CUDA Implementation of RSHA-1 and Code Optimization

Neha Kishore[1], Dr. Bhanu Kapoor[2]

[1]Chitkara University, Himachal Pradesh, India; [2]Mimasic, Dallas, TX USA

neha.kishore@chitkarauniversity.edu.in; bkapoor@mimasic.com;

ABSTRACT

A new algorithm RSHA-1 is proposed which serves the same purpose as SHA-1 Cryptographic Hash Algorithm with less computational complexity and delay by using the power of GPU's. The target is not only to reduce the delay but also to lead green computing by saving power consumption.

The algorithm implements recursive tree hash to break the chain dependencies of the standard hash function to enable parallel computation of hash code of heavy files. We discuss here the theoretical foundation for the work and the performance implications. The result analysis of the algorithm has been done by implementing and optimizing it using CUDA on GPU's and comparing the results with standard SHA-1, implementation in OpenMP API.

MOTIVATION

Conventional Cryptographic hash algorithms like MD5, SHA-1, SHA-2 [1] are mainly serially implemented and are used in various applications like Digital Signatures, Forensics, SSL protocol, authentication etc. The computation of hash value requires lot of execution time.

In forensics, the hash process is normally used during acquisition of the evidence, during verification of the forensic image, and again at the end of the examination to ensure the integrity of the data and forensic processing. These algorithms are also currently used to validate the integrity of downloaded files in information technology applications.

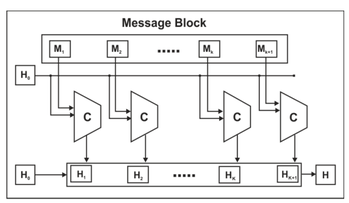
The amount of data often exceeds 1 terabyte and it takes too much time to calculate the hash code for such heavy files.

So the power of multiprocessors on a GPU machine can be exploited to parallelize the hashing algorithms which can lead to its fast and secure implementation.

RSHA-1

Most of the computation in SHA-1 hashing algorithm takes place in its compression function which accounts for the majority of the time consumption by the algorithm. To speed up the whole process, a new algorithm Redesigned SHA-1(RSHA-1)[2] is proposed which is based upon recursive framework in which the standard SHA-1 algorithm is tried to parallelize. In this the chain dependency in the function is removed by recursively calling the compression function with a single Initialization Vector (IV) i.e. H_0 and generating the hash value.

RSHA-1's compression function is data parallel. In order to gain the maximum degree of concurrency, multiple compression functions can be executed in parallel on different processing units simultaneously. The design of the algorithm is based on SIMD architecture performing both recursive and data decomposition in order to make the optimum use of resources. The structure of RSHA-1 framework is illustrated in "Fig. 1" and described in Algorithm 1.



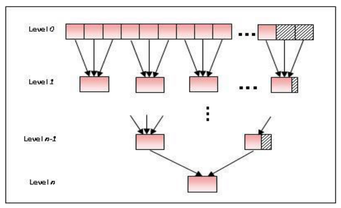


Figure 1: RSHA-1 framework

In RSHA-1 each message block has the same initial vector H_0 to generate hash value. So the calculation of $H(M_{(i)})$ for each $M_{(i)}$ is fully parallelized without dependences, and then reducing H_i using $H(H_i)$ as similar to $H(M_{(i)})$ to produce an output hash value.

1. Append padding bits to the original message if required.
2. Calculate the length of the original message and append it to the end of the message
3. Split new message space M into blocks M(i) of 512 bits each.
4. Cut out the dependences among hash values Hi; setting H0 (IV) for all blocks M(i)
5. Calculate $H(M_{(i)})$ of 160 bits for each $M^{(i)}$ to generate $H_1,\ldots, H_i,\ldots,H_p$.
6. Combine $H_1,\ldots, H_i,\ldots,H_p$ to form a single message.
7. Use recursive hash as similar to $H(M_{(i)})$ to produce an output hash value with 160-bits if required.

Algorithm 1: RSHA-1 Algorithm

SECURITY MEASURES

- *Pre-Image Resistance*: Given a hash h, it is infeasible to generate M such that H(M)=h.
- *Weak Collision resistance:* Given M, it is hard to find another message, M', such that H(M)=H(M'). Functions that lack these properties are vulnerable to pre-image and second-pre-image attacks.
- *Collision resistance:* It is not easy to find two different messages, M and M' with the same hash value, such that H(M) = H(M').
- *Partial Pre-Image Resistance*
- *Non- Correlation Resistance*

The RSHA-1 has Avalanche Effect and satisfies all the above stated properties of CHF to make it useful for the authentication purpose.

