UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: _____
_____
_____

Academic Session: _____

I _____

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____

(Author's signature) (Supervisor's signature)

Address:

_____

_____ _____

_____ Supervisor's name

Date: _____ **Date**: _____

PERFORMANCE COMPARISON of ENCRYPTION ALGORITHM UNDER CPU and GPU PLATFORM

By

CHEONG HON SANG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Perak Campus)

January 2015

DECLARATION OF ORIGINALITY

I declare that this report entitled "PERFORMANCE COMPARISON of ENCRYPTION ALGORITHM UNDER CPU and GPU PLATFORM" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : _____

Name : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my super visor, Mr. Lee Wai Kong who has given me this bright opportunity to engage in this research project. This is my first step to run a research. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support and continuous encouragement throughout the course.

Abstract

Nowadays, technologies are getting more and more advance and this causes the processing speed of data become faster. Due to this improvement of technology, data that send through the internet are getting larger. Other than that, the technology of internet speed are getting more mature and can send more information through internet but the processing speed of the processor cannot catch up will the internet speed. In this report, research will be carried out to get the performance of the symmetric key block cipher, which is used during internet data transfer, under multiple platforms to show that the performance speed of symmetric key block cipher can be increase through parallel programming.

The different platform selected in this report is normal computing platform, multiple core platform and GPU platform. C programming is used in normal computing platform. OpenMP is used in multiple core platforms and GPU platform is coded with CUDA.

The time taken for both of the symmetric key block cipher algorithm to process will be recorded. Graph will be plotted with data throughput against input data size. From the result collected through experiment, the objective of this report will be answered.

Other than that, random numbers are needed to ensure the security of a symmetric key block cipher. Random number can help ensuring that the security of the algorithm to be unpredicted thus increase the safety of using it. Therefore, random numbers are essential and critical to any security algorithm.

This paper also carried out research to generate pseudo random number with symmetric key block cipher in counter mode under GPU platform. After generating the pseudo random number, that set of pseudo random number will be tested with statistical tests: TestU01 and NIST Test Suite. Only after the generated pseudo random number passed the statistical test, then the pseudo random number will only be considered random enough to be applied in security.

Table of content

List of Tables

List of Figures

List of Abbreviations

CUDA	Compute Unified Device Architecture
OpenMP	Open Multi-Processing
OpenSSL	Open Source Toolkit for SSL/TLS
GPU	Graphic Processing Unit
GPGPU	General-purpose Computing on Graphics Processing Unit
IDEA	International Data Encryption Algorithm

Chapter 1 Introduction

1.1 Problem Statement

1.1.1 To optimize the performance of the block cipher algorithm so that the algorithm with be faster than other researches.

Nowadays many researchers had research on the use of GPU in cryptography algorithm. The main concern of this paper is to ensure that the algorithm that runs under GPU platform can perform better than what other researches show.

1.1.2 To generate pseudo random number that passes the statistical test

It will also easy to generate pseudo random number but it will be hard to generate pseudo random number that will pass statistical test that prove randomness of the pseudo random number generated.

1.2 Motivation

Nowadays, the rapid advancement of cloud technology and the improvement of internet technology increase the demand for high speed secure internet connection. The speed of the processor cannot catch up with the advancement of cloud technology and internet technology. The used of all this secure internet connection will required the used of random number which can maintain the security of the connection.

Therefore, many researchers had targeted their study on fast implementation of cryptography algorithm under GPU platform.

1.3 Project Background

The importance of cryptography on ensuring security or integrity of the electronic data transaction had become higher during past few years. Each day millions of users apply block cipher on their daily activity that involve internet. The data packet had to be encrypted with block cipher in order to be sent out through internet.

Therefore, 1000 of block of data is encrypted and decrypted with the block cipher algorithm when a user use internet for his or her daily activity. A great amount of time is used for this encryption and decryption with the block cipher algorithm. Other than that, server also had to

use up along time for block cipher encryption and decryption in order to reply to the client request.

For user, the block cipher algorithm will not bring out much effect to the performance of the computer, since the input data block receive from the internet is relatively small and acceptable. Whereas, the block cipher algorithm will slow down the performance of the server as the input data block received by the server is larger than normal personal computer. The block cipher algorithm will consume a huge amount of computer resources and thus causes slow reply to the client.

In order for the entire block cipher algorithm to stay secure, a source of random data is required in the block cipher algorithm. Generating a large amount of distinct random number would be a time consuming task. When there are a lot of client try to connect to the server, server will used a lot of time and computer resources to generating the random number and causes slow reply to client.

The target of this paper is to research on the performance speed of the 3 block cipher algorithm (Blowfish, IDEA and Threefish) in counter mode with different platform. The research will be split into 3 parts. The first part is to test the performance speed of the 2 block cipher algorithm in sequential programming under normal environment. Second part is to test the performance speed of the 2 block cipher algorithm in parallel programming with OpenMP library under multi-threaded environment. This part will require a multi-core processor in order to be able to test the performance. Third part is to test the performance speed of the 2 block cipher algorithm in parallel programming with CUDA framework provided by NVIDIA under GPU platform.

The second target of this paper is to research on the pseudo random number generate by the selected 3 block cipher algorithm (Blowfish, IDEA and Threefish) in counter mode. The research will be split into 2 parts. The first part is to generate pseudo random number from the selected 3 block cipher algorithm in counter mode. Then, conduct a statistical analysis on the 3 random number generators. The second part of the research is to optimize the random number generator and then conduct statistical analysis on the optimized generator.

1.4 Project Scope

In this report, parallel programming will be applied on counter mode symmetric block cipher Blowfish and IDEA by using CUDA and OpenMP. In counter mode symmetric block cipher, encryption algorithm is applied to the counter that is 1 to N. Then, the result will be XOR with the original text to get the cipher text. The decryption of the counter mode is the same as encryption except the original text that is used to XOR with the encrypted counter is replace with the cipher text.

1.5 Objective

1.5.1 To parallelize the serial block cipher algorithm

The first objective of this paper is to apply parallelism on the serial block cipher algorithm so that the performance of the block cipher can be compare.

1.5.2 To compare the performance of the block cipher on normal CPU platform, multiple cores platform and CUDA platform

The second objective of this paper is to apply parallel programming on counter mode symmetric block cipher is to find out whether the performance changes according to different platform. This project will run on 3 different platforms. First, normal processors that can only execute the counter mode block cipher sequentially. Second, multi-cores processors that can run few threads of counter mode block cipher at the same time. Third, GPU that can run 1000+ threads of counter mode block cipher at the same time.

Before this project, other already attended to apply parallel programming on block cipher. This project is to extend the previous work by apply parallel programming on counter mode block cipher and identify difference in the performance of these different platform.

This project will focus on applying parallel programming on Blowfish algorithm and IDEA algorithm in counter mode. The key size for Blowfish algorithm is fixed to 128 bits while key size for IDEA algorithm is also 128 bits, the only key size for IDEA algorithm. The input block for Blowfish and IDEA algorithm is the same, 64 bits.

1.5.3 To generate random number from block cipher algorithm in counter mode under GPU platform

The third objective of this paper is to generate the random number from the block cipher algorithm in counter mode. Appropriate statistical analysis will be conducted to the random number generator to test the randomness of the random number generated.

1.5.4 To generate pseudo random number that pass statistical test

The fourth objective of this paper is to ensure that pseudo random number generated passes the statistical test. Statistical test conducted is to ensure the randomness of the pseudo random number generated.

1.6 Report Organization

The following are the organization of this report.

Chapter 2 Literature Review

This chapter is about which report from other research that this paper reference.

Chapter 3 Method / Technologies Involved

This chapter explains out all the method and technology involved in this paper to achieve the objective of this paper.

Chapter 4 System Design

This chapter includes the flow chart of all the research done and detail about each research carried out in this paper.

Chapter 5 Implementation and Optimization

In this chapter, there is explanation on how the research is implemented. Other than that, this chapter also explains what method used in this paper to optimize the performance of different research.

Chapter 6 Performance and Analysis

This chapter shows all the result from different research carried out in this paper and these chapters also compare and analyze the performance of different research.

Chapter 7 Conclusion

This chapter include the summarization of this paper, did the objective being achieved and who will be benefit from this research.

Reference

This is the last chapter of this paper and it include all the reference of this paper.

Chapter 2 Literature Review

2.1 GPU Acceleration of Block Ciphers in the OpenSSL Cryptographic Library

This paper had contributed the parallel programming on block cipher in OpenSSL cryptographic library. They had shown that the implementation of GPU on block cipher had increase the performance by a factor of 10 as show in figure 2.1.1. Other than that, they also show the guidelines to implement benchmarking cryptographic and other program with OpenSSL and CUDA from their experience in preparing this paper.

The symmetric key block cipher algorithm that is involved in this paper is AES, Blowfish, IDEA, DES, CAST5 and Camellia. The operational mode implemented in this paper is cipher block chaining (CBC) mode in decryption because only decryption of CBC mode can be parallelized. Electronic codebook (ECB) mode in both encryption and decryption are implemented in this paper. This 2 operational mode is selected because only this 2 mode which is parallelizable are available in the OpenSSL packages at the time of the paper is prepared.

There are 2 optimizations done in this paper : remove of redundant flexibility and remove of non-coalesced memory access. This paper removed redundant flexibility by passing as few parameter to the kernel as possible. The key is stored as T-table which is referenced using a pointer. Then, the non-coalesced memory access is remove so that multiple memory transaction can be avoid in this paper to improve the performance.

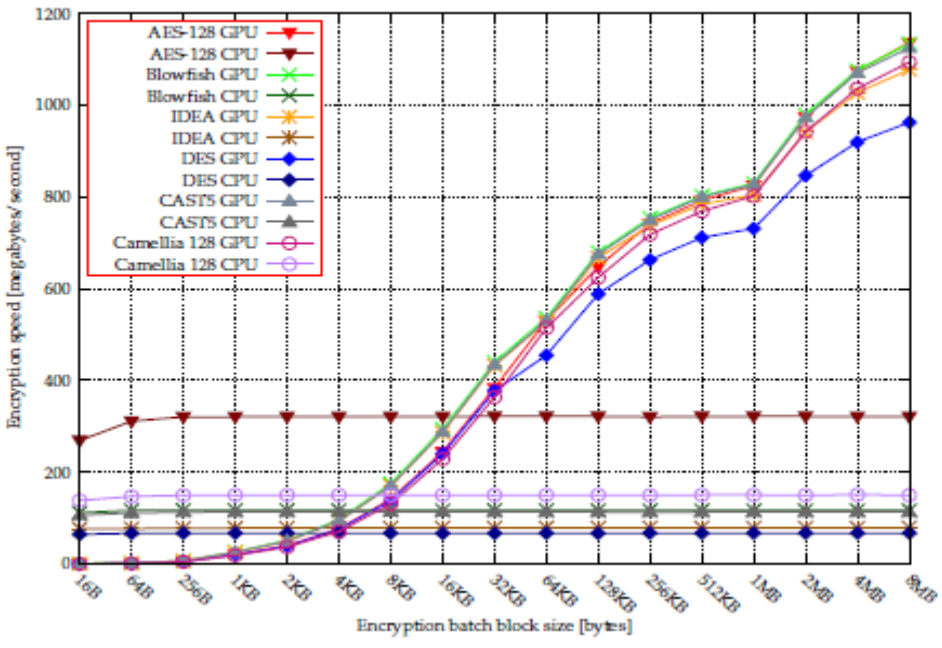

Figure 2.1.1 ECB encryption with CUDA on one PCB of a GeForce GTX 295 (Johannes Gilger, Johannes Barnickel & Prof. Dr. Ulrike Meyer)

There are some benchmark problems faced by the team. Most of the time the specific CPU implementation and number of cores in CPU was not mentioned. Even in one case, java was used to measure the performance of the CPU. Almost all the team that involved in this paper used different GPU and thus made the comparison of the result to be difficult.

2.2 High-Performance Symmetric Block Ciphers on CUDA

With the improvement of computer technology, internet speed had become more and more fast every day. In order to make sure the internet technology did not advanced ahead cryptographic module, a better performance with good cost of cryptographic module are needed in future. Since the performance of current processor is slow, therefore, study on GPGPU implementation on GPU for symmetric cryptographic had been conducted and this technique is highly depend on programmer's programming technique. The symmetric cryptographic that is research in this paper is 128-bit block ciphers AES, Camellia, CIPHERUNICORN-A, and Hierocyrpt-3.

There are a lot of factors that cause problem in implementing symmetric cryptographic with GPGPU on GPU. This paper had been able to gain 2 insights despite of so many limitations on this technique. First, this paper had concluded that the granularity that with give out the best throughput is 16 bytes/thread granularity. Second, extended key and substitution table are stored in the shared memory. The following image shows the best throughput on GPGPU implementation of the 4 selected encryption algorithm.

This paper only dealt with one operational mode which is electronic codebook (ECB) mode. This is because this mode contains the basic properties for the discussion of the parallel programming.

		Environment 1 [Gbps]	Environment 2 [Gbps]
AES	W/O TF	48.6	35.2
	With TF, With OL	27.5	22.0
	With TF, W/O OL	15.7	13.6
Camellia	W/O TF	50.6	35.4
	With TF, With OL	27.5	22.0
	With TF, W/O OL	15.9	13.7
CIPHER UNICORN-A	W/O TF	26.1	14.6
	With TF, With OL	21.3	12.8
	With TF, W/O OL	12.3	8.8
Hierocrypt-3	W/O TF	24.8	18.2
	With TF, With OL	20.4	15.5
	With TF, W/O OL	12.0	10.0

* TF, Transfer; OL, Overlap

Figure 2.2.1 the best throughput on GPGPU implementation (Naoki Nishikawa, Keisuke Iwai and Takakazu Kurokawa, 2011)

2.3 Parallel Random Numbers: as Easy as 1, 2, 3

There are a lot of pseudo random number generators available but most of the generator only works well in sequential instead of massive parallel high computation environment. This paper demonstrates how to use block cipher in counter mode to generate pseudo random numbers with excellent statistical properties.

There are 2 types of counter based pseudo random number generator: pseudo random number generator based on cryptography standard (Threefry) and a completely new pseudo random number generator (Philox).

All the pseudo random number presented in this paper was tested with statistical test and passed the test. The generators had passed the most comprehensive statistical test, TestU01 and able to generator at least of 2^{64} unique parallel stream of pseudo random numbers. The generators had passed smallcrush, crush and bigcrrush battery test from TestU01.

This paper also presented that Philox is faster than CURAND on a single NVIDIA GPU with Philox being able to generate pseudo random number at 201.6 GB per second per chip.

Method	Max. input	Min. state	Output size	Intel CPU cpB	Intel CPU GB/s	Nvidia GPU cpB	Nvidia GPU GB/s	AMD GPU cpB	AMD GPU GB/s
Counter-based, Cryptographic									
AES(sw)	(1+0)×16	11×16	1×16	31.2	0.4	–	–	–	–
AES(hw)	(1+0)×16	11×16	1×16	**1.7**	**7.2**	–	–	–	–
Threefish (Threefry-4×64-72)	(4+4)×8	0	4×8	7.3	1.7	51.8	15.3	302.8	4.5
Counter-based, Crush-resistant									
ARS-5(hw)	(1+1)×16	0	1×16	0.7	17.8	–	–	–	–
ARS-7(hw)	(1+1)×16	0	1×16	**1.1**	**11.1**	–	–	–	–
Threefry-2×64-13	(2+2)×8	0	2×8	2.0	6.3	13.6	58.1	25.6	52.5
Threefry-2×64-20	(2+2)×8	0	2×8	2.4	5.1	15.3	51.7	30.4	44.5
Threefry-4×64-12	(4+4)×8	0	4×8	1.1	11.2	9.4	84.1	15.2	90.0
Threefry-4×64-20	(4+4)×8	0	4×8	**1.9**	**6.4**	15.0	52.8	29.2	46.4
Threefry-4×32-12	(4+4)×4	0	4×4	2.2	5.6	9.5	83.0	12.8	106.2
Threefry-4×32-20	(4+4)×4	0	4×4	3.9	3.1	15.7	50.4	25.2	53.8
Philox2×64-6	(2+1)×8	0	2×8	2.1	5.9	8.8	90.0	37.2	36.4
Philox2×64-10	(2+1)×8	0	2×8	4.3	2.8	14.7	53.7	62.8	21.6
Philox4×64-7	(4+2)×8	0	4×8	2.0	6.0	8.6	92.4	36.4	37.2
Philox4×64-10	(4+2)×8	0	4×8	3.2	3.9	12.9	61.5	54.0	25.1
Philox4×32-7	(4+2)×4	0	4×4	2.4	5.0	3.9	201.6	12.0	113.1
Philox4×32-10	(4+2)×4	0	4×4	3.6	3.4	**5.4**	**145.3**	**17.2**	**79.1**
Conventional, Crush-resistant									
MRG32k3a	0	6×4	1000×4	3.8	3.2	–	–	–	–
MRG32k3a	0	6×4	4×4	20.3	0.6	–	–	–	–
MRGk5-93	0	5×4	1×4	7.6	1.6	9.2	85.5	–	–
Conventional, Crushable									
Mersenne Twister	0	312×8	1×8	2.0	6.1	43.3	18.3	–	–
XORWOW	0	6×4	1×4	1.6	7.7	5.8	136.7	16.8	81.1

Figure 2.3.1 Memory and Performance Characteristics (John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw, 2011)

2.4 High Speed Implementation of Symmetric Block Cipher on GPU

This paper present implementation and benchmark of symmetric block cipher algorithms: AES, Camellia and CAST5 with existing solution.

