International Journal of Advanced Research in IT and Engineering

ISSN: 2278-6244
Impact Factor: 6.111

Study of Various Algorithms for Computing GCD of Bulk Pair of Numbers and Parallelization Techniques

Shehanaz, Department of computer science and engineering, NMAMIT, Nitte, Karnataka,India

Raju K, Department of computer science and engineering, NMAMIT, Nitte, Karnataka,India

***Abstract*—RSA algorithm is used commonly for encrypting and decrypting the data, so that data can be transferred safely on insecure channel. But this algorithm heavily depends on finding the factors of large numbers which affects the performance and make its use limited in various applications. When RSA modulus of two encryption keys share the same prime numbers then that prime key can be decomposed by computing their GCD (greatest common divisor). When there is huge collection of encryption keys GCD computation becomes worst in practical computational time. There are several algorithms which is used for GCD computation. This paper is about the study on various algorithms for finding the GCD of pair of numbers.**

Keywords- *Weak RSA keys, GCD, RSA algorithm.*

I. INTRODUCTION

Graphics processing units (GPUs) provides high performance processing power for computing various applications.GPU is a logic circuit built to render the images and videos. Now GPUs are designed for the computation purpose and can compute the applications which are particularly processed by the CPUs. Heavy computation on CPU can run efficiently on GPU. Therefore GPUs have become more popular in the minds of many application developers. When CPU and GPU works together, serial portion of any operation executed on CPU and parallel portion will be executed on GPU. GPUs manufactured by Nvidia Corporation provide parallel computing capacity to research scholars and science experts which allow them to run large computations efficiently [1].

NVIDIA creates application programming interface and a parallel processing platform which is known as Compute Unified Device Architecture (CUDA), allows application developers to make use of a CUDA-enabled GPU for general purpose computation [2].CUDA provides advantage of huge processing power which is provided by Nvidia's graphics cards. CUDA gives 128 cores jointly working towards the same end. These cores will communicates to exchange the information with each other[3].

RSA is a public key cryptosystem used for safe transfer of data. RSA algorithm uses two types of keys for encryption and decryption of data. Both the key consists of pair of parameter : one is RSA modulus and other an exponent for encrypting and decrypting the data. When there is huge collection of encryption keys generated randomly, RSA modulus of pair of encryption keys will share the pair numbers. RSA algorithm faces the difficulty in finding the factors of RSA modulus when the RSA modulus is the product of two very large prime numbers. Since the factorization is very costly, decomposing the *RSA modulus* into two prime numbers is not possible in experimental computing time. Factorization is the main disadvantage of RSA public key cryptography.

Suppose, we have a set of public encryption key which consist of RSA modulus which share or reuse the common prime numbers, such encryption keys are called *weak RSA keys*. In such keys, RSA modulus can be decomposed by calculating the GCD. Consider two RSA modulus, Rmd_1 and Rmd_2, if these two modulus share the same prime number *pr1*, this *pr1* can be obtained by computing the GCD (Rmd_1,Rmd_2). Thus $pr_{12}=Rmd_1/pr_{11}$ and $pr_{21}=Rmd_2/pr_1$. Once both *pr1* and *pr2* are available, we can get the decryption keys. Once we get the common prime numbers used in the RSA modulus of pair of encryption keys by GCD computation, we can break the weak RSA keys. The idea of this paper is to study various algorithms to calculate the GCD of bulk pair of numbers.

II. BACKGROUND

A. Division algorithm.

This algorithm is defined as, if two numbers $n1, n2 \in Z, n2>0$ then, there is distinct *quotient* and *reminder* such that $n1=n2*quotient + reminder$, where $0 \leq reminder < n2$.

B. Divisibility.

Divisibility property states that, for any two number *n1* and *n2*, where $n2 \neq 0$, *n2* divides *n1*, i.e. $n2|n1$, if there is an integer *m*, such that $n1=m*n2$.

i.e $n2|n1 \Leftrightarrow \exists m, n1=m*n2$.

C. Prime numbers

A number is said to be prime, if a number cannot be divisible by any other number, except 1 and itself.

D. RSA algorithm

RSA is a public key cryptosystem used for transferring data safely. It consists of Public key for encryption, which is open

to all and Private key for decryption. Steps in the RSA encryption and Decryption are as follows:

1. Select two dissimilar prime numbers *pr1* and *pr2*
2. Now calculate *Rmd=pr1*pr2.*

   This *Rmd* is used as the modulus for encryption key and decryption key.
3. Calculate

   $$\Phi(Rmd)=(pr1-1)(pr2-1)$$
4. Select exponent for encryption, *ekey* such that

   $1<ekey<\varphi(Rmd)$ and this *ekey* and $\varphi(Rmd)$ should not share any divisors i.e. *ekey* and $\varphi(Rmd)$ should be co-prime.
5. Find exponent for decryption, *dkey* using modular arithmetic such that it should satisfies the congruence relation.