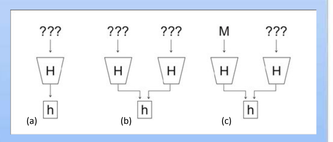


Figure 2: (a) Pre-image Resistance (b): Weak Collision Resistance (c) Collison Resistance

PERFORMANCE ANALYSIS

In order to assess the new algorithm for CHF applications, the code was implemented in CUDA and executed on a GPU machine with the following configuration:

The test environment was a server with the following configuration:

- **Processor:** Intel Core 2 Duo CPU E7500@2.93GHz X 2
- **RAM:** 3.0 GiB
- **O.S.:** Ubuntu Linux 11.04 32 bit
- **Platform:** GCC Infrastructure- gcc 4.5.4
- **GPU:** Tesla C2075
- **API:** CUDA 5.5

EXPERIMENTAL RESULTS

The thread size and block size in the code was decided on the basis of the file size. The experiment was executed on varied file sizes: random files of sizes 512KB, 1MB, 2MB, 4MB and 8MB were taken for the same. It was observed that simple implementation of code on CUDA when compared with implementation on OpenMP API (on the same CPU) and with standard SHA-1 could not give good results as shown in graph below.

Sr. No.	File Size	Execution Time(in ms)		
		SHA-1	RSHA-1 on CPU	RSHA-1 on GPU
1	512 KB	0.01426	0.01307	0.05917
2	1 MB	0.02829	0.02476	0.05991
3	2 MB	0.05836	0.05064	0.06219
4	3 MB	0.08543	0.07175	0.06989
5	4MB	0.11312	0.09544	0.07898
6	8 MB	0.21336	0.17754	0.13363

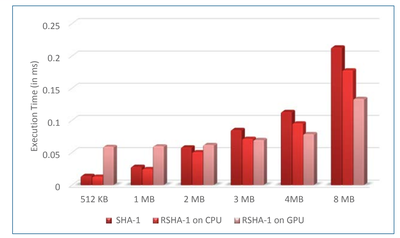


Figure 3: Execution time of SHA-1 and RSHA-1 for different file sizes

The reason for the non performance was analyzed by using the profiler and was found that time of memory copies from Host to Device and Device to Host was the main overhead of the code. So a new technique of Pinned Memory was used to optimize the code and save memory flaws.

CODE OPTIMIZATION

PINNED MEMORY:

In Pinned Memory concept there is no need to copy the data from Host Memory to Device Memory. Pinned memory is used as a staging area for transfers from the device to the host. The cost of transfer between pageable and pinned host arrays can be avoided by directly allocating the host arrays in pinned memory.

Figure 4 shows the results after using Pinned Memory concept. The speedup achieved now is approximately 3x.

Sr. No.	File Size	Execution Time(in ms)		
		SHA-1	RSHA-1 on CPU	Optimized RSHA-1 on GPU
1	512 KB	0.01426	0.01307	0.01001
2	1 MB	0.02829	0.02476	0.01579
3	2 MB	0.05836	0.05064	0.03137
4	3 MB	0.08543	0.07175	0.04167
5	4MB	0.11312	0.09544	0.05026
6	8 MB	0.21336	0.17754	0.08036

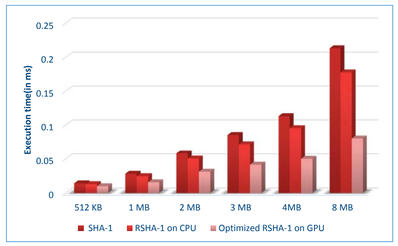


Figure 4: Performance Comparison after code optimization

FUTURE WORK

The main focus now is to more optimize the implementation of RSHA-1 on multi-core machines on GPUs to explore the power of GPUs. Undoubtedly the RSHA-1 is meant for large data flows. More speedups can be achieved on large chunks of data on GPU machines. The future course of work will include the usage of Shared Memory concept as well so as to have more control on memory related pitfalls. It would also include analyzing the collision resistance of the algorithm in adverse conditions.

REFERENCES

[1] Stalling W., Cryptography and Network Security, 3rd ed., New Jersey: Pearson Education International, 2003.

[2] N. Kishore and B. Kapoor, "An efficient parallel algorithm for hash computation in security and forensics applications," in *Advance Computing Conference (IACC), 2014 IEEE International*, 2014, pp. 873-877.