Camellia is block cipher algorithm that takes in 128 bits input and used either 128 bits, 192 bits or 256 bits key. This is an algorithm designed based on Feistal network of 18 rounds for 128 bits key and 24 rounds for 192 bits and 256 bits key. 4 32 bits S-Boxes are used in Camellia.

CAST5 is a block cipher algorithm that takes in 64 bits input and used key range with size from 40 bits to 128 bits. This paper presented CAST5 implemented with 128 bits key with 16 rounds. CAST5 used 8 32 bits S-Boxes. 4 S-boxes used in the key scheduling and the other 4 S-boxes used in the encryption or decryption algorithm.

All block cipher algorithm implemented in this paper is implemented in counter mode under GPU platform.

Throughput (Gbps)					
	TF	GPU	Camellia	CAST5	SEED
Nishikawa et al. [6]	W/O	Tesla C2050	50.6	N/A	N/A
Our work	W/O	GTX680	61.1	45.53	47.4
J. Gilger et al. [9]	W/O	GTX295	N/A	38.6	41.5
S. Lee et al. [10]	With	GTX285	N/A	N/A	9.5
Nishikawa et al. [6]	With	Tesla C2050	15.9	N/A	N/A
Our work	With	GTX680	44.9	40.5	38.6

*TF: Data Transfer

Figure 2.4.1 Comparison of Performance (Wai-Kong Lee, Bok-Min Goi, Raphael C.-W. Phan and Geong-Sen Poh, 2014)

Chapter 3 Method / Technologies Involved

Blowfish Algorithm

Blowfish algorithm is a symmetric-key block cipher, which means Blowfish algorithm will used the same key in both encryption and decryption process. Each of the input block of data must be 64 bits and the key used in the encryption can be any length 32 bits up to 448bits but key size is set to be 64 bits for Blowfish in this project.

Blowfish Encryption Algorithm

The input block will be first separate to left block and right block each with 32 bits of data. Then, the left block of data will XOR with the first P-array and perform Feistel cipher. After that, the result is XOR with the right block of data. The left and right block of data with be swap. This steps will be repeated for 14 times but access to incremented index of array P every repetition, i.e. : the left block of data is XOR with second P-array in first repetition and the left block of data is XOR with third P-array in second repetition. After the 14 times of repetition, the left block of data will XOR with the sixteenth P-array and then perform Feistel cipher. The results will XOR with the right block of data but this time do not need to swap the left and right block of data. The left block of data is then XOR with the seventeenth P-array and the right block of data is XOR with eighteenth P-array. The left block of data is merged with the right block of data to form cipher text.

Blowfish Decryption Algorithm

Blowfish decryption algorithm is same as the encryption algorithm except the P-array is used in reverse order, ie: eighteenth P-array is used in first XOR and seventeenth P-array is used in the second XOR.

International Data Encryption Algorithm (IDEA)

IDEA algorithm is another symmetric-key block cipher which is designed by James Massey, Xuejia Lai and was first described in year 1991. This encryption was to replace for Data Encryption Standard (DES). This algorithm was used to be patented in some countries but the last patent was expired in 2012 and IDEA is now patent free so it is free to use. IDEA uses 128 bits of key and the input block must be 64 bits.

IDEA Encryption Algorithm

The 64 bits input block will be split into 4 16 bits block. Then, 2 16 bits input block perform 16 bits addition modulo and another 2 16 bits perform 16 bits multiplication modulo using appropriate part of the round key. Then, perform 16 bits addition modulo and 16 bits multiplication modulo according to figure 3.1 using appropriate part of the round key and also perform XOR operation according to the figure below. Swap the middle 2 blocks of data. Repeat the same steps or encryption round for 8 times. Swap the middle 2 blocks of data and perform 16 bits addition modulo and 16 bits multiple modulo according to figure 4.2. Finally combine the 4 blocks of data as the final result.

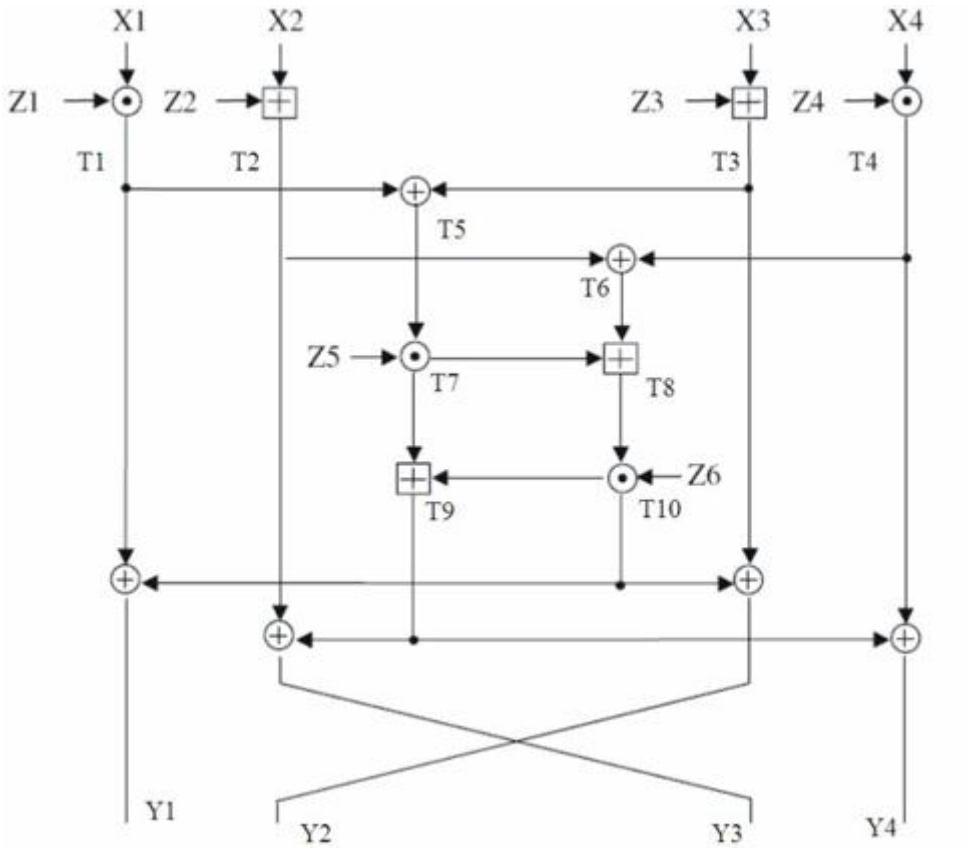

Figure 3.1 IDEA encryption round (Network App. Related Cores)

IDEA Decryption Algorithm

IDEA decryption work the same as the encryption but the difference is that the order of the round key is inverted and the sub keys is replaced with inverse value of the original sub keys.

Threefish Algorithm

Threefish algorithm is a symmetric-key tweakable block cipher with large input block size. It was designed as part of the Skein hash function, which is an entry in the NIST hash function competition. Threefish algorithm did not look up any table to avoid cache timing attack. The input block size for Threefish algorithm can be 256 bits, 512 bits or 1024 bits and the key size is the same as the input block size.

Threefish Encryption Algorithm

The 512 bits input block will be first split into 8 64 bits input block. Then, the input block will be pass to MIX function. A MIX function will consists of a single normal addition, a rotation by a constant, and an XOR.

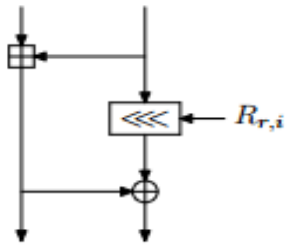

Figure 3.2 Mix Function (The Skein Hash Function Family)

The following figure show that how MIX function are used in the Threefish algorithm. Threefish algorithm consists of 72 rounds and consists of 4 MIX function followed by permutation of the split 8 64 bits input block. Different subkey is injected into the input block every 4 rounds. Each permutation is just 3 MIX function with constant and without adding key. After running the following cycle for 18 times, the final 8 input block is added with the key and tweak to get the cipher text.

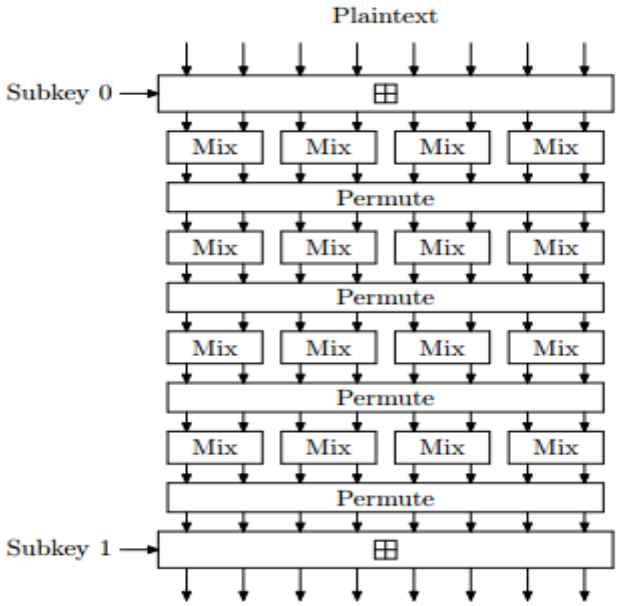

Figure 3.3 How Mix Function is used in Threefish Algorithm (The Skein Hash Function Family)

Threefish Dncryption Algorithm

Threefish decryption algorithm is exactly reverse process of the encryption. It started by minus out the key and tweak from the cipher text. Then, run the permutation reversely for 18 times to get the original message.

Mode of Operation

Mode of operation is an algorithm that uses block cipher to provide authentication service. There are 5 common operation modes for block cipher but counter mode is selected for this project.

First, a counter will be calculated out from 1 until n depend on the size of the input data. Then, the counter will be encrypted using Blowfish algorithm or IDEA algorithm. After that, the encrypted result will be XOR with the 64 bits input data. Finally, the result will be the cipher text for Blowfish algorithm or IDEA algorithm in counter mode.

For decryption, it is just the same as encryption. The only difference is that the encrypted result from Blowfish algorithm or IDEA algorithm is XOR with the cipher text to get back the original text as shown in figure 4.4.

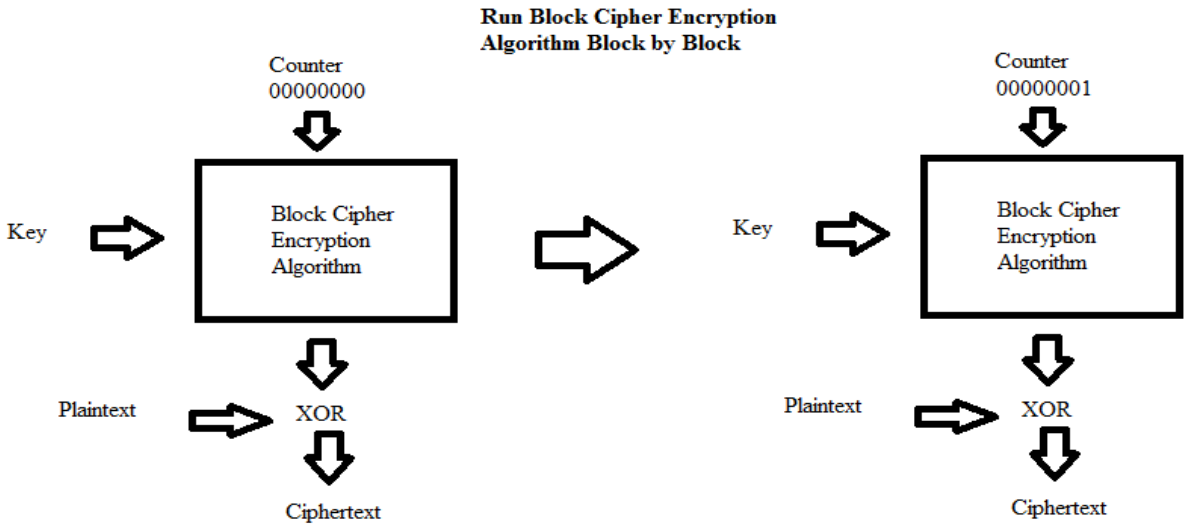

Figure 3.4 Block cipher encryption in counter mode

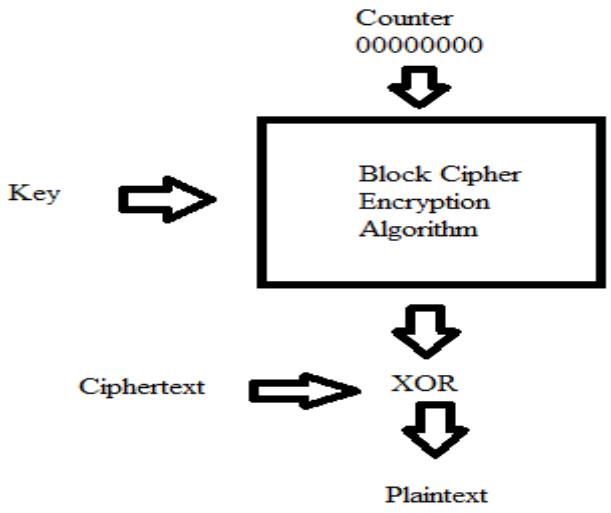

Figure 3.5 Block cipher decryption in counter mode

OpenMP

OpenMP is an API that can support multi-platform shared memory multiprocessing programming that is available in C, C++ and Fortran. OpenMP is based on the it's directive that programmer added into the source code to inform compiler which part of the code should be run in parallel. Since the source code is written for OpenMP to execute in parallel with the optimize speed, therefore, it can be sometimes hard to be understand.

OpenMP runtime library run in the operating system layer and the OpenMP library s work in program layer. Developer will develop application that will perform parallelism by making

use of the OpenMP library in program layer. The following image show the basic overview of OpenMP.

