MD5 Calculation and Decryption Using CUDA on GPU

LEI Mingshan, JIANG Yanjun, GAO Zhanchun
(*School of Computer, Beijing University of Posts and Telecommunications, Beijing*, 100876)

Abstract: It is difficult to get the original information from a MD5 hash since MD5 is a one-way hash algorithm. MD5 decryption is based on MD5 calculation for brute force attack, which requires great computing resources. This paper presents an approach for MD5 calculation and decryption on GPU, which has high concurrency. The CUDA program performed on a PC with NVIDIA GTX 560TI graphics card. The experimental result has shown that the calculating speed is 150 million words per second, increasing from 10 to 20 times compared to program run on CPU (Core i7-950 @3.07GHz). GPU has great potential in future general-purpose computing and concurrent computing.
Key words: MD5; GPU; CUDA; multithreading; parallel computing

0 Introduction

The MD5 message-digest algorithm is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. MD5 has been utilized in a wide variety of security applications, and is also commonly used to check data integrity. MD5 digests have been widely used across the world for security applications [1]. For example, MD5 can provide some assurance that a transferred file has arrived intact.

The 128-bit MD5 hashes are typically represented as a sequence of 32 hexadecimal digits. The following phase demonstrates a 10-byte ASCII input and the corresponding MD5 hash:

MD5 ("Hello MD5") = E5DADF6524624F79C3127E247F04B548

The hash is totally different if we change only one character:

MD5 ("Hello Md5") = F940261A5B6321092532EA909D3973C0

Talking about MD5 security, MD5 hash function is severely compromised. It has since been shown that MD5 is not collision resistant; as such, MD5 is not suitable for applications like SSL certificates or digital signatures that rely on this property. And most U.S. government applications now require the SHA-2 family of hash functions to replace MD5 [2].

MD5 hash is a one-way function and nowadays it is impossible to decrypt MD5 using an algorithm directly. Brute force decryption is used in most of MD5 decryption situations. This method consumes large amounts of computing resources for large password space and it's not practical for common computer [3]. General-purpose computing on graphics processing units (GPGPU) is the utilization of a graphics processing unit (GPU), and it has provided us a way to parallelize original programs based on the multi-core architecture of GPU. The dominant proprietary framework is NVIDIA's CUDA.

In this paper, we perform parallelism for MD5 calculation on CPU and GPU. CUDA is used to develop a program running on GPU. The experimental evaluations have shown us strong ability of parallel computing of GPUs and their potentials in future general-purpose computing.

1 MD5 Algorithm

Step 1: Padding. The input message is padded so that its length is divisible by 512.

Step 2: Initialization. The main MD5 algorithm operates on a 128-bit state, divided into four 32-bit words, denoted A, B, C and D. These are initialized to certain fixed constants.

Brief author introduction:Lei Mingshan(1988-),Male,Master,Computer Networks
Correspondance author: Jiang Yanjun(1967-),Male,Associate Professor,Computer Networks. E-mail: jiangyanjun0718@163.com

Step 3: Calculation. MD5 consists of 64 basic operations, grouped in four rounds of 16 operations based on a non-linear function F. There are four possible functions F, which is shown in Fig. 1; a different one is used in each round [4].

$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

$\oplus, \wedge, \vee, \neg$ denote the XOR, AND, OR and NOT operations respectively.

Fig. 1 Non-linear function F

This step includes most of the calculations in MD5 algorithm. We will focus on this step for parallelism and multithreading.

Step 4: Finalization. End an MD5 message-digest operation, and the message digest is stored in A, B, C and D.

2 MD5 Calculation on CPU Using OpenMP Multithreading

Classic MD5 Algorithm's sample code is shown in RFC2312. In this chapter we perform multithreading MD5 calculation on CPU. We choose OpenMP to accomplish the task.

OpenMP (Open Multi-Processing) is an API that supports multi-platform shared memory multiprocessing programming in C, C++ and FORTRAN on most processor architectures and operating systems.

The pragma omp parallel is used to fork additional threads to carry out the work enclosed in the construct in parallel. The program used a recursive function to generate words based on the given character set and the word length. After that each word is given to one thread to calculate its MD5 hash and parallelism is implemented.

