Parallel Implementation of RSHA-1 for Cryptographic Applications using CUDA

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

The SHA-1 hash algorithm is widely used for identification and authentication purpose. In digital evidence forensics, hashing is used for verification of integrity and correctness of evidence. But there is a huge computational complexity and delay due to its data dependency in the critical path iterations. To deal with the problem a new algorithm RSHA-1 is proposed which serves the same purpose as SHA-1 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 hash to break the chain dependencies of the standard hash function. We discuss here the theoretical foundation for the work and the performance implications. The algorithm is implemented using the OpenMP API as well as CUDA on GPU's.

MOTIVATION

Traditional Cryptographic hashing algorithms like MD5, SHA-1, SHA-2 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) 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. 2" and described in Algorithm 1.

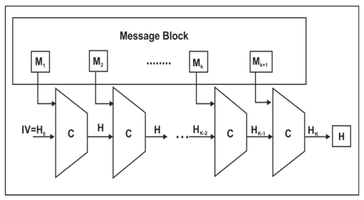


Figure 1: SHA-1 framework

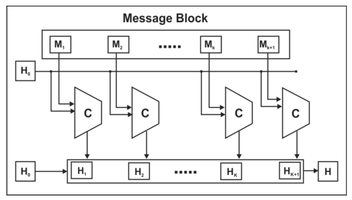


Figure 2: RSHA-1 framework

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

COMPLEXITY CALCULATION

Let the Complexity of calculating $H(M_{(i)})$ for hash value of each message blocks is $Cmplx(H(M_{(i)}))$, and the number of original message blocks as p, with n (512)- bit, m (160)- bit be length of hash value and k= log(16/5)p.

So the computational complexity for RSHA-1 can be given by $f(R)$:

$$f(R) = (p + n\_call) \,.\, Cmplx(H(M_{(i)})),$$

where n_call is number of calls to $H(M_{(i)})$ after reduction and is given by:

$$n\_call=[(m/n)^1.p] + [(m/n)^2.p]+\ldots\ldots+ [(m/n)^k.p]$$

Although RSHA-1 has more computations than that of standard SHA-1 but RSHA-1 is more secure than standard SHA-1 due its recursive nature.

PERFORMANCE MEASURES

Let the time taken by standard SHA-1 to calculate hash value of each message block = $T(M_{(i)})$

The processing time of standard SHA-1 is given by: $T_s = p.\, T_{M(i)}$

and the processing time of RSHA-1 is given by: $T_r = [(\log_{n/m} p)+1].\, T_{M(i)}$

So, **Speedup** $= T_s / T_r$

$= \Theta\,(p/(k+1))$

and **Efficiency** = Speedup / (no.of processors)

$= \theta\,(1/\log(16/5)\,p + 1)$

Where p is number of original message blocks with n (512)- bit, m (160) is length of hash value and k= log(16/5)p.

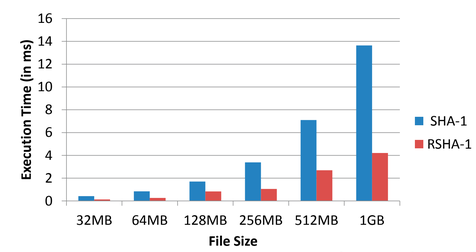
Hence, as the size of data and the number of processors increases, the performance shown by the algorithm is better than the standard SHA-1.

EXPERIMENTAL RESULTS

In order to assess the new framework properly, the algorithm was implemented in both using OpenMP API and CUDA on GCC infrastructure. Firstly the experiment was conducted in OpenMP and following readings were found:

The test environment was a server with the following configuration:

- Processor: AMD FX(tm)-8120 8-Core Processor X 8, 3.10 GHz
- RAM: 4.00 GB
- Kernel .: Ubuntu Linux 11.04
- GCC: gcc- 4.5.4
- Parallel Code: OpenMP 3.1



Sr. No.	File Size	Execution Time(in ms) on 8-Core machine	
		SHA-1	*RSHA-1*
1	32MB	0.42478	0.13172
2	64MB	0.84093	0.26237
3	128MB	1.70663	0.83703
4	256MB	3.38537	1.05623
5	512MB	7.10087	2.69129
6	1GB	13.64857	4.21023

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

"Fig. 3" depicts these results graphically showing the improvement of 3.5x when compared with sequential SHA-1 using OpenMP API. The following table shows the results of same code on different file and different core machine, so as to analyze the speedup gained.

Sr. No.	File Size	Time Consumption (in ms) by RSHA-1 on			
		2 Core	*4 Core*	*6 Core*	*8 Core*
1	32MB	0.46171	0.24848	0.15824	0.13172
2	64MB	0.97682	0.48278	0.31533	0.26237
3	128MB	1.9325	0.94026	0.64362	0.83703
4	256MB	3.94609	1.90022	1.27396	1.05623
5	512MB	7.76	3.88596	2.55713	2.69129
6	1GB	15.63747	7.78528	5.15977	4.21023

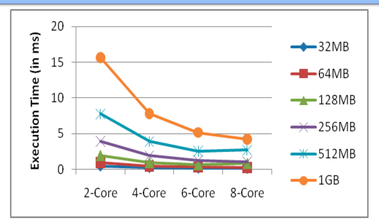


Figure 4: Performance Comparison in terms of file size and cores

In OpenMP, it was observed that RSHA-1 can give good results for producing hash code for the heavy files. So, we compiled the serial implementation of SHA-1 on CPU and GPU first and found that with nvcc, code gives a speedup of around 5x (runtime 82 sec on CPU , runtime 19 sec on nvcc compiler) on CUDA GPU (GTX 570, Fermi Architecture). Now we are implementing RSHA-1 in CUDA on a GPU machine to achieve high throughput. Although, RSHA-1 is not giving any speedup right now as compared to SHA-1 due to overhead of copying the data. But by introducing the concept of pinned memory in RSHA-1 implementation the expected speedup is 13x to 15x.

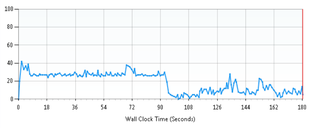


Figure 5: Execution Time of SHA-1 on

The main focus now is to optimize the implementation and analyze the performance gain by 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 also include analyzing the collision resistance of the algorithm in adverse conditions as well. The target is to develop a parallel hash function to resolve the problem of computational delays and power consumption.

GOAL VALIDATION

It is to be noted that the hash values generated by standard SHA-1 and RSHA-1 will never be the same as both work on different frameworks. The RSHA-1 will still have Avalanche Effect and will satisfy weak collision resistance and strong collision resistance properties of CHF to make it useful for the authentication purpose.