PARC4-I: PARALLEL IMPLEMENTATION OF ENHANCED RC4A USING PASCS AND LOOP UNROLLING MECHANISM

Disha Handa[1] and Bhanu Kapoor[2]

[1]Computer Science Engineering Department, Chitkara University, Himachal Pradesh, India
[2]Computer Science Engineering Department, Chitkara University, Himachal Pradesh, India

ABSTRACT

In today's network-based cloud computing era, software applications are playing big role. The security of these software applications is paramount to the successful use of these applications. These applications utilize cryptographic algorithms to secure the data over the network through encryption and decryption processes. The use of parallel processors is now common in both mobile and cloud computing scenarios. Cryptographic algorithms are compute intensive and can significantly benefit from parallelism. This paper introduces a parallel approach to symmetric stream cipher security algorithm known as RC4A, which is one of the strong variants of RC4. We present an efficient parallel implementation to the compute intensive PRGA that is pseudo-random generation algorithm portion of the RC4A algorithm and the resulted algorithm will be named as PARC4-I. We have added some functionality in terms of lookup tables. Modified algorithm is having four lookup tables instead of two and is capable of returning four distinct output bytes at each iteration. Further, with the help of Parallel Additive Stream Cipher Structure and loop unrolling method, encryption/decryption is being done on multi core machine. Finally, the results shows that PARC4-I is a time efficient algorithm.

KEYWORDS

Parallel Algorithms, Encryption, RC4, Symmetric, WEP, PRGA, Stream Cipher

1. INTRODUCTION

The Internet is the worldwide interconnection of networks operated by industry, academia, government and private parties. Initially the Internet is used to interconnect laboratories affianced in government research projects, and since 1994 it has been expanded to serve millions of users for simple routine as well as complex tasks. The Internet, as no other communication medium, has become the Universal source of information for millions of people at school, at home and at work. But the other side of the coin is we have to make sure that our communications are safe while transmitting over the internet. It is sensible to suppose that at some point someone may capture and alter your transmissions. So we should use some form of encoding at the sender level and decoding at receiver to protect the data from unauthenticated parties. To safeguard the data from unauthorized use over the channel encryption/decryption process is used. It allows data to be transmitted that will be ineffective to anyone who intercepts it. Different encryption algorithms have been used to secure information communications over the network. Furthermore the encryption algorithms are categorized into two categories: Public key infrastructure based known as Asymmetric algorithm and private key infrastructure based also known as symmetric algorithm. In private key algorithms, same key is used for both encryption and decryption process. It's called private key encryption because sender and receiver must know before the message is sent how to interpret the message. RC4 [1-2] is a very popular symmetric stream cipher algorithm. RC4A is the variant of RC4 with more security in terms of key space. Some other algorithms include DES, 3-DES, and AES [3]. Public key algorithms use different keys for encryption and decryption process. Public key means that anyone can publish his or her method of encryption publish a key for his or her messages, and only the recipient can read the messages. This type of encryption is based on the mathematical logic of finding two prime factors of a very large number. In general it is easy to multiply two very large numbers together, but it is very difficult to take a very large number and find its two prime factors. RSA [3] is the commonly used asymmetric encryption algorithm.

Although security algorithms have many advantages like security of data transmitted over the channel yet there are some disadvantages involved in these algorithms. Intensive computation and their sequential structure affect the speed of the cipher. The speed of the encryption and decryption is a very important aspect of security algorithms [3] in working with applications. A slow cryptographic algorithm can slow the speed of an application and reduce its effectiveness. Sequential security algorithms can be made faster using parallelization. Fortunately, with the advent of parallel processors in computing, we now have easily available means to parallelize the algorithms to make them faster. The symmetric multiprocessors (SMP) such those readily available from companies such as Intel can be used in conjunction with parallel programming APIs such as OpenMP to make security algorithms parallel and faster. It is possible to use parallel algorithms for any of the cryptographic techniques currently in use.

This paper introduces a parallel approach to execute RC4A in parallel to encrypt a large set of data. Because RC4A encrypt data as per the architecture of synchronous additive stream cipher structure, first we have designed Parallel framework of additive stream cipher structure. During the initial study of parallel computing, we have implemented RC4 using data parallel model and got good speedup [6]. In this paper, we have presented parallel implementation of RC4A because its structure allows us to use loop unrolling technique for code optimization. We chose this stream cipher algorithm for two major reasons. One is, this algorithm is the strong variant of RC4 which is very popular and used in many popular protocols and applications and another is, its simplicity.

Rest of the paper is organized as follows: Section 2 shows some useful work which is already been done on this emerging area. We discuss the existing RC4A algorithm in section 3. We follow this up with an introduction to the parallel additive stream cipher framework which is the base of this implementation. Parallel methodology and related issues are discussed in section 5. Section 6 gives the ample knowledge of PARC4-I, a parallel approach of RC4A. This includes a discussion on how we parallelize it to gain performance benefits and various techniques and fundamentals to parallelize the PARC4-I. In section 7, security analysis of PARC4-I is done using ENT tool to check whether parallelization effects security or not. Next section discusses the experimental setup along with the results and speedup measurements. In section 9, Performance analysis for the same is being done. We then conclude the paper and summarize the results.

2. Related Work

In the evolving era of data-intensive computing and low-cost internet connections for global data communications, there is a higher demand for data security and computational speedup. In recent years, successful studies have been made using hardware acceleration technique, FPGA implementations, using CUDA's GPGPU platform to speed up the execution of cryptographic algorithms. This section is giving some glimpse about the work that has already done or in progress on RC4 family ciphers. K.H. Tsoi et al [1] presented a parallel FPGA implementation of RC4 algorithm. That FPGA design employs parallelism at the logic level to increase the number of operations per cycle by RC4 search engine. In their design they have used on-chip memories to attain very high memory bandwidth, floor planning to condense routing delays and multiple decryption units to accomplish further parallelism. Total 96 number of RC4 decryption engines were integrated on a single Xilinx Virtex XCV1000-E field programmable gate array (FPGA). The resulting design operates at a 50 MHz clock rate and gained a search speed of 6.06 × 106 keys/second, which is a speedup of 58 over a 1.5 GHz Pentium 4 PC.

