2014 Fifth International Conference on Intelligent Systems, Modelling and Simulation

Speeding Up RSA Encryption Using GPU Parallelization

Chu-Hsing Lin, Jung-Chun Liu, and Cheng-Chieh Li
Department of Computer Science
Tunghai University
Taichung 40704, Taiwan
{chlin, jcliu, s993957}@thu.edu.tw

***Abstract*—Due to the ever-increasing computing capability of high performance computers today, in order to protect encrypted data from being cracked, the bit number used in RSA, a common and practicable public-key cryptosystem, is also getting longer, resulting in increasing operation time spent in executing the RSA algorithm. We also note that while the development of CPU has reached limits, the graphics processing unit (GPU), a highly parallel programmable processor, has become an integral part of today's mainstream computing systems. Therefore, it is a savvy choice to take advantage of GPU computing to accelerate computation of the RSA algorithm and enhance its applicability as well. After analyzing the RSA algorithm, we find that *big number* operations consume most parts of computing resources. As the benefit acquired from combining *Montgomery* with GPU-based parallel methods is not high enough, we further introduce the Fourier transform and *Newton's method* to design a new parallel algorithm to accelerate the computation of *big numbers*.**

Keywords-RSA; Montgomery method; CUDA; Fast Fourier Transform; Newton's method, Cooley-Tukey algorithm, big number operation

I. INTRODUCTION

In recent years, due to the vigorous development of the information technology, various industries have integrated with the IT industry, and the intelligent (smart) city has been introduced to highlight the growing importance of Information and Communication Technologies (ICTs). In addition, wireless networks and handheld devices have been widespread; consequently, their related technologies have also become hot issues. Along with the rapid development of the information technology, communications and transmissions of information have become an integrated part of our everyday life; as a result, security of data cannot be overemphasized to protect information from being misused by malicious parties. In view of this, people have paid much attention to information security and governments have set laws, e.g., Personal Data Protection Act, to protect personal information security and privacy. In the information technology industry, basic security encryption techniques have been further extended and developed, such as improving hashing algorithms to make it faster and more secure, or to strengthen performance of asymmetric key cryptography and add more bits into the key length in order to enhance the security level of the algorithm.

This paper has two focuses on: (1) heterogeneous computing to develop a Parallel Universal Kernel Operations (PUKO for short) algorithm for general operations of cryptographic components, such as modulo division, addition/subtraction and multiplication/division for *big numbers*; (2) the application of the proposed method on the RSA encryption system. We use high-speed computing capability of GPU parallel hardware architecture to enhance performance of cryptographic systems, so as to make the encryption/decryption and authentication systems more efficient and applicable.

II. BACKGROUND

A. Heterogeneous computing

Heterogeneous computing has become popular in dealing with complex operations; in which CPU handles complex logic operations and GPU handles simple operations of huge amount. The differences between GPU and CPU architectures are illustrated in Figure 1.

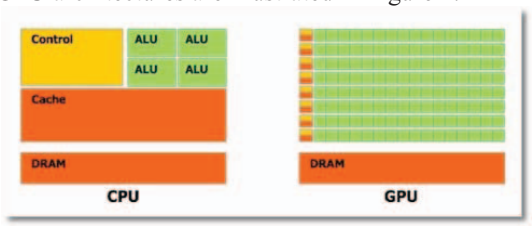

Figure 1. Cpu And Gpu Architectures.

B. Computing unit in GPU

A streaming multiprocessor (SM) is composed of many streaming processors, also called core. So the number of GPU cores depends on the number of SMs. Each SM, with its own control units, registers, execution pipelines, and caches, performs the actual computations. Every SM may contain 8 streaming processers (SPs).

C. CUDA kernel

The *CUDA* kernel is essentially executed by multiple blocks of threads; threads in the same block need to share data, therefore they must be transmitted in the same SM. Each thread in one block is passed to an SP for execution.

2166-0662/14 $31.00 © 2014 IEEE
DOI 10.1109/ISMS.2014.97

IEEE computer society

D. *Active blocks in SM*

In the operation of SM, many blocks will be kept running for execution and are called active thread blocks; the reason for this design is to hide delays. The next block waiting for execution can occupy resources when a block performs memory accesses which may causes delay in operation.

E. *Memory model*

Figure 2 depicts a *CUDA* memory model. In the practical application, the global memory and shared memory will be used.