The key parallel pseudo code is shown below:

```
Calc(word, index)
{
    if (index == 0) {
        MD5calculate(word)
    } else {
#pragma omp parallel for num_threads(4)
        for (i = 0 to char_set length) {
            word[index - 1] = char_set[i];
            Calc(word, index - 1);
        }
    }
}
```

[Pseudo code of parallel MD5 hash calculation]

The performance test on CPU is shown in Fig. 2.

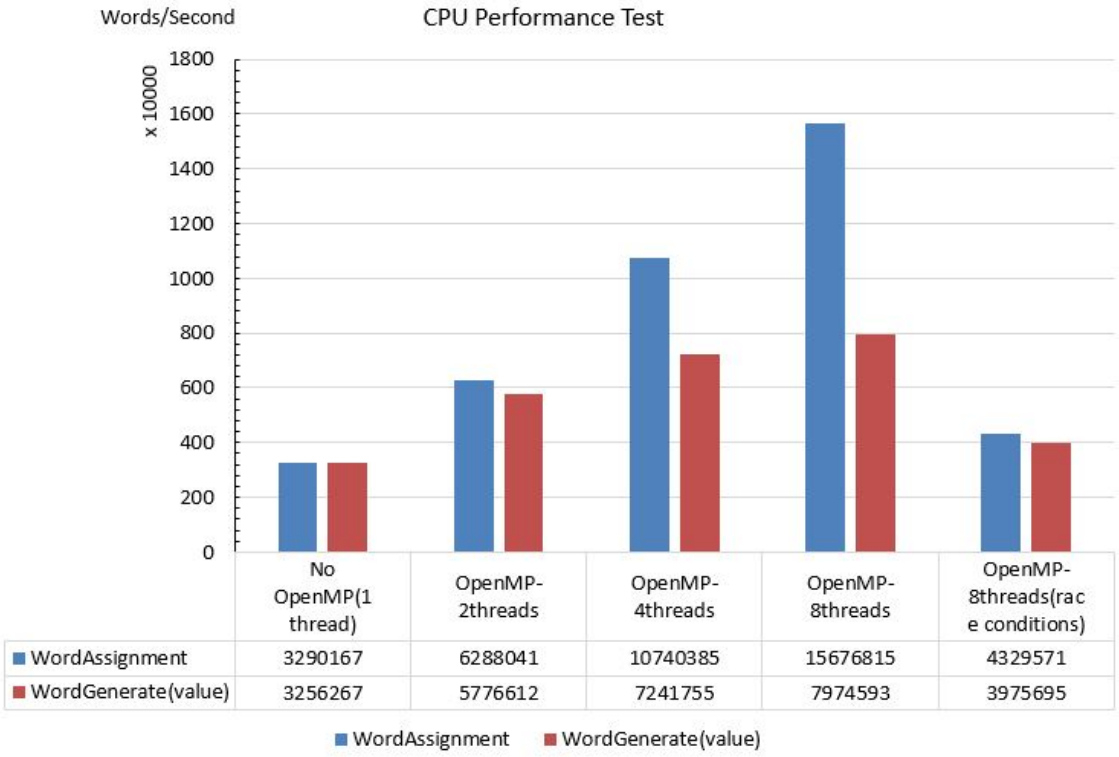

Fig. 2 Performance Test on CPU

We have designed two functions for performance test. One is a recursive function shown before and the other is to generate a word based on values. We set OpenMP for 2 threads, 4 threads and 8 threads for test. The performance will slow down if race condition happens. With the highest concurrency, the testing CPU was able to calculate about 10 million words per second.

3 MD5 Decryption Using CUDA on GPU

3.1 GPU and CUDA

In recent years, driven by the insatiable market demand for real-time, high-definition 3D graphics, the programmable Graphic Processor Unit (GPU) has evolved into a highly parallel, multithreaded, multi-core processor with tremendous computational power and high memory bandwidth [5]. The new generation of NVIDIA's flagship graphics card - GeForce GTX Titan has reached 4500 theoretical GFLOP per second, almost 10 times over Intel's Sandy Bridge CPU, which is about 480 GFLOPS per second. CPU and GPU emphasize on different usage: CPU is good at data caching and flow control while GPU is specialized for compute-intensive, highly parallel computation - exactly what graphics rendering is about.