Changxin Li et al [7] have presented an efficient implementation for MD5-RC4 encryption/decryption algorithm using NVIDIA's Graphics Processing Unit with CUDA programming framework. The algorithm was implemented on NVIDIA GeForce 9800GTX GPU and they got 3-5X speedup.

T.D.B Weerasinghe [8] presented a java based multithreaded implementation of RC4 algorithm using i3 and i7 series processors. The proposed method does not parallelize RC4, instead it introduces a way that multithreading can be used to perform encryption and decryption when the message is in the form of text file.

S.S Gupta [21] presents "High Performance Hardware Implementation". The paper is published in IEEE Transactions on Computers [Volume 62, issue 4]. The paper proposes the fastest hardware implementation for RC4 cipher and this is achieved by combining two key facts one is hardware pipeline and another is loop unrolling. This structural design produces two RC4 key stream bytes for each clock cycle. The proposed architecture is being implemented using VHDL , amalgamate with 130 , 90,65 nm manufacture technologies at clock frequencies 625MHz, 1.37 GHz, and 1.92 GHz, correspondingly. With the help of this architecture authors succeeded to attain the high throughput of 10, 21.92, and 30.72 Gbps for RC4 key stream.

This paper presents efficient hardware implementation of new stream cipher, RC4A [22]. The proposed hardware implementation achieves a data throughput up to 22.28 MB/sec at frequency of 33.33 MHz and the performance in terms of throughput to area ratio equal to 0.37. The implementation is also parameterized in order to support variable key lengths, 8-bit to 512-bit. The cipher was designed using Verilog hardware description language and implemented into a single Altera APEX TM 20K200E Field Programmable Gate Array (FPGA).

3. Existing RC4A Algorithm

Souradyuti Paul and Bart Preneel have proposed an RC4 variant, which they call RC4A. RC4A uses two state arrays S1 and S2, and two indexes j1 and j2. Each time i is incremented, two bytes are generated. First, the basic RC4 algorithm is performed using S1 and j1, but in the last step, S1 [i] + S1 [j1] is looked up in S2. Second, the operation is repeated (without incrementing i again) on S2 and j2, and S1 [S2 [i] +S2 [j2]] is output. The algorithm has two main parts: the key scheduling algorithm (KSA) and the pseudo random generation algorithm (PRGA) [1-2].

The key-scheduling algorithm

The key scheduling algorithm is used to generate the permutation array. In this algorithm the key stream is generated with the help of a variable length key with an internal state comprised of the following key elements:

1) Four 256 bytes S1-S2 arrays that contains a transformation of these 256 bytes
2) Three index pointers i, j1 and j2 which will use to point elements in the S1 and S2 arrays

The algorithm will start with initializing two arrays with the values from 0-255 that means the values in the array are equal to their index. Once the arrays are initialized, the next step is to generate random numbers and store in these two arrays to make them permutation arrays. For this we simply iterate the array 256 times, compute the value of j1 and j2 pointers with the help of j1 = j1 + S[i] + key[i mod key-length] formula where key is the user's input value and The "key length" is the number of bytes in the key and the range of "key length" is from 1 to 256.

The pseudo-random generation algorithm

In this step, generated key stream is XOR with plaintext to produce encrypted text in the form of a sequence of bytes. All arithmetic is performed modulo 256. The iterations in the PRGA algorithm depend on the input size. In each of the iterations, there is a different value for ranging from 0 to 255. If the input length is more than 256 then the process again starts from 0 and continues until the last byte. For each of the iterations, the value for j1 and j2 is calculated, S1 [i] and S1 [j1] are swapped, and the sum of S1 [i] and S1 [j1] mod 256 is looked up in the S2 array to return one byte. Same operation applies to S2 array. Returned bytes are then XOR with one individual letter in plaintext to convert it into cipher text.

Proposed Technique:

4. PARALLEL ADDITIVE STREAM CIPHER STRUCTURE

For stream ciphers, the whole encryption process is based on bit by bit encryption. One key bit is generated, and corresponding one plaintext bit is fetched and using XOR operation, one bit of Ciphertext is produced. This mechanism is not best fitted in today's supercomputing era because multiple core processors are available in market where each core is having its own cache memory. If multiple bits can be processed at the same time and multiple Ciphertext bits can generated at a single cycle, the encryption process can be much faster as compared to the traditional one where only single bit is encrypted at a time. Mostly software implemented stream ciphers are synchronous additive stream cipher structure which is sequential in nature. Parallel framework for the same will help to process many bits concurrently and make the application faster. Moreover, there are many stream cipher algorithms which are intensively used to secure file systems, communications etc. Parallel framework will help those to have parallel implementations for effective usage.

According to the new architecture, a key is supplied to the key stream generator which will produce random key stream. The plaintext is in the form of fixed sized blocks. The key stream is supplied to each individual block to process plaintext concurrently. The following figure illustrates the complete process.

