MANAS Journal of Engineering

MJEN

Volume 5 (Issue1) (2017) Pages 53-63

Comparative Analysis of Integer Factorization Algorithms Using CPU and GPU

Gulida Kimsanova
Department of Computer Engineering, Kyrgyz-Turkish Manas University, Bishkek city, Kyrgyz Republic
gulida.kms@gmail.com

Rita Ismailova
Department of Computer Engineering, Kyrgyz-Turkish Manas University, Bishkek city, Kyrgyz Republic
rita.ismailova@manas.edu.kg

Rayimbek Sultanov
Department of Computer Engineering, Kyrgyz-Turkish Manas University, Bishkek city, Kyrgyz Republic
rayimbek.sultanov@manas.edu.kg

Received: 24.01.2017 ; Accepted:01.05.2017

Abstract: *In this work we have evaluated the running time of four integer factorization algorithms, namely, trial division algorithm, Fermat algorithm, Pollard rho and Brent algorithms. Implementation of these algorithms was performed in three ways on c programming language, on c++ programming language, using GMP 6.0.0 library and on CUDA architecture to run on GPU. Results showed that Fermat algorithm and trivial division algorithm had the fastest running time in parallel implementation on CUDA architecture. The difference of running times between CUDA implementation and GMP implementation was up to 10 times. The difference between c and c++ implementation was mainly due to difference in these programming languages.*

Keywords: *Integer factorization, GPU, GMP, trial division algorithm, Fermat algorithm, Pollard rho algorithm, Brent algorithm.*

Сравнительный анализ алгоритмов целочисленной факторизации при работе на Центральном ЦПУ и ГПУ

Abstract: *В данной работе была проведена оценка времени работы четырех алгоритмов целочисленной факторизации, а именно тривиального алгоритма факторизации, алгоритма Ферма, алгоритмов Полларда Ро и Брента. Реализация этих алгоритмов была выполнена тремя способами: на языке программирования с, на языке программирования с ++, используя библиотеку GMP 6.0.0 и на архитектуре CUDA для работы на ГПУ. Результаты показали, что алгоритм Ферма и тривиальный алгоритм деления имели самое быстрое время при параллельной реализации в архитектуре CUDA. Разница между временем выполнения факторизации при реализации на CUDA и реализации на GMP доходила до 10 раз. Разница между временем выполнения факторизации при реализации на с и с ++ была в основном связана с различиями в этих языках программирования.*

Keywords: *Целочисленная факторизация, ЦПУ, GMP, тривиальный алгоритм факторизации, алгоритм Ферма, алгоритм Полларда Ро, алгоритм Брента.*

1. INTRODUCTION

In 1976 Diffie and Hellman introduced the idea of Public Key Cryptosystems [7]. Unlike symmetric key cryptosystems such as DES [17] and AES [24], in public key algorithms two keys are used – one for encryption and the other is for decryption. Two year later, in 1978, Rivest, Shamir and Adleman proposed a working system based on this idea – the RSA algorithm [25]. The security of the RSA algorithm, which is commonly used in many software products, is based on the integer factorization problem: if the large number, taken as a modulus, is factored, the algorithm is broken. Integers, used in the algorithm, are very large and practically it is very hard to find its prime factors. Thus, factorization takes very long time even when computers with high performance are used. And although the integer factorization problem has been there for more than 2000 years, yet, no efficient algorithm for factoring large primes was proposed [16]. According to [16], the recent progress in factoring is mainly based on the methods which allow to use many computers and perform for factorization in parallel. However, the larger the number, the more computers need to work in parallel to factor.

The parallel processing possibility, offered by high performance GPU (Graphics Processing Unit), was quickly adopted by several non-graphic applications [21]. In [6] it was proposed to use of GPUs for symmetric key ciphers, however the study indicate that given that level of API, GPUs were not suitable to be used for AES. As for public key cryptosystems, the parallel computing became one of the widely used methods to enhance the performance since they involve complex calculations on large numbers, which requires large memory and power consumption [26]. With the fast grow rate of GPU [14] and due to its high parallel processing power, GPU came to be considered more suitable [30] and fast [13], [10], [2], to perform calculations in public key cryptosystems than CPU.

In this work we implemented four integer factorization algorithms utilizing the GPU and analyzed their speed acceleration compared to the implementation on CPU. In the next section basics of implemented integer factorization algorithms are presented. Next, state of the art GPU based implementations are reviewed followed by our results and conclusion of the paper.