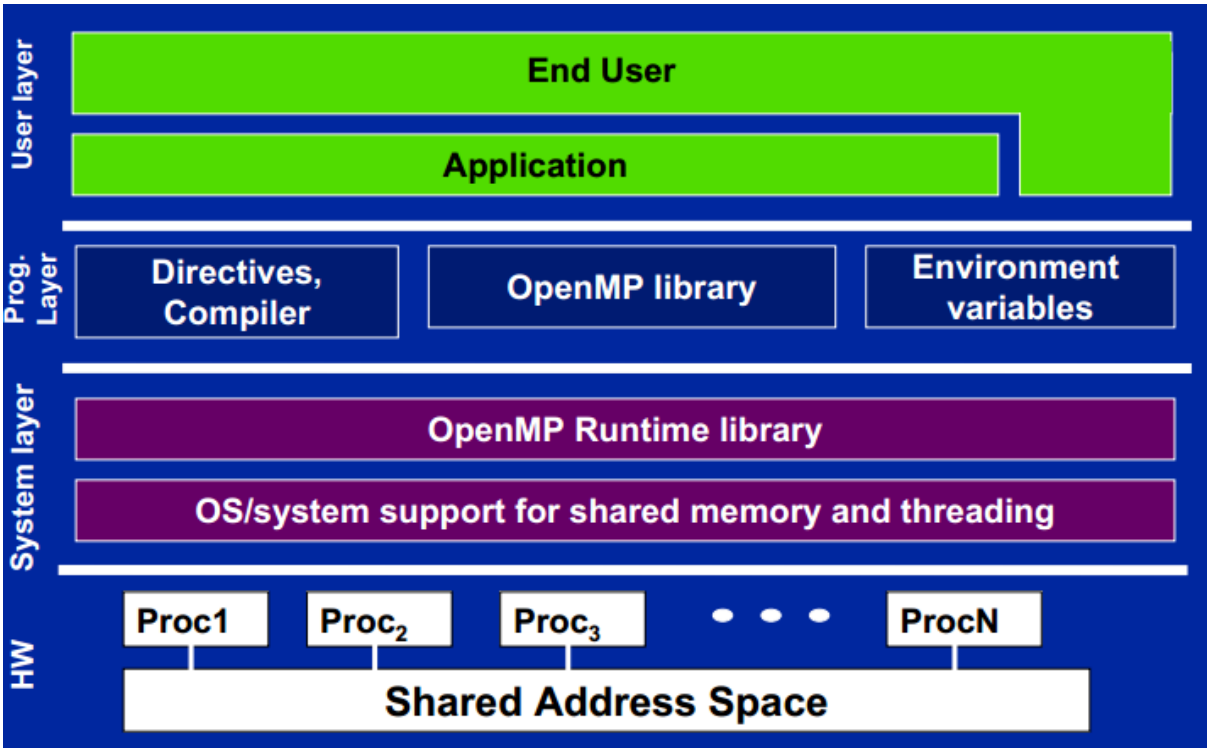

Figure 3.6 OpenMP Basic Overview (Tim Mattson & Larry Meadows. A)

OpenMP works in Fork-Join Parallelism. OpenMP working can be split into 2 parts : fork and join. In fork, the master thread will spawn multiple worker thread ass needed in the program. Then, all the worker threads will terminate and synchronize when the process is completed. Leaving only the master thread with the result collected from all of the worker threads. The following image is the programming model for OpenMP.

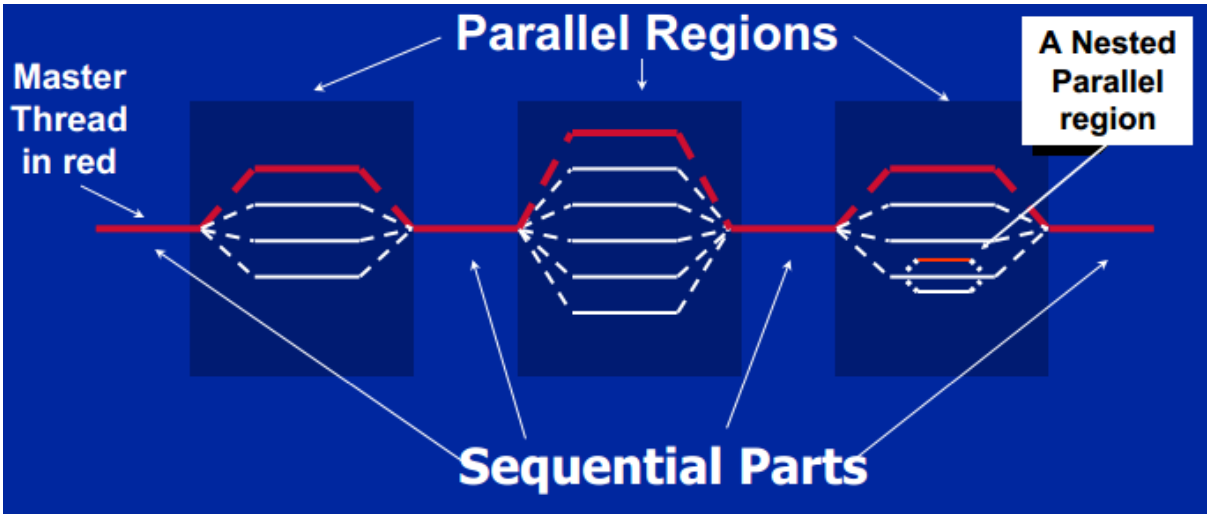

Figure 3.7 OpenMP Programming Model (Tim Mattson & Larry Meadows. A)

Compute Unified Device Architecture (CUDA)

CUDA is a parallel computing platform and programming model that is created by NVIDIA and can only be used or intergrated with graphic processing unit (GPU) created by NVIDIA. If the algorithm or program cannot break down to have at least 1000 threads, then that algorithm is not good for CUDA. In this project, the encryption algorithm can easily break down to 1000 threads and above. So, this project is suited for CUDA.

CUDA provide programmer or developer with a virtual instruction set and memory in the GPU created by NVIDIA. With CUDA, GPU can be used for general purpose processing also known as GPGPU. Unlike the way normal processor run thread, GPU run a huge amount of thread concurrently and slowly instead of running one thread very fast at a time.

CUDA is run using an array of threads and each thread will execute the same block of code. Each of the thread will have an id that it uses to calculate the memory addresses and make all the control decisions found in the source code. The following image shows how CUDA work. The thread may require cooperation on each other. Thread in CUDA can cooperate on each other on memory access, bandwidth reduction for some situation and avoid of redundancy by sharing result.

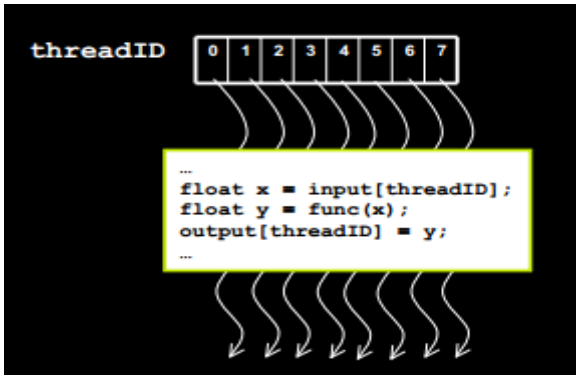

Figure 3.8 CUDA Basic Overview (NVIDIA corporation, 2008)

CUDA kernel is executed by grid of thread in block. Each block contain of threads that can cooperate with each other. Thread from different block within the same grid or different grid cannot cooperate with each other. The following image is CUDA programming model that shows how CUDA's kernel works. This allows the program to transparently scale to different

GPU. Transparent scalability means that the hardware, which is the GPU, is free to schedule thread block to any processor. Figure 4.9 shows the transparent scalability of NVIDIA GPU.

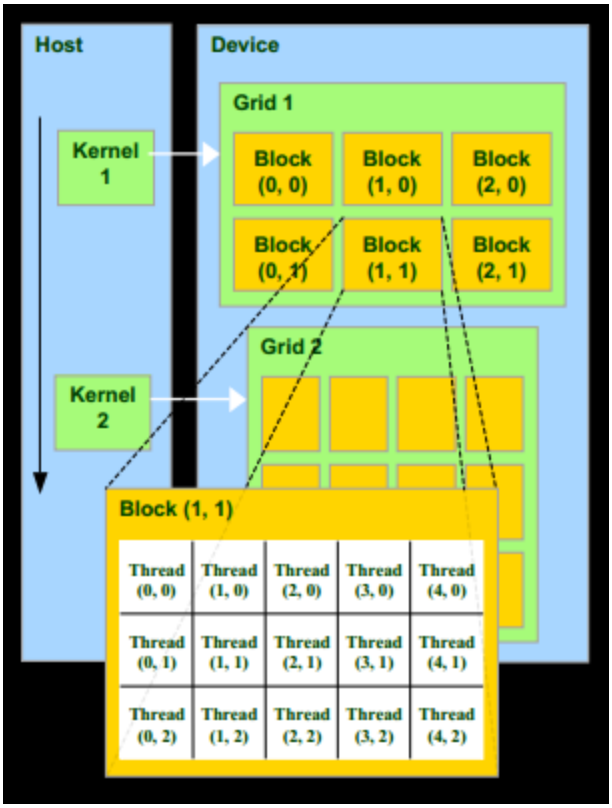

Figure 3.9 CUDA Programming Model (NVIDIA corporation, 2008)

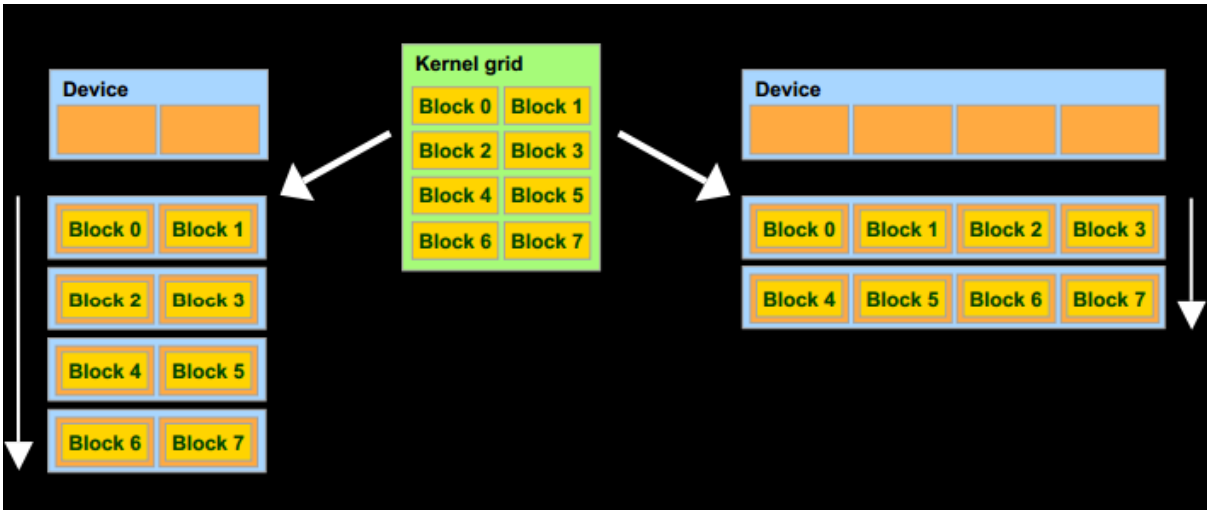

Figure 3.10 Transparent Scalability (NVIDIA corporation, 2008)

CUDA Warp Shuffle

There is always need for CUDA program to communicate with value between different parallel thread. The conventional way to do this is to use shared memory but there is only a limited amount of shared memory available for data communication between parallel thread. On NVIDIA GPU Kepler architecture, there is a new method to share data between parallel thread in the same warp.

A warp is an implicit synchronized group of threads and a thread in the same warp can read each other's register with a new instruction called SHFL on Kepler architecture. There is no shared memory is used in this method.

Figure 3.11 Warp Shuffle (GPU Technology Conference)

Streams and Concurrently

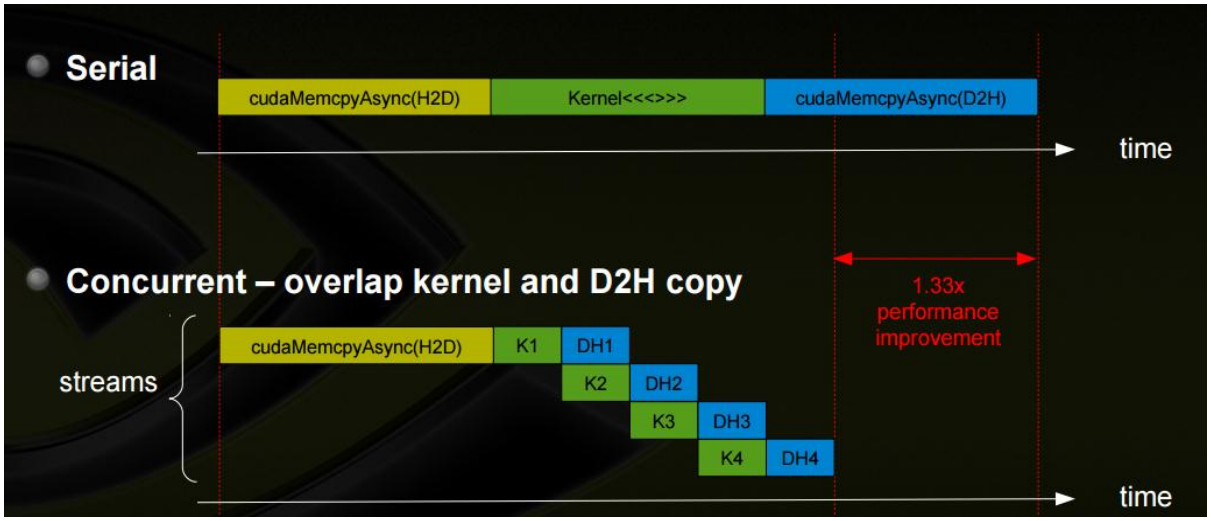

Figure 3.12 Streams and Concurrently (GPU Technology Conference)

The traditional parallel programming in CUDA is just copy the data from hard disk to the GPU. Then, execute kernel in parallel. After that, copy the result back from GPU to hard disk. This method wastes a lot of resource waiting for the slow memory copy to be done.

Another way to do it is by running the kernel and memory copy concurrently. From the figure above, the kernel had split into 4 streams: k1, k2, k3 and k4 and the memory copy from GPU to Hard disk is split into 4 streams: DH1, DH2, DH3 and DH4. By doing this, the kernel execution time can be improve by 1.33 times compare to the traditional way.

In this paper, the kernel is executed concurrently in 3 ways. This can improve the performance of the block cipher under GPU platform even more.

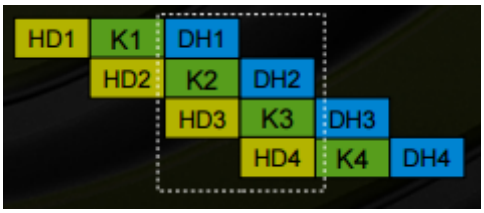

Figure 3.13 Concurrently in 3 ways (GPU Technology Conference)

Inline PTX Assembly

Inline PTX assembly is a parallel thread execution instruction set architecture for using NVIDIA GPU as a data-parallel computing device. This is an assembly language for NVIDIA CUDA GPU. This method can slightly improve the performance of the algorithm but it can only be applied to calculating or computing command but not memory related command.

Operation Mode of Block Cipher

Electronic Codebook Mode

Electronic codebook mode is the simplest operation mode for block cipher. The encryption algorithm work by passing plaintext as an input to the block cipher encryption algorithm with a key. Then, the result is the cipher text. The decryption algorithm work by passing cipher text as an input to the block cipher decryption algorithm with a key and the result is the plaintext. This mode can be applied with parallelism easily but the encryption and decryption algorithm is 2 different algorithms. In order to save time in this paper, this mode is rejected.

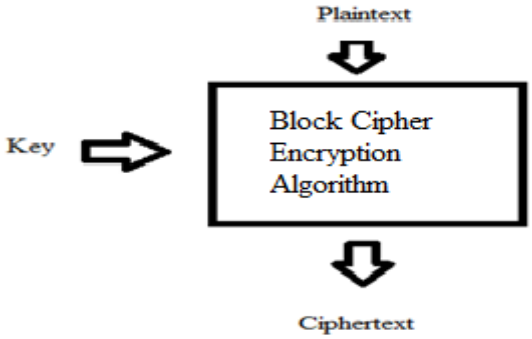

Figure 3.14 Electronic Codebook Mode Encryption

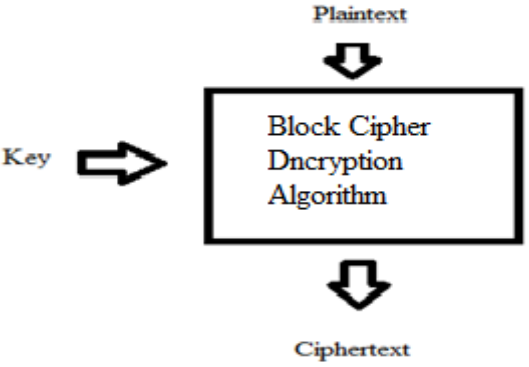

Figure 3.15 Electronic Codebook Mode Decryption

Cipher-block Chaining Mode

Plaintext will be the input of the encryption algorithm in this mode. For the first encryption in this mode, plaintext will be XOR with the initialization vector. Then, the result is pass to the block cipher encryption algorithm with a key. The result of this will be the cipher text. Next, the cipher text will be the input for next round. For the decryption algorithm in this mode, cipher text will be the input. Cipher text is passing as an input to the block cipher decryption algorithm with a key. The result is XOR with an initialization vector to get the plaintext. The following round will repeat same step will the cipher text from the previous round as an input. This mode is rejected because parallelism cannot be applied to the encryption algorithm of this mode.