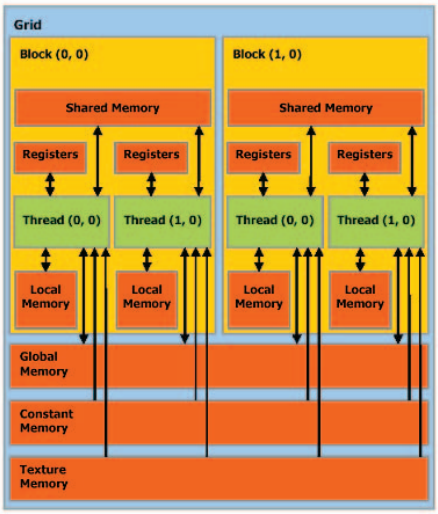

Figure 2. Cuda Memory Model.

III. METHODOLOGY

A. *RSA encryption algorithm*

RSA, one of the first practicable public-key cryptosystems and widely used for secure data transmission, was first publicly described by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. It uses public key cryptography to encrypt data as a way to achieve data encryption and digital signature.

RSA is divided into three steps: key generation, public key encryption, and private key decryption as shown in Table 1. For illustration, an application for confidentiality is described as follows:

(1) Entity A uses entity B's public key to perform encryption operation on message *M* to generate a ciphertext *C*.

(2) Entity A transmits the ciphertext to entity B.

(3) Entity B uses his/her own private key to conduct the decryption operation on the cipthertext from entity A, and the message M is revealed.

B. *Speedup of RSA*

Many important cryptosystems such as RSA are based on arithmetic operations, such as multiplications, and modulo on a large number. Today mainstream parallel algorithms use *Montgomery* reduction to do modular multiplication parallelism, but there is a problem that seriously affects efficiency of the *Montgomery* iterative algorithm (refer to Table 2): as the computation time increases, the active threads decreases and consequently, the available amount of parallel decreases. Thus, it is important to accelerate large number operations for RSA in other approach.

TABLE I. THREE STEPS OF RSA

1. RSA key generation
Randomly select two prime numbers p and q, and p is not equal to q.
Calculate N = p * q, and use N to compute the public key and private key.
Calculating φ (N) = (p-1) (q-1), φ representative of Euler function.
Choose an integer e, such that 1 <e <φ (N) and the GCD (e, φ (N)) = 1, this time (e, N) is the public key.
Calculate d, d ≡ e ^ (-1) mod φ (N), i.e., to obtain the private key (d, N).
2. Encryption
C ≡ M ^ e mod N
3. Decryption
M ≡ C ^ d mod N

TABLE II. ITERATIVE ALGORITHM OF MONTGOMERY REDUCTION

```
c[0] = 0
for i = 0 to k − 1 loop
p = (c[i] + a_i × b)(mod 2)
c[i + 1] = (c[i] + a_i × b + p × n)(div 2)
end loop
return c[k]
```

C. *Multiplication*

In this paper we use Cooley-Tukey Fast Fourier Transform (*FFT*) algorithm; its complexity is $O(n\log n)$, which can be further reduced to $O(\log n)$ using GPU. The key to perform Fast Fourier Transform in parallel lies in dealing with the butterfly diagram inside the Cooley-Tukey algorithm, as shown in Figure 3. We created two array and

indirectly accessed data so as to perform parallel computing without interference.

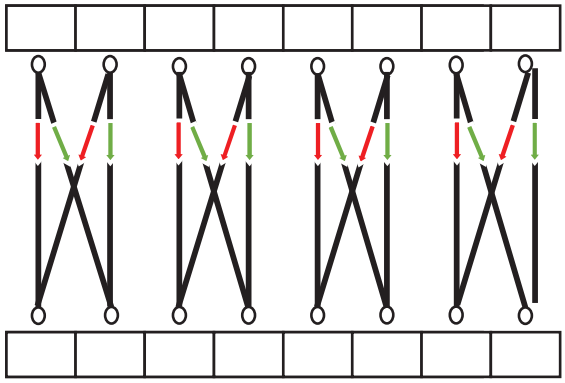

Figure 3. Butterfly Diagram In Cooley-Tukey Algorithm.

The *FFT* operations in each of the GPU thread is listed in Table 3.

