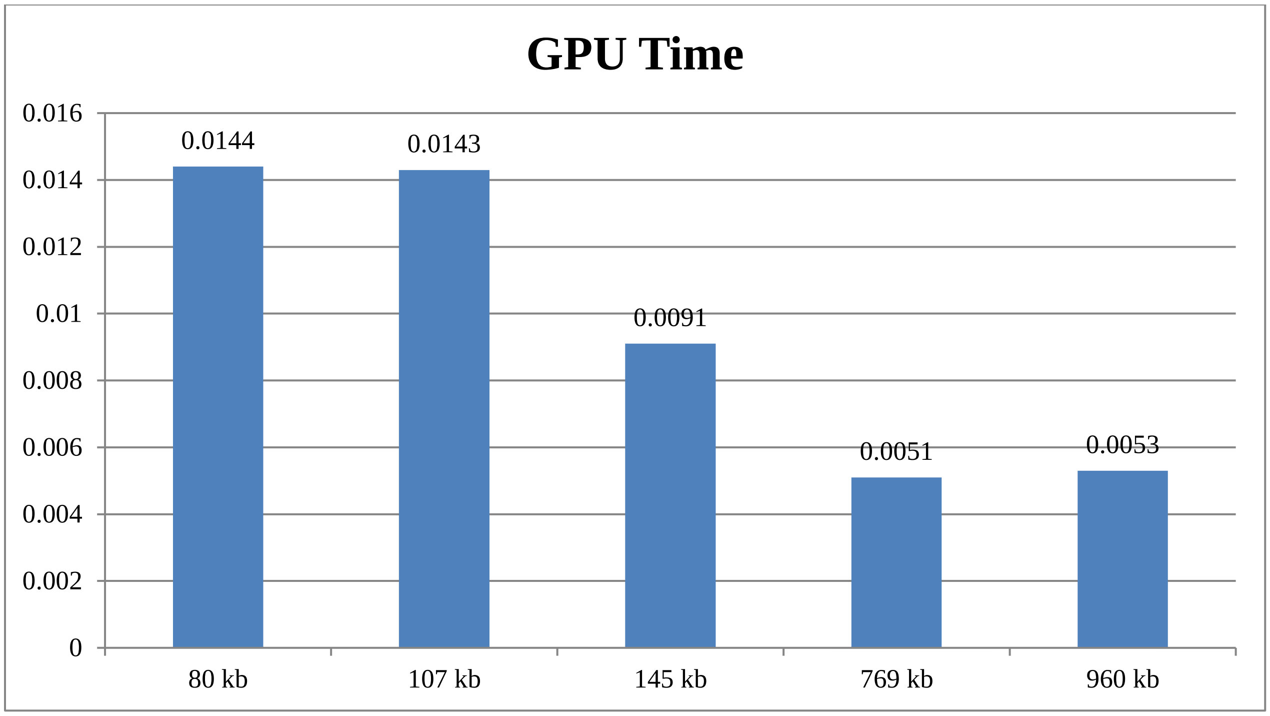


Figure 4.5.4.3: GPU Time for Blake for Image files

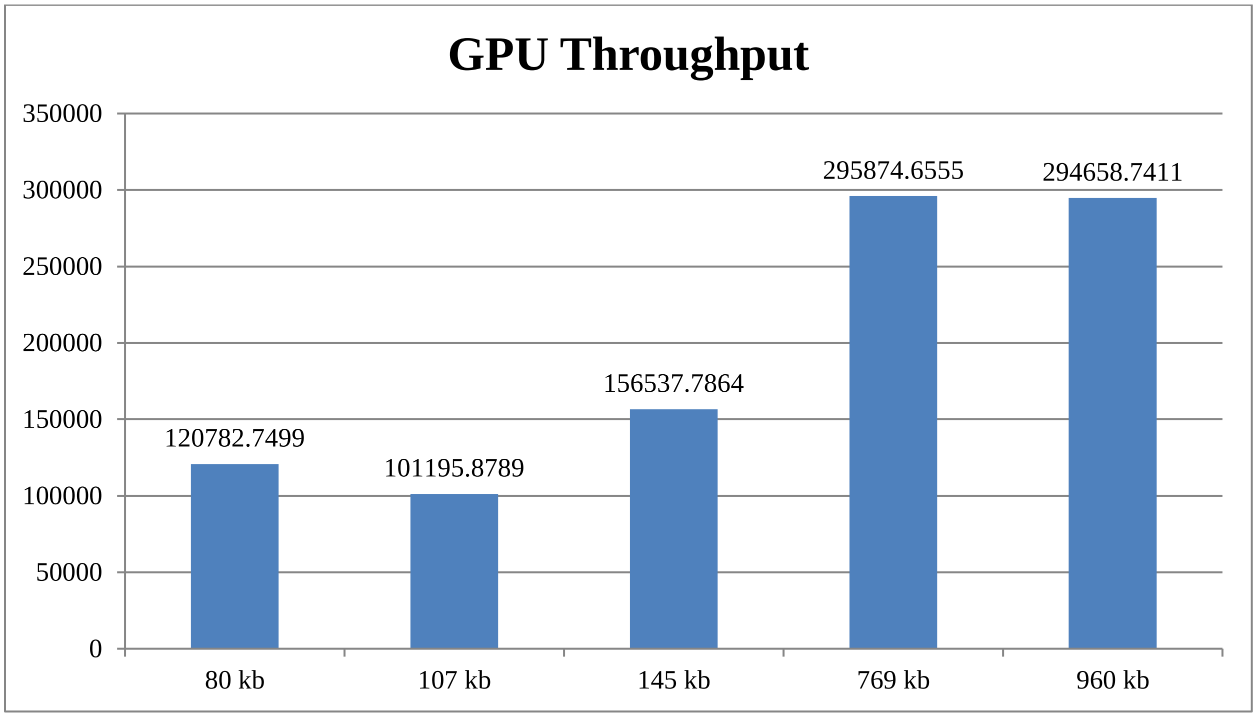


Figure 4.5.4.4: GPU Throughput for Blake for Image files

4.6 Comparison for Text Dataset

4.6.1 CPU Time (Bytes per Second)

In terms of CPU Time, Keccak was faster than all other contenders. Second comes Skein, Blake stood third.

Table 4.6.1: Comparison of CPU Time on Text dataset

Text File	**Groestl**	**Keccak**	**JH**	**Skein**	**Blake**
94 kb	14.6	20.47	9.9	19.92	19.26
319 kb	14.45	19.63	9.57	19.45	19.56
538 kb	13.06	19.66	9.53	19.56	19.52
963 kb	14.59	20.26	9.49	20.35	19.89
977 kb	14.3	20.45	9.73	20.54	20.14
1.1 mb	14.57	20.32	9.8	20.5	20.19

Below graph depicts the performance of all five SHA-3 finalist candidates based on CPU Time on text dataset. It can be seen that Keccak outperformed all other contenders w.r.t CPU Time.

