International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 05 | May 2019 www.irjet.net p-ISSN: 2395-0072

Performance Analysis of RSA Algorithm with CUDA Parallel Computing

Rishikesh Kadam[1], Vishakha Vidhani[2]

[1,2] *Student, Department Of Computer Engineering, V.E.S.I.T., Mumbai University, India.*

Abstract - *In today's data driven and digitally connected world, computer system security is of paramount importance. The computer security relies on cryptographic security techniques for protection of data. RSA is one of the most used cryptographic algorithms which can be sped up using parallel computing techniques. The advances in computing capability of Graphics Processing Unit as a co-processor of the system CPU allows parallelization of RSA encryption and decryption. This paper focuses on parallel implementation of RSA using Nvidia's Compute Unified Device Architecture (CUDA). First, the RSA cryptographic algorithm is studied. Then, it is designed for implementation in CUDA framework. In the last step, the RSA function is called parallely by all nodes in a Hadoop cluster. For this step, JCUDA is used. The results are analyzed mainly for the speed up introduced by GPU as compared to the CPU-only implementation.*

Key Words: **Cryptography, RSA, CUDA, GPU, Hadoop**.

1. INTRODUCTION

RSA algorithm is named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman. It is a public key cryptographic algorithm whose security relies on the difficulty of finding prime factors of large numbers.[1] But the modular power used in this algorithm acts as a bottleneck for its performance which makes its large scale implementations inefficient. Thus, the optimization of RSA has been a research focus. The advancements in multi-core processing and parallel computing is huge boost for RSA optimization.

In last decade, the use of Graphical Processing Unit in training of machine learning models and neural networks has been a major trend, but along with that, GPU turns out to be exclusively beneficial for general purpose parallel computing [2] after the advent of Compute Unified Device Architecture (CUDA) technology. Nvidia's CUDA platform makes thousands of stream processors available to provide parallel speed up. This paper explores the implementation of CUDA for parallelization of RSA algorithm and presents the performance analysis. First, the RSA algorithm is encapsulated in Java Native Interface (JNI). Then, JCUDA is used to implement the parallelization.

In section 2 of the paper, traditional principle of RSA algorithm is studied. Section 3 gives an architectural overview of the system hardware used for the implementation. Section 4 presents the design and implementation of RSA parallelization and section 5 shows the results of performance analysis.

2. RSA ALGORITHM

RSA algorithm is the most commonly used asymmetric cryptography algorithm. It works on two different keys i.e. public key and private key. The public key is known to everyone and is used for encrypting messages. Encrypted Messages can only be decrypted using the private key.

The keys for the RSA algorithm are generated in the following way:

1. Choose two distinct large prime numbers, p and q.
2. Calculate n = pq, n is used as the modulus for both the public and private keys.
3. Compute $\varphi(pq) = (p - 1)(q - 1)$. ($\varphi$ is totient function).
4. Pick an integer e such that $1 < e < \varphi(pq)$, and e and $\varphi(pq)$ are coprime (e and $\varphi(pq)$ share no divisors other than 1). The public key consists of e (often called public exponent) and the modulus n.
5. Find d which satisfies the relation $d*e = 1 \pmod{\varphi(pq)}$. The private key consists of d (private exponent) and the modulus n.

The message M is encrypted using formula $C \equiv M^e \bmod n$, where C is the encrypted message. And it is decrypted using the formula $M \equiv C^d \bmod n$.

Example :

1. The plaintext M to be encrypted: 1314
2. Let p=397 and q=401
3. n= pq = 159,197
4. $\varphi(pq) = 396 * 400 = 158,400$
5. The public key(e,n) is (343, 159197) and private key d is 12,007 ($d*e = 1 \bmod \varphi(pq)$)
6. $C \equiv M^e \bmod n$, $C \equiv 1314^{343} \pmod{159,197} = 33,677$
7. $M \equiv C^d \bmod n$, $M \equiv 33,677^{12,007} \pmod{159,197} = 1314$

3. CUDA ARCHITECTURE

NVIDIA has designed a special C-based language CUDA to utilize the massively parallel nature of GPU.[3] Parallel execution is expressed by the kernel function that is executed on a set of threads in parallel on GPU; GPU is also

called the device. This kernel code is a C code for only one thread. The number of thread blocks, and the number of threads within those blocks that execute this kernel in parallel are given explicitly when this function is called.

NVIDIA's CUDA architecture includes a number of multiprocessors, each multiprocessor consists of:

- 8 scalar processor cores
- 2 special function units for transcendentals
- 1 multithreaded instruction unit
- On-chip shared memory

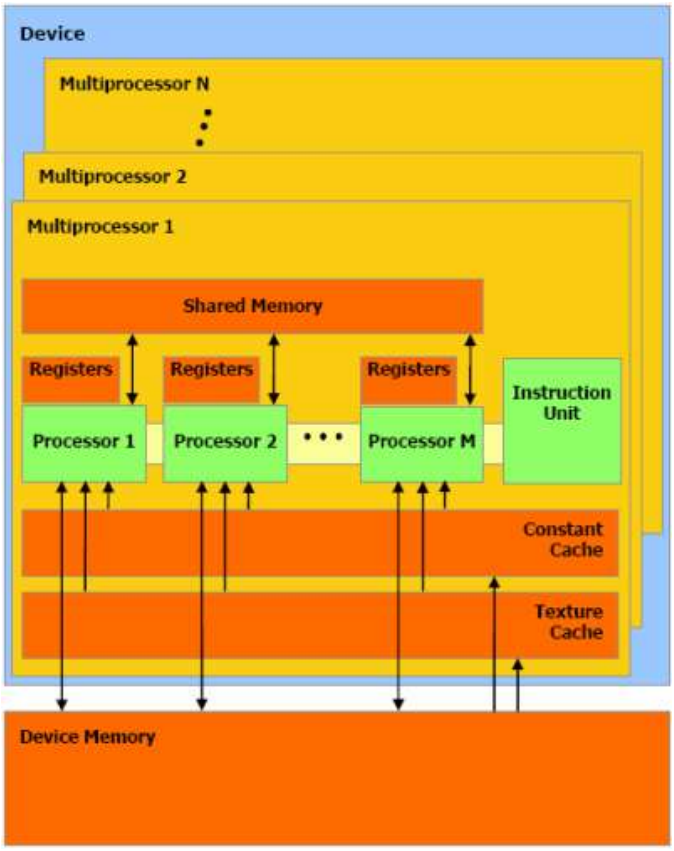

Fig -1: CUDA Hardware

Each multiprocessor has SIMD architecture (single instruction multiple data). Each processor of the multiprocessor executes a different thread but all the threads run the same instruction but operate on different data.

Several threads (up to 512) may execute concurrently within a multiprocessor and communicate through a small shared memory bank (16KB) which can be seen in Fig -1. Shared memory is local to each multiprocessor unlike device memory and allows local synchronization to be more efficient. It is divided into many parts. Each thread block within multiprocessor accesses its own part of shared memory. This part of shared memory is not accessible by any other thread block of this multiprocessor or of some other multiprocessor.

4. PARALLELIZATION

This paper presents a two-level parallelization of RSA algorithm, node level and thread level as shown in Fig -2.

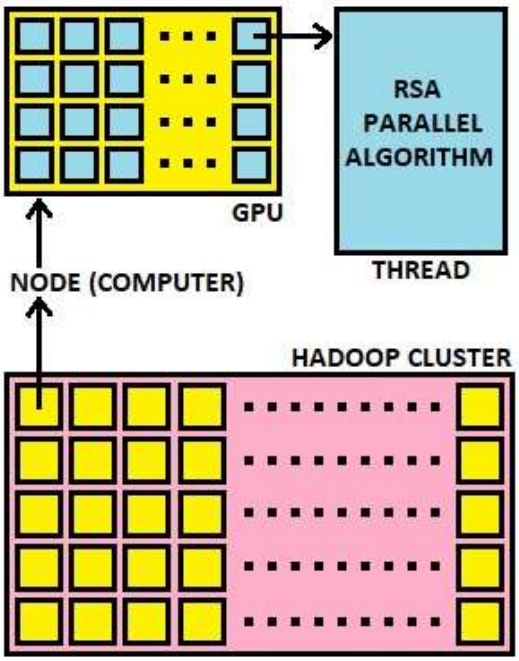

Fig -2: Two Level Parallelization

CUDA framework is used for parallel implementation on thread level. RSA algorithm makes packets with the same length by dividing the plaintext or ciphertext. The same function will be executed for all the packets. As mentioned in Section 2, the text is considered as integers. All the packets are considered as arrays with same number of integer elements. The encryption and decryption processes are carried out on these packets. We can get the thread and block index and then, they are used to assign packets (elements) to each thread. Thus, on thread level, CUDA multi-thread programming improves the performance of RSA algorithm. [4]

The large size of plaintext or ciphertext can be troublesome to handle on a single computer. We used a distributed file system to store the text to be processed. Each node (computer) in the cluster executes the parallel RSA algorithm function. This is node level or computer level parallelization. In such clusters, each node gets a small part of data. In this case, a part of text is processed by each node. Then, each node performs thread level multi-processing.

This paper uses Hadoop Distributed File System (HDFS) [5] for distribution of the text to all the nodes in the cluster. HDFS is block-structured file system where each file is divided into multiple blocks. The blocks are stored

across the cluster of nodes. It is a Java based software framework. Google's MapReduce [6] and Google File System (GFS) [7] papers built the foundation for HDFS. As Hadoop is based on pure Java, we need to find a method to implement our C language code of RSA on Hadoop. JCUDA [7] provides an interface for invoking CUDA kernels using Java code or framework such as Hadoop. The syntax and usage of JCUDA are beyond the scope of this paper. A sample of kernel code is as follows:

```
__global__ static void RSACUDA(int* arr)
{
  long p,q,e,d,m,n,t,c,i;
  int x = blockDimension.x * blockId.x + threadId.x;
  p=191;
  q=223;
  n=p*q;
  t=(p-1)*(q-1);
  e=t-1;
  d=1;
  while(((e*d)%t)!=1)
    d++;
  for(i=0;i<50;i++)
  {
    c = encrypt(*(arr+i+x*50),e,n);
    m = decrypt(c,d,n);
    *(arr+i+x*50) = m;
  }
}
```

In next section, performance of parallel implementation is discussed.

5. VERIFICATION

For verification, we divide the tests into two parts. First, we run the RSA program using a CPU-only processing. Then, in the next step, we implement the RSA algorithm on CUDA framework.

5.1 TEST ENVIRONMENT

The computers used for the testing have following configuration:

- CPU: Intel Core i5-5200U 2.20GHz
- GPU: Nvidia GeForce 920M (2048MB Memory)
- Memory: 2.20GHz DDR3 (8192MB)

For development and testing, Visual Studio 2017 is used. For GPU processing, CUDA Toolkit 10.1 is used. Using the Hadoop framework, the text was distributed on a cluster of 3 nodes. JCUDA was involved to invoke CUDA kernels from Java code on each HDFS node.

5.2 RESULTS

Table -1: Time taken for execution (in milliseconds) for different text sizes.

No. of chars	CPU	10 Threads	100 Threads	500 Threads
10000	10176.45	12023.84	1256.73	276.49
20000	20312.82	24096.33	2303.34	530.18
30000	31034.47	36201.93	3678.01	760.48
40000	40583.05	47936.17	4907.38	1015.74
50000	51396.11	61034.28	6179.57	1346.05
60000	61564.26	71124.05	7389.18	1592.38
70000	71891.49	83456.61	8599.06	1904.75
80000	82392.73	96345.34	9803.49	2267.11

Table -1 displays the relationship between the size of input text and the time taken for execution (in milliseconds) in traditional CPU-only mode and GPU implemented multithreading mode.

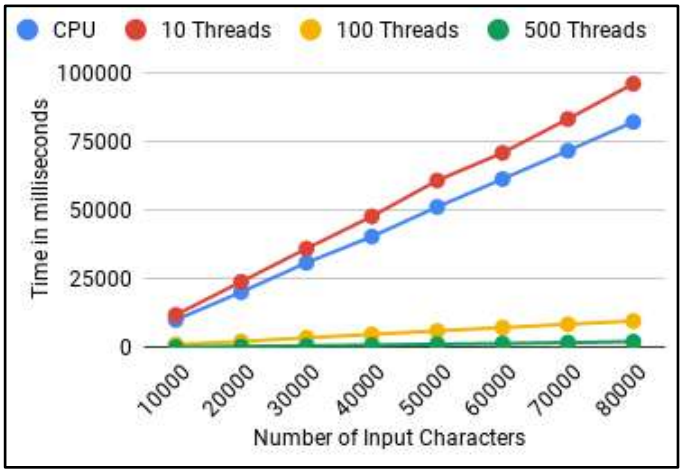

Chart -1: Time taken for execution by CPU-only and GPU multithreading mode.

Chart 1 represents this output graphically. The relationship between size of input text and execution time is linear. As we can see, the execution time is very less when we implement RSA algorithm in 500-thread mode. It is almost 40 to 50 times faster than the CPU-only mode. This output proves that higher the degree of parallelism, better is the performance.

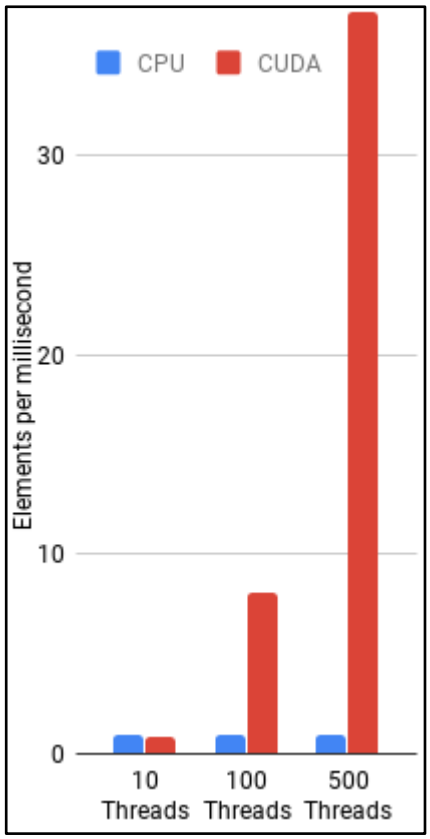

Chart -2: Elements processed per millisecond in CPU-only and multithreading mode

If number of threads are less, about 10, RSA algorithm takes longer takes longer than CPU-only mode. This is the result of co-operation processing and communication overhead between CPU and GPU. It majorly consists of copying of data from the host computer (CPU) to the device (GPU). This can be verified in Chart -2.

6. CONCLUSIONS

This paper presented the CUDA based implementation of RSA algorithm. We described the traditional RSA cryptosystem. The large size of text and modular power function act as the bottleneck for performance which is a drawback for large scale implementation.

We used Nvidia's Compute Unified Device Architecture (CUDA) for parallel implementation of RSA. First, RSA was executed in CPU-only mode. Then, it was implemented in CUDA framework in 10-thread, 100-thread and 500-thread modes. The results were compared which made it clear that higher the degree parallelism, better us the performance. The 10-thread mode gave a poor performance as compared to CPU-only mode. It was a result of communication and data transfer overhead between CPU and GPU.

Apache Hadoop framework can be used for distributed storage of large text files. All the nodes in the cluster store a part of the text file and execute RSA in parallel on the data.

REFERENCES

[1] R.L. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, iss. 2, pp. 120-126, February 1978.

[2] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, Tim Purcell, "A survey of general-purpose computation on graphics hardware," Computer Graphics Forum, vol. 26, pp. 80-113, March 2007.

[3] NVIDIA Corporation, NVIDIA Compute Unified Device Architecture Programming Guide, Version 10.1, February 2019.

[4] Daniel Page, Nigel P. Smart, "Parallel cryptographic arithmetic using a redundant montgomery representation," IEEE Transactions on Computers, vol. 53, no. 11, pp. 1474-1482, November 2004.

[5] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System," Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, pp. 1-10, May 2010.

[6] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Sixth Symposium on Operating System Design and Implementation, San Francisco, pp. 137-150, December 2004.

[7] Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung, "The Google File System," Proceedings of the 19th ACM Symposium on Operating Systems Principles, NY, pp. 20-43, October 2003.

[8] Yonghong Yan, Max Grossman, Vivek Sarkar, "JCUDA: A Programmer-Friendly Interface for Accelerating Java Programs with CUDA," Proceedings of the 15th International Euro-Par Conference on Parallel Processing, pp. 887–899, August 2009.