TABLE III. FFT OPERATIONS IN EACH GPU THREAD

```
d_buffer[x].r=(Array[x+(loop/2)].r*d_wm[(x%(loop/2))*(len/loop)].
r-Array[x+(loop/2)].i*d_wm[(x%(loop/2))*(len/loop)].i)+Array[x].r

d_buffer[x].i=(Array[x+(loop/2)].r*d_wm[(x%(loop/2))*(len/loop)].
i+Array[x+(loop/2)].i*d_wm[(x%(loop/2))*(len/loop)].r)+Array[x].i
```

In order to let all the threads have the same computational complexity, we prebuilt in the HOST a look-up table as shown in Table 4 for ω values, which are passed to the GPU; in this way, each thread operation needs only one look-up table access to retrieve the corresponding ω value.

TABLE IV. PREBUILT LOOK-UP TABLE FOR ΩVALUE

```
h_w[0].r=1;
h_w[0].i=0;
h_wm.r = cos(on*2*PI/len)
h_wm.i = sin(on*2*PI/len)
for i=1 to len/2 ;i++ loop
h_w[i]=Virt_Multi(h_w[s-1],h_wm)
end loop
```

D. Division

For division, we used *Newton's method* based iterative algorithm, as tabulated in Table V.

Since in the RSA public key system, the modulus N is a fixed value, we can use two multiplications, as shown in Equation (1), to replace the complex modulo operation, greatly reducing the operation complexity and thus the computation time; the higher the efficiency of multiplication is, the more the improvement of the modulo operation achieves.

TABLE V. ALGORITHM OF NEWTON'S ITERATIVE METHOD

```
int div
X1=1/B // Take the top three
for int i=0 to 5 loop
X1=(2-X1*B)*X1
end loop
A=A*X1
div=A
```

$$A \bmod N = A - (A * X1) * N \qquad (1)$$

Where A is the large number, *N* is the modulus, and *X1* is (1/*N*)'s convergence number.

E. Addition and Subtraction

For addition and subtraction operations, we are faced with algorithms with floating-point numbers due to the use of the Newton's iterative method; accordingly, we try to perform addition and subtraction in parallel on big floating-point numbers.

For addition operation, in order to minimize logic operations in the GPU side, most logic judgments are performed in advance inside the CPU, as shown in Table 6, then passed to the GPU to achieve maximum computation efficiency.

TABLE VI. LOGIC OPERATIONS FOR ADDITION PERFORMED IN CPU

```
If a.len>b.len
If pointflag
Add_a<<<blocks,threads>>>(d_Aa,d_Ab,len,point,pointflag);
else
Add_b<<<blocks,threads>>>
(d_Aa,d_Ab,len,point,pointflag);
end else  end if
else
if !pointflag
pointflag=1;
Add_a<<<blocks,threads>>>(d_Ab,d_Aa,len,point,pointflag);
end if
else
pointflag=0;
Add_b<<<blocks,threads>>>(d_Ab,d_Aa,len,point,pointflag);
end else  end else
```

As for subtraction operations, the logic judgments are not as complicated as those in addition operations since the order of operands are decided before subtraction operations; besides, operations in the GPU for performing subtraction are the same as addition.

IV. Results and Discussion

For the experiments, we use CPU: Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz 2.33GHz with RAM: 4.00 GB. For the graphic processing unit, we use GPU: GTX 650Ti. And we use Windows 7 as the operation system (64bit). Performance comparisons of *FFT* and multiplication operations executed by CPU and GPU are listed in Table 7 and Table 8, and are plotted in Figure 4 and Figure 5. As shown in Table 8, after applying results of fast Fourier transform into multiplication operations, we find more than 50 folds speedup using GPU compared to using CPU when the data bits are more than 8192 bits. Besides, for any amount of data within a block, GPU always manage to complete operations in a time span, and can output the accelerated computation results to be applied in the *Montgomery* method for further operations.