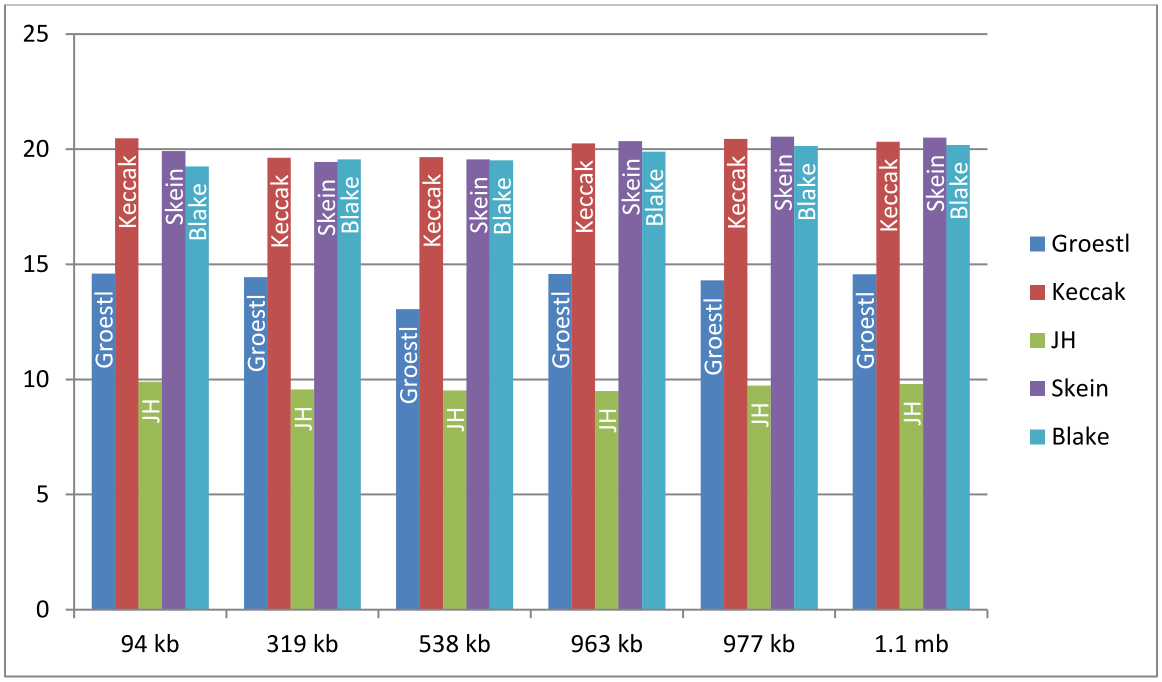


Figure 4.6.1: Comparison of CPU Time for Text files

4.6.2 CPU Throughput (Bytes per Cycle)

In terms of CPU Throughput, Keccak performed the best, Groestl stood second, Skein stood third.

Table 4.6.2: Comparison of CPU Throughput on Text dataset

Text File	Groestl	Keccak	JH	Skein	Blake
94 kb	101.12	152.96	72.87	76.47	79.24
319 kb	100.63	152.81	72.64	75.41	72.64
538 kb	100.52	152.77	72.66	73.25	72.66
963 kb	100.58	153.21	72.63	76.52	72.63
977 kb	100.5	152.98	72.64	74.85	72.64
1.1 mb	100.62	153.65	72.65	75.75	72.65

Following graph depicts the performance comparison of all SHA-3 finalists based on CPU Throughput on text dataset.

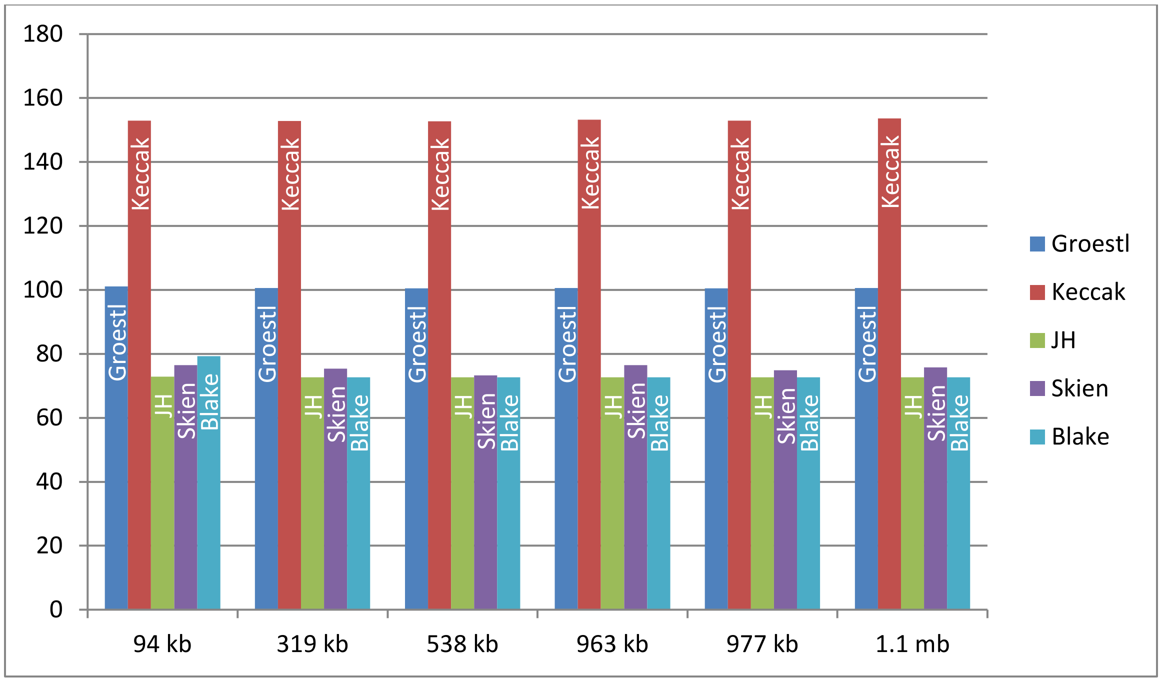


Figure 4.6.2: Comparison of CPU Throughput on Text files

4.6.3 GPU Time

In terms of GPU Time, Blake turned out to be fastest, Skein stood second, and Keccak stood at third. Next table shows the results for GPU Time on Text dataset.

Table 4.6.3: Comparison of GPU Time on Text dataset

Text File	Groestl	Keccak	JH	Skein	Blake
94 kb	0.0247	0.0346	0.0168	0.0337	0.0326
319 kb	0.0105	0.0143	0.007	0.0321	0.0345
538 kb	0.0095	0.0143	0.0069	0.0252	0.0347
963 kb	0.0085	0.0118	0.0055	0.0152	0.0411
977 kb	0.0084	0.012	0.0059	0.0198	0.0314
1.1 mb	0.0085	0.0119	0.0061	0.0185	0.0215

Graph below depicts the results of all finalist candidates on GPU Time for text dataset.

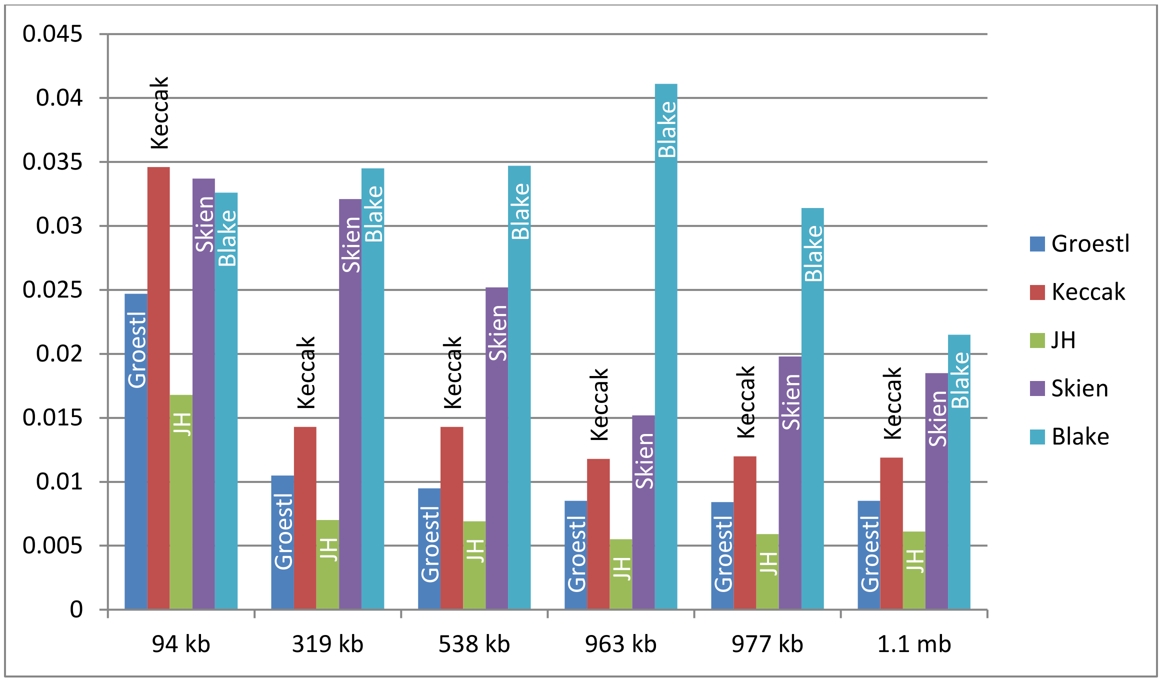


Figure 4.6.3: Comparison of GPU Time for Text files

4.6.4 GPU Throughput

Keccak stood first in terms of GPU throughput whereas Groestl scored second and Skein scored third position.