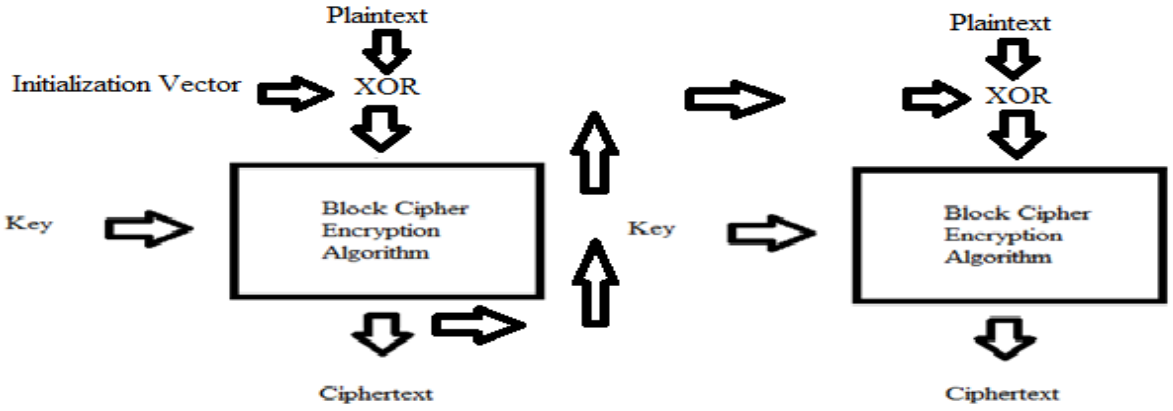

Figure 3.16 Cipher-block Chaining Mode Encryption

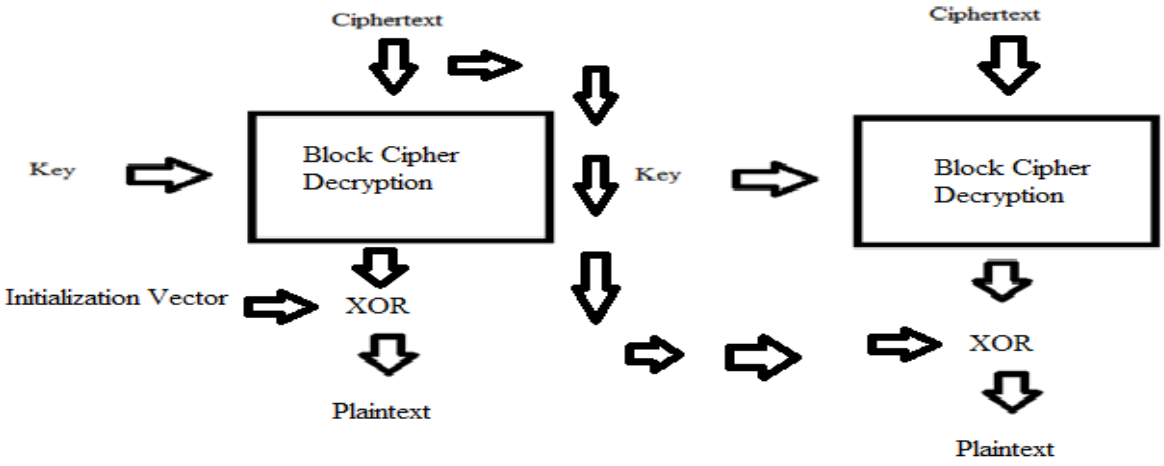

Figure 3.17 Cipher-block Chaining Mode Decryption

Cipher Feedback Mode

Cipher feedback mode of block cipher's encryption work by passing initialization vector to the block cipher encryption algorithm with a key as an input for the first round of encryption algorithm. Then, the result is XOR with the plaintext to get the cipher text. The following round of encryption will use the cipher text from the previous round as an input. The decryption algorithm of this mode is almost the same as the encryption algorithm. The only difference of the encryption algorithm with the decryption algorithm is the position of the plaintext swap with the cipher text. This mode is also rejected because parallelism cannot be applied to the encryption of this mode.

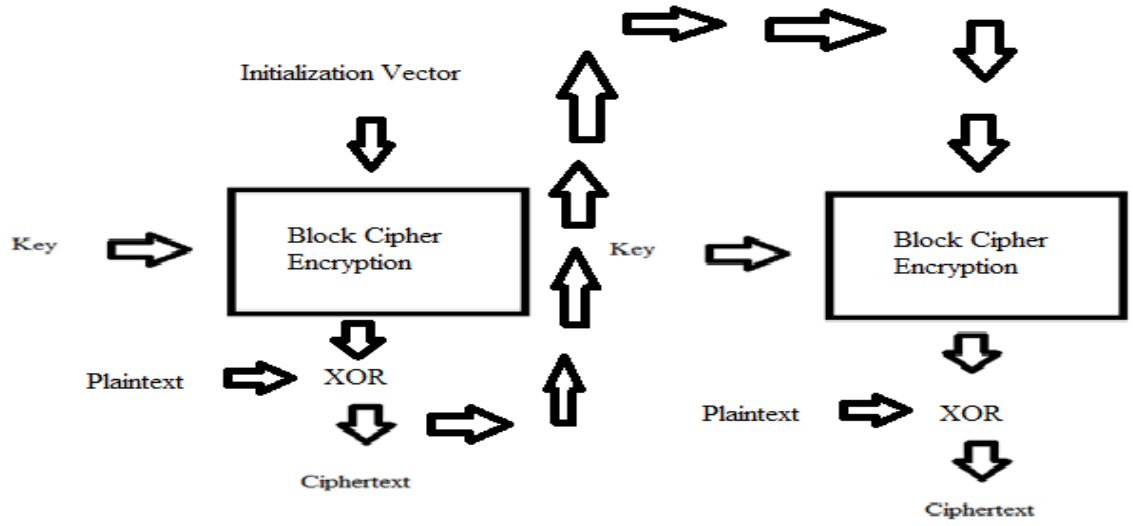

Figure 3.18 Cipher Feedback Mode Encryption

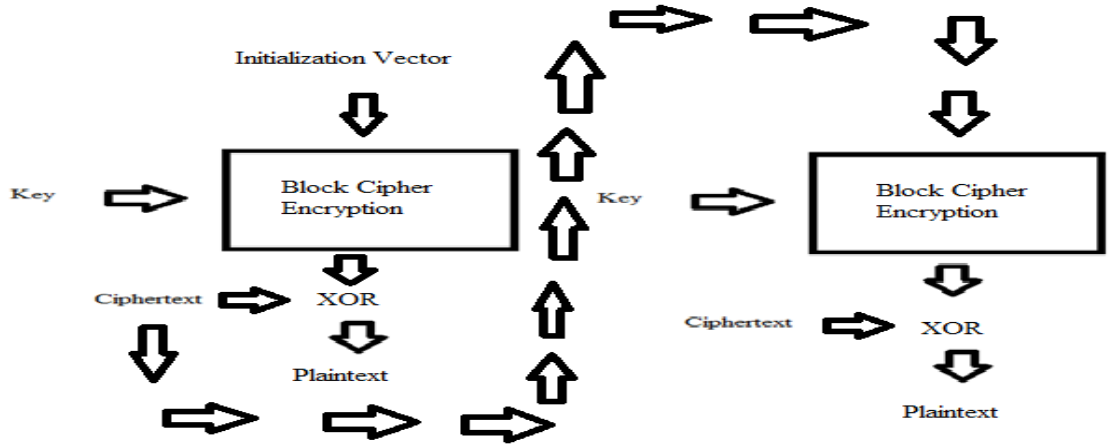

Figure 3.19 Cipher Feedback Mode Decryption

Output Feedback Mode

This mode is almost similar with the cipher feedback mode. The only different of this mode with the previous mode is that the second round onwards for both encryption and decryption in this mode will use the result from the encryption before XOR as an input. This mode is still rejected because both encryption and decryption of this mode cannot be applied with parallelism.

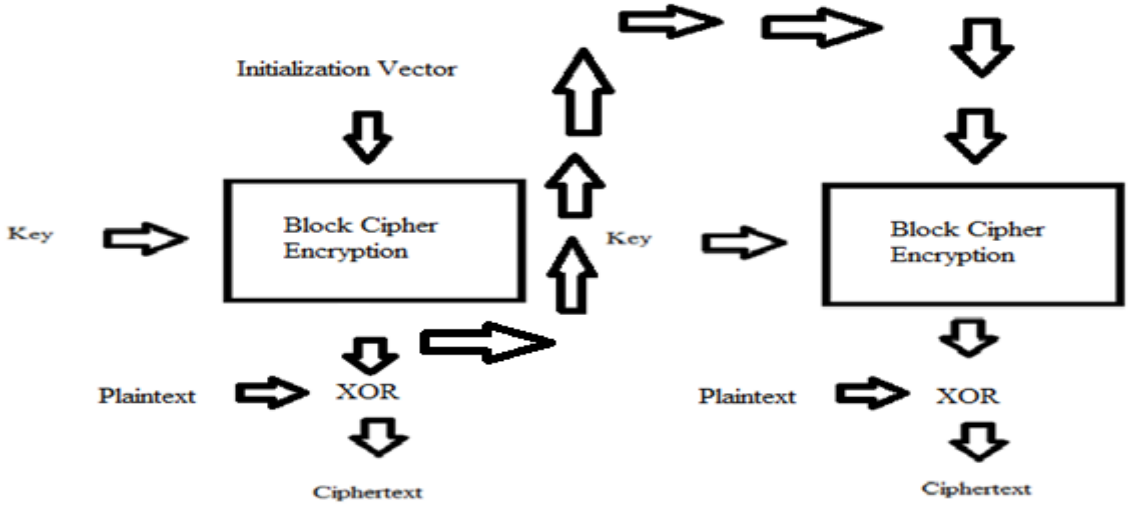

Figure 3.20 Output Feedback Mode Encryption

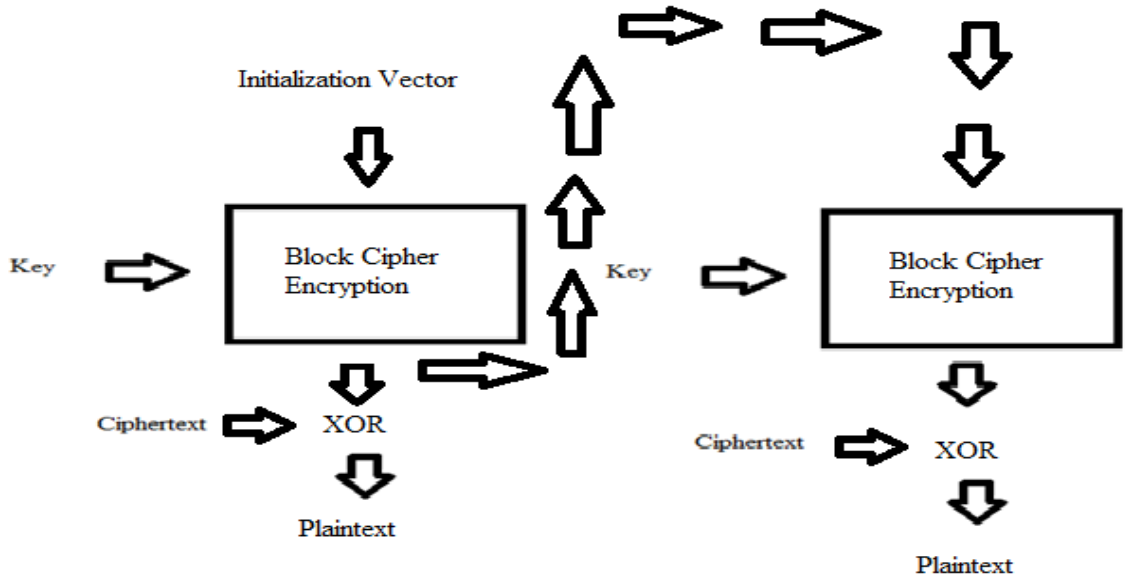

Figure 3.21 Output Feedback Mode Decryption

Counter Mode

Counter mode is selected in this paper because this mode is one of the modes that parallelism can be applied. Other than that, both encryption and decryption of this mode uses the same algorithm, which is the block cipher encryption algorithm. So, this can save time in the implementation of the block cipher algorithm. Since parallelism can be applied to counter mode, therefore, the whole counter mode operation can be implementing in parallel programming form. The parallel programming can increase the speed of the encryption according to the number of cores in the processor theoretically. That means instead of running the block cipher algorithm block by block, this project will run the block cipher algorithm with multiple input data block at the same time as shown in figure below and the performance speed should increase 4 times with 4-cores processor theoretically.

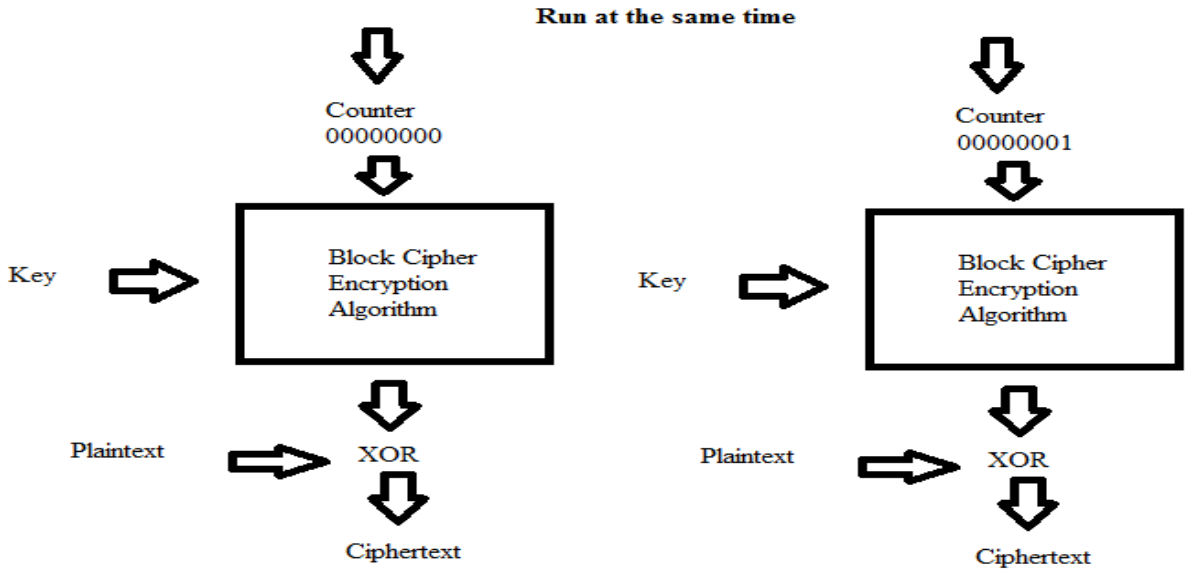

Figure 3.22 Run encryption algorithm at the same time

This parallel programming is done with OpenMP and CUDA. This is to test the performance of the block cipher algorithm with different platform. Using OpenMP to code this block cipher algorithm is to test the performance of the block cipher algorithm with multi-cores processor while using CUDA to code is to test the performance of the block cipher algorithm in GPU environment which contain more cores than multi-cores processor.

For parallel programming that is done with OpenMP, the number of thread that is spawn is 64 threads. Meanwhile, thousands of threads are spawning for the parallel programming that is done with CUDA.

Pseudo Random Number

Pseudo random number is not truly random. It is a sequence of number that is approximates the properties of random number. Pseudo random number is not truly random because the generation of this pseudo random number is because the generation of pseudo random number is depends completely on a set of initial value.

Pseudo random numbers are mainly needed in electronic game and cryptography. In electronic game, pseudo random number is used to ensure that the randomness of the game so that player cannot predict what will happen in the game. In cryptography, pseudo random number is used to ensure that the output of the encryption will be unpredicted.