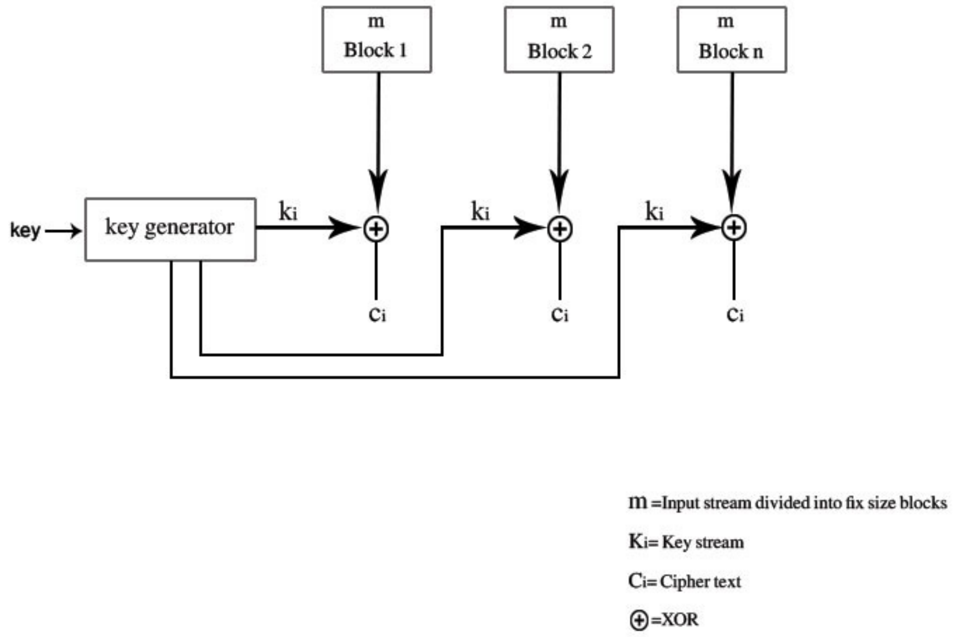


Figure-1: Design of Parallel Additive Stream Cipher Structure

This parallel structure can be used by any stream cipher which is of synchronous type. The size of the block depends upon the algorithm's architecture. PASCS is based on the concept of vernam cipher where corresponding to each bit of plain text there is individual key bit. To keep the essence of vernam cipher and maintain its randomness, each block should have different key stream. Hence, modification in architecture is required while implementing it in stream cipher algorithms.

5. Parallel Methodology and Related Issues

The major objective that serves as our motivation for this research is to have a faster and efficient cipher. But, Apart from the requirement of faster execution of cipher, a diversity of additional issues has also become vital over the same time.

5.1 Issues of Parallel Computing

One of the most conspicuous issue of parallel or high performance computing is the ability of the memory system to feed data to the processor at the required rate. The mismatch between processor speed and DRAM latency is typically bridged by a hierarchy of successively faster memory devices called caches that rely on locality of data reference to deliver higher memory system performance. Parallel platforms typically yield better memory system performance as compared to sequential one. The reason behind is: Larger aggregate caches and Higher aggregate bandwidth to the memory system both typically linear in the number of processors and locality of data reference. This argument can be extended to disks where parallel platforms can be used to achieve high aggregate bandwidth to secondary storage. The other benefit of parallelism is to get energy efficiency. In this section, we will introduce the methodology which is used to redesign the sequential RC4A cryptographic algorithm along with the implementation challenges.

PARC4-I execution model is based on Data parallel model because PASCS can be implemented using this model easily. According to this model, the tasks are statically mapped onto processes and each task performs similar operations on different data. Since all tasks perform similar set of computations, the decomposition of the problem into tasks is based on data partitioning because a uniform partitioning of data followed by a static mapping is sufficient to guarantee load balance. Data parallel algorithms can be implemented in both shared address space and message passing paradigms. We have used shared address space paradigm for this implementation. We chose this model because of its key characteristic that for most problems, the degree of data parallelism increases with the size of the problem, making it possible to use more processes/cores to effectively solve larger problems. At the same time, reducing the interaction overhead among concurrent tasks is important for an efficient parallel program. The overhead that a parallel program incurs due to interaction among its processes depends on many factors, such as the volume of data exchanged during interactions, the frequency of interaction, the spatial and temporal pattern of interactions. Thus to reduce interaction overheads we have considered following techniques [19]:

1) *Maximizing Data Locality*: The interaction overheads can be reduced by using techniques that promote the use of local data or data that have been recently fetched.
2) *Minimize volume of data exchange*: A fundamental technique for reducing the interaction overhead is to minimize the overall volume of shared data that needs to be accessed by concurrent processes. This is akin to maximizing the temporal data locality.
3) *Minimize frequency of interactions*: Minimizing interactions frequency is important in reducing the interaction overheads in parallel programs because there is a relatively high startup cost associated with each interaction on most of the architectures. Interaction frequency can be reduced by restructuring the algorithm such that shared data are accessed and used in large pieces.

5.2 Parallel Methodology

Apart from this there are some key concepts which serve as the basis of our parallelization methodology. These concepts are: Type of parallel computer, Decomposition technique and mapping technique for load balancing. In this section, we have discussed briefly all these concepts.

- Type of Parallel Computer*:*

  There are different methods to categorize parallel computers. According to Flynn's Taxonomy multi-processor computer architecture system is organized according to how they can be classified along with the two independent proportions of Instruction Stream and Data Stream. Each of these proportions can have only one of two possible states: Single or Multiple. There are four possible classifications:

  Single Instruction, Single Data (SISD):
  Single Instruction, Multiple Data (SIMD):
  Multiple Instructions, Single Data (MISD):
  Multiple Instructions, Multiple Data (MIMD):

- Decomposition*:*

  After identifying the type of parallel computer the next step involves the decomposition of data [9]. In case of large data sets, it is important to decide that how to decompose data so that an algorithm can execute in parallel. The optimization objective for decomposition is to balance the work-load among processing units and to minimize the inter process communication requirements. The number of data sets generated by the partitioning step may not be equal to the processing units/cores, thus a core may be idle or loaded with multiple processes. There are two techniques to decompose data: Domain partitioning and Functional partitioning. In *Domain partitioning* the data associated with an algorithm is decomposed and In *Functional* partitioning the computations involved in executing an algorithm is decomposed among multiple cores rather than data.

- Mapping for Load Balancing:

  After decomposing data the next step is to do load balancing [9]. Load balancing refers to the approach of distributing approximately equal amounts of work among cores so that all cores/processing units are kept busy all of the time. The primary optimization purpose of mapping is to balance the task load of processing unit/cores and to minimize the inter-processor communication cost. Commonly, the task of load balancing is to develop decomposition and mapping algorithm for the purpose of achieving their respective optimization objectives. Furthermore Load balancing algorithms can be broadly categorized into two categories, static or dynamic. Where Static load balancing algorithms distribute the processes to processors at compile time. This type of mapping is being used when we have a known data set. While dynamic algorithms bind processes to processors at run time. This approach is being used when we have unknown data set.

- Loop unrolling mechanism:

  Loop unrolling is also acknowledged as loop unwinding. It is a loop alteration technique that tries to optimize a program's execution speed at the cost of its binary mass/size. The conversion can be undertaken manually by the programmer. The objective of loop unwinding is to boost a program's speed by dipping instructions that control the loop, for example 'end of loop' test on every iteration, dropping branch penalties, and hiding latencies, particularly, the waiting time used to read data from memory. To eradicate this overhead, one can use the mechanism in which loops can be re-written as a repetitive series of alike independent statements.

6. PARC4-I Algorithm

As RC4A, PARC4-I is having two sub algorithms. One is for key generation and other is for encryption.

Enhanced KSA:

The enhanced approach comprised of the following key elements:

1) Four 256 bytes S1-S4 arrays that contains a transformation of these 256 bytes
2) Five index pointers i and j1, j2, j3, j4 which will use to point elements in the S1-S4 arrays

The algorithm will start with initializing four arrays with the values from 0-255 that means the values in the array are equal to their index. Once the arrays are initialized, the next step is to generate random numbers and store in these S1-S4 arrays to make them permutation arrays. For this, we simply iterate the array 256 times, compute the value of j1 to j4 pointers with the help of j1 = j1 + S[i] + key[i mod key-length] formula where key is the user's input value and The "key length" is the number of bytes in the key and the range of "key length" is from 1 to 256. Below is the pseudo-code corresponding to the key-scheduling algorithm:

```
1.  Start loop from i:= 0 to 255
2.  Fill the following arrays:
     S1 [i]:= i
     S2 [i]:=i
     S3 [i]:=i
     S4 [i]:=i
3.  End loop
4.  Set j1:= 0
5.  Start loop from i: =0 to 255
6.  Calculate j1:= (j1 + S1 [i] + key [i mod key length]) mod 256
7.  Swap values of S1 [i] and S1 [j1]
8.  End loop
9.  Set j2:= 0
10. Start loop from i: =0 to 255
11. Calculate j2:= (j2 + S2 [i] + key [i mod key length]) mod 256
12. Swap values of S2 [i] and S2 [j2]
13. End loop
14.   Set j3:= 0
15. Start loop from i: =0 to 255
16. Calculate j3:= (j3 + S3 [i] + key [i mod key length]) mod 256
17. Swap values of S3 [i] and S3 [j3]
18. End loop
19. Set j4:= 0
20. Start loop from i: = 0 to 255
21. Calculate j4:= (j4 + S4 [i] + key [i mod key length]) mod 256
22. Swap values of S4 [i] and S4 [j4]
23. End loop
```

Algorithm-1: Enhanced key scheduling algorithm (KSA)

The pseudo-random generation algorithm

As we have four look up tables and more index pointers, additional steps are required in PRGA. Now the sum of S1[i] and S1[j1] mod 256 is looked up in the S2 array to return one byte , sum of S2[i] + S2[j2] is looked up into s1 array , sum of S3[i] + S3[j3]] is looked up into s4 array and lastly sum of S4[i] + S4[j4 is looked up into s3 array. That means at each iteration it returns four distinct bytes and which are then used to encrypt four plaintext bytes in the single iteration jump using loop unrolling. Below mentioned algorithm 2 explains the enhanced PRGA.

```
1.  Set i , j1 , j2 , j3 , j4 := 0
2.  While Generating Output:
3.  i := i + 1
4.  Calculate j1:= j1 + S1[i]
5.  Swap values of S1[i] and S1[j1]
6.  Output S2[S1[i] + S1[j1]]
7.  Calculate  j2:= j2 + S2[i]
8.  Swap values of S2[i] and S2[j2]
9.  Output S1[S2[i] + S2[j2]]
10. Calculate J3:= j3 + S3[i]
11. Swap values of S3[i] and S3[j3]
12. Output S4[S3[i] + S3[j3]]
13. Calculate J4:= j4 + S4[i]
14. Swap values of S4[i] and S4[j4]
15. Output S3[S4[i] + S4[j4]]
```

Algorithm-2: Enhanced pseudo-random generation algorithm (PRGA)

Method that adhered parallelism

In this approach, the modification takes place in PRGA method. PRGA will return four distinct bytes at each iteration. The input text is divided into fixed size blocks, where each block size is 256 bytes. Afterwards, each individual block is encrypted using PRGA, where at each index pointer increment PRGA will generate four bytes so we can take first four bytes of plaintext altogether to encrypt or decrypt. Using loop unrolling technique we can do the process without loop jumps, and finally, the output of each block is concatenated to make the complete cipher text. Individual blocks are getting executed on multiple cores to have good speedup. The overhead associated with the function calls is also reduced as for each 64 byte of data PRGA is executed only 16 times instead of 32 in RC4A or 64 times in RC4. Figures 2 depict the mechanism followed for PARC4-I