2. FACTORIZATION ALGORITHMS

Trial Division

Trial division is the simplest, yet, the most time consuming algorithm for integer factorization. As its name suggests, in this algorithm the number is consequently divided by all numbers less than (or equal to) its square root, since, according to the theorem, if *s* and *t* are nontrivial factors of a number *N* with $s \leq t$, then $s \leq \lfloor \sqrt{N} \rfloor$. If no such *s* found, then *N* is a prime number.

The algorithm can be optimized by using the list of primes less than (or equal to) its square root of *N* [5].

Fermat Factorization Algorithm

In 1643 Fermat introduced a new idea to consider a number as a difference of squares [11], i.e., to find a prime factor of the form $x + y$ and $x - y$, he proposed to find number *x* and *y* such that

$N = x^2 - y^2$. The complexity of the algorithm is of the order $O\left(cN^{1/2}\right)$. Thus, since it is the opposite of the trial division algorithm and starts at square root of the number to be factored, Fermat factorization algorithm is effective when the difference between prime factors of the number is relatively small [23], [28].

Pollard rho Algorithm

Another idea to factor large prime *N* was proposed by Pollard [22] and utilizes polynomial iteration modulo *N*. Algorithm starts with random integer x_0, then an iteration is defined as follows:

$$x_{n+1} = f(x_i) (\mathrm{mod}\, N)$$

Usually the polynomial used in the algorithm is $f(x_i) = x_i^2 + a$, where $a \neq 0, -2\ mod\ N$. The idea is to find integers x_i and x_j which are congruent modulo *N*. If such integers are found, then $\gcd(|x_j - x_i|, N)$ is a prime factor *p* of *N*. It is suggested that $j = 2i$. The expected run time to find a prime factor *p* of *N* is $O\left(p^{1/2}(\log N)^2\right)$.

Brent Algorithm

In the Pollard rho algorithm, the sequence $x_i, x_{i+1}, x_{i+2}, \ldots$ will repeat after some period, since x_i's are element of a finite set modulo *p*. In [4] Brent proposed modification of the Pollard rho algorithm, using Floyd's cycling method to detect iteration. Unlike Pollard rho algorithm, where elements x_i and x_j with $j = 2i$ are compared, in Brent algorithm is was proposed to compare elements x_n and x_m, with *m* being the largest integral power of 2 less than *n*. Due to this modification, the algorithm runs up to 25% faster than the original Pollard rho algorithm [4].

3. RELATED WORKS

In [27] compared the performance of three algorithms on CPU and GPU and concluded that an idealized GPU can deliver better performance. However, they also found that two modern quad-core CPU sockets approximately match one or two GPUs in performance.

In [29] CPU-based Pollard's *p*-1 Factorization Algorithm (CPFA) and GPU-based Pollard's *p*-1 Factorization Algorithm (GPFA) was tested using 173,057,268 prime numbers ranged among 32-bit integers. GPFA algorithm was implemented on GTX-260, S1070, and C2050 GPU architectures. To perform operations on large primes, a custom integer system (CIS) was designed and implemented for both CPFA and GPFA. In this system, the unsigned integer type was used to store the integer each digit by one byte. Results showed the 1197.5x average speedups by comparing GPFA with CPFA under various platforms for RSA-64 integers.

In [3] developed a software package, where Number Theory Algorithms, including integer factorization, were implemented on NVIDIA using CUDA (Compute Unified Device

Architecture) technology [18]. For the integer factorization the elliptic curve factorization algorithm by Lenstra, which can be considered as a generalization of Pollard's $p - 1$ and Williams' $p + 1$ methods [12], was implemented. Results of this work showed that compared to the implementation by using Message Passing Interface, the GPU-based implementation of algorithm on GMP showed commensurable results with very good performance. [1] showed that compared to implementation on 8-core CPU, a 448-core GPU implementation showed up to 45x speedup and 88% energy saving. The integration of GPU and current Number Field Sieve software was used for factorization of 768 bit integer by [15] and resulted in less time consumption.

4. METHOD

4.1. Materials

In this work we have evaluated the running time of four integer factorization algorithms, namely, trial division algorithm, Fermat algorithm, Pollard rho and Brent algorithms. The aim was to compare the running time of algorithms on central processing unit (CPU) and graphical processing unit (GPU). For the experimental part of this work, two computers with the following characteristics were used:

Table 1. Characteristics of computers used in this work.

	1st computer	*2nd computer*
Processor	Intel(R) Core(TM) 2 Quad CPU Q8300 @ 2.50GHz	Intel(R) Core(TM) i3-2310M CPU @ 2.10GHz
RAM	4,00 GB	4,00 GB
Operating system	64-bit Operating System	64-bit Operating System
GPU	-	NVIDIA GeForce GT 630

On both machines, the integrated development environment by Microsoft - Visual Studio 2010 package was installed. Next, the GMP 6.0.0 Multiple Precision Arithmetic Library was added, following the instructions given in [8]. In one of the machines, the CUDA architecture, which allows use of graphical processing unit, was installed. The CUDA parallel computing architecture was used since it allows computation using graphic processors like GeForce, ION, Quadro and Tesla [20].

Programs were developed using c, c++ and CUDA programming languages following.

4.2. Method

In this work, 4 factorization algorithms, namely, trial division, Fermat method, Pollard rho and Brent methods were implemented. Algorithms were implemented in three ways. Fist implementation is an original implementation on c programming languages and uses no libraries. The second implementation was done on c++ using GMP (GNU Multiple Precision Arithmetic Library) library [8], and the third one on GPU using CUDA (Compute Unified Device Architecture) platform by NVIDIA [19].

Next, the running time of selected 4 algorithms were recorded and analyzed. Analysis included two cases: first, we compare running time of algorithms based on the input size, and next, based on the distance between prime cofactors.

Since there was a limitation on the computational power and because no open source libraries are present to work with large integers on CUDA, no large integers were used as an input for implementation on GPU.

5. RESULTS

5.1. Factorization of integers with close factors

First, the test were conducted on the numbers, whose factors are close to each other. First, the running times of algorithms developed using GMP library and running on CPU were obtained. Next, the same numbers were factored using algorithms developed on CUDA and working on GPU. Last, the variants of algorithms developed on c programming language and running on CPU were executed. Results are presented in the Figure 1 for the trivial division algorithm, in Figure 2 for Fermat algorithm and in the Figure 4 for Brent algorithm. Result for Pollard rho and Fermat algorithms are omitted since the running time was too small.

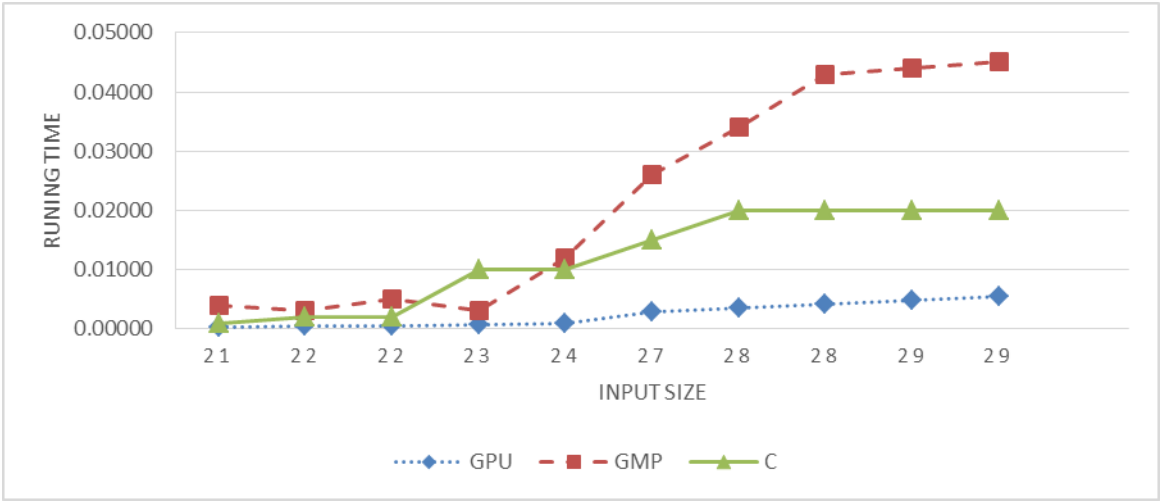

Figure 1. Running time of Trial Division Algorithm

As it can be seen from the figure, the best running time was observed in the CUDA implementation of the trial division algorithm on GPU, the worst – on the implementation of the algorithms using GMP library. The CUDA implementation's running time was almost 10 times faster than those of c++ implementation with GMP library. The difference between c and c++ implementation was smaller. Thus, the parallel implementation works well for the trial division algorithms.