Pseudo Random Number Generation

Pseudo random number is generated with block cipher under counter mode in this paper. The whole process of generating pseudo random number with block cipher is the same process as normal block cipher encryption under counter mode. The only difference is that the output of the encryption will not XOR will the original message instead it will straight away being used as pseudo random number.

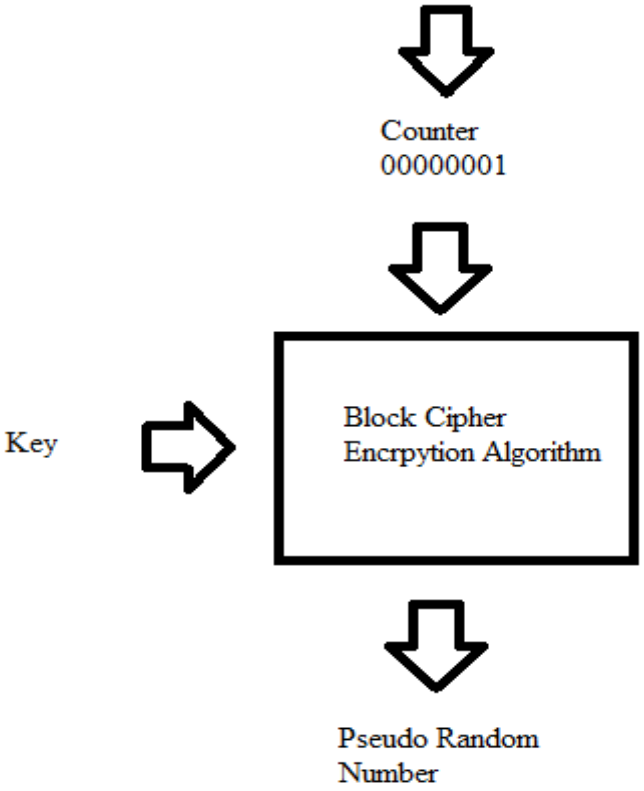

Figure 3.23 Pseudo Random Number Generation

Chapter 4 System Design

4.1 IDEA / Blowfish / Threefish Algorithm in Counter Mode under CPU Platform

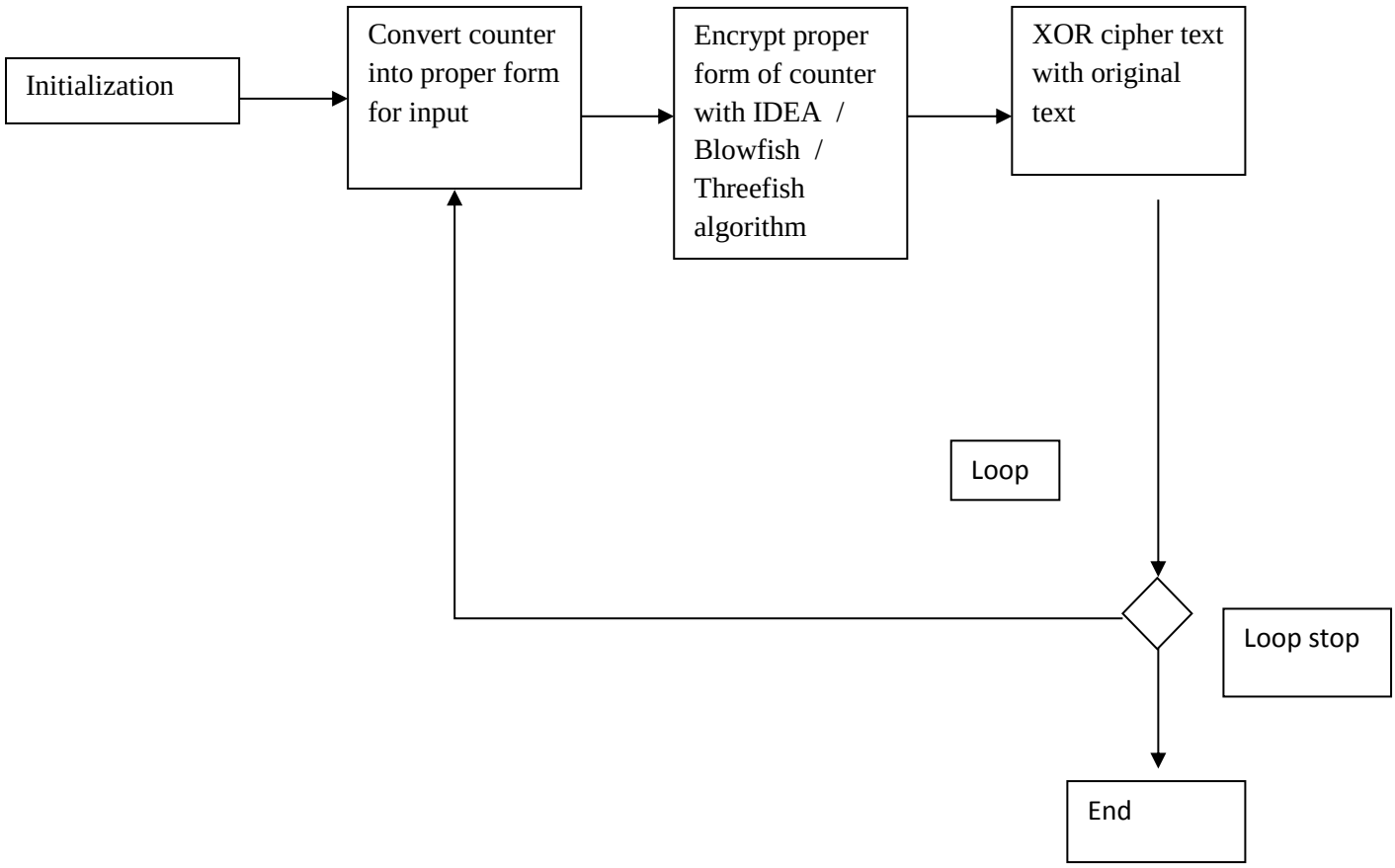

Figure 4.1 IDEA / Blowfish / Threefish Algorithm in counter mode under CPU platform

First, the key for the whole encryption process must be schedule during the initialization process. The same key will be used in the whole process. Second, convert counter to a proper form of array as an input to the IDEA, Blowfish or Threefish algorithm. The counter that is converted to array is then encrypting with IDEA, Blowfish or Threefish algorithm to get a cipher text. After that, the cipher text is then XOR with the original text to get the output for IDEA, Blowfish or Threefish in counter mode. The process will keep on looping until the loop is stop.

4.2 IDEA / Blowfish / Threefish Algorithm in Counter Mode under Multiple Cores Processor Platform

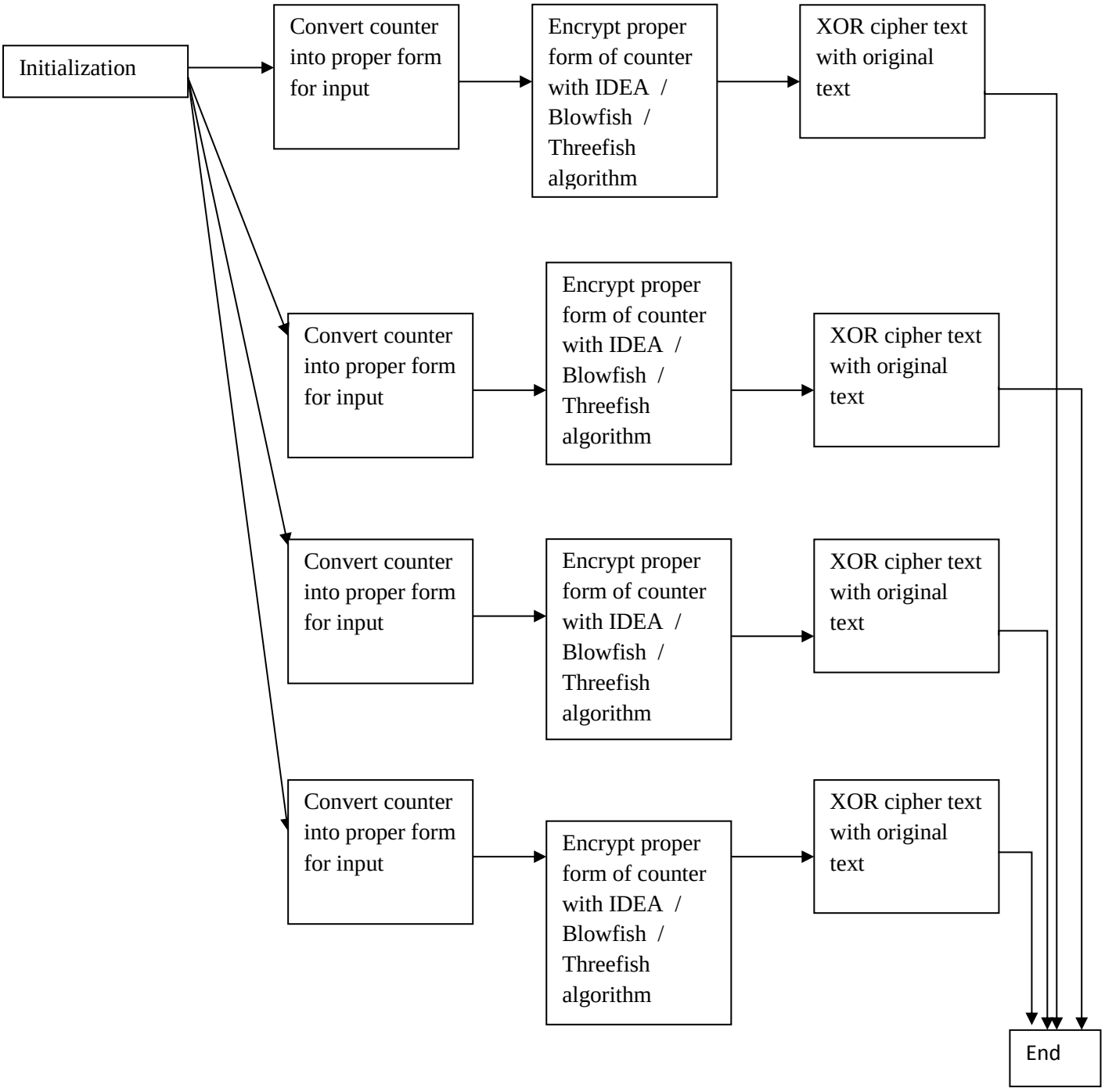

Figure 4.2 IDEA / Blowfish / Threefish Algorithm in counter mode under multiple cores processor platform

First, the key for the whole encryption process must be schedule during the initialization process. The same key will be used in the whole process. Then, convert counter to a proper form of array as an input to the IDEA, Blowfish or Threefish algorithm. The counter that is converted to array is then encrypt with IDEA, Blowfish or Threefish algorithm to get a cipher text. After that, the cipher text is then XOR with the original text to get the output for IDEA, Blowfish or Threefish in counter mode. The process starting from converting the counter to proper array until XOR with cipher text is the process that will be executed simultaneously in multiple core processor. After the process is executed, the result is then collected from the multiple core processor in right order.

4.3 IDEA / Blowfish / Threefish Algorithm in Counter Mode under GPU Platform

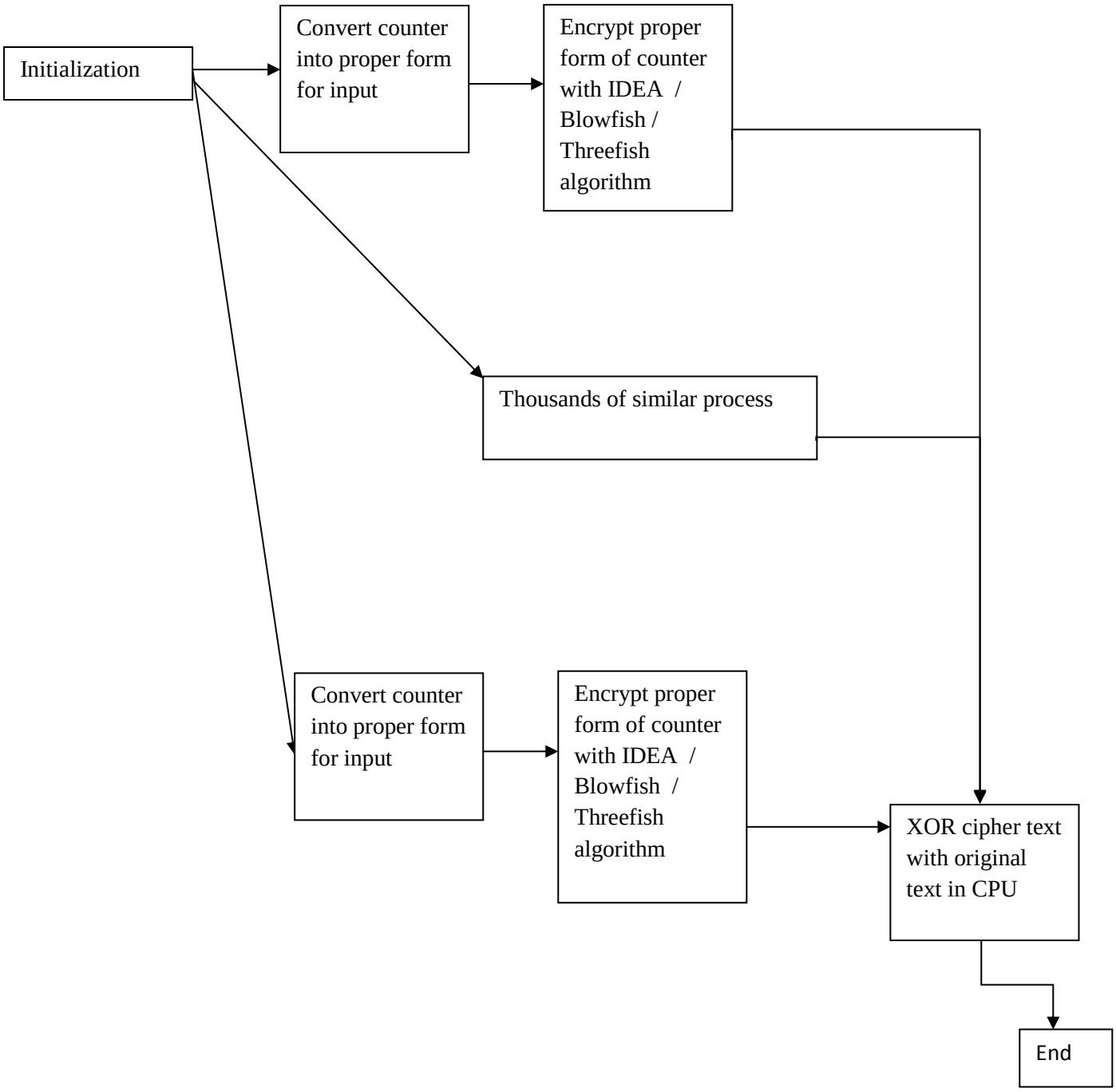

Figure 4.3 IDEA / Blowfish / Threefish Algorithm in counter mode under GPU platform

First, the key for the whole encryption process must be schedule during the initialization process. The same key will be used in the whole process. Then, convert counter to a proper form of array as an input to the IDEA, Blowfish or Threefish algorithm. The counter that is converted to array is then encrypting with IDEA, Blowfish or Threefish algorithm to get a cipher text. After that, the cipher text is then XOR with the original text to get the output for IDEA, Blowfish or Threefish in counter mode. The process starting from converting the counter to proper array until XOR with cipher text is the process that will be executed simultaneously in GPU. After the process is executed, the result is then collected from the GPU in right order. The only difference of the execution of the algorithm in multiple core platform and GPU platform is that the GPU will allow a larger amount of thread executed at the same time if compare to multiple core platform.