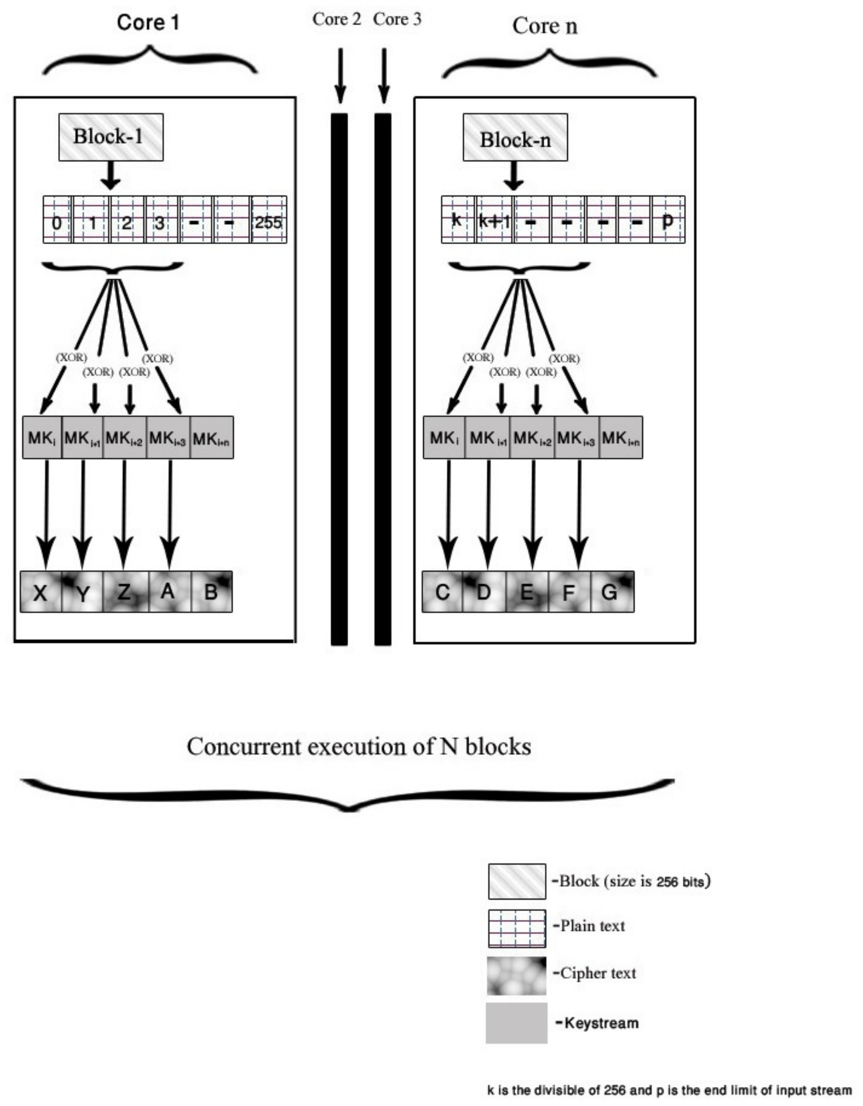
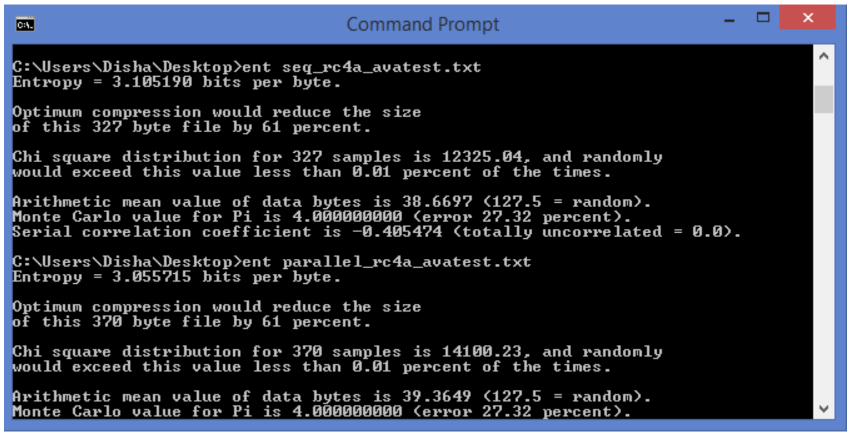

Core 1
Core 2
Core 3
Core n
Block-1
0 1 2 3 - - 255
(XOR) (XOR) (XOR) (XOR)
MKi MKi+1 MKi+2 MKi+3 MKi+n
X Y Z A B
Block-n
k k+1 - - - - p
C D E F G
Concurrent execution of N blocks
-Block (size is 256 bits)
-Plain text
-Cipher text
-Keystream
k is the divisible of 256 and p is the end limit of input stream

Figure: 2- Mechanism used to implement PARC4-I on SMPs

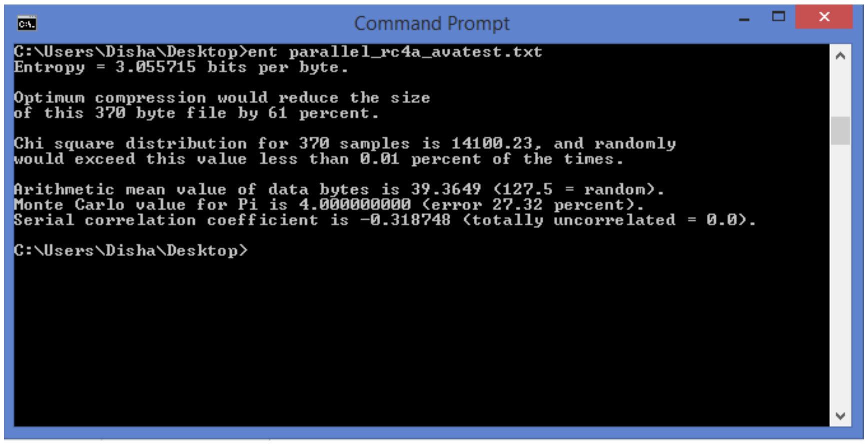
7. SECURITY ANALYSIS

To prove that the changes made during the parallel design does not affect the security of the cipher, ENT software is used to test the randomness of generated key stream bytes. Results are compared on the basis of entropy value for both the versions sequential as well as parallel and the results show that PARC4-I is as secure as RC4A. Consider the two figures, i.e. fig.3 and fig.4 given below:



```
C:\Users\Disha\Desktop>ent seq_rc4a_avatest.txt
Entropy = 3.105190 bits per byte.

Optimum compression would reduce the size
of this 327 byte file by 61 percent.

Chi square distribution for 327 samples is 12325.04, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 38.6697 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is -0.405474 (totally uncorrelated = 0.0).

C:\Users\Disha\Desktop>ent parallel_rc4a_avatest.txt
Entropy = 3.055715 bits per byte.

Optimum compression would reduce the size
of this 370 byte file by 61 percent.

Chi square distribution for 370 samples is 14100.23, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 39.3649 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
```

Figure-3: Entropy value of RC4A is 3.105 for 327 byte file



```
C:\Users\Disha\Desktop>ent parallel_rc4a_avatest.txt
Entropy = 3.055715 bits per byte.

Optimum compression would reduce the size
of this 370 byte file by 61 percent.

Chi square distribution for 370 samples is 14100.23, and randomly
would exceed this value less than 0.01 percent of the times.

Arithmetic mean value of data bytes is 39.3649 (127.5 = random).
Monte Carlo value for Pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is -0.318748 (totally uncorrelated = 0.0).

C:\Users\Disha\Desktop>
```

Figure-4: Entropy value of PARC4-I is 3.055 for 370 byte file

8. EXPERIMENTAL SETUP

To estimate the speed up gains, firstly, the sequential RC4A cryptographic algorithm has been executed to evaluate the execution time. The sequential results serve as the baseline for comparison with the results of the improved parallel algorithm PARC4-I. The same compiler along with same configuration options have been used to compile all data files. To disable debugging, Compiler option –g0 is used. Similarly, -O3 is used to support third level of optimization and -March=native to enable usage of CPU specific instructions. All the data has been tested on a server having configuration as mentioned below:

- Processor - AMD FX(tm) - 8320 , eight core processor running @ 3500 MHz
- RAM - 8 GB
- System Type - 64 bit operating system,x64- based processor
- Operating System - Linux/Ubuntu 12.04 version
- OpenMP - 4.0
- Compiler- GCC

- Programming Language - C with OpenMP

To assess the parallel framework, all tests use the text files from t5-corpus11 with amendments. Below tables show the time taken by RC4A and PARC4-I using multiple cores. The first column of each of the table shows the number of input bytes used for encryption and decryption. Second and third column shows the execution time [in seconds] for encryption and decryption processes and last column shows the overall time taken by both the processes.

Table-1: Time [In Seconds] taken to encrypt/decrypt bits of input stream by uniprocessor

Size of input stream[in bits]	Encryption time	Decryption time	Overall Time
9,34,15,936	0.63482	0.71552	1.35034
18,68,31,872	1.58461	1.5705	3.15511
28,02,47,808	2.52451	2.50427	5.02878
37,36,63,744	3.59566	3.57956	7.17522
46,70,79,680	4.77162	4.54654	9.31816
56,04,95,616	6.5028	6.21054	12.71334
65,39,11,552	7.8804	7.14834	15.02874
74,73,27,488	8.91011	8.77074	17.68085
84,07,43,424	9.43049	9.59899	19.02948
93,41,59,360	10.71573	10.49485	21.21058

Table-2: Time [In Seconds] taken to encrypt/decrypt bits of input stream by 4 cores

Size of input stream[in bits]	Encryption time	Decryption time	Overall Time
9,34,15,936	0.17993	0.17989	0.35982
18,68,31,872	0.41564	0.41558	0.83123
28,02,47,808	0.66285	0.66277	1.3256
37,36,63,744	0.97849	0.97845	1.95694
46,70,79,680	1.24622	1.24616	2.49238
56,04,95,616	1.68213	1.68207	3.36421
65,39,11,552	2.01168	2.01162	4.0233
74,73,27,488	2.36608	2.36602	4.7321
84,07,43,424	2.50562	2.50559	5.0112
93,41,59,360	2.81159	2.81151	5.62311

Table-3: Time [In Seconds] taken to encrypt/decrypt bits of input stream by 6 cores

Size of input stream[in bits]	Encryption time	Decryption time	Overall Time
9,34,15,936	0.11847	0.11843	0.2369
18,68,31,872	0.2766	0.276	0.5526
28,02,47,808	0.4415	0.4407	0.8822
37,36,63,744	0.6297	0.6291	1.2588
46,70,79,680	0.8177	0.8171	1.6348
56,04,95,616	1.1157	1.1148	2.2304
65,39,11,552	1.31328	1.31322	2.6265
74,73,27,488	1.55089	1.55081	3.1017
84,07,43,424	1.6659	1.6655	3.3314
93,41,59,360	1.8606	1.8605	3.7211

Table-4: Time [In Seconds] taken to encrypt/decrypt bits of input stream by 8 cores

Size of input stream[in bits]	Encryption time	Decryption time	Overall Time
9,34,15,936	0.09248	0.09242	0.1849
18,68,31,872	0.21614	0.21606	0.4322
28,02,47,808	0.34438	0.34432	0.6887
37,36,63,744	0.49144	0.49136	0.9828
46,70,79,680	0.63823	0.63817	1.2764
56,04,95,616	0.87077	0.87073	1.7415
65,39,11,552	1.0294	1.0293	2.0587
74,73,27,488	1.21104	1.21097	2.422
84,07,43,424	1.30337	1.30333	2.6067
93,41,59,360	1.45277	1.45273	2.9055

Table 1 to table 4 shows the execution time of PARC4-I for large input streams. The first column shows the number of input bytes/bits used for encryption and decryption. Second and third column shows the encryption/decryption time in seconds. Last column represents the overall time. We have specified the execution time of single core, four cores, six cores and eight cores. We can see the speedup increases as the input size increases.

9. PERFORMANCE AND SCALABILITY ANALYSIS

To observe the performance gains using parallelism, following metrics have been used for proposed algorithm.

Parallel Run Time: The parallel run time is a measure represented as P_T (n), where P_T (n) is the parallel execution time of a program on a SMP having n cores. If n=1, P_T (1) denotes the serial run time of a program using single processor core. Below figure visualizing the parallel run time on each core using PARC4-I.

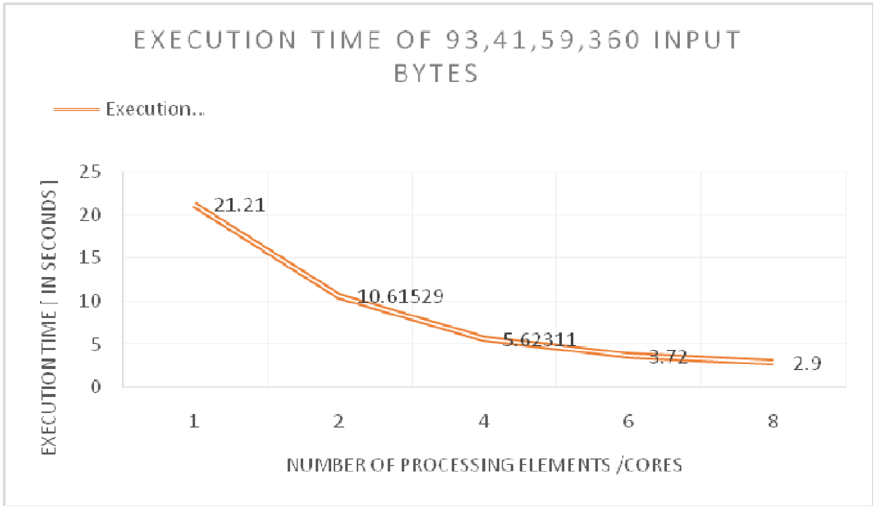


Figure: 5- Execution time of 93, 41, 59,360 bytes using PARC4-I

It is visible in above graphical representation that increasing cores will be drastically reducing execution time.

Speedup: Speedup is a quantitative measure of performance gain that is accomplished by parallel implementation using SMPs of a specific program over a sequential implementation using uniprocessor system. But to capture the relative benefit of running a program in parallel, the sequential algorithm should be the fastest algorithm. From above tables 1 to table 4, it can be observed that PARC4-I is giving 7.3x speedup using eight cores and 5.7x using 6 cores. Figure-6 is showing the speedup comparison of PARC4-I using multiple cores.

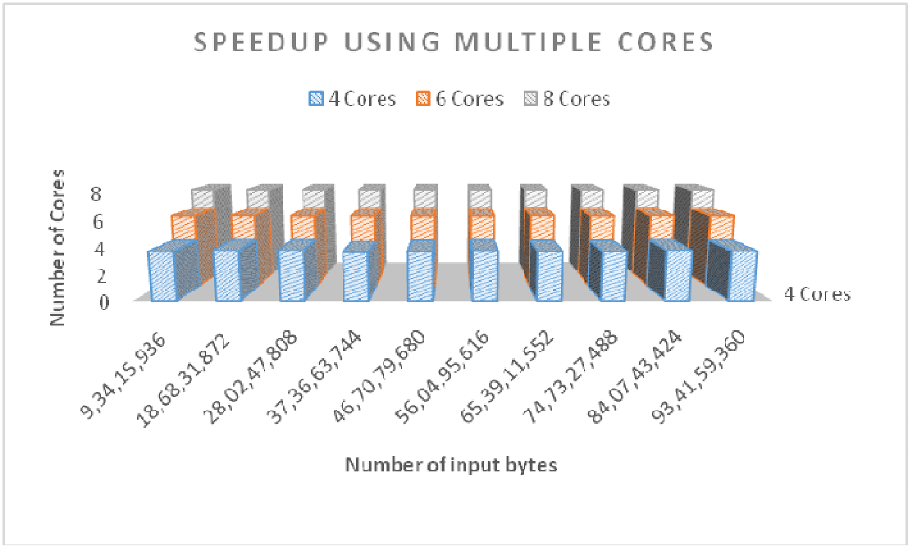


Figure-6: Speedup comparison of PARC4-I using multiple cores

It is visible from the graph above that speedup is increasing as the number of cores are increasing. But as per the consequences of Amdahl's law, speedup tends to saturate and efficiency can drop at some specific point. Now the point where saturation occurs, depend on the type of parallel execution model used to execute the problem, the size of data associated with the problem and the number of cores to process that data. If the processing elements and problem data are increasing, the overhead of decomposition and distribution of tasks among processors are also increased. Thus at certain point it must tend to saturate which can affect the overall speedup. Similarly, if execution time is observed in above tables for large input streams like 93, 41, 59,360 bytes of data, the total time for executing the complete data is the function of number of cores employed to complete the task. Every time if two processing cores are added, it takes approximately half of the time it takes to process with less number of cores. This way it can be inferred that after adding two additional cores for constant file size, the execution time will be half of the time it takes for previous attempt.