Table 4.6.4: Comparison of GPU Throughput for Text dataset

Text File	Groestl	Keccak	JH	Skein	Blake
94 kb	59747.3421	90377.3086	43055.6647	45182.7457	46819.4164
319 kb	138128.2034	209752.2683	99708.1655	99777.445	48987.4654
538 kb	137977.2136	209697.363	99735.6182	98754.254	88654.6548
963 kb	172119.3628	261430.454	124289.4146	121245.2642	110324.213
977 kb	171982.4614	261289.7	124306.5273	123548.594	123446.546
1.1 mb	172187.8136	262328.63	124323.64	125648.987	124878.963

Following graph depicts the results of all finalists on GPU Throughput for text files.

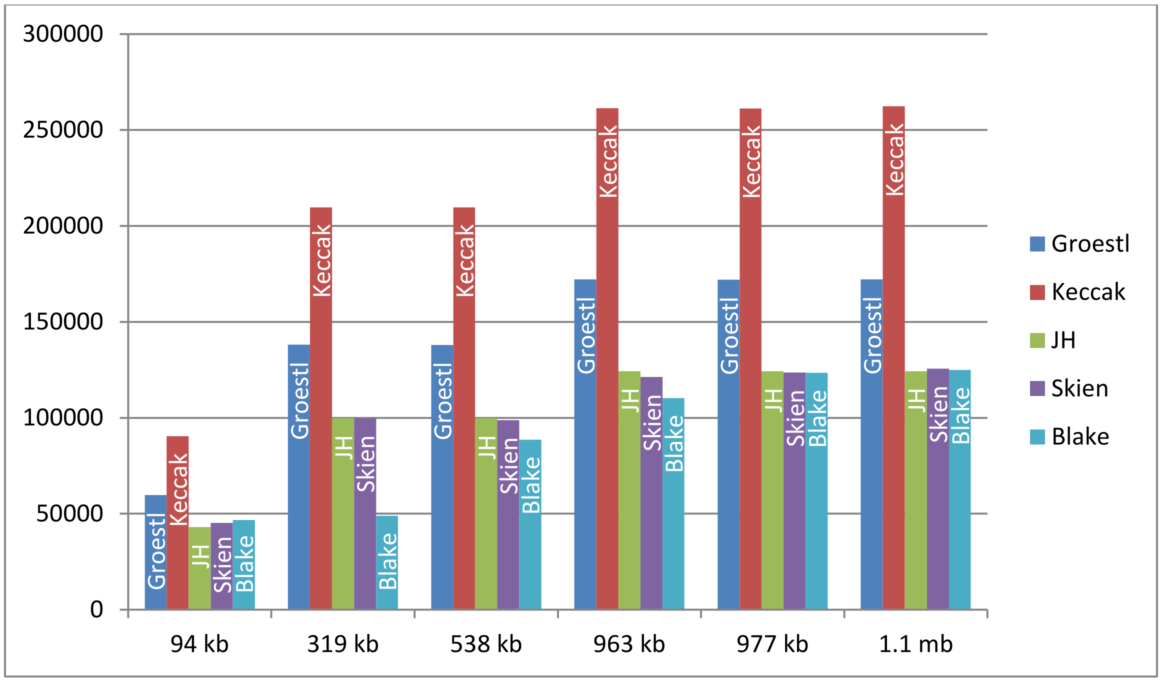


Figure 4.6.4: Comparison of GPU Throughput for Text files

4.7 Comparison of Audio Dataset

4.7.1 CPU Time

In terms of CPU Time, Keccak was faster than all other contenders. Second comes Skein, Blake stood third.

Table 4.7.1: Comparison of CPU Time for Audio dataset

Audio File	Groestl	Keccak	JH	Skein	Blake
384 kb	14.25	19.1	9.81	19.83	18.65
457 kb	14.46	20.97	9.74	20.02	19.57
719 kb	14.23	18.89	9.82	19.92	19.45
1.1 mb	14.59	20.91	9.8	20.05	19.6

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Time on Audio dataset.

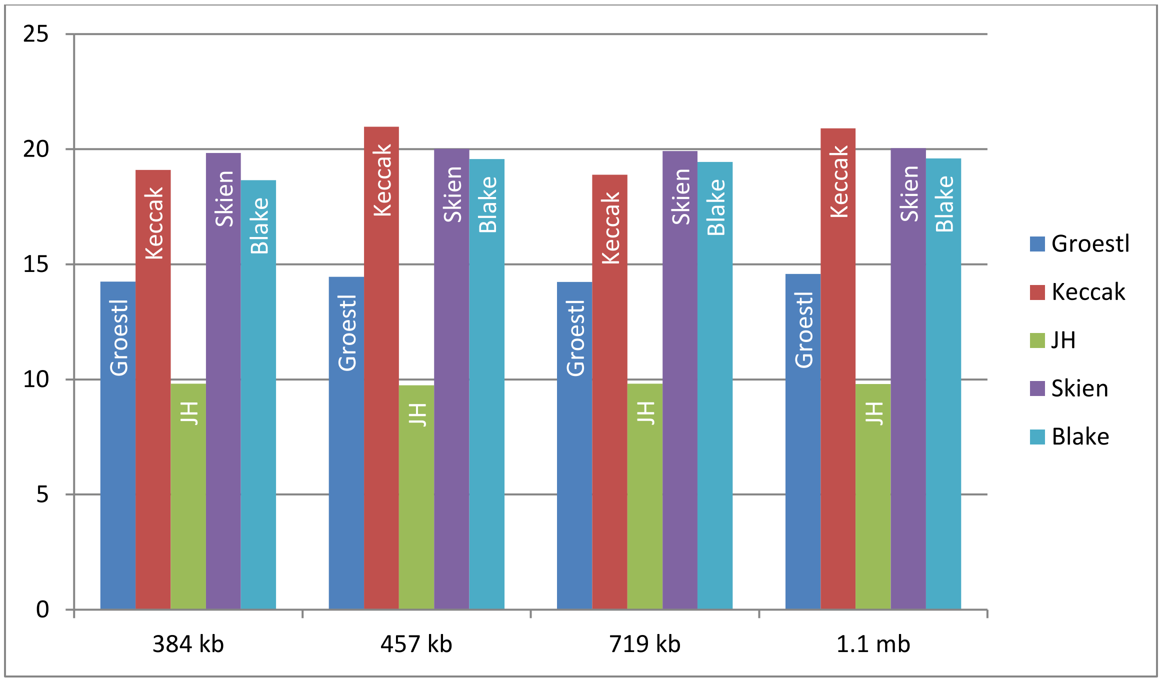


Figure 4.7.1: Comparison of CPU Time for Audio dataset

4.7.2 GPU Time

In terms of GPU Time, Blake was faster than all other contenders. Second comes Keccak, and Skein stood third.

Table 4.7.2: Comparison of GPU Time for Audio dataset

Audio File	**Groestl**	**Keccak**	**JH**	**Skein**	**Blake**
384 kb	0.042	0.0563	0.0289	0.0585	0.055
457 kb	0.0427	0.0619	0.0287	0.0591	0.0577
719 kb	0.042	0.0557	0.029	0.0588	0.0574
1.1 mb	0.0085	0.0122	0.0057	0.0117	0.0584

Following graph depicts the performance of all five SHA-3 finalist candidates based on GPU Time on Audio dataset.

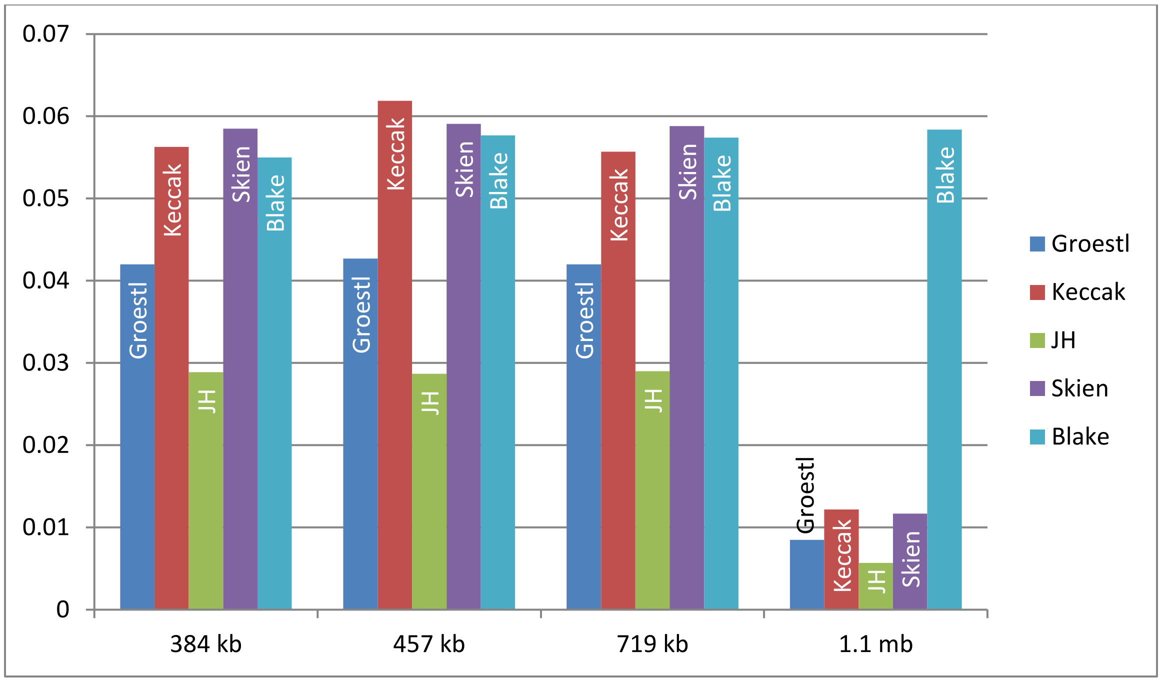


Figure 4.7.2: Comparison of GPU Time for Audio dataset

4.7.3 CPU Throughput

In terms of CPU Throughput, Keccak stood first and Groestl stood second. Blake stood third.

Table 4.7.3: Comparison of CPU Throughput for Audio dataset

Audio File	Groestl	Keccak	JH	Skein	Blake
384 kb	101.13	152.85	72.83	76.42	79.28
457 kb	101.18	152.95	72.89	76.46	79.47
719 kb	101.19	152.95	72.87	76.56	79.48
1.1 mb	101.2	152.96	72.87	76.49	80.01

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Throughput on Audio dataset.

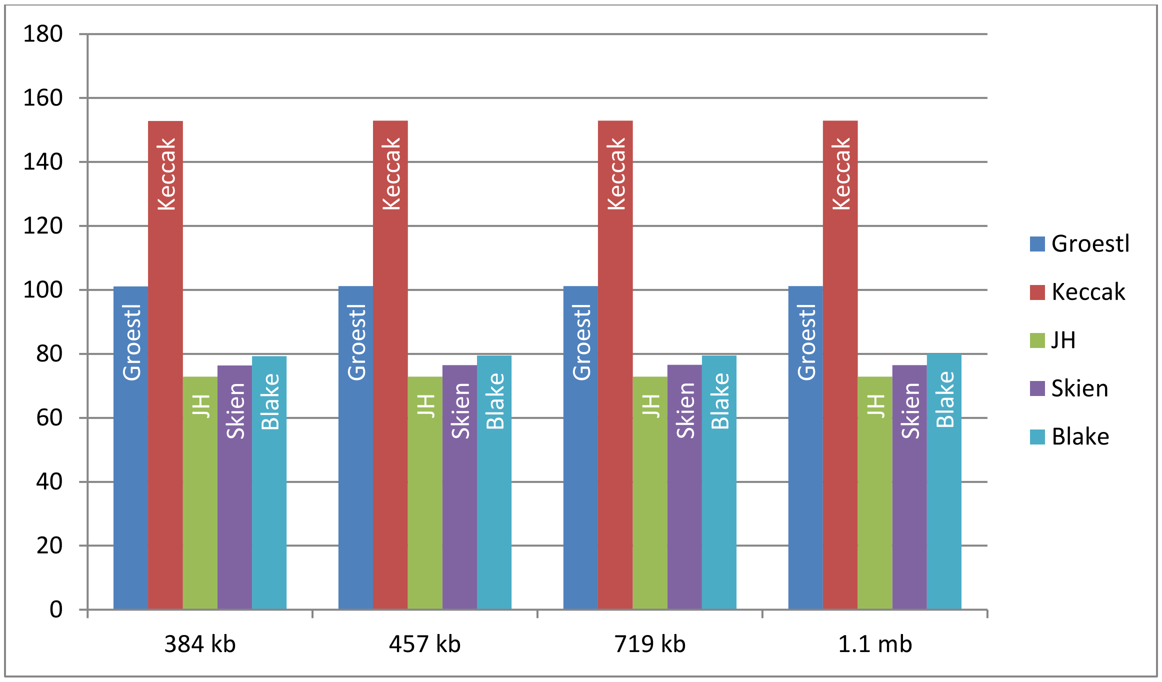


Figure 4.7.3: Comparison of CPU Throughput for Audio dataset

4.7.4 GPU Throughput

In terms of GPU Throughput, Keccak stood first, Groestl stood second, and Blake stood third.

Table 4.7.4: Comparison of GPU Throughput for Audio dataset

Audio File	Groestl	Keccak	JH	Skein	Blake
384 kb	34282.2758	51814.9496	24688.78	25905.7798	26875.2974
457 kb	34299.2254	51848.8488	24709.1376	25919.3395	26939.7059
719 kb	34302.6153	51848.8488	24702.3577	25919.3395	26943.0958
1.1 mb	173180.3492	261755.595	124700.119	130843.5721	26945.0955

Graph below depicts the performance of all five SHA-3 finalist candidates based on GPU Throughput on Audio dataset.

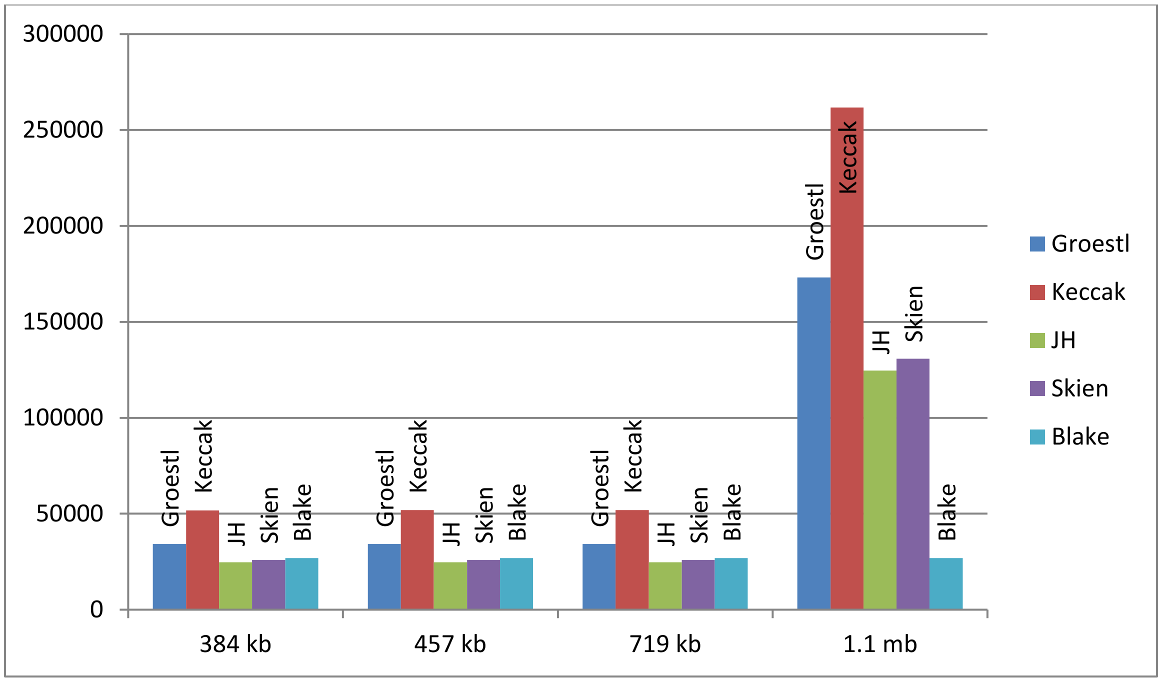


Figure 4.7.4: Comparison of GPU Throughput for Audio dataset

4.8 Comparison of Video Dataset

4.8.1 CPU Time

In terms of CPU Time, Keccak was faster than all other contenders. Second comes Skein, Blake stood third.

Table 4.8. 1: Comparison of CPU Time for Video dataset

Video File	Groestl	Keccak	JH	Skein	Blake
99 kb	4.96	7.57	3.47	7.02	7.37
141 kb	5.39	7.79	3.7	7.49	7.75
240 kb	5.33	8.61	3.88	8.33	7.76
769 kb	5.96	8.72	4.17	8.41	8.33
960 kb	6.11	8.88	4.24	8.65	8.54

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Time on Video dataset.

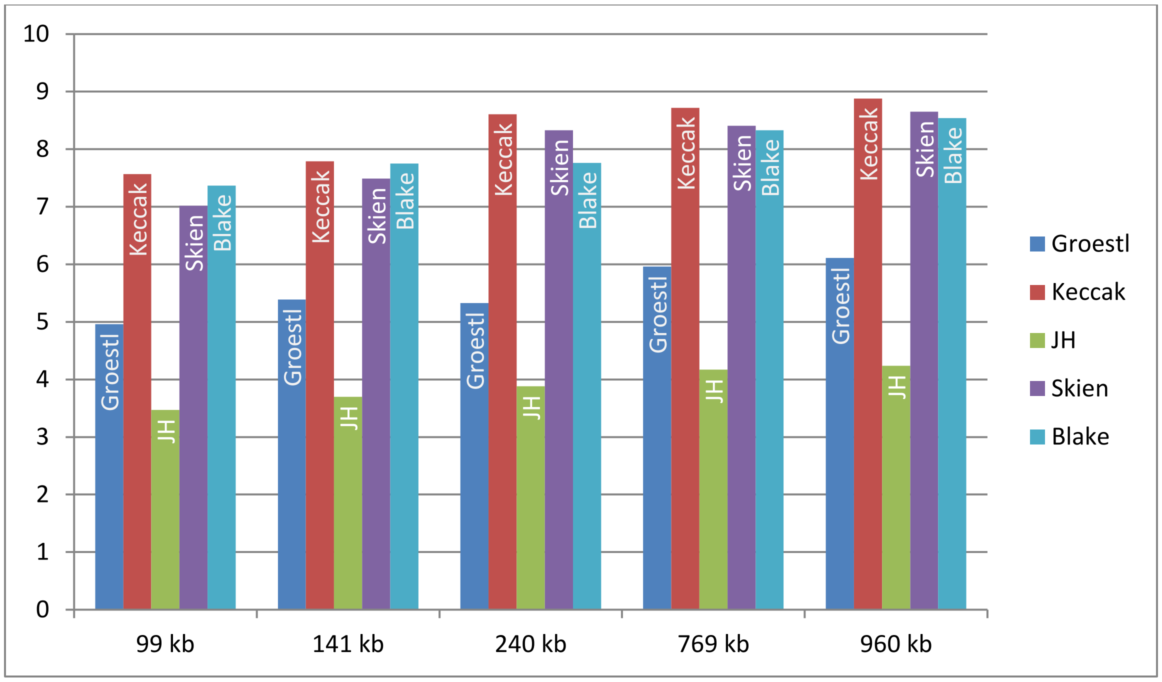


Figure 4.8.1: Comparison of CPU Time for Video dataset

4.8.2 GPU Time

In terms of GPU Time, Keccak was faster than all other contenders. Blake and Skein performed almost the same.

Table 4.8.2: Comparison of GPU Time for Video dataset

Video File	Groestl	Keccak	JH	Skein	Blake
99 kb	0.0084	0.0128	0.0059	0.0119	0.0125
141 kb	0.0091	0.0132	0.0063	0.0127	0.0131
240 kb	0.0058	0.0093	0.0042	0.009	0.0084
769 kb	0.0035	0.0051	0.0024	0.0049	0.0049
960 kb	0.0036	0.0052	0.0025	0.0051	0.005

Following graph depicts the performance of all five SHA-3 finalist candidates based on GPU Time on Video dataset.

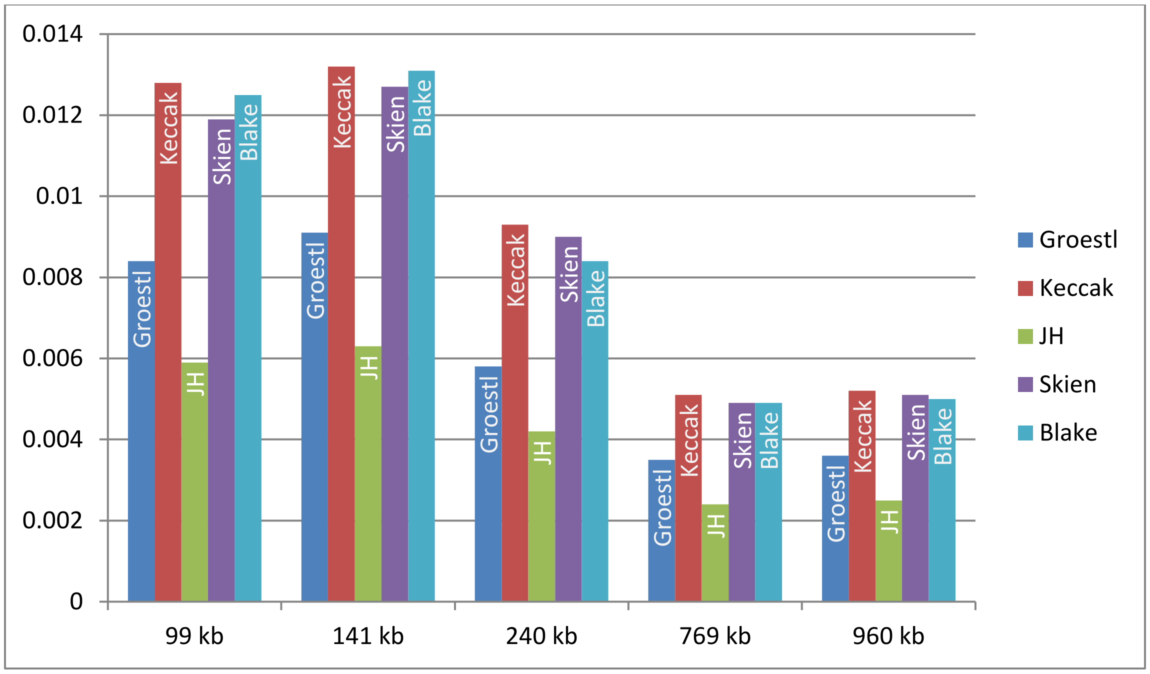


Figure 4.8.2: Comparison of GPU Time for Video dataset

4.8.3 CPU Throughput

In terms of CPU Throughput, Keccak was faster than all other contenders. Second comes Groestl, Blake stood third.

Table 4.8.3: Comparison of CPU Throughput for Video dataset

Video File	Groestl	JH	Keccak	Skein	Blake
99 kb	218.75	155.92	330.74	163.16	169.97
141 kb	217.93	154.99	330.67	163.17	174.81
240 kb	218.39	155.9	329.5	164.22	169.58
769 kb	218.3	155.15	330.39	163.03	171.89
960 kb	218.32	155.14	327.34	163.24	169.56

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Throughput on Video dataset.