4.4 Pseudo Random Number Generation with IDEA / Blowfish Algorithm under CUDA platform

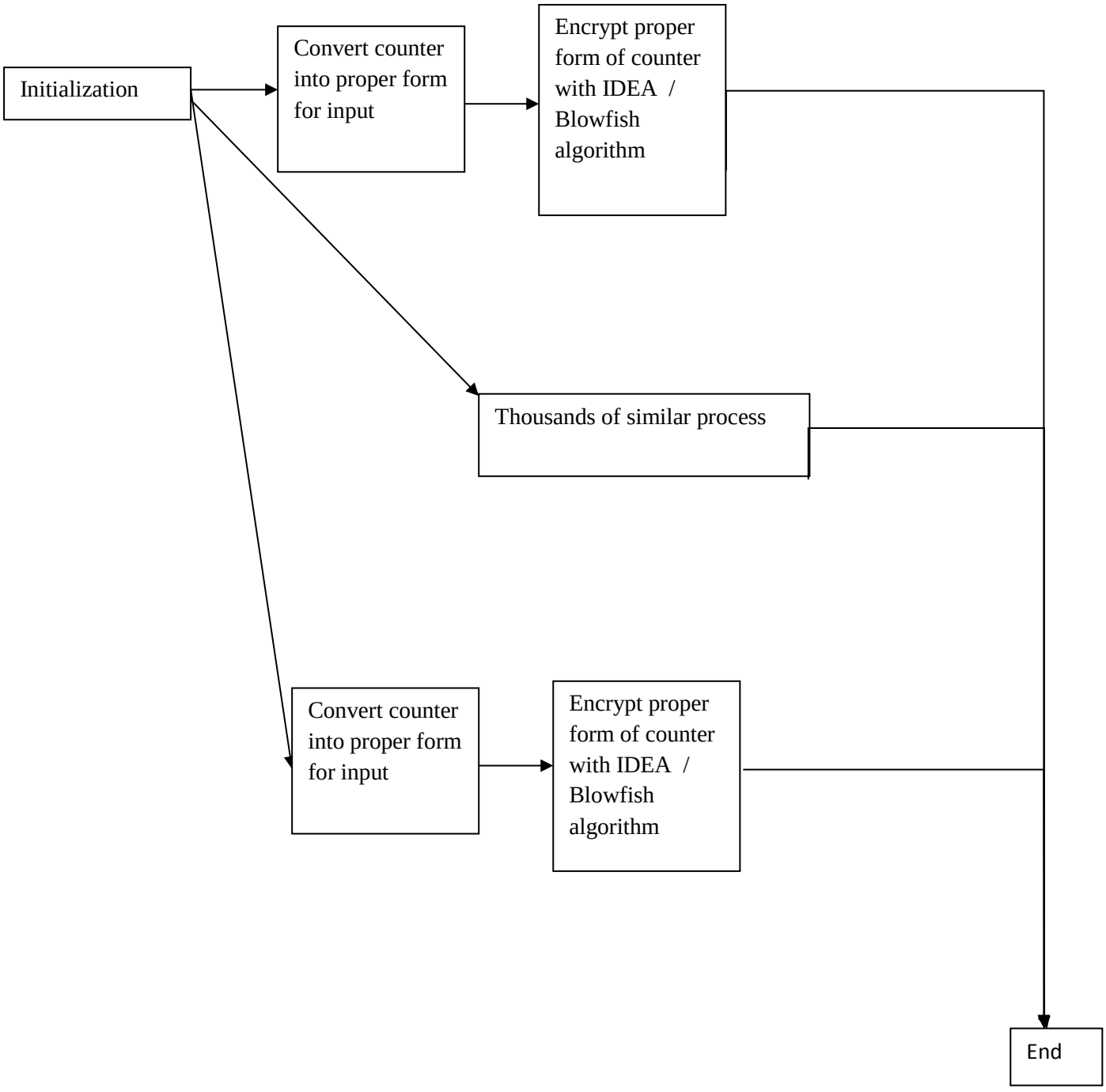

Figure 4.4 Pseudo Random Number Generation with IDEA / Blowfish Algorithm under CUDA platform

First, the key for the whole encryption process must be schedule during the initialization process. The same key will be used in the whole process. Then, convert counter to a proper form of array as an input to the IDEA or Blowfish algorithm. The counter that is converted to array is then encrypting with IDEA or Blowfish algorithm to get a cipher text. After the encryption is executed, the cipher text generated can be use as pseudo random number.

The pseudo random number generated with reduced round of IDEA or Blowfish algorithm also been carried out with the same design like the full round. The only difference is that the reduced round had lesser round compare to full round.

4.5 Pseudo Random Number Generation with modified IDEA algorithm under GPU platform

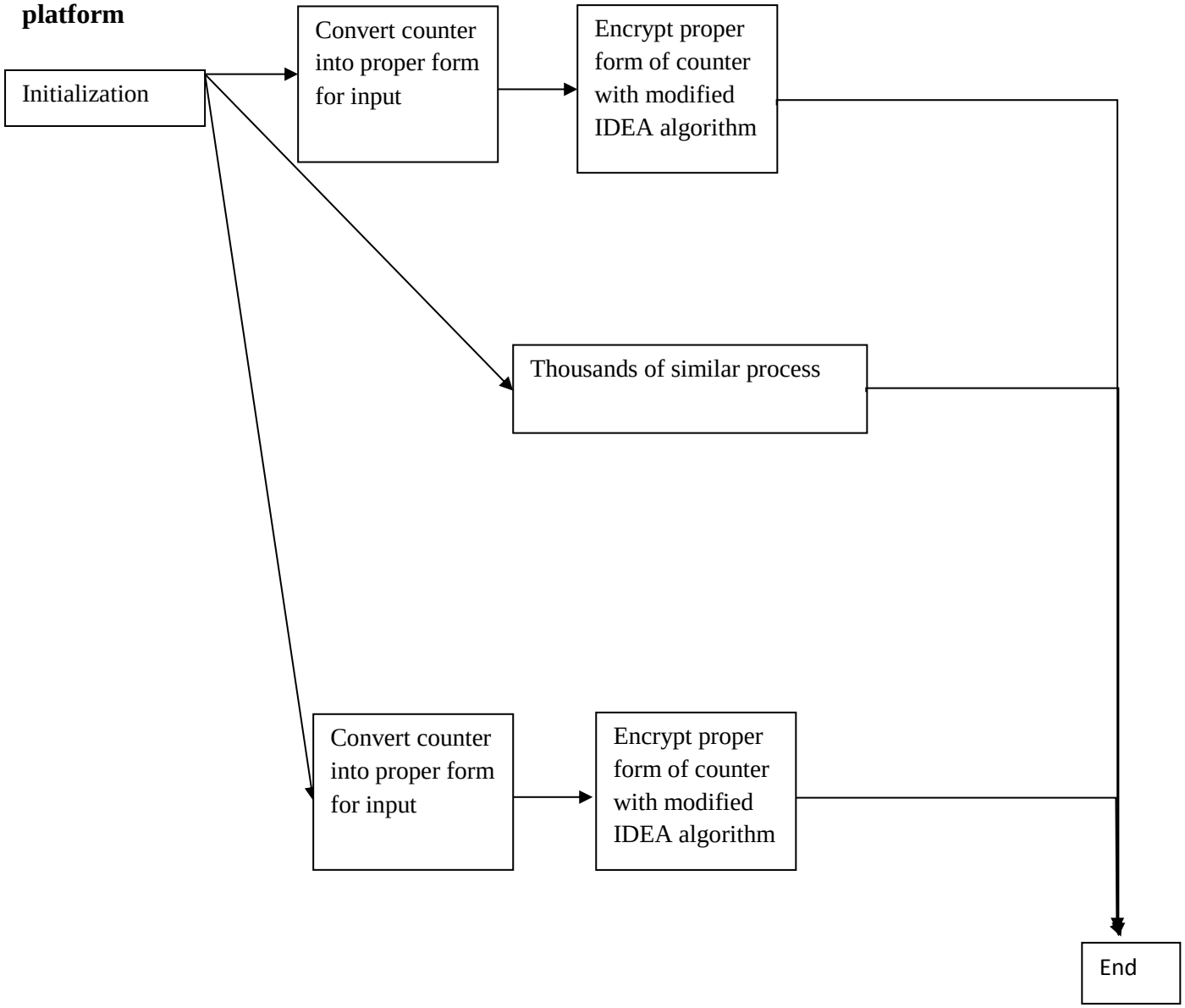

Figure 4.5 Pseudo Random Number Generation with modified IDEA algorithm under GPU platform

First, the key for the whole encryption process must be schedule during the initialization process. The same key will be used in the whole process. Then, convert counter to a proper form of array as an input to the modified IDEA algorithm. The counter that is converted to array is then encrypting with modified IDEA algorithm to get a cipher text. After the encryption is executed, the cipher text generated can be use as pseudo random number.

The modified IDEA algorithm is the same as the original IDEA algorithm except that all the unnecessary code are remove from the original IDEA algorithm to optimize the speed of generating pseudo random number.

Chapter 5 Implementation and Optimization

5.1 Implementing Block Cipher algorithm under normal CPU platform

First, ensure that the block cipher algorithm can be executed under normal cpu platform. Check the result generated from the block cipher algorithm with the test vector available online.

Then, execute block cipher algorithm with counter mode. First, all the data will be initialize with the proper size. After that generating the counter from for loop, insert the counter as the input for the block cipher algorithm. Finally, XOR original message with the cipher text generated out from the block cipher algorithm. The time taken for the block cipher algorithm to complete the execution is recorded without include time taken to XOR original message with cipher text.

The implementation of block cipher algorithm in normal CPU platform must be correct because this code will be used to verify result generated under OpenMP platform and GPU platform. Therefore, this code will be tested .Only after the results are correct for certain, then the progress can be move on. The code is tested will another same block cipher algorithm but with different structure.

5.2 Implementing Block Cipher algorithm under OpenMP platform

The variable will be first initialized so that it can be used. Then, counter will be generated through for loop and then counter is passed as input to the block cipher algorithm to generate a cipher text. All of this will be executed with 4 threads running in parallel.

Finally, the original message will be XOR with the cipher text. The time taken for the block cipher algorithm to complete the execution is recorded without include time taken to XOR original message with cipher text.

The result generated will be tested using the same block cipher algorithm under normal CPU platform. I will only progress to GPU platform only after the results are correct.

5.3 Implementing and Optimizing Block Cipher algorithm under GPU platform

The data required for the block cipher algorithm are initializing at the beginning. Then, the kernel is executed. Inside the kernel, the counter is generated by using the thread id and block id with the following code.

Counter = threadIdx.x + blockDim.x * blockIdx.x;

Then, the block cipher algorithm is used to encrypt the counter and copy back the result to the CPU. After that, the result is XOR with the original text to obtain the cipher text.

In CUDA, the part that will slow down the performance is the part that copy memory either from hard disk to the GPU or vice versa. This will cased the kernel to wait for the slow memory copy. Therefore, multiple streams and asynchronous memory copy are used in this paper to optimize the performance of the block cipher algorithm.

There are different types of memory available in CUDA. Shared memory are used in this paper to speed up the performance of the block cipher algorithm because shared memory are fast and the size of shared memory available are large enough for the algorithm.

Even with shared memory, the performance of the block cipher is still not fast enough. Therefore, warp shuffle are implemented to improve the performance. Warp shuffle allow threads within the same warp to read each other's register and register is much faster than shared memory.

The performance of the block cipher algorithm can be improved even more by applying inline PTX assembly into the algorithm. Inline PTX assembly is a parallel thread execution instruction set architecture for using NVIDIA GPU as a data-parallel computing device. This can only be apply to some of the calculation in block cipher algorithm and can only slightly improve the performance of the block cipher.

The time taken for the block cipher algorithm to complete the execution is recorded without include time taken to XOR original message with cipher text.

The result generated will be tested using the same block cipher algorithm under normal CPU platform.

5.4 Implementing Pseudo Random Number Generator with Block Cipher Algorithm in Counter mode under GPU platform

The pseudo random number can be generated using block cipher in counter mode. This is exactly the same as normal block cipher encryption in counter mode but the only difference is that the cipher text generated from block cipher algorithm will not be XOR with the original message and that result contains statistical properties.

After generating the pseudo random number, the random number will be tested with 2 statistical tests: TestU01 and NIST test suite. Only if the pseudo random number pass the tests, then the randomness of the pseudo random number will be proved. TestU01 is hard for pseudo random number to pass the entire test and many known random number generator cannot pass all the test provided in TestU01.

Chapter 6 Performance and Analysis

Table 6-1 Hardware Specification

Hardware	**Specification**
Intel (R) Core (TM) i5-3330 CPU (4 CPUs)	3.00GHz
RAM	8.0 GB
NVIDIA GeForce GTX 690	
Software	**Specification**
Windows 7	64-bits
Linux	64-bits
Visual Studio 2010	64-bits
CUDA SDK 6.0	
TestU01	
NIST Test Suite	

Experiment Result

The experiment is carried out with different data size under different platform. The size of the input data is 64KB, 128KB, 256KB, 512KB, 1MB, 2MB, 4MB, 8MB, 16MB, 32MB and 64MB. The platform involve is normal CPU platform, multiple core platform and GPU platform. The execution time recorded is only the time taken for the algorithm to complete. The time recorded did not include time taken for initialization and other process. The result expected to be faster from normal CPU platform to GPU platform. This is because the algorithm is executed in parallelism but more optimization should be done on the code to improve the performance speed.

Data Initialization is allocating memory to the variable, copy the memory from hard disk to GPU and schedule the key for encryption.

Table 6-2: Performance in Normal CPU Platform

Time taken / Data Size	64KB	128KB	256KB	512KB	1MB	2MB
Blowfish (s)	0.00074	0.00142	0.00287	0.00567	0.01132	0.02232
Blowfish with Data Initialization (s)	0.01193	0.02293	0.03415	0.04696	0.09694	0.18543
IDEA (s)	0.00147	0.00292	0.00578	0.01213	0.02503	0.04735
IDEA with Data Initialization (s)	0.00279	0.00564	0.01110	0.02029	0.03947	0.04698
Threefish (s)	0.00087	0.00179	0.00356	0.00712	0.01446	0.02837
Threefish with Data Initialization (s)	0.00237	0.00516	0.00892	0.01805	0.03557	0.07190

Time taken / Data Size	4MB	8MB	16MB	32MB	64MB
Blowfish (s)	0.04473	0.08939	0.17833	0.35744	0.76618
Blowfish with Data Initialization (s)	0.37267	0.73388	1.47459	2.94522	5.89900
IDEA (s)	0.10319	0.19890	0.38420	0.74654	1.49831
IDEA with Data Initialization (s)	0.10651	0.19981	0.39689	0.76486	1.50639
Threefish (s)	0.05701	0.11215	0.22385	0.45048	0.89127
Threefish with Data Initialization (s)	0.14310	0.28152	0.56569	1.13738	2.27577

Table 6-3: Performance in Multiple Core Platform

Time taken / Data Size	64KB	128KB	256KB	512KB	1MB	2MB
Blowfish (s)	0.00393	0.00260	0.00675	0.00889	0.01495	0.03258
Blowfish with Data Initialization (s)	0.01219	0.02509	0.03796	0.05202	0.10380	0.20765
IDEA (s)	0.00233	0.00443	0.00758	0.00976	0.01830	0.02779
IDEA with Data Initialization (s)	0.00390	0.00467	0.00826	0.01030	0.01895	0.02819
Threefish (s)	0.00147	0.00204	0.00283	0.00483	0.00517	0.00911
Threefish with Data Initialization (s)	0.00312	0.00514	0.00848	0.01454	0.03135	0.05273

Time taken / Data Size	4MB	8MB	16MB	32MB	64MB
Blowfish (s)	0.10914	0.12657	0.16902	0.39439	0.77708
Blowfish with Data Initialization (s)	0.36840	0.81207	1.60170	3.07840	6.52363
IDEA (s)	0.04326	0.07373	0.16162	0.30055	0.53594
IDEA with Data Initialization (s)	0.05098	0.07797	0.19203	0.34889	0.57452
Threefish (s)	0.02543	0.03173	0.05859	0.13309	0.30548
Threefish with Data Initialization (s)	0.10504	0.22632	0.40158	0.84121	1.65003

Table 6-4: Performance in GPU platform

Time taken / Data Size	64KB	128KB	256KB	512KB	1MB	2MB
Blowfish (ms)	0.04138	0.05770	0.06704	0.10733	0.18864	0.35226
Blowfish with Data Initialization (ms)	4.20502	4.61178	5.04691	4.56170	5.15859	5.17907
IDEA (ms)	0.04134	0.04922	0.05754	0.08189	0.13917	0.24595
IDEA with Data Initialization (ms)	3.82752	4.02342	4.22970	4.00205	4.35930	4.78067
Threefish (ms)	0.03542 (256 threads)	0.04128 (512 threads)	0.05786	0.06208	0.11146	0.21302
Threefish with Data Initialization (ms)	1.24541 (256 threads)	3.01037 (512 threads)	2.00646	2.87773	2.90141	4.7305

Time taken / Data Size	4MB	8MB	16MB	32MB	64MB
Blowfish (ms)	0.67296	1.32470	2.62112	5.50224 5.25427	10.66723
Blowfish with Data Initialization (ms)	7.13763	8.44810	12.16803	18.09661	28.13652
IDEA (ms)	0.46810	0.89674	1.76518	3.54653	7.33488
IDEA with Data Initialization (ms)	5.08893	6.14122	9.34698	15.66730	24.68416
Threefish (ms)	0.41377	0.80051	1.56794	3.05834	6.04851
Threefish with Data Initialization (ms)	5.50688	8.26637	11.59987	17.91654	71.96749

Experiment Analysis

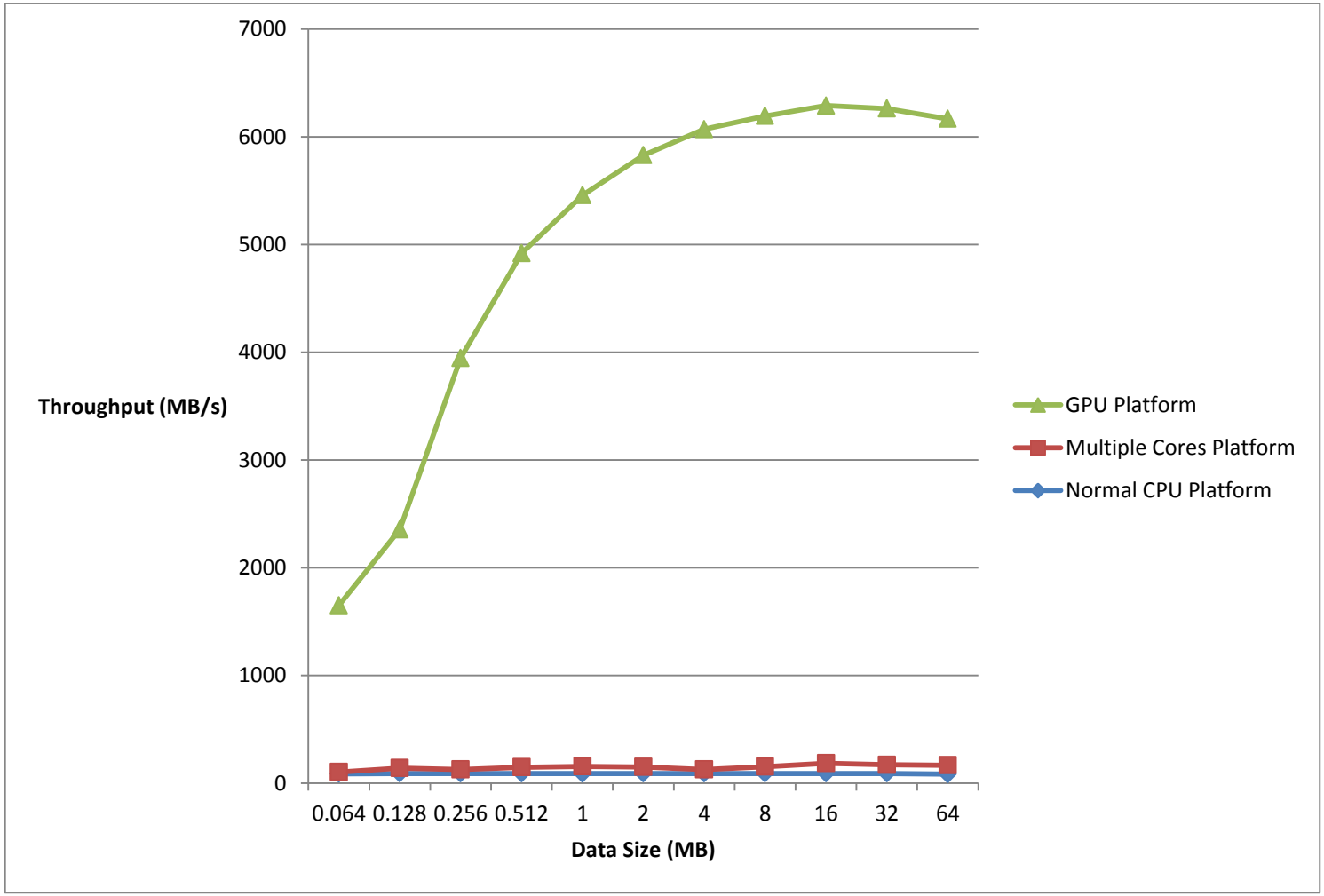

Figure 6.1 Blowfish Throughput

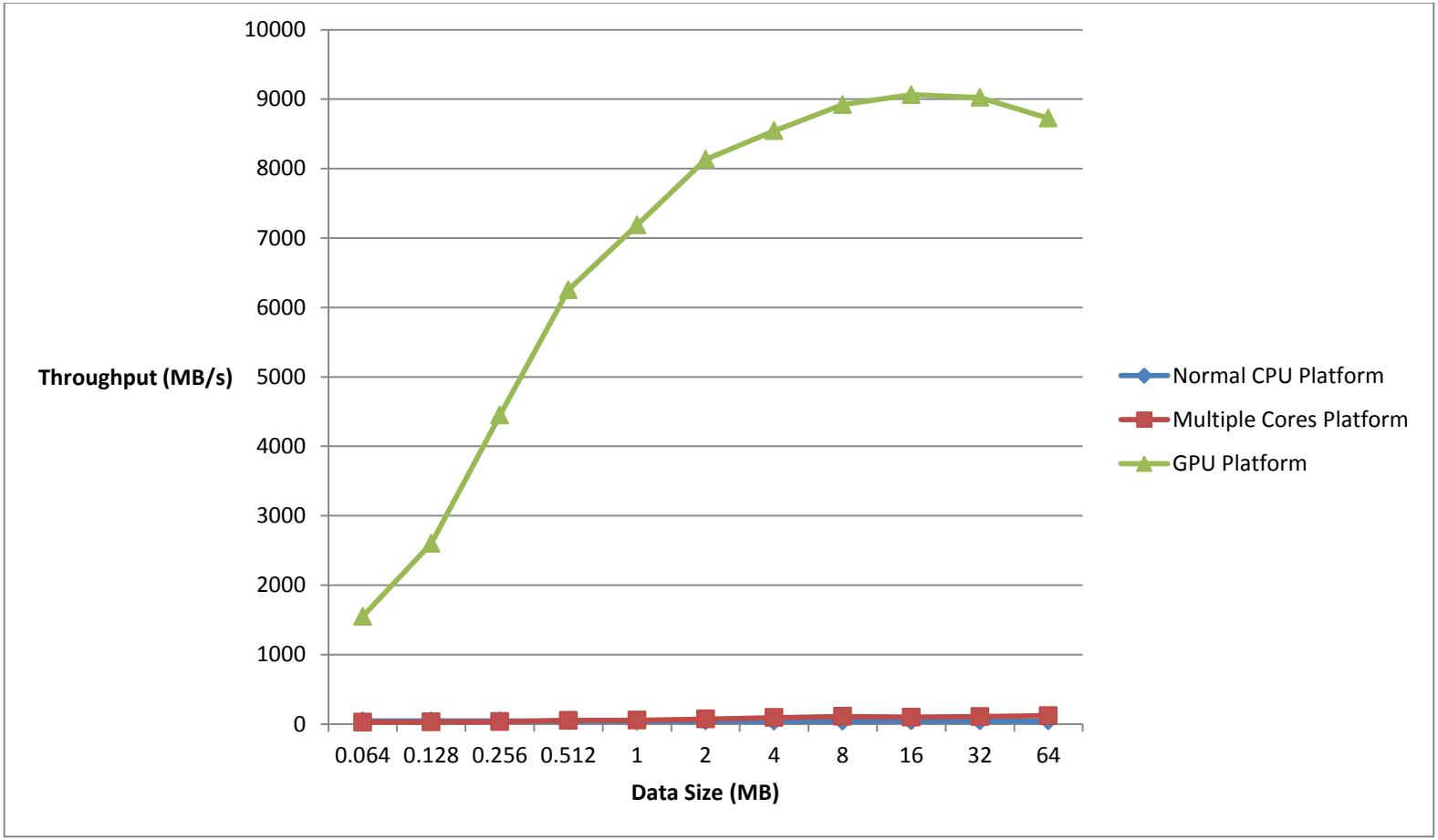

Figure 6.2 IDEA Throughput

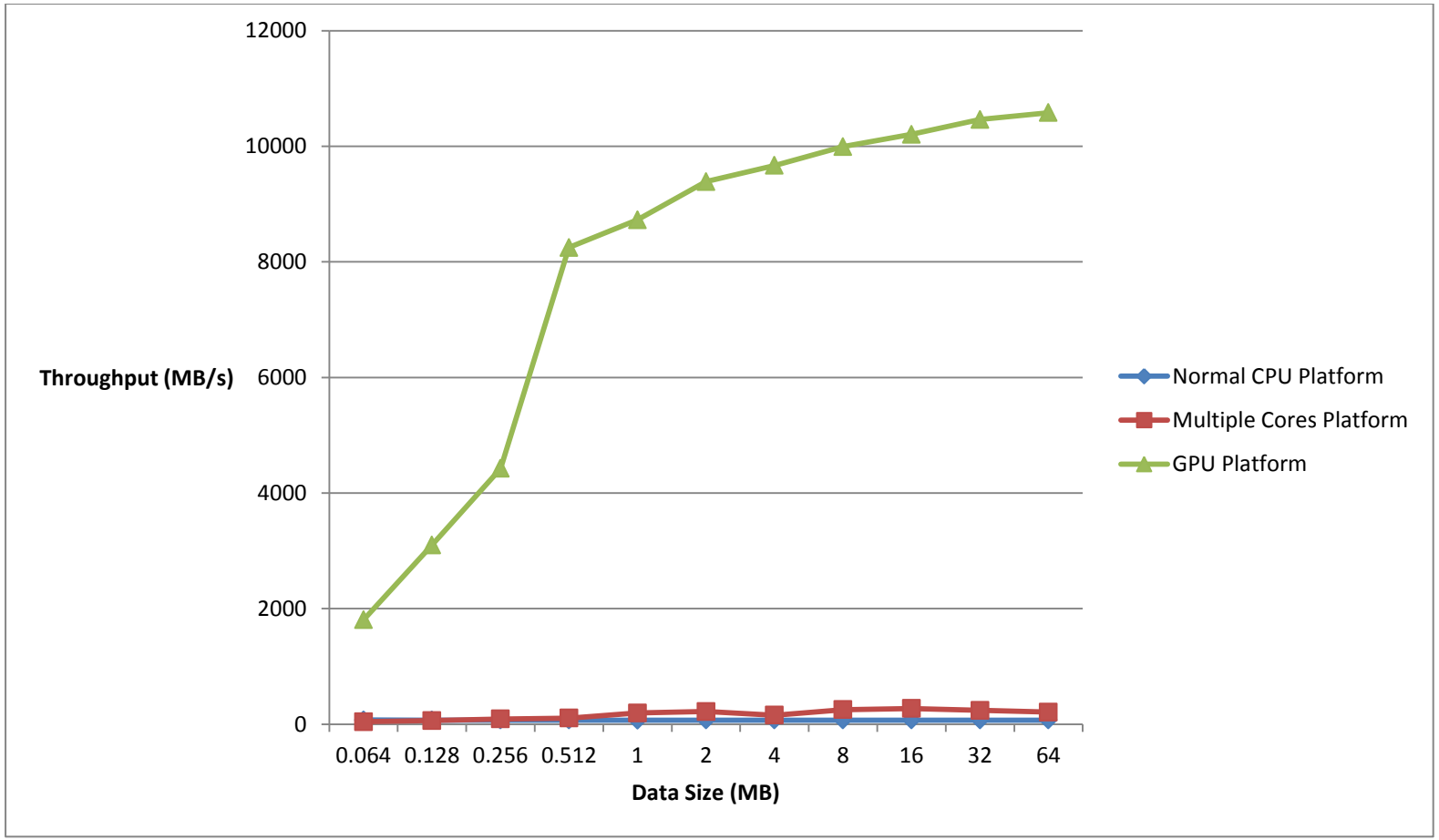

Figure 6.3 Threefish Throughput

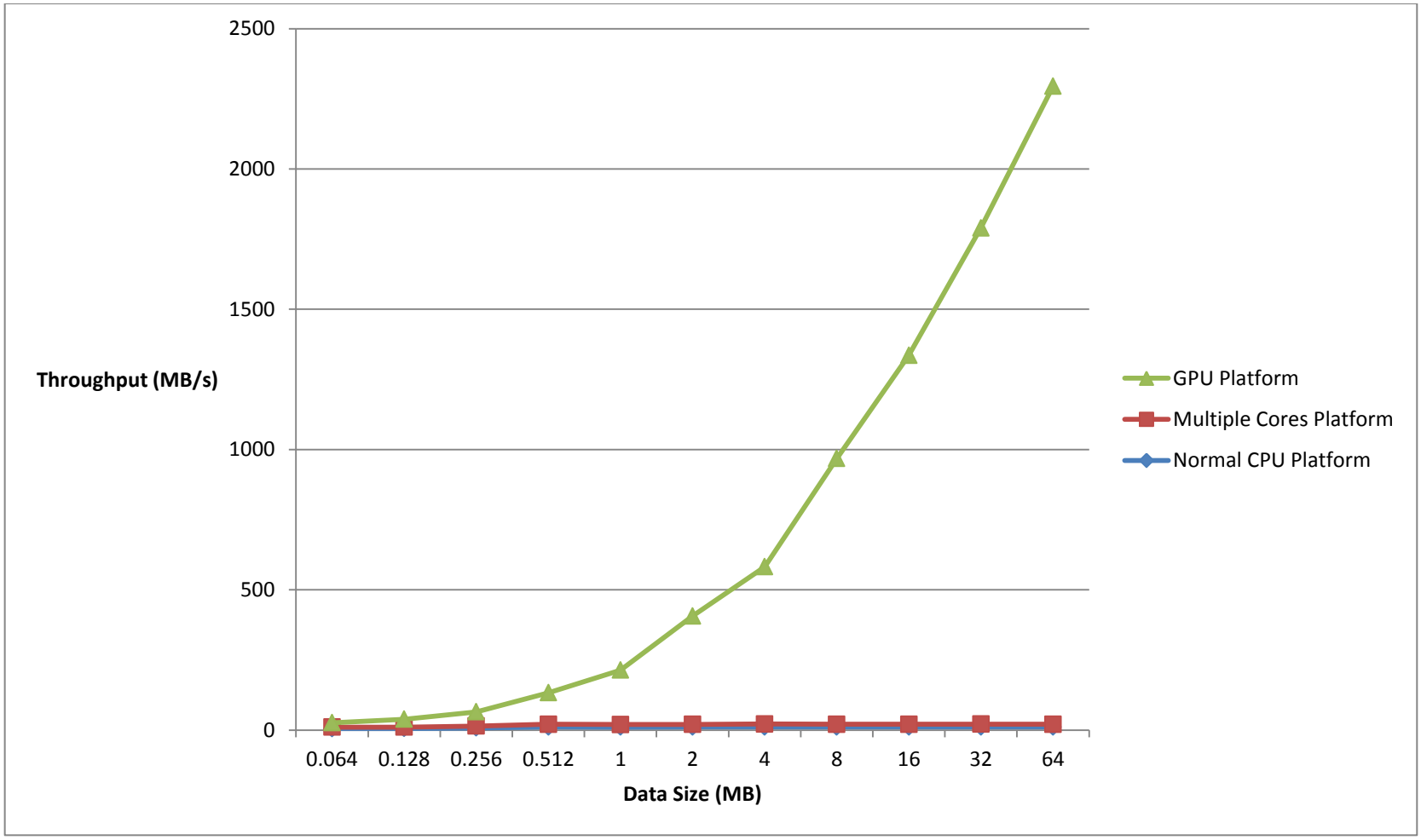

Figure 6.4 Blowfish Throughput with Data Initialization

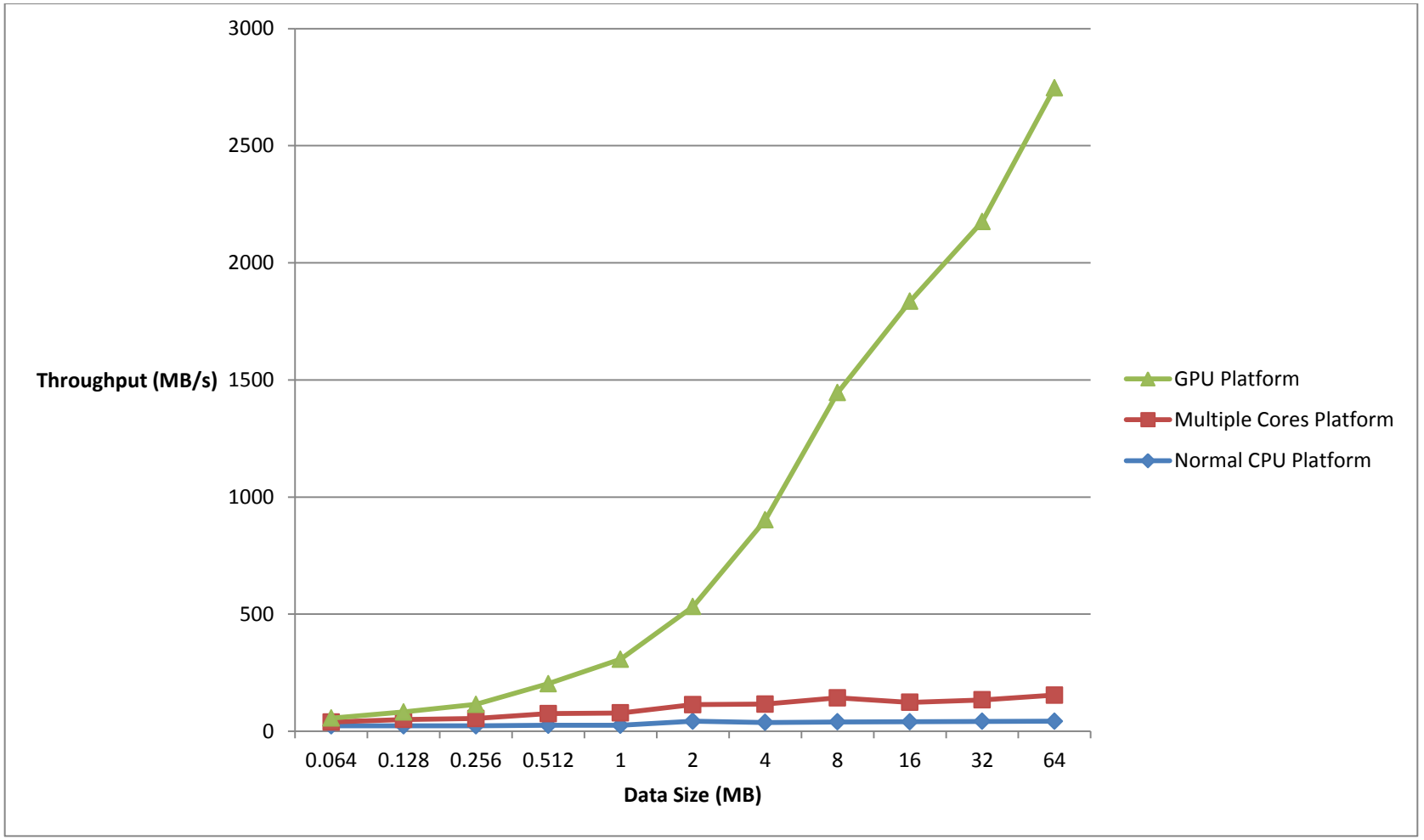

Figure 6.5 IDEA Throughput with Data Initialization

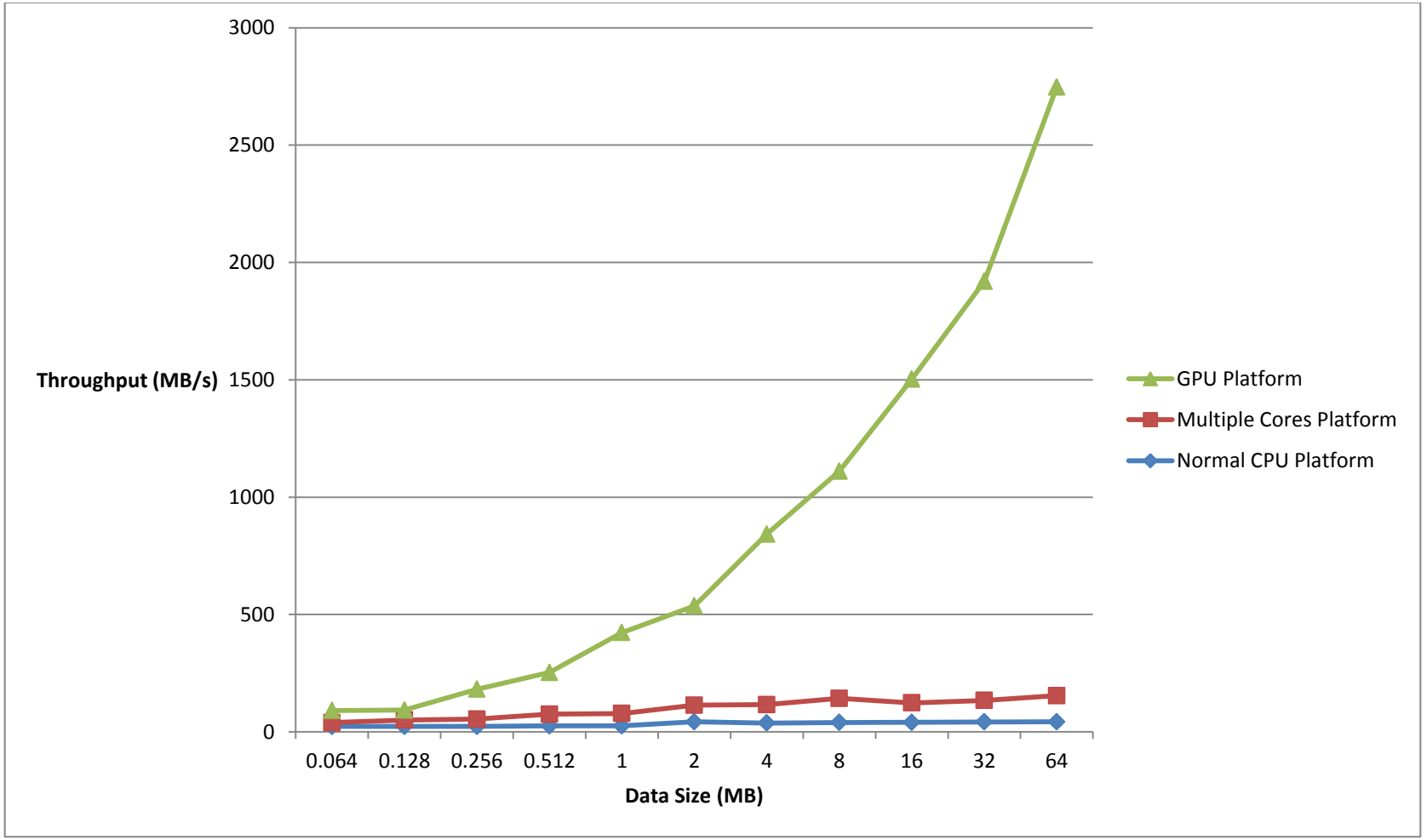

Figure 6.6 Threefish Throughput with Data Initialization

From the figure 6.1, figure 6.2 and figure 6.3, the graph shown that the throughput of GPU platform is the highest throughput among the 3 platform experimented. The second highest throughput is the multiple cores platform and the lowest throughput is the normal CPU platform. For both Blowfish and Threefish algorithm, the throughput of the normal CPU platform and multiple cores platform showed a stable increase from 64KB data size until 64MB. Meanwhile, the throughput of the GPU platform shown a dramatically increase at the small input data size. Then, the throughput will increase stably until 64 MB. For Blowfish algorithm, there is no increase in the throughput under OpenMP platform but there is a huge increase under GPU platform because Blwofish algorithm is a table lookup based block cipher. Table lookup is a very expensive operation in OpenMP and cause the entire algorithm to become very slow.

Figure 6.4, figure 6.5 and figure 6.6 showed the graph of Blowfish, IDEA and Threedish with data initialization. These 3 graphs shown difference compare to figure 6.1, figure 6.2 and figure 6.3. The throughput of the entire research platform showed a significant decrease. This decrease is especially obvious on GPU platform and Blowfish algorithm in multiple cores platform. The huge decrease in throughput shown in the Blowfish algorithm under multiple cores platform is because the algorithm used 2d array and the initialization of the 2d array are expensive. The dramatic decrease of throughput of all block cipher algorithm under GPU platform is because data initialization in GPU platform is expensive.

Pseudo Random Number Generation with Block Cipher under Counter Mode

Experiment Result

This experiment is carried out with the exact same block cipher algorithm that is experimented under different platform. Therefore, there is no need to collect the throughput of these 3 block cipher algorithms under GPU platform.

To prove that the pseudo random number generated out from the block cipher algorithm is random enough to be applied, statistical test are carried out to test the randomness of the pseudo random number generated. The statistical test carried out is TestU01 and NIST test suite.

In this experiment, both TestU01 and NIST test suite is run in Linux.

Table 6-5:TestU01 Result

Statistical Test	**SmallCrush**	**Crush**	**BigCrush**
Pseudo Random Number Generator			
IDEA (full round)	Passed all test.	Passed all test.	Test no. 35, 36 and 37 failed.
IDEA (6 round)	Passed all test.	Test no. 74 failed.	Test no. 35, 36 and 37 failed.
IDEA (4 round)	Passed all test.	Test no. 2 failed.	Test no. 35, 36 and 37 failed.
IDEA (2 round)	Passed all test.	Failed 7 tests.	Test no. 35, 36 and 37 failed.
Blowfish (full round)	Passed all test.	Passed all test.	Test no. 36 and 37 failed.
Blowfish (Round 0 to Round 11)	Passed all test.	Test no. 73 and 79 failed.	Test no. 35, 36 and 37 failed.
Blowfish (Round 0 to Round 7)	Passed all test.	Test no. 1 and 56failed.	Test no. 36 and 37 failed.
Blowfish (Round 0 to Round 3)	Passed all test.	Passed all test.	Test no. 36, 37 and 74 failed.
Modified IDEA	Passed all test.	Passed all test.	Test no. 21, 34, 35,

			36, 37 and 65 failed.

Table 6-6: NIST Test Suite Result

Statistical Test	**NIST Test Suite**
Pseudo Random Number Generator	
IDEA (full round)	Passed all test.
IDEA (6 round)	Passed all test.
IDEA (4 round)	Passed all test.
IDEA (2 round)	Passed all test.
Blowfish (full round)	Passed all test.
Blowfish (Round 1 to Round 12)	Passed all test.
Blowfish (Round 1 to Round 8)	Passed all test.
Blowfish (Round 1 to Round 4)	Passed all test.
Modified IDEA	Passed all test.

Chapter 7 Conclusion