GPU is especially well-suited to solve problems that can be expressed as data-parallel computations, such as calculating MD5 hashes. The same program of MD5 calculating runs in parallel on many data elements (words) - with high arithmetic intensity. It turns out performance of parallel computations of CPU is higher than that of CPU.

CUDA is a general-purpose parallel computing platform and programming model that leverages the parallel compute engine in NVIDIA GPUs. CUDA is designed to support various languages and application programming interfaces [5].

With the key features of GPU and CUDA, GPU is an appropriate solution to solve MD5 calculating and decryption problems, where each CUDA thread is able to calculate MD5 for one word and GPU is able to get thousands of thread run in parallel at the same time.

3.2 Algorithms in CUDA on GPU

3.2.1 Definitions

According to thread hierarchy of CUDA, thread running on GPU are organized into blocks. A kernel function can be executed by multiple equally-shaped thread blocks, so that the total number of threads, denoted as P, is equal to the number of threads per block (up to 1024 threads) multiple the number of blocks [5]. Assuming that there are totally N passwords, the kernel should run N/P times to finish the task [6].

For MD5 decryption, we are given the passwords' maximum length and a character set which includes all possible characters that could be shown in one password. A MD5 hash is given for us to find out the original password [7].

First we should check out the password space. The password space is defined as follows:

The character set is denoted as C_S. Here is one example C_S = {a-z, A-z, 0-9}.

The length of one C_S, denoted as N, is the number of element of the C_S. For C_S = {a-z, A-z, 0-9}, N = 62.

The maximum length of password is denoted as M_L.

M_L subsets will be generated and S_i is the set of passwords having the length of i which i is no more than M_L.

For example, assuming that

C_S = {a-z, A-z, 0-9} N = 62 M_L = 4 P = 1024

Information about this C_S is shown in Tab. 1:

Tab. 1 C_S Information

i	*Si(word quantity)*	*Processing Times*
1	62	1
2	3844	4
3	238328	233
4	14776336	14431

3.2.2 Algorithm in CUDA

Algorithm 1: The following code shows us how the passwords in a certain password space is divided and calculated in parallel on GPU. First we get a subset of passwords of length i. Each time P threads could run in parallel on GPU, so we get values of 0 to P-1, P to 2P-1, 2P to 3P-1, etc. Each value represent a password. Second we generate the password based on the value. Finally we calculate the MD5 hash in one thread on GPU and compare it to the target to get the result.

```
for (i = 1 to M_L) {
    num = length(Si); // the total number of passwords in Si (length of i)
    count = num / P + 1;
    for (j = 0 to (count - 1)) {
        base_num = j * P; // the set's first password's value
        for (index = 0 to (P-1) execution in parallel on GPU) {
            password = GeneratePassword(base_num + index, i); // generate a password according to the value and word length
            md5 = md5_calculate(device, password); // launch the kernel of the CUDA device
            compare(md5, target_md5); // compare the md5 to the target we find
```

}

}

Algorithm 2: Calculate the word based on value. The following code shows how it works. Assume the password's length is n:

Password = GeneratePassword(value, n)=$C_0C_1C_2……C_{n-2}C_{n-1}$

C_{n-1}=C_S[value mod N] C_{n-2}=C_S[(value div N) mod N]…

C_1=C_S[(value div N^{n-2}) mod N] C_0=C_S[(value div N^{n-1}) mod N]

For example, assuming that C_S={abcd0123} N=8, we get:

Value = 0, n = 5, word = aaaaa

Value = 1, n = 5, word = aaaab

Value = 5934 = $1*8^4 + 3*8^3 + 4*8^2 + 5*8^1 + 6*8^0$, n = 5, word = bd012

4 Performance Test and Evaluations

The program implemented algorithms in CUDA and was deployed on test PC with one NVIDIA graphics card GTX560Ti and CUDA 5.5 SDK installed.