   *i.e. dkey*ekey=1(modulus(φ(Rmd)).*
6. Encrypt the message Msg using the encryption key.

   i.e. $Csg=Msg^{ekey}$ *modulus Rmd.*
7. Decrypt the encrypted message $i.e\ Csg$, a cipher text of message, using decryption key.

   i.e. $Msg=Csg^{dkey}$ *modulus Rmd.*

III. Literature Survey

RSA cryptography is one of the applications which are used in sending the data over the insecure channel. The main disadvantage of this algorithm is its computation time involved in factoring the RSA modulus. By GCD calculation factorization of RSA modulus can be obtained. There are various techniques to calculate the GCD. Euclidean algorithm (EA) can efficiently calculate the GCD of any two numbers. But modulo computation very big numbers is expensive with respect to its execution time.

Greatest common divisor of two number gives the greatest divisor which divides both the numbers.GCD can be calculated by finding the prime factors of the numbers and the common prime factors of those numbers will gives the GCD of those numbers. For example: if *n1*and *n2* are two numbers whose prime factors can be as shown below:

Algorithm 1: Prime factorization algorithm

Input: Any two numbers

Output: GCD of two numbers

1. *factorize n1 into the multiples of prime numbers*

   *for example: n1=a*b*c*d*
2. *factorize n2 into the multiples of prime numbers*

   *for example: n2=a*b*e*f*
3. *compare the common prime factors in both the numbers*

   in our case a, b are common
4. *multiply the common prime factors which give the GCD of those two numbers*

   *i.e. GCD (n1, n2) =a*b*

   Where a, b, c, d, e, f are the prime numbers

Euclid's algorithm is the popular algorithm to find the GCD of any two numbers. There are two versions of this algorithm. One uses subtraction method and the other one is division method. As this algorithm is introduced by the Euclid, the name of the algorithm is Euclid's algorithm.

The basic version of this algorithm uses subtraction method which is as shown below:

Algorithm 1: Subtraction version of Euclid's algorithm

Input: Any two numbers

Output: GCD of two numbers

1. *Check whether n1=n2, if true then go to step 2 else go to step 3*
2. *GCD(n1,n2)=n1;*
3. *while n1≠n2 do*
4. *Check whether n1>n2, if true then go to step 5 else go to step 6*
5. *n3=n1-n2*

   n1=n3
6. *n3=n2-n1*

   n2=n3
7. *end while*
8. *return n1*

The time complexity of this version is *O (n).*

The division version of the Euclidean algorithm can be shown using the following pseudo code:

Algorithm 3: Division version of Euclid's algorithm

Input: Any two numbers

Output: GCD of two numbers

1. *for two numbers n1 and n2 where n1≥n2:*
2. *Check whether n2|n1, if true, then go to step 3else go to step 4*
3. *GCD(n1,n2)=n2*

4. *GCD (n1, n2)=GCD(n2, n1 modulus n2).*

This Euclidean algorithm (EA) with division version can be explained as: smaller of the two number divides the bigger number and then repeats this division by taking the remainder obtained in the previous division and the divisor which we used in previous division until the reminder becomes zero and the GCD of those numbers will be the last non zero remainder.

The time complexity of this version is the sum of *n1* and *n2*. i.e. *O(log(n1+n2)).*

When there is large integers, the above algorithms may not be well suited to be used to find the GCD of large numbers, so there is one more method called Binary Euclidean algorithm (BEA) which is introduced by the Israeli physicist and Josef Stein [9], to calculate the GCD of two numbers, which uses Subtraction, and checks whether the number is even or odd and dividing it by 2 if it is even.

The BEA can be shown as the following pseudo code:

GCD (n1, n2) can be given as,

Algorithm 4: Binary Euclidean algorithm

Input: Any two numbers

Output: GCD of two numbers

1. *Check whether n1=n2, if true then go to step 2 else go to step 3.*
2. *GCD(n1,n2)=n1*
3. *Check whether n1 and n2 is even, if true then go to step 4 else go to step 5*
4. *GCD(n1,n2)=2*GCD(n1/2,n2/2)*
5. *Check whether n1 is even, if true then go to step 6 else go to step 7*
6. *GCD(n1,n2)=GCD(n1/2, n2)*
7. *Check whether n2 is even, if true then go to step 8 else go to step 9*
8. *GCD(n1,n2)=GCD(n1,n2/2)*
9. *Check whether n1>n2, if true then go to step 10 else go to step 11*
10. *GCD(n1,n2)=GCD((n1-n2)/2,n2)*
11. *GCD(n1,n2)=GCD(n1,(n2-n1)/2)*

This algorithm is well suited for large numbers, as computation in this algorithm is done depends on the input size.