The objective of this report is to get the performance of symmetric block cipher encryption on different platform: normal computing platform, parallel programming with OpenMP under multi-core platform and parallel programming with CUDA under GPU platform. The block cipher algorithm involved in this paper was Blowfish and IDEA. The operation mode of both encryptions selected for this paper is counter mode because of the possible of applying parallelism to the counter mode.

In order to achieve the goal of this paper, multiple techniques are used in this paper. C programming is used in implement block cipher algorithm on normal computing platform. OpenMP library is used together with C programming to implement block cipher algorithm in parallel manner under multi-core platform. CUDA framework is used to implement block cipher algorithm under GPU platform.

The result collected from the experiments had shown that the objective of this report is achieves. The performance speed is faster from normal computing platform to parallel programming with CUDA under GPU platform. This result is true only for IDEA and Threefish block cipher but not applicable to Blowfish. This is because Blowfish is a table lookup based block cipher and cause the whole encryption to become slow by waiting for the table lookup.

Parallel programming can in deed hugely increase the performance speed of the symmetric block cipher but more optimization should be done on the source code in order to get a faster performance speed especially for OpenMP platform.

Next, pseudo random numbers are generated from block cipher in counter under GPU platform, which is tested in this paper. The pseudo random number generated had pass all NIST Test Suite but only pass some of the test in TestU01. Only a few random number generators can pass all the test in TestU01 so this proved that the objectives of this paper are achieved.

This paper will benefit any party or cooperation that had a server that involved a lot of internet traffic every day by increase the encryption speed every time when something are uploads to the internet. Other than that, this paper also benefits any party or organization that is seeking for new pseudo random number generator to be applied in their algorithm.

Reference

1. GPU Technology Conference, *Shuffle: Tips and Tricks*, NVIDIA, Available from: <https://www.google.com.my/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&sqi=2&ved=0CDgQFjAE&url=http%3A%2F%2Fon-demand.gputechconf.com%2Fgtc%2F2013%2Fpresentations%2FS3174-Kepler-Shuffle-Tips-Tricks.pdf&ei=WkorVZ3UFc7_ugSxjoGADg&usg=AFQjCNFjjKyYsoqfGfSZYOmlt21zmc1F3g&bvm=bv.90491159,d.c2E&cad=rja>
2. GPU Technology Conference, *CUDA STREAMS*, NVIDIA, Available from: < https://www.google.com.my/url?sa=t&rct=j&q=&esrc=s&source=web&cd=5&ved=0CDYQFjAE&url=http%3A%2F%2Fon-demand.gputechconf.com%2Fgtc%2F2014%2Fpresentations%2FS4158-cuda-streams-best-practices-common-pitfalls.pdf&ei=6UsrVZyPAYuVuASIkoCwAQ&usg=AFQjCNHaiSJvVZho-UHtK2N0SAN2IKW6Eg&bvm=bv.90491159,d.c2E&cad=rja>
3. *Introduction to Blowfish*, Available from : < http://www.splashdata.com/splashid/blowfish.htm> [1 – 8 – 2014]
4. Johannes Gilger, Johannes Barnickel & Prof. Dr. Ulrike Meyer, 'GPU-Acceleration of Block Ciphers in the OpenSSL Cryptographic Library'. [1 – 8 - 2014]
5. John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw, 2011, 'Parallel Random Numbers: As Easy as 1, 2, 3', 2011 International Conference, pp1 – pp12. Available from: IEEE Computer Society. [18 – 11 – 2011]
6. Modes of operation. Available from : <http://cs.ucsb.edu/~koc/cs178/docx/w04x-modes.pdf>. [1 - 8 - 2014]
7. Naoki Nishikawa, Keisuke Iwai and Takakazu Kurokawa, 2011, 'High-Performance Symmetric Block Ciphers on CUDA', 2011 Second International Conference on Networking and Computing, pp221 – pp227. Available from : IEEE Computer Society. [1- 8 - 2014]
8. *Network App. Related Cores*, Available from : < http://www.unistring.com/network.htm> [1 – 8 – 2014]
9. NVIDIA corporation, 2008, CUDA Programming Model Overview, Available from : < http://www.sdsc.edu/us/training/assets/docs/NVIDIA-02-BasicsOfCUDA.pdf> [1 – 8 - 2014]

10. Tim Mattson & Larry Meadows. A “Hands-on” Introduction to OpenMP. Available from : <http://openmp.org/mp-documents/omp-hands-on-SC08.pdf>. [1 - 8 - 2014]
11. *The Skein Hash Function Family*, Available from: <https://www.schneier.com/skein1.3.pdf> [2 – 3 – 2014]
12. Wai-Kong Lee, Bok-Min Goi, Raphael C.-W. Phan and Geong-Sen Poh, 2014, ‘High Speed Implementation of Symmetric Block Cipher on GPU’, 2014 International Symposiu8m on Intelligent Signal Processing and Communication Systems (ISPACS), pp102 - pp107. Available from: IEEE Computer Society. [4 – 12 – 2014]