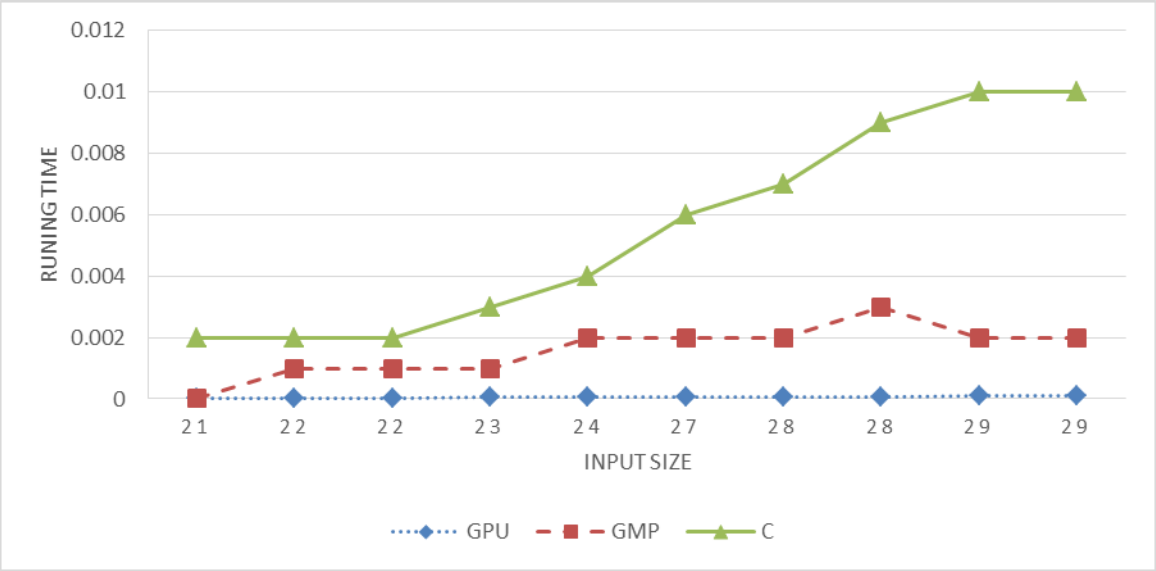

Figure 2. Running time of Fermat algorithm

A key feature of Fermat factorization algorithm is that the algorithm works more quickly if the difference between factors is small, but as the distance between the factors gets larger, the algorithm requires more time. Also, the algorithm works fast for factoring small numbers. The best running time in the implementation of Fermat algorithm was again observed in the CUDA implementation on GPU, the worst – on the implementation of the algorithms using c programming language. However, comparison of c and c++ implementation with GMP library showed that c implementation of the algorithm worked slower.