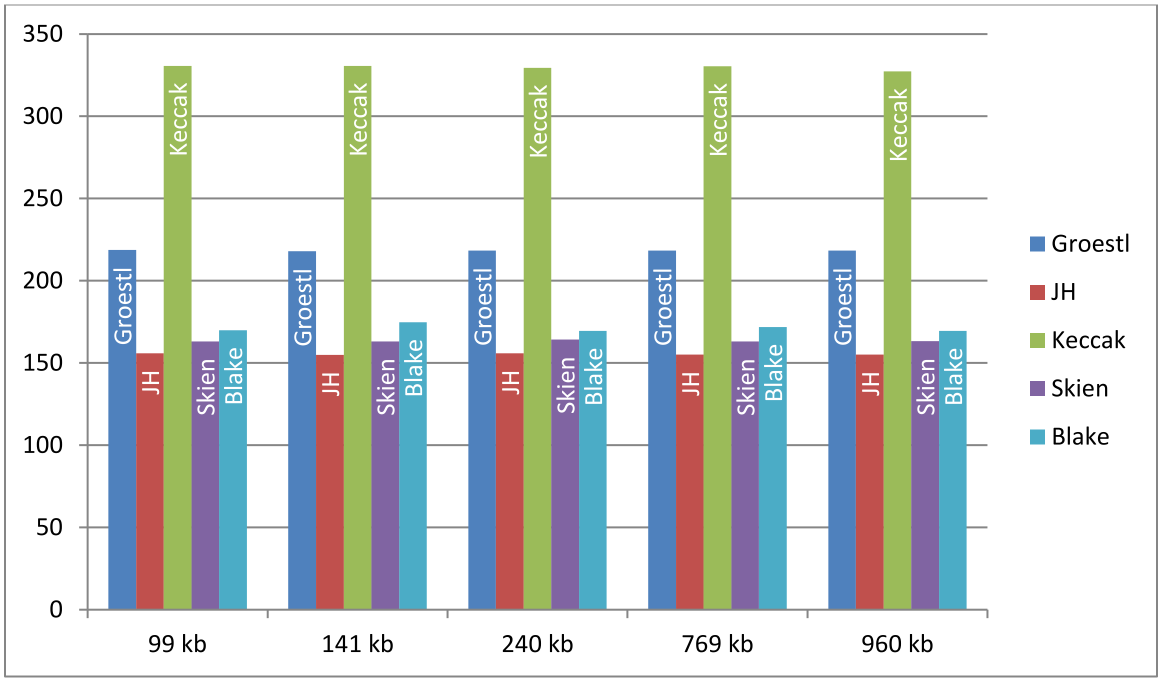


Figure 4.8.3: Comparison of CPU Throughput for Video dataset

4.8.4 GPU Throughput

In terms of GPU Throughput, Keccak was faster than all other contenders. Second comes Groestl, Blake stood third.

Table 4.84: Comparison of GPU Throughput for Video dataset

Video File	Groestl	JH	Keccak	Skein	Blake
99 kb	129249.714	92126.2418	195419.659	96404.0381	100427.7663
141 kb	128765.2122	91576.7459	195378.2991	96409.9467	103287.5086
240 kb	202345.5885	144446.528	305292.6939	152155.2843	157121.5024
769 kb	373569.8639	265503.2725	565385.9245	278988.0665	294149.9034
960 kb	373604.0893	265486.1598	560166.5563	279347.4328	290162.6483

Following graph depicts the performance of all five SHA-3 finalist candidates based on GPU Throughput on Video dataset.

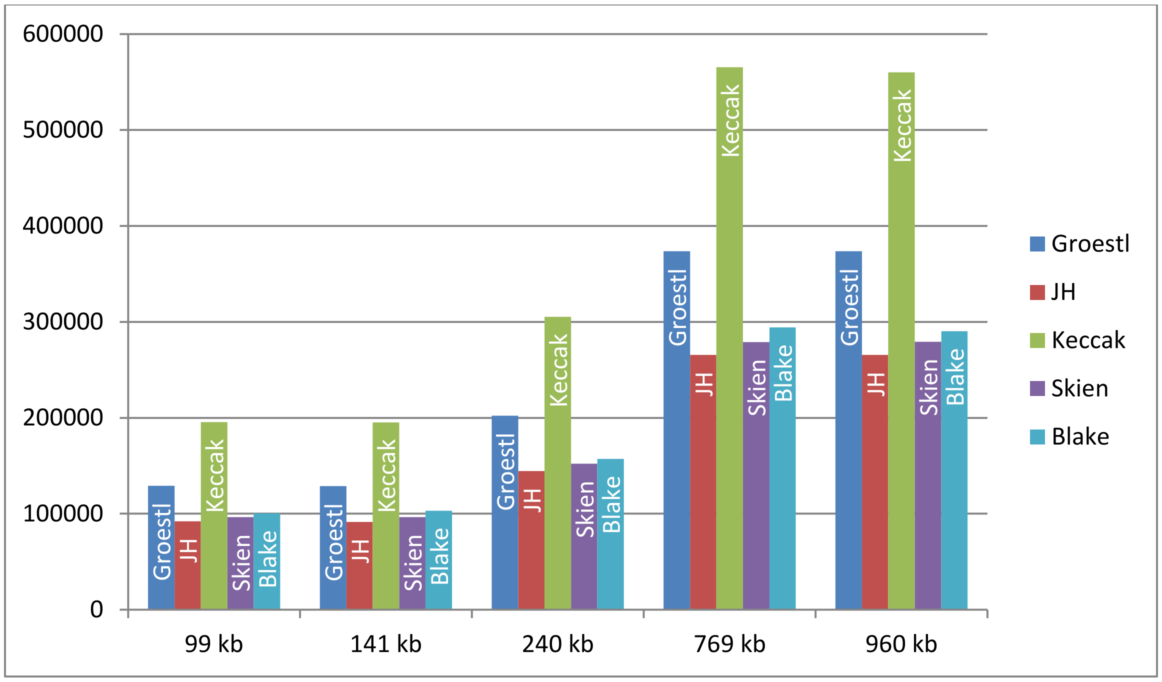


Figure 4.8.4: Comparison of GPU Throughput for Video dataset

4.9 Comparison for Image Dataset

4.9.1 CPU Time

In terms of CPU Time, Keccak was faster than all other contenders. Second comes Blake while Skein stood third.

Table 4.9.1: Comparison of CPU Time for Image dataset

Image File	Groestl	Keccak	JH	Skein	Blake
80 kb	6.09	8.76	4.15	7.81	8.52
107 kb	5.92	8.7	4.17	7.37	8.45
145 kb	6.01	8.67	4.17	7.32	8.41
769 kb	6.22	8.75	4.16	8.41	8.43
960 kb	6.35	8.94	4.3	8.52	8.67

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Time on Image dataset.

