International Journal of Computer Networks and Communications Security

VOL. 2, NO. 7, JULY 2014, 232–235
Available online at: www.ijcncs.org
ISSN 2308-9830

Image Encryption Using Parallel RSA Algorithm on CUDA

Vaibhav Tuteja[1]

School of Information Technology and Engineering, VIT University, Vellore, India

E-mail: vaibhav.tuteja2011@vit.ac.in

ABSTRACT

In this paper we discuss Image Encryption and Decryption using RSA Algorithm which was earlier used for text encryption. In today's era it is a crucial concern that proper encryption decryption should be applied so that unauthorized access can be prevented. We intend to build a general RSA algorithm which can be combined with other image processing techniques to provide new methodologies and better encryption decryption efficiency. One such implementation is by using edge detection method and converting the images to their filtered form. CUDA is a platform for parallel algorithm implementation using CPU with GPU support. The following technique has been implemented on CUDA considering host and device interaction process. Thus, to make the algorithm more efficient we parallelize the algorithm using CUDA block and grid methodology.

Keywords: *Encryption, Decryption, RSA, GPU, Host, Device.*

1 INTRODUCTION

In 1977, an algorithm called as the RSA Algorithm was published by R. Rivest, A. Shamir, and L. Adleman. RSA is an asymmetric encryption system which uses two keys to encrypt and decrypt data, namely the public key and the private key. Depending on the requirement one of the keys can be used for encryption while the other can act as the decryption key. Various image processing techniques can be used to encrypt and decrypt the images like edge detection, histogram distortion, pixel filtration, etc. The following algorithm suggest the use of text encryption using RSA for the generation of encrypted keys which can be used and to initiate the encryption and decryption process. Thus in order to initiate the image encryption an encryption key (cipher form) must be generated using the RSA technique and corresponding to it a decryption key will be generated (cipher form). Both these keys can be passed as a parameter to a switch case holding the image processing algorithms. If the generated key matches with the case value of the image processing algorithm the process will begin. Thus a ciphered image will be generated which can be passed on to the receiver to decrypt the image. This methodology helps in better and secure transfer of data. In order to increase the efficiency of the algorithm, parallel computing is required which can be accessed using CUDA host-device computation technology. Each image will be processed on different block and grid combinations controlled by different devices in parallel. The host accepts the data and copies it to the GPU devices for parallel execution using thread processing techniques.

2 EXISTING RSA ALGORITHM

Generation of public and private keys using RSA text based encryption.

A. Key generation

Choose two large distinct primes p and q and then form the public modulus n = pq.

Choose public exponent e to be coprime to $(p-1)(q-1)$, with $1 < e < (p-1)(q-1)$.

- The pair (n, e) is the public key.
- The private key is the unique integer $1 < d < (p-1)(q-1)$ such that $ed = 1 \bmod (p-1)(q-1)$. Encryption: Split a message M into a sequence of blocks M1, M2, . . . , Mt where each Mi satisfies $0 \leqslant Mi < n$. Then encrypt these blocks as

(1)Decryption: Given the private key d and the cipher text C, the decryption function is:

(2) Note that encryption does not increase the size of a message. Both the message and the cipher text are integers in the range 0 to n - 1.The encryption key is thus the pair of positive integers (e; n). Similarly, the decryption key is the pair of positive integers (d; n). Each user makes his encryption key public, and keeps the corresponding decryption key private.

B. Image Encryption

Any image processing approach can be used with the RSA algorithm to encrypt or decrypt the image.

One of the technique used in implementation is the **edge detection technique** which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities**.**

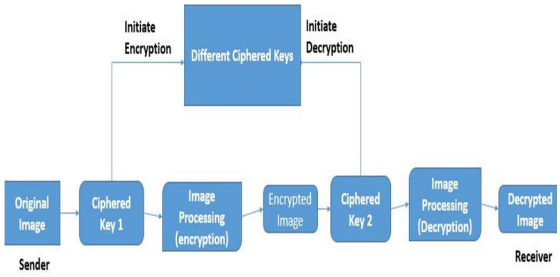
- Two filters to detect horizontal and vertical change in the image.
- Computes the magnitude and direction of edges.
- We can calculate both directions with one single CUDA kernel.

$$C_{horizontal} = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

$$C_{vertical} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

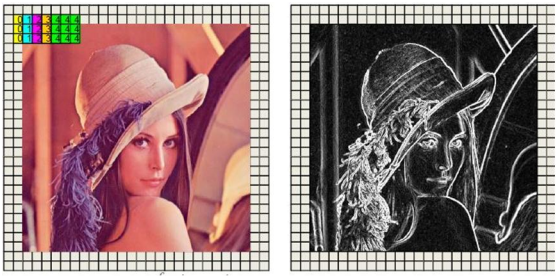
$$Magnitude_{Sobel} = norm \bullet \sqrt{G_{horizontal}^2 + G_{vertical}^2}$$

$$Direction_{Sobel} = \arctan\left(\frac{G_{vertical}}{G_{horizontal}}\right)$$



Proposed Architecture

3 SCHEME USED



With each iteration a particular pixel value is fetched and is compared with the neighbouring pixel values. The process continues till the highest pixel intensity is found out. Comparing the picture pixel values and the window pixel values an edge is formed corresponding to the image pixel value. Thus at the end of the process only those pixel values are seen which have a particular intensity from the image matrix.

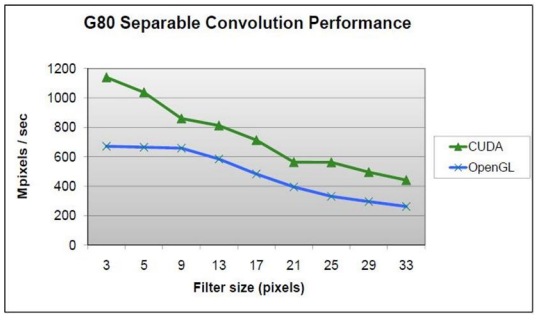


Fig. 1. The compatibility of CUDA with OPENGL

4 PARALLEL ARCHITECTURE

A. The Programming Model

The CUDA parallel hardware architecture is accompanied by the CUDA parallel programming model that provides a core to core of abstractions that enable expressing fine-grained and coarse-grain data and task parallelism. The following languages that can be chosen to express parallelism in high-level languages like C, C++, FORTRAN or driver APIs such as OpenCL™ and DirectX™-11 Compute

B. Advantages and Limitations of CUDA

- *Advantages*

CUDA is better than several other traditional general purpose computation on GPU-GPU (GPGPU) using graphics APIs.

Scattered reads – code can read from arbitrary addresses in memory.

Shared memory – CUDA exposes a fast shared memory region (16KB in size) that can be shared amongst the threads.

In order to have a higher bandwidth shared memory can be used as a user-managed cache, using texture lookups

Fast access to GPU for copying and retrieving of data. Supports integer and bitwise operations, with additional integer texture lookups.

- *Limitations*

Just like C, CUDA has a recursion-free, function-pointer-free support, with simple extensions. However, a single process should run spread across multiple disjoint memory spaces, making it different from other C language runtime environments. Fermi GPUs now have full support of C++.

Texture rendering is not supported.

For double precision (only supported in newer GPUs like GTX 260) there are some deviations from the IEEE 754 standard: round-to-nearest-even is the only supported rounding mode for division, reciprocal, and square root. In, denormals, signalling NaNs and single precision are not supported; only two IEEE rounding modes are supported (chop and round-to-nearest even), and those are specified on a per instruction basis rather than in a control word; and the precision of division/square root is slightly lower than single precision.

The latency and bus bandwidth between the GPU and the CPU may be limited.

In order to achieve the best performance, 32 threads should be grouped together, with total of 1000 threads. Branches in the code do not affect performance to a countable rate, provided that each of 32 threads takes the same path for execution; One of the limitation is the SIMD execution model for inherently divergent tasks (e.g. traversing a space partitioning data structure during ray tracing).

Unlike OpenCL, CUDA-enabled GPUs are only available from NVIDIA (GeForce 8 series and above, Quadro and Tesla).

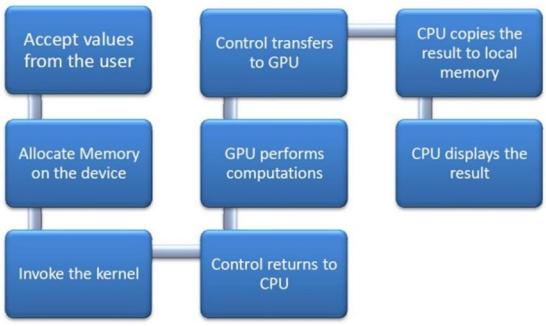


Fig. 2. The basic working principle for CUDA.

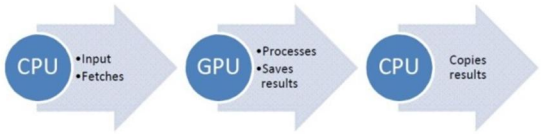


Fig. 3. Data Flow Diagram

C. Design Model

- Scan for CUDA enabled devices:
- IF(device not found)
- Exit;
- ELSE
- Copy to device
- Start the kernel
- Generate Ciphered Keys (for encryption and decryption).
- WHILE(k!=0)
- {
- IF(k>=2)
- C=C*((m*m)%n) and k=k-2;
- ELSE IF(k<2&&k>0)
- C=C*(m%n) and k=0;
- ELSE
- C=C%n;
- }
- Scan image and key parameters {k,n}
- Switch(generated key)
- Match the case values
- IF match found (will be used while encryption as well as decryption),
- Start the image encryption or Decryption Process in the block (will vary with different image processing techniques),
- Else exit
- Exit the kernel
- Copy values from device to host

5 CONCLUSION

The new fusion of RSA with image processing techniques is better than various other techniques since it is fast and efficient because of the parallelism embed within. Secondly the algorithm is independent of a particular image processing algorithm. One may use any algorithm based on his/her choice or requirement of the system. There is a lot more flexibility in the system – one may replace RSA for the ciphered key generation process with any other encryption techniques. On the whole this approach is a fusion of security, integrity, flexibility and efficiency.

6 REFERENCES

[1] Zhao, Gaochang, et al. "RSA-based digital image encryption algorithm in wireless sensor networks." Signal Processing Systems (ICSPS), 2010 2nd International Conference on. Vol. 2. IEEE, 2010.

[2] Chang, Chin-Chen, Min-Shian Hwang, and Tung-Shou Chen. "A new encryption algorithm for image cryptosystems." Journal of Systems and Software 58.2 (2001): 83-91.

[3] Manavski, Svetlin A. "CUDA compatible GPU as an efficient hardware accelerator for AES cryptography." Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on. IEEE, 2007.

[4] Nickolls, John, et al. "Scalable parallel programming with CUDA." Queue 6.2 (2008): 40-53.

[5] Ryoo, Shane, et al. "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA." Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. ACM, 2008.