The time complexity of this BEA is *O(log(n1*n2))* i.e. *O(log n1 +log n2)*. For a large number it may take the time complexity of $O((\log n)^2)$. As each iteration take the time complexity of *O(log n), where n=n1+n2.*

In [10], Mohamed has introduced an algorithm to calculate the GCD. He mainly concentrated on average sized numbers. He used both basic Euclidean algorithm and binary Euclidean algorithm to design its algorithm. since his algorithm is a mixture of basic Euclidean and binary Euclidean algorithm, hence the name of his algorithm is, *sequential Mixed Binary Euclid Algorithm* is as shown below:

Algorithm 5: Sequential Mixed binary Euclid

Input: two numbers n1 and n2, n1 ≥n2≥1, with n2 being odd

Output: GCD of two numbers

1. *while n2>1 do*
2. *Rem=n1 modulus n2.*
3. *Diff=n2-Rem.*
4. *While Rem>0 and Rem=0 do*
5. *Rem=Rem/2.*
6. *End while*
7. *While Diff>0 and Diff=0 do*
8. *Diff=Diff/2.*
9. *End while*
10. *Check whether Diff<Rem, if true then go to step 11, else go to step 12*
11. *n1=Rem and n2=Diff.*
12. *n1=Diff and n2=Rem.*
13. *End while*
14. *Check whether n2=1 if true go to step 15 else go to step 16*
15. *Return n2*
16. *Return n1*

To improve the performance of this *sequential mixed binary Euclid algorithm*, he also developed a Parallel algorithm for GCD computation. This Parallel algorithm is based on the reduction method and the Reduction algorithm is as shown below:

Algorithm 6: Reduction Algorithm

Input: two numbers n1 and n2, n1 ≥n2≥1, with n2 being odd

Output: n1 and n2 after the reduction.

1. *Begin Kin=2^n is integer parameter where Kin is the power of 2.*
2. *For i= 1 to m do Rem[i]=n1, Diff[i]=n2, Qnt_i=floor($n1_i/n2_i$)*

3. *For i= 1 to m do rem_i=abs (i*n1-Qnt_i*n2) and $diff_i$=n2-rem_i.*
4. *While rem_i>0 and rem_i%2=0 do*
5. *rem_i= rem_i/2.*
6. *End while*
7. *Check whether rem_i<2*n2/kin if true then go to step 8 else go to step 4*
8. *Rem[i] = rem_i.*
9. *While $diff_i$>0 and $diff_i$%2=0 do*
10. *$diff_i$= $diff_i$/2.*
11. *End while*
12. *Check whether $diff_i$<2*n2/kin if true then go to step 13 else go to step 9*
13. *Diff[i] = $diff_i$.*
14. *Rem=min{Rem[i]} and Diff=min{Diff[i]}*
15. *Check whether Rem≥Diff,if true then go to step 16 else go to step 17*
16. *Return n1=Rem and n2=Diff*
17. *Return n1=Diff and n2=Rem.*

In this above algorithm step 2, step 3, step 7 and step 12 can be performed parallel and step 14 will take O(1) parallel time. Hence because of this reduction algorithm, values of n1 and n2 will be reduced and then GCD can be calculated based on the sequential algorithm.

In [11], David et al., designed an algorithm based on the SRT division which is similar to that of non-restoring division algorithm where it uses its MSB to get the quotient and this SRT division algorithm it makes use of the look up table to get its quotient and the remainder digits. Here they have designed their algorithm such a way that, the algorithm will normalize the integers whose GCD has to be calculated to get its redundant value. This reduction is based on the three most significant signed bits of the integers. Since this algorithm is used to get the redundant values of the integers so that the number of iteration in GCD computation is reduced, the name of the algorithm is Redundant Binary Euclidean GCD algorithm.

In [12], Chor and Goldreich have designed an algorithm which works parallel. They have developed this algorithm based on an algorithm known as Brent-Kung GCD(BKGCD) algorithm. As this Brent-Kung GCD(BKGCD) is based on the parity check using LSB to know the number is odd or even, parallel execution of the developed algorithm doesn't depend on the complete execution of previous iteration, once the LSB of previous iteration is known, execution can be continued so that intermediate steps in GCD computation is done parallel.

In [6], Pavel proposed an algorithm to find the GCD of polynomials with single variable having large integer as its variable's coefficient which can be implemented on Graphics processing units (GPUs). In this method, he has used the modular mechanisms to divide the big problem into multiple smaller problems and then assign each smaller problem into blocks to calculate the GCD. Now this intermediate result of each block having the GCD for multiple smaller problems is then combined to form a result for actual problem. For combining the result of each block, he has used the mixed-radix converting method which is also implemented on Graphics processing units. For obtaining the GCDs of each smaller problem of a big problem, he has considered a polynomial as one matrix and calculated the GCD using the Gaussian elimination method, where non-zero elements from the matrix after the Gaussian elimination gives the GCD for that sub-problem of polynomial which is considered as matrix for calculating GCD. Once all the GCDs for all multiple smaller problems are obtained, we can then get the GCD of actual problem by using the mixed-radix conversion technique to get the GCD of actual problem in polynomial form. As in this they have tried a method which involves: representing polynomial into matrix then once GCD is found by applying Gaussian elimination method and converting that matrix into polynomial using mixed-radix conversion, it is quite time consuming. It is also not suitable for the large numbers.

In [7], Joseph et al. have designed the algorithm to find the GCD of RSA modulus. This algorithm is developed mainly to break the weak RSA keys by computing the GCD of RSA modulus of various pair of encryption keys to show that RSA modulus of the collection of encryption keys which are generated randomly by verifying that they share the same prime key to obtain the RSA modulus. Here Joseph et al. used the Binary Euclidean Algorithm for implementing their algorithm, they have parallelized the intermediate steps involved in GCD computation such as right shift, subtraction and comparison of greater than or equal, so that performance is improved.

In [8], Miller et al. have worked to parallelize then GCD computation to speed up the performance. They have implemented their algorithm by packing the number of iterations in the Euclidean algorithm. Their final answer as GCD will be very close to the actual GCD but not the exact GCD. They have used the CRCW model so that cost of each bit operation is unit cost. The main purpose of this algorithm is to implement the algorithm which will replace n1 into i*n1-Qnt*n2 where Qnt is the quotient when i*n1 is divided by n2 and i=1 to n, instead of n1-q*n2, where q s the quotient when n1 is divided by n2. There will be two problems which arises in this approach. One is GCD(n1,n2) may change due to replacement and the other one is it may not be clearly known

how to execute the whole steps of this algorithm. the fisrt problem can be solved by showing that if GCD(n1,n2)=a and GCD(i*n1-Qnt*n2)=b, then a|b and a or b divides i.

The second problem can be solved by satisfying the following two conditions. They are:

i. $0 \leq$ i*n1-Qnt*n2 $\leq$ n2-1.

ii. The most significant log n bits of i*n1-Qnt*n2 are zero.

This algorithm uses the pigeon hole principle which helps in finding the combination of the two integer n1 and n2 which has the lesser bits compared to n. here arithmetic operations performed in parallel in each iterations only once. This algorithm works well only for numbers which takes less time for GCD computations.

IV. Conclusion and Future Work

Secure transfer of data over insecure channel is done using RSA algorithm. Factorization is main disadvantage of RSA algorithm as it is expensive and time consuming when used for large numbers. Factorizing of RSA modulus can be done using GCD computation. Study on various algorithms is done. Hence as a future work, GPU based parallel algorithm for computing GCD can be developed to speed up GCD Computation.

References

[1] https://en.wikipedia.org/wiki/Nvidia

[2] https://en.wikipedia.org/wiki/CUDA

[3] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni,and Amit Bawaskar “GPGPU processing in CUDA Architecture,” Advanced Computing: An International Journal (ACIJ), Vol.3, No.1, January 2012.

[4] NVIDIA Corporation, “NVIDIA CUDA C programming guide version 5.0,” 2012.

[5] Ambedkar, B. R., and S. S. Bedi. "A New Factorization Method to Factorize RSA Public Key Encryption." IJCSI International Journal of Computer Science Issues 8.6 (2011).

[6] N.Fujimoto,”High Throughput Multiple-Precision GCD on the CUDA Architecture”, IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), IEEE, 507512, 2009.

[7] K. Scharfglass, D. Weng, J. White, and C. Lupo, “Breaking weak 1024bit RSA keys with CUDA,” in Proc. of Internatinal Conference of Breaking weak 1024-bit RSA keys with CUDA, Dec. 2012, pp. 207 – 212.

[8] Kannan, Ravindran, Gary Miller, and Larry Rudolph. "Sublinear parallel algorithm for computing the greatest common divisor of two integers." SIAM Journal on Computing 16.1 (1987): 7-16.

[9] https://en.wikipedia.org/wiki/Binary_GCD_algorithm

[10] Sedjelmaci, Sidi Mohamed. "The mixed binary euclid algorithm." Electronic Notes in Discrete Mathematics 35 (2009): 169-176.

[11] Parikh, Shrikant N., and David W. Matula. "A redundant binary Euclidean GCD algorithm." Computer Arithmetic, 1991. Proceedings, 10th IEEE Symposium on. IEEE, 1991.

[12] Chor, Benny, and Oded Goldreich. "An improved parallel algorithm for integer GCD." Algorithmica 5.1-4 (1990): 1-10.