Efficiency: This metric reflects how effectively all processing elements are working. Actually, it is the function of load balancing. The efficiency of a given problem using n processing elements, *E* (n), is defined as the ratio of the speedup attained and the number of processors used to attain it:

$$E(n) = \frac{T(1)}{n * T(n)} \qquad (1)$$

In Eq.1, Let *T* (1) = 21.21 seconds, *T* (n) = 2.9 seconds and *n*=8 to process 93, 41, 59,360 bytes. The Efficiency of PARC4-I is: = $\frac{21.21}{8*2.9}$ = 0.91, since $1 < S(n) \leq n$, there is $\frac{1}{n} \leq E(n) \leq 1$. PARC4-I achieved $0.125 \leq 0.91 \leq 1$ for increasing p and n.

Scalability: As stated above, PARC4-I maintained efficiency 0.9 for all increasing cores with respect to scalable data. This property of proposed algorithm indicates that the algorithm has scalability feature as it is promising to keep the efficiency fixed by increasing the problem size and processing elements instantaneously.

CONCLUSION AND FUTURE SCOPE

This paper introduces PARC4-I, a parallel approach to the well-known RC4A cipher algorithm. The basic idea behind this implementation is to use some loop unrolling optimization techniques along with the parallel methodology to improve the performance gains. The implementation shows promising results with the use of loop unrolling method. The following conclusions are drawn from the discussion about this implementation. As a result of use of the PASCS framework along with the loop unrolling optimization techniques, we get better performance gains along with the gains in efficiency and throughput. In addition, the new algorithm is as scalable as the algorithm that doesn't use the optimization.

REFERENCES

[1] K.H.Tsoi et al, “A massively parallel RC4 key search engine(With FPGA)”,10 Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2002 .

[2] Hisham A. Kholidy, Khaled S. Alghathbar, “Adapting and accelerating the stream cipher algorithm “RC4” using “ultra gridsec” and “HIMAN” and use it to secure “HIMAN” data”, journal of information assurance and security 4(2009) 474-483. 11

[3] D Elminaam et al., “Evaluating the performance of symmetric encryption algorithms”, Int. journal of Network security, 2010 May, Vol.10, No.3. 3

[4] V.A.Korthikanti and Gul Agha, “Energy-Performance trade-off analysis of parallel algorithms”,unpublished.

[5] D Handa and B Kapoor, “State of the Art Realistic Cryptographic Approaches for RC4 Symmetric Stream Cipher ”IJCSA, Vol.4, No.4, pp. 27-37, August 2014

[6] D.Handa and B.Kapoor,”PARC4: High Performance Implementation of RC4 Cryptographic Algorithm using Parallelism”, International Conference on Reliability Optimization & Information Technology, Feb,2014

[7] Changxin Li et al,” Efficient implementation for MD5-RC4 encryption using GPU with CUDA”, 3rd International Conference

[8] T.D.B Weerasinghe,” Improving throughput of RC4 algorithm using multithreading techniques in multicore processors”, International Journal of Computer Applications(0975 – 8887) Volume 51– No.22, August 2012

[9] D Handa and B Kapoor, Performance Analysis of PBlock Algorithm Implemented Using SIMD Model to Attain Parallelism”, in International Conference on Emerging ICT for Bridging Future , Dec 2014

[10] Gene M. Amdahl,”Validity of the single processor approach to achieving large scale computing capabilities”,AFIPS spring,ACM,New York,NY,USA,pp.483-485,1967

[11] http://www.ibm.com/developerworks/library/l-gnuprof.html

[12] http://sourceware.org/binutils/docs/gprof/

[13] B. Chapman,G. Jost and Ruud Pas, Using OpenMP: Portable Shared Memory Parallel Programming , MIT Press, 2008.

[14] C. Hughes, T. Hughes, Professional Multicore Programming: Design and Implementation for C++ Developers, Wrox

[15] M Damrudi et al.,” State of the Art Practical Parallel Cryptographic Approaches”, Aus. Journal of Basic and Applied Sciences,2011,ISSN 1991-8178

[16] P.Ruangchaijatupon and P.Krishnamurthy,”Encryption and power consumption in wireless LAN”, The third IEEE workshop on wireless LANs, Newton, Massachusetts Sep.27-28,2001, pp.148-152.

[17] Chen Liu,Rolando Duarte,Omar Granados,Jie Tang,Shaoshan Liu,Jean Andrian,”Critical Path based hardware Acceleration for Cryptosystems,” journal of information processiong system(JIPS),Vol.8,No.1,pp.133-144,2012

[18] V.U.K Sastry and K. Anup Kumar,”A Modified Feistel Cipher Involving Substitution, shifting of rows, mixing of columns, XOR operation with a Key and Shuffling”, Int. Journal of Advanced Computer Science and Applications,2012,Vol.3,No.8.

[19] A. Grama,A. Gupta , G. Karypis and V. Kumar, An Introduction to parallel Computing , Pearson , Second Edition.

[20] E.J. Swankoski et al,”A parallel architecture for secure FPGA Symmetric Encryption” unpublished

[21] S.S.Gupta et al,”High-Performance Hardware Implementation for RC4 Stream Cipher", IEEE Transactions on computers, pp: 730 - 743 Volume: 62, Issue: 4, April 2013

[22] Abdullah Al Noman et al,"Hardware Implementation of RC4A Stream Cipher",International Journal of Cryptology Research,pp.224-233,2009