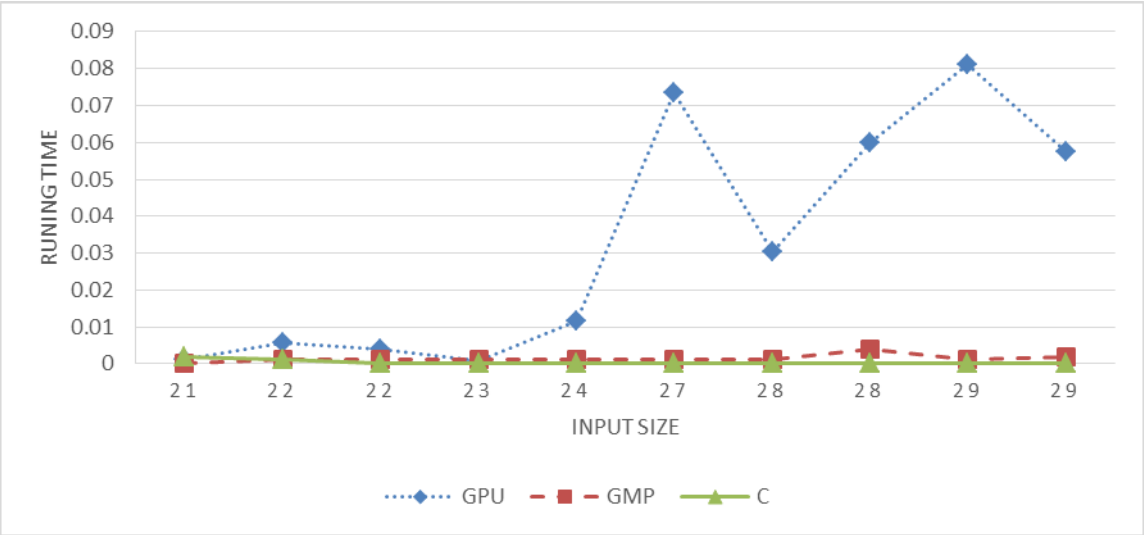

Figure 3. Running time of Pollard rho Algorithm.

However, implementation of the Pollard rho algorithm showed the best running time on c implementation, while the worst running time was observed for CUDA implementation on GPU. The difference between c implementation and c++ implementation using GMP library was small for this algorithm.

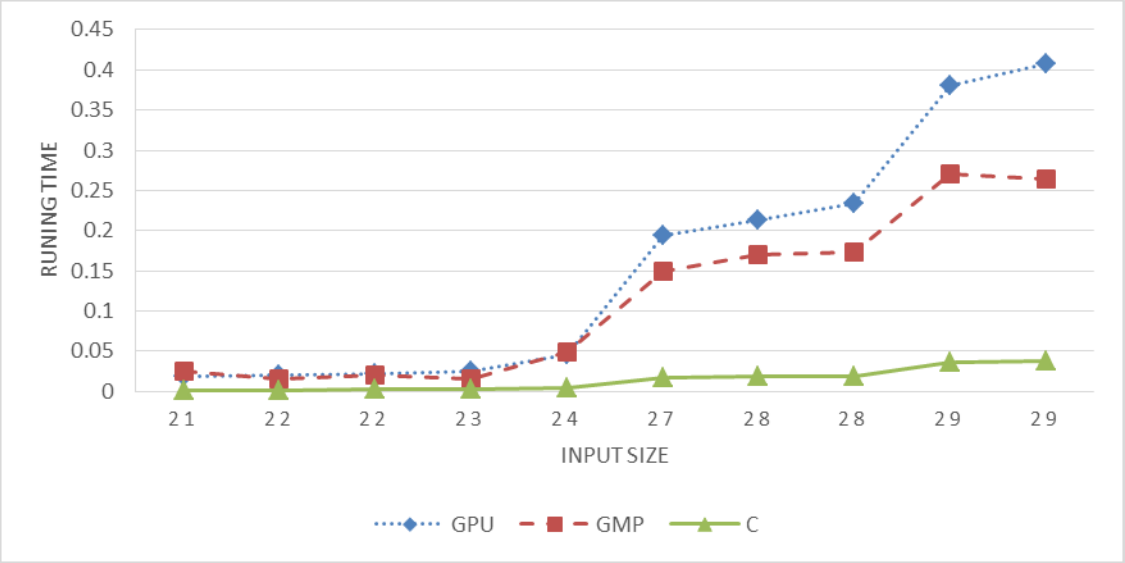

Figure 4. Running time of Brent algorithm

The same behavior was observed in implementation of the Brent algorithm, where the fastest implementation was on c implementation, and CUDA implementation on GPU showed the worst running time among all three implementations. However, adding GMP library slowed the algorithm down more than in case of Pollard rho algorithm.

5.2. Factorization of integers, dependence on distance between factors

According to [9], the factoring of a composite number *N* with difference of the factors *p* and *q* being $|p-q| < N^{5/18}$ can be performed in a polynomial time, which is a big issue for the security of the integer factorization based algorithms. The FIPS recommends that prime divisors of the modulus *N*, that is numbers *p* and *q* should not be close together. In [9] there was given an example, according to which for the 1024 bit RSA modulus *N*, primes *p* and *q* should not be identical in their 171 most significant bits.

In the scope of this work we factorized numbers, for which the ratio of its prime factors different, since it is considered to be hard to factors such numbers. The implementation was performed on the CUDA platform, which allows parallel computation. Therefore, we evaluated the running time of algorithms when the distance between the prime factors changes. Results are presented in the Figure 5 and Figure 6.

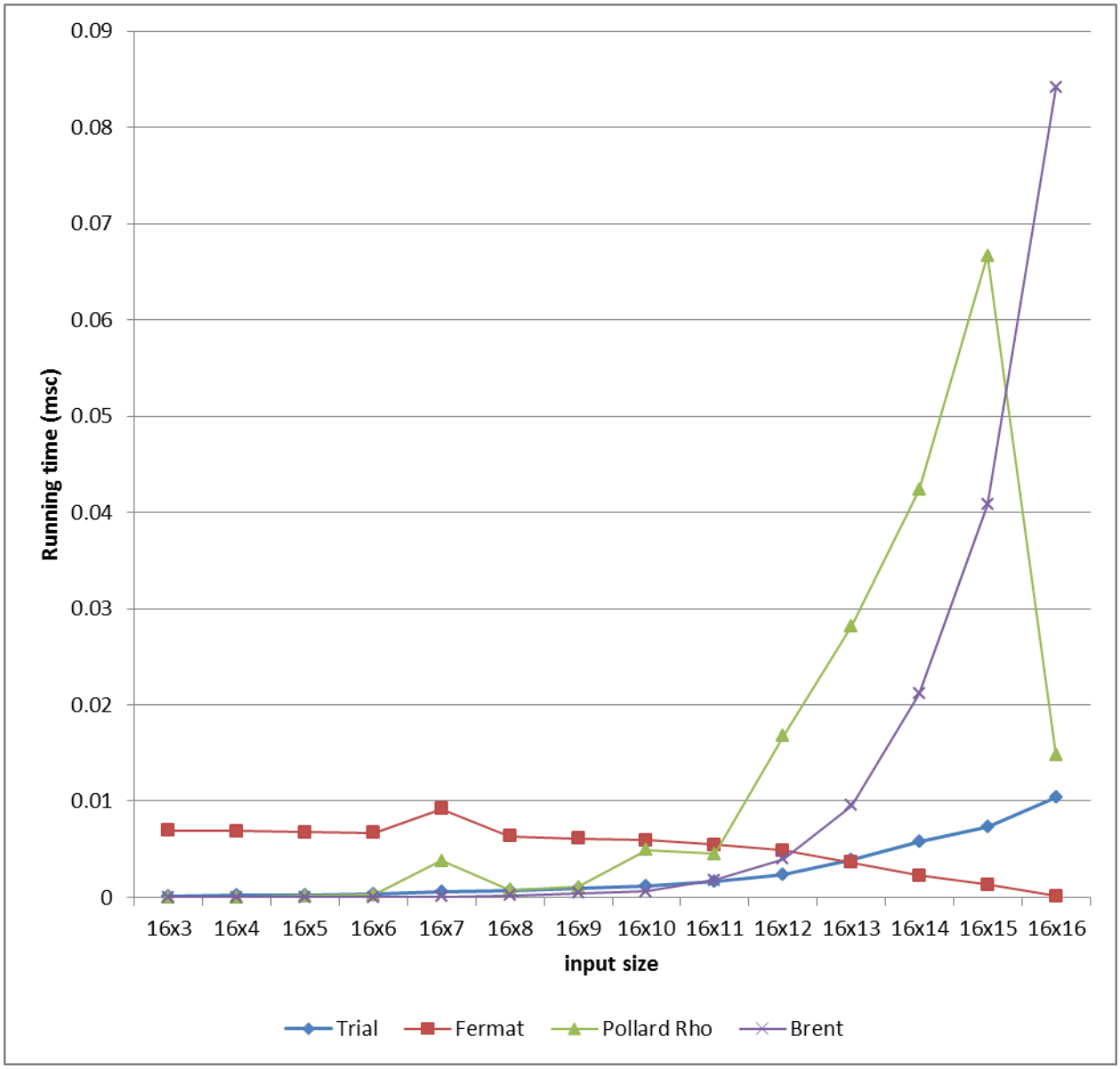

Figure 5. Running time of algorithms implemented on GPU

As it can be seen from the Figure 1, the running time of Trial division algorithm is in direct ratio with the input size, that is, with the number of bits of the integer to be factored. In Fermat algorithm, however, the running time mostly depend on the distance between factors, with the running time getting smaller with the decrease in distance between factors. Same was observed for the Pollard rho algorithm; however, the increase of running time was up to the last number, whose factors were both of 16 bits long. Running time of Brent algorithm showed constant growth when the size of input numbers increased.