TABLE VII. *FFT* PERFORMANCE COMPARISON BY GPU AND CPU (THE UNIT OF THE DATA LENGTH IS IN BITS, AND THE TIME IN MS)

	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144
GPU	0	1	1	1	1	1	1	2	3	5	11	24
CPU	1	1	2	3	6	14	28	55	115	217	550	1159

TABLE VIII. MULTIPLICATION PERFORMANCE COMPARISON BY GPU AND CPU (THE UNIT OF THE DATA LENGTH IS IN BITS, AND THE TIME IN MS)

	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144
GPU	1	1	1	1	1	1	1	2	4	7	13	26
CPU	1	1	2	5	11	27	58	126	198	543	990	2685

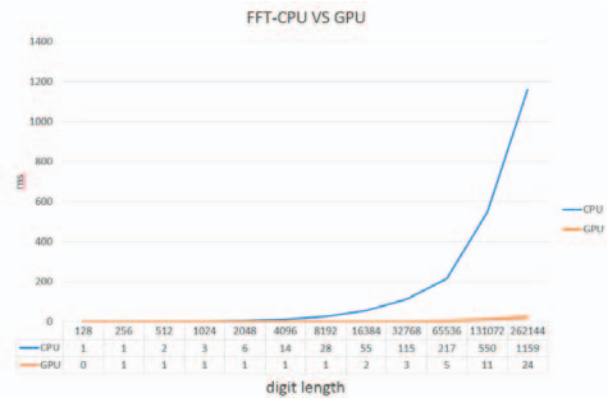

Figure 4. Performance Comparisons Of *Fft* Operations By Gpu And Cpu.

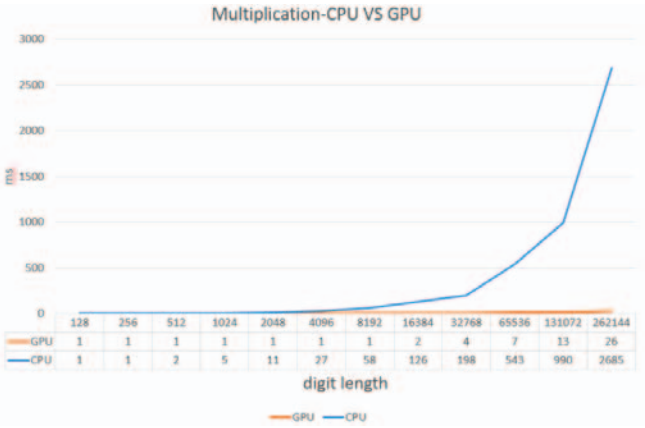

Figure 5. Performance Comparisons Of Multiplication Operations By Gpu And Cpu.

V. CONCLUSION

We take advantage of GPU computing to accelerate computation of the RSA algorithm and enhance its applicability. The parallel efficiency of Montgomery iterative algorithm is seriously reduced since as the computation time increases, the active threads decreases and the available amount for parallel operations decreases. Thus, we accelerate large number operations for RSA algorithm by FFT. We also use the fact that the modulus N is a fixed value for the RSA algorithm; as a result,) can be performed by one iteration in Newton's iterative method, and each modulo operation can be computed by two multiplication operations. Furthermore, the proposed method can be applied to other asymmetric encryption methods for accelerating computing; of course, some adjustments must be made according to characteristics of the algorithm to maximize its performance.

Acknowledgement: this research is in part supported by the National Science Council of Taiwan, under grant number NSC 102-2221-E-029-013

REFERENCES

[1] McIvor, C. McLoone, M. and McCanny, J.V.. "Modified Montgomery modular multiplication and RSA exponentiation techniques," IEE Proc.-Comput. Digit. Tech., Vol. 151, No. 6, November 2004

[2] Hong, Sunpyo, and Hyesoon Kim. "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. , 2009.

[3] Nukada, Akira, et al. "Bandwidth intensive 3-D FFT kernel for GPUs using CUDA." IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. .

[4] Frigo, Matteo, and Steven G. Johnson. "FFTW: An adaptive software architecture for the FFT," Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, 1998.

[5] Owens, John D., et al. "GPU computing," Proceedings of the IEEE 96.5 (2008): 879-899.

[6] Ciet, Mathieu, et al. "Parallel FPGA implementation of RSA with residue number systems-can side-channel threats be avoided?" 2003 IEEE 46th Midwest Symposium on Circuits and Systems, Vol. 2., 2003.

[7] Kirk, David. "NVIDIA CUDA software and GPU parallel computing architecture," International Symposium on Memory Management, 2007.

[8] Montgomery, Peter L. "Modular multiplication without trial division," Mathematics of Computation, 44.170 (1985): 519-521.