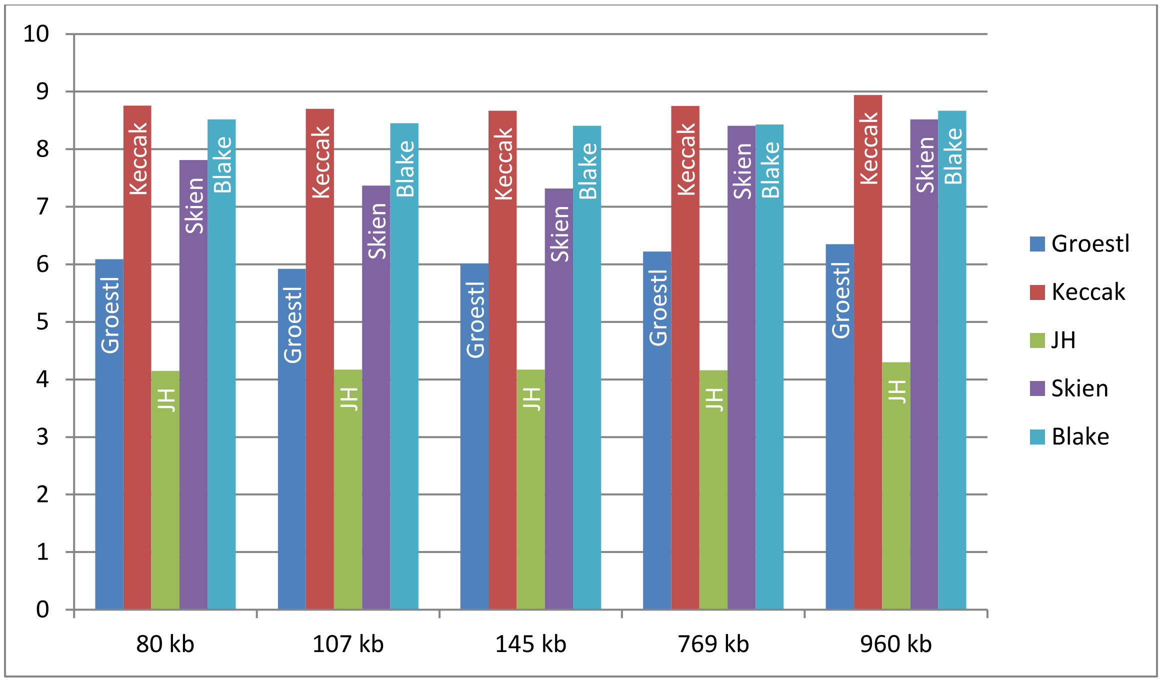


Figure 4.9.1: Comparison of CPU Time for Image dataset

4.9.2 GPU Time

In terms of GPU Time, Keccak was faster than all other contenders. Second comes Groestl, Blake stood third.

Table 4.9.2: Comparison of GPU Time for Image dataset

Image File	Groestl	Keccak	JH	Skein	Blake
80 kb	0.0103	0.0148	0.007	0.0132	0.0144
107 kb	0.01	0.0147	0.0071	0.0125	0.0143
145 kb	0.0065	0.0094	0.0045	0.0079	0.0091
769 kb	0.0069	0.0074	0.0024	0.0049	0.0051
960 kb	0.0074	0.0064	0.0027	0.0055	0.0053

Following graph depicts the performance of all five SHA-3 finalist candidates based on GPU Time on Image dataset.

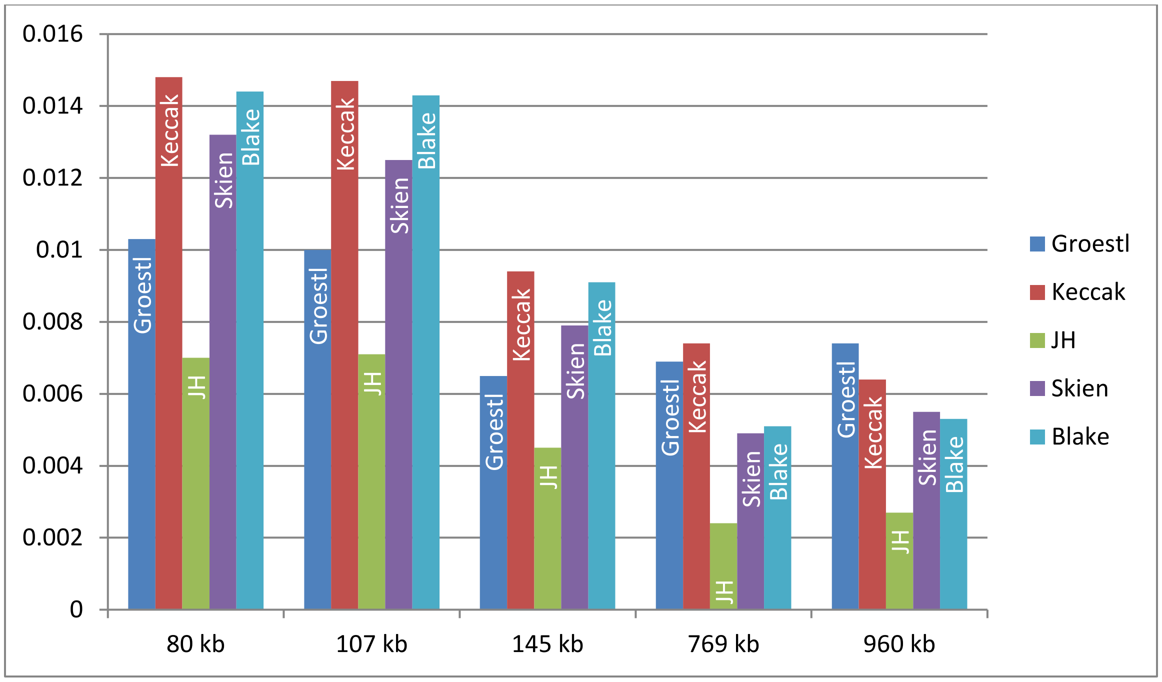


Figure 4.9.2: Comparison of GPU Time for Image dataset

4.9.3 CPU Throughput

In terms of CPU Throughput, Keccak was faster than all other contenders. Second comes Groestl, Blake stood third.

Table 4.9.3: Comparison of CPU Throughput for Image dataset

Image File	Groestl	JH	Keccak	Skein	Blake
80 kb	217.82	155.09	330.71	163.37	204.42
107 kb	221.55	155.13	331.4	163.15	171.27
145 kb	222.41	155.12	326.9	163.51	168.95
769 kb	223.3	155.74	331.82	163.03	172.15
960 kb	231.32	155.8	324.74	163.24	171.55

Following graph depicts the performance of all five SHA-3 finalist candidates based on CPU Throughput on Image dataset.

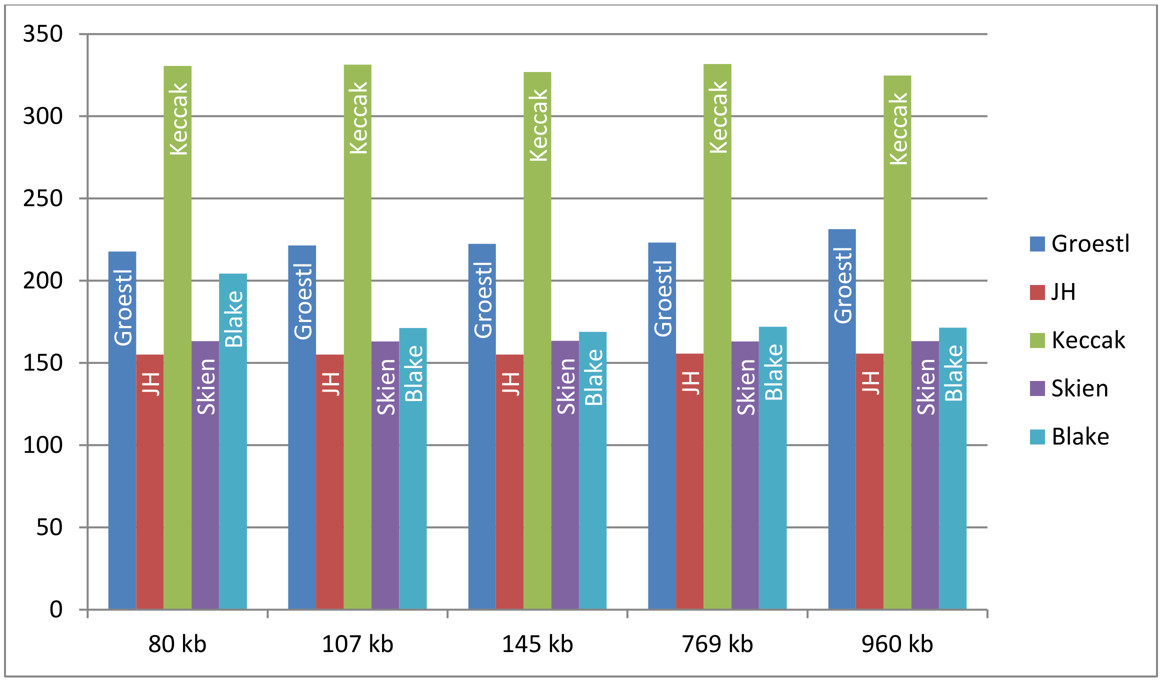


Figure 4.9.3: Comparison of CPU Throughput for Image dataset

4.9.4 GPU Throughput

In terms of GPU Throughput, Keccak was faster than all other contenders. Second comes Groestl, Blake stood third while.