Next, we compared the running time of algorithms on different platforms. This time, we aimed at measuring the difference between running times when the distance between factors changes. Due

to small size of input numbers, for other algorithms, implementations on CPU showed 0 as running time. Results, obtained for Brent algorithm are presented in the Figure 6.

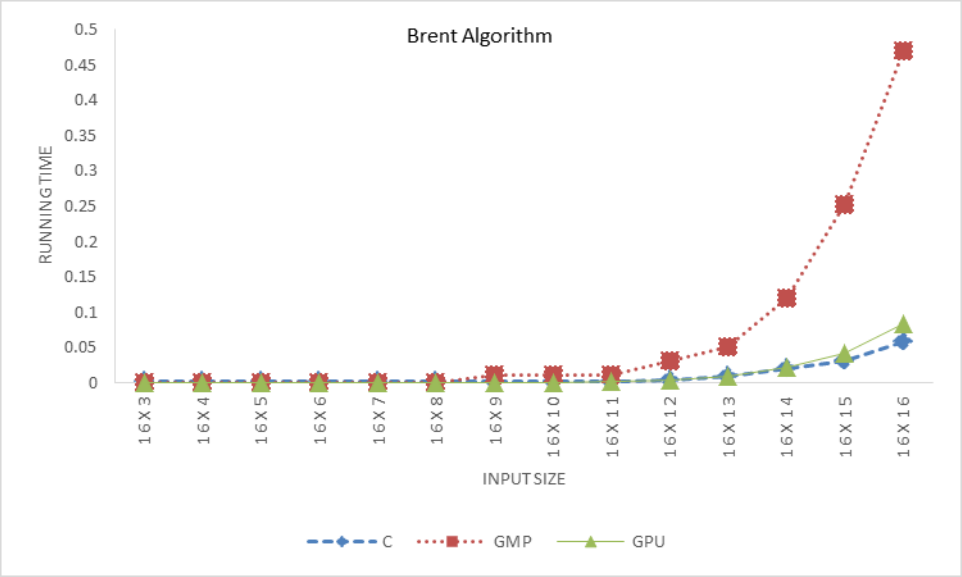

Figure 6. Running time of Brent algorithm on different platforms

Results showed that performance of algorithm, running on GPU is better for Brent algorithm. However, for small numbers the pure c implementation works faster. As for the Fermat and Pollard rho algorithms, the implementation of these algorithms on CPU using GMP library showed all zero results. However, test with big numbers showed that parallel implementation was faster that implementation with GMP library.

Overall, it was observed that the fastest running time is on the CUDA implementation of algorithms, which work on GPU, while the slowest results were obtained for the c++ implementation using GMP library. In all cases, the pure c implementation showed better results that c++ implementation. However, it should be noted that in this study, small numbers were factored.

6. DISCUSSIONS AND CONCLUSION

In this work we have evaluated the running time of four integer factorization algorithms, namely, trial division algorithm, Fermat algorithm, Pollard rho and Brent algorithms. Implementation of these algorithms was performed in three ways: first, an implementation on c programming language was performed. Next, we have implemented algorithms on c++ programming language, using GMP 6.0.0 library. Finally, all four algorithms were implemented on CUDA architecture to run on GPU. First, we run algorithms to factor numbers, whose prime divisors were close to each other. Next, all algorithms were tested as the distance between factors change. Every time, the running times of algorithms were measured.

Results showed that Fermat algorithm and trivial division algorithm had the fastest running time in parallel implementation on CUDA architecture. The difference of running times between CUDA implementation and GMP implementation was up to 10 times. The difference between c and c++ implementation was mainly due to difference in these programming languages.

However, these results were obtained for small numbers, with the size of prime factors being up to 16 bits long. Since there was limitation due to computational power, we could not use numbers with bit length more than 33 bits. Also, no library for big numbers on CUDA architecture was available. However, according to obtained results it can be concluded that parallel implementation of integer factorization algorithms run faster.

REFERENCES

[1] Asaduzzaman A., Yip, C. M., Maiti, ACUDA-assisted energy-efficient primality test. In Southeastcon 2014, Ieee, (2014), 1-5.

[2] Asaduzzaman A., Gummadi, D., Waichal, P. A promising parallel algorithm to manage the RSA decryption complexity. In *SoutheastCon 2015*, IEEE, (2015), 1-5.

[3] Atanassov, E., Georgiev, D., Manev, N. Number Theory Algorithms on GPU Clusters. In High-Performance Computing Infrastructure for South East Europe's Research Communities, Springer International Publishing, (2014), 131-138

[4] Brent R. P., Pollard J. M., "Factorization of the eighth Fermat number", Mathematics of Computation 36.154, (1981), 627-630.

[5] Brent, R. P. Parallel algorithms for integer factorisation. Number theory and cryptography, 154, (1990), 26-37.

[6] Cook D. L., Ioannidis J., Keromytis A.D., Luck, J. CryptoGraphics: Secret key cryptography using graphics cards. In Topics in Cryptology–CT-RSA 2005, Springer Berlin Heidelberg, (2005), 334-350.

[7] Diffie W., Hellman, M. E.New directions in cryptography. Information Theory, IEEE Transactions on, *22*(6), (1976), 644-654.

[8] Du Z., GMP Install Instruction for Windows Platform, [E-documentation], (2006), Available: http://cs.nyu.edu/~exact/core/gmp/index.html.

[9] Erra, R., Grenier, C. The Fermat factorization method revisited. IACR Cryptology ePrint Archive, (2009), 318.

[10] Fleissner, S. GPU-accelerated Montgomery exponentiation. In Computational Science–ICCS 2007, Springer Berlin Heidelberg, (2007), 213-220.

[11] Kleiner I., "Fermat: The founder of modern number theory." Mathematics Magazine, (2003), 3-14.

[12] Lenstra Jr, H. W. Factoring integers with elliptic curves. Annals of mathematics, (1987), 649-673.

[13] Mahajan, S., Singh, M. Analysis of RSA algorithm using GPU programming. *arXiv preprint arXiv:1407.1465*, (2014).

[14] Manocha, D. General-purpose computations using graphics processors.*Computer*, (8), (2005), 85-88.

[15] Miele, A., Bos, J. W., Kleinjung, T., & Lenstra, A. K. Cofactorization on graphics processing units. In Cryptographic Hardware and Embedded Systems–CHES 2014. Springer Berlin Heidelberg, (2014), 335-352.

[16] Müller, S., & Müller, W. B. The security of public key cryptosystems based on integer factorization. *In Information Security and Privacy, Springer Berlin Heidelberg, (1998), 9-23.*

[17] National Bureau of Standards, Data Encryption Standard, FIPS-Pub.46. National Bureau of Standards, U.S. Department of Commerce, Washington D.C., (1977).

[18] NVIDIA. Fermi architecture whitepaper, http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf (2010).

[19] NVIDIA Developer Zone, https://devtalk.nvidia.com/default/topic/491799/gtx-590-cuda-power-tests/ (2011).

[20] NVIDIA. Cuda programming guide 5 (2013), http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[21] NVIDIA. Parallel thread execution isa version 3.2 (2013), http://docs.nvidia.com/cuda/parallel-thread-execution/index.html

[22] Pollard J. M., "Monte Carlo methods for index computation ($mod\ p$)."Mathematics of computation 32.143, (1978), 918-924.

[23] Riesel H., " Prime numbers and computer methods for factorization" (2nd ed.), Birkhauser Verlag, Basel, Switzerland, Switzerland, (1994).

[24] Rijmen, V., Daemen, J. Advanced encryption standard. Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology, (2001), 19-22.

[25] Rivest, R. L., Shamir, A., Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM, 21(2), (1978), 120-126.

[26] Saxena, S., & Kapoor, B. State of the art parallel approaches for RSA public key based cryptosystem. arXiv preprint arXiv:*1503.03593*, (2015).

[27] Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M., & Shringarpure, A. On the limits of GPU acceleration. In Proceedings of the 2nd USENIX conference on Hot topics in parallelism. USENIX Association, (2010, June), 1-6.

[28] Williams H. C., Shallit J. O., "Factoring Integers Before Computers, Mathematics of Computation 1943–1993: a half-century of computational mathematics" (Providence) (Walter Gautschi, ed.), American Mathematical Society, (1991), 481–531.

[29] Lin Y.-S., Lin C.-Y., Lou D.-C., "Efficient parallel RSA decryption algorithm for many-core GPUs with CUDA," Proc. International Conference on Telecommunication Systems, Modeling and Analysis, (2012), 85-94.

[30] Zhang H., Zhang D., Bi X., "Comparison and Analysis of GPGPU and Parallel Computing on Multi-Core CPU," International Journal of Information and Education Technology. 2, 2, (2012), 185-187.