The Performance test for GPU used the following parameters: C_S = {a-z, A-Z, 0-9}, N = 62, maximum password length is 5, number of threads per block is 512, and the number of blocks is 384. The GPU on GTX560Ti has 8 multiprocessors and each multiprocessor has 48 CUDA Cores, and 384 CUDA Cores could be used in CUDA program in total.

The program written in CUDA C ran in parallel on the above testing environment. The calculating speed was highly increased compared to CPU-OpenMP versions. GPU could calculate 155 million words per second, about 10 times of CPU when CPU runs in 8 threads in parallel, which is 15 million hashes per second.

Tab.2 shows time for calculating in password space on GPU for C_S = {ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789}.

Tab. 2 Time for Calculating in password space

Max Word Length	*Password Number*	*GPU Time*	*CPU Time*
1	62	1.268ms	0.234ms
2	3,906	2.281ms	2.510ms
3	242,234	4.631ms	32.19ms
4	15,018,570	0.1066s	0.9890s
5	931,151,402	6.049s	68.33s
6	57,731,386,986	5m 12s	1h 18m
7	3,579,345,993,195	5h 26m	3d 8h
8	221,919,451,578,091	13d 8h	187d

5 Conclusion

In this paper, we implemented MD5 calculation and decryption algorithm in parallel on CPU and GPU and compared the experimental results between them. The result showed that GPU has greatly increased the speed of calculating MD5 hashes and showed higher concurrency and higher performance than CPU.

Further work could be done to improve the performance of program in CUDA. We plan to do some optimizations to maximize GPU instruction throughput and utilization to get a higher speed to calculate MD5.

References

[1] Wiki. MD5 Algorithm Wiki[OL].[2013]. http://en.wikipedia.org/wiki/MD5
[2] NIST.gov-Computer Security Division. NIST's policy on hash functions[OL].[2010]. http://csrc.nist.gov/groups/ST/hash/policy.html
[3] Feng Wang, Canqun Yang, Qiang Wu, Zhicai Shi. Constant memory optimizations in MD5 Crypt cracking algorithm on GPU-accerlerated supercomputer using CUDA[A]. Feng Wang. 2012 7th International Conference on Computer Science & Education[C]. Melbourne: IEEE, 2012. 638~642
[4] Network Working Group. RFC1321:The MD5 Message-Digest Algorithm[L].[1992.04]. http://www.ietf.org/rfc/rfc1321.txt
[5] NVIDIA. CUDA C Programming Guide[OL].[2013]. http://docs.nvidia.com/cuda/index.html
[6] Duc Huu Nguyen, Thuy Thanh Nguyen, Tan Nhat Duong, et al. Cryptanalysis of MD5 on GPU Cluster[A]. Duc Huu Nguyen. 2010 International Conference on Information Security & Artifical Intelligence(ISAI 2010)[C]. Chengdu, China: IEEE, 2010. 910~914
[7] Hongwei Wu, Xiangnan Liu, Weibin Tang, et al. A Fast GPU-based Implementation for MD5 Hash Reverse[A]. Hongwei Wu. 2011 International Conference on Anti-Counterfeiting, Security and Identification(ASID 2011)[C]. Xiamen, China: IEEE, 2011. 13~16

基于 GPU CUDA 的 MD5 计算与破解

雷鸣山，蒋砚军，高占春
（北京邮电大学计算机学院，北京，100876）

摘要：由于 MD5 算法是单路哈希算法，要从一个 MD5 的哈希值中提取出原始的数据是非常困难的。MD5 破解通常是基于 MD5 的计算算法进行暴力破解，需要大量的计算资源。本文提出了一种基于 GPU 高并行性的 MD5 计算与破解的方法。CUDA 程序运行在配备有 NVIDIA GTX 560TI 显卡的 PC 上，实验数据表示基于 GPU 的 CUDA 程序的计算速度可以达到每秒 1.5 亿单词，相对于运行于 CPU（Core i7-950 @3.07GHz）上的程序有 10 到 20 倍的提升。GPU 在未来的通用计算和并行计算方面具有非常巨大的潜力。

关键词：MD5 算法；GPU；CUDA；多线程；并行计算
中图分类号：TP309.7