Table 4.9.4: Comparison of GPU Throughput for Image dataset

Image File	**Groestl**	**JH**	**Keccak**	**Skein**	**Blake**
80 kb	128700.2181	91635.8315	195401.9333	96528.1178	120782.7499
107 kb	130904.1103	91659.4657	195809.6238	96398.1295	101195.8789
145 kb	202364.1192	143723.8321	302883.7075	151497.4457	156537.7864
769 kb	374545.5231	266412.2125	553546.985	278988.0665	295874.6555
960 kb	375745.6589	267545.6963	525398.6324	279347.4328	294658.7411

Following graph depicts the performance of all five SHA-3 finalist candidates based on GPU Throughput on Image dataset.

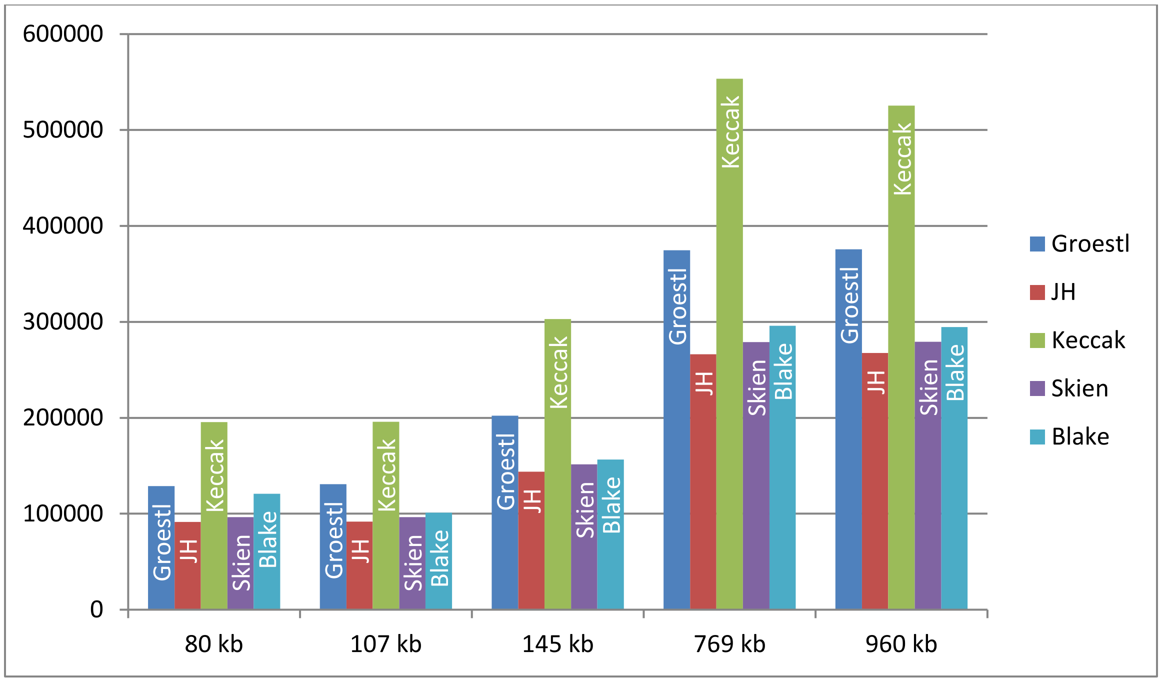


Figure 4.9.4: Comparison of GPU Throughput for Image dataset

CHAPTER 5: CONCLUSION

The objective of this study was to implement Batch mode-based JH, GROESTL,KECCAK, BLAKE, and SKEIN algorithms on both Intel Core i3-4005U 1.70GHz CPU and Nvidia 940 M GPU platforms. Its purpose was to conduct a performance evaluation of various SHA-3 candidates on different data formats i.e., Text, Audio, Video and Image.

The findings of this study suggests that KECCAK performed the best in most data formats across CPU and GPU. However, Blake performed better than Skein, JH, and Groestl on Text and Video datasets. Performance of JH was the lowest among all candidates.

This study yielded that Keccak's algorithm showed 95% to 99% reduction in time on GPU as compared to CPU with speedups ranging from 420x to 1200x for different datasets. Similarly, in terms of Throughput, Keccak showed a gain of upto 1250x on GPU.

Moreover, this study also validates results of Kuznetsov et al. [4] showing that Keccak is the best performing algorithm for smaller file sizes seconded by Blake, where Blake has performed almost the same as Keccak on file sizes reaching or exceeding 1 MB. Therefore, these two algorithms are most suitable for use in blockchain technology.

The results of this research will provide a valuable insight in understanding how these algorithms perform on different architectures, this knowledge will aid in optimizing the algorithms for specific hardware, enhancing its efficiency and speeding up the hashing process. In real-world application, outcomes of this research will guide organizations for better selection of hardware, by choosing the most suitable hardware and algorithm for optimal performance. Furthermore, this research will aid researchers in further evaluating the performance of SHA-3 algorithms on more specialized hardware and software environments and improving the efficiency of these algorithms where applicable.

REFERENCES

[1] Hassan, M., Hassan, S., & Hussain, R. (2017). Optimization of Keccak Algorithms on NVIDIA GPU Platform for High-Performance Computing. International Journal of Scientific Engineering and Applied Science, 3(12), 1-6

[2] Wang, C., & Chu, X. (2019). GPU Accelerated Keccak (SHA3) Algorithm. arXiv preprint arXiv:1902.05320. Retrieved from https://arxiv.org/abs/1902.05320

[3] Sideris A, Sanida T, Dasygenis M. High Throughput Implementation of the Keccak Hash Function Using the Nios-II Processor. *Technologies*. 2020; 8(1):15. https://doi.org/10.3390/technologies8010015

[4] Kuznetsov, Alexandr & Oleshko, Inna & Tymchenko, Vladyslav & Lisitsky, Konstantin & Rodinko, Mariia & Kolhatin, Andrii. (2021). Performance Analysis of Cryptographic Hash Functions Suitable for Use in Blockchain. International Journal of Computer Network and Information Security. 13. 1-15. 10.5815/ijcnis.2021.02.01.

[5] Y. Jararweh, L. Tawalbeh, H. Tawalbeh and A. Moh'd, "Hardware Performance Evaluation of SHA-3 Candidate Algorithms," *Journal of Information Security*, Vol. 3 No. 2, 2012, pp. 69-76. doi: 10.4236/jis.2012.32008.

[6] Hanser, Christian H.. "Performance of the SHA-3 Candidates in Java." (2012).

[7] Sobti, Rajeev & Ganesan, Geetha & Anand, Sami. (2012). Performance comparison of Grøestl, JH and BLAKE - SHA-3 final round candidate algorithms on ARM cortex M3 processor. 220-224. 10.1109/ICCS.2012.57.

[8] C. Schmidt and A. Izraelevitz, "A Fast Parameterized SHA3 Accelerator", vol. 1, 2015.

[9] Singh, Gurpreet & Sobti, Rajeev. (2015). SHA-3 Blake Finalist on Hardware Architecture of ARM Cortex A8 Processor. International Journal of Computer Applications. 123. 975-8887. 10.5120/ijca2015905583.

[10] Lowden, Jason. “Analysis of KECCAK Tree Hashing on GPU Architectures.” (2014).

[11] Cayrel, Pierre-Louis & Hoffmann, Gerhard & Schneider, Michael. (2011). GPU Implementation of the Keccak Hash Function Family. 200. 33-42. 10.1007/978-3-642-23141-4_4.

[12] Rao, Muzaffar & Newe, Thomas & Grout, I.A. & Mathur, Avijit. (2016). High Speed Implementation of a SHA-3 Core on Virtex-5 and Virtex-6 FPGAs. Journal of Circuits, Systems and Computers. 25. 1650069. 10.1142/S0218126616500699.

[13] T. N. Dat, K. Iwai and T. Kurokawa, "Implementation of High-Speed Hash Function Keccak Using CUDA on GTX 1080," 2017 Fifth International Symposium on Computing and Networking (CANDAR), Aomori, Japan, 2017, pp. 475-481, doi: 10.1109/CANDAR.2017.47.

[14] https://www.github.com/K2/HashLib