USD 004.312.46:681.2

R.K. APIKYAN

ELAPSED TIME COMPARISION OF THE LFSR PARALLEL AND SEQUENTIAL OUTPUT GENERATION

The linear feedback shift registers (LFSR) are designed for pseudo-random number generation. These numbers consist of one and zero sequences and have a repeating period. LFSR has a maximum repeating period if the unique output bits length is equal to *2n-1*. The LFSR used in encryption and signal processing (CDMA). The time of generation of LFSR's output data depends on the required output bit's length, the longer is the required output length the more time it will require for generation. If the data or signal is encrypted with long-length LFSR output data it will require a long time for the receiver to decrypt the received data as it will first start the local generation of LFSR's output data and then start the decryption process. The parallel generation of the LFSR's output bits allows shorting the generation time and, based on that, the time for data encryption and decryption is short too.

The output bits of LFSR can be generated in a sequential and parallel manner. In this article, we will generate the output data in both ways, and calculate and compare the generation time.

Keywords: LFSR, parallel and sequential generation, Java, encryption, CDMA.

Introduction. For LFSR simulation we are going to use the Java library from [1]. In this library, we can find all the required tools that are needed for LFSR's output generation. The library contains the abstraction of LFSR, which means that we can create the generic LFSRs with n length and the randomly positioned feedback values. By default, it is up to the user to specify the algorithm of generating the output data: can be generated in a sequential manner or parallel. For sequential generation, we just need to create the required LFSR's object and call a "process" method in iteration, where the length of the iteration is the maximum length of the LFSR [2]. For a parallel generation, we are going to use the class "ParallelLFSR" [3] which uses a special algorithm that allows parallelizing the output data generation. The algorithm is based on LFSR's upcoming state generation and it defines the upcoming state and starts the output generation from that state. Before starting the generation time testing, we should note that the parallel generation method uses recursion as a soft solution which is realized done using the Java language. The sequential generation uses iteration, and the iteration in Java working is much faster than the recursion [4]. The solution for parallel generation shows time advantages when it is implemented in the scheme and not in a high-level programming language. However, we can measure the speed between the iteration and recursion on the tested device (computer) and define a coefficient

of relativity between the time of iteration and recursion that will be equal to $y = IT / RT$ where IT is the speed of iteration and the RT is the speed of recursion for the same number of iterations. After performing the generation testing for the parallel method, we need to multiply the elapsed time with y to get the corrected elapsed time related to the iteration method.

Defining the time relativity between iteration and recursion. For defining the y coefficient we need to run the program that will perform an iteration of n times and record the elapsed time between the start and end of iteration. Then we need to run the recursion n times and measure the elapsed time between the start and end of the recursion process. Let's run the "RecusionVsIteration" [5] program from the library and define the time relativity for different n values. The results are displayed in Table .

Table

Iteration and recursion elapse times

n	R_T	I_T
100	19299	4760
500	97368	7933
1000	166839	11278
5000	742013	43381
8000	1051972	65485

As we can see from Table the iteration process takes much less time than the recursion. And while measuring the LFSR output generation time we need to consider this.

LFSR sequential and parallel output data generation. For testing the output data generation let's look at the program "ParallelAndSequentialLfsrOutputGeneration" [6] from the library. In the "main" method, we have defined the LFSR with $n = 17$, and the feedback positions are $f_1 = 2, f_2 = 3, f_3 = 6, f_4 = 8, f_5 = 9, f_6 = 17$.

After running the program, we can see the elapsed times for each kind of generation parallel and sequential as shown in Figure.

```
Sequential  17596273    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
Parallel    123622027   [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
```

Fig. LFSR output generation elapsed times in sequential and parallel manner

As we can see, the parallel generation time is much longer than the sequential generation, because the parallel generation is based on recursion. For equalizing the recursion effect we need to multiply the parallel generation time with the

relativity coefficient between recursion and iteration. The tested LFSR's length is 17 which means it produces 2^{17}-1 period output data. The algorithm which is based on the parallel generation will start recursion after every *17* steps, which means that we need to divide the 2^{17}-1/17 = 7710 to define the recursion count. The value is close to $n = 8000$ from Table 1. The $y = I_T / R_T = 65485 / 1051972 = 0.0622$ and $P_{T\,real} = P_T * y = 7689290$, which is ***2.28*** times faster than the P_T sequential method.

Conclusion. The article demonstrates LFSR's parallel and sequential output data generation and the elapsed time for LFSR with length $n = 17$ and $f_1 = 2, f_2 = 3, f_3 = 6, f_4 = 8, f_5 = 9, f_6 = 17$ feedback positions for each of sequential and parallel manners.

REFERENCES

1. GPS generator library reference
   https://github.com/RobertApikyan/GpsGenerator/tree/parallel_and_sequential_generation
2. John F. Brendle Jr. (2000) Pseudorandom Code Generation for Communication and Navigation System Application, 17-23.
   https://apps.dtic.mil/dtic/tr/fulltext/u2/a336311.pdf
3. ParallelLFSR class reference
   https://github.com/RobertApikyan/GpsGenerator/blob/parallel_and_sequential_generation/src/src/main/ParallelLFSR.java
4. Recursion vs Iteration article in Medium
   https://medium.com/backticks-tildes/iteration-vs-recursion-c2017a483890
5. RecusionVsIteration class reference
   https://github.com/RobertApikyan/GpsGenerator/blob/parallel_and_sequential_generation/src/src/main/RecursionVsIteration.java
6. ParallelAndSequentialLfsrOutputGeneration class reference
   https://github.com/RobertApikyan/GpsGenerator/blob/parallel_and_sequential_generation/src/src/main/ParallelAndSequentialLfsrOutputGeneration.java

Ռ.Կ. ԱՊԻԿՅԱՆ

ԳԾԱՅԻՆ ՀԱԿԱԴԱՐՁ ԿԱՊՈՎ ՏԵՂԱՇԱՐԺՄԱՆ ՌԵԳԻՍՏՐՆԵՐԻ ԵԼՔԱՅԻՆ ՏՎՅԱԼՆԵՐԻ ԳԾԱՅԻՆ ԵՎ ՋՈՒԳԱՀԵՌ ԳԵՆԵՐԱՑԻԱՅԻ ԺԱՄԱՆԱԿՆԵՐԻ ՀԱՄԵՄԱՏՈՒՄԸ

Գծային հետադարձ կապով տեղաշարժման ռեգիստրները (ԳՀԿՏՌ) նախատեսված են քվազի պատահական համարների գեներացիայի համար: Այս համարները բաղկացած են զրոներից և մեկերից և ունեն կրկնման պարբերություն: ԳՀԿՏՌ-ն օժտված է առավելագույն կրկնման պարբերությամբ, եթե նրա չկրկնվող ելքային բիթերի երկարությունը հավասար է 2n-1-ի: ԳՀԿՏՌ-ները օգտագործվում են տվյալների կոդավորման և

ազդանշանների մշակման համար (CDMA). ժամանակը, որն անհրաժեշտ է ԳՀԿՏՌ-ի ելքային տվյալների գեներացիայի համար, ուղիղ համեմատականորեն կախված է անհրաժեշտ ելքային բիթերի քանակից. որքան մեծ է պահանջված ելքային բիթերի քանակը, այնքան ժամանակ է անհրաժեշտ դրանց գեներացիայի համար: Բավականաչափ երկար ժամանակ է անհրաժեշտ տվյալների ապակոդավորման համար, եթե ընդունիչի ընդունված տվյալները կոդավորված են ԳՀԿՏՌ-ով, որը բաղկացած է մեծ քանակի ռեգիստրներից, քանի որ ընդունիչը, մինչ դեկոդավորման պրոցեսի սկսելը, լոկալ գեներացնում է ԳՀԿՏՌ-ի ելքային տվյալները, որից հետո սկսում է դեկոդավորման պրոցեսը: ԳՀԿՏՌ-ի ելքային տվյալների զուգահեռ գեներացիան թույլ է տալիս ավելի կարճ ժամանակահատվածում գեներացնել ելքային տվյալները, և հետևաբար, ավելի արագ կատարել տվյալների կոդավորում և դեկոդավորում:

ԳՀԿՏՌ-ի ելքային տվյալները հնարավոր է գեներացնել երկու եղանակով՝ զուգահեռ և հաջորդական: Աշխատանքի շրջանակներում իրականացվում է երկու տեսակի գեներացիա, հաշվարկվում և համեմատվում են գեներացման համար ստացված ժամանակները:

Առանցքային բառեր. ԳՀԿՏՌ, զուգահեռ և հաջորդական գեներացիա, Java, կոդավորում, CDMA:

Р.К. АПИКЯН

СРАВНЕНИЕ ВРЕМЕН ПАРАЛЛЕЛЬНОЙ И ЛИНЕЙНОЙ ГЕНЕРАЦИИ ВЫХОДНЫХ ДАННЫХ РЕГИСТРОВ СДВИГА С ЛИНЕЙНОЙ ОБРАТНОЙ СВЯЗЬЮ

Регистры сдвига с линейной обратной связью (РСЛОС) предназначены для генерации псевдослучайных чисел. Эти числа состоят из одной и нулевой последовательностей и имеют период повторения. РСЛОС имеет максимальный период повторения, если длина выходных битов равна *2n-1*. РСЛОС используются при шифровании и обработке сигналов (CDMA). Время генерации выходных данных РСЛОС зависит от требуемой длины: чем больше длина выходных битов, тем больше времени необходимо для генерации. Приемнику потребуется много времени для дешифрования полученных данных, если данные зашифрованы с использованием длинных выходных данных РСЛОС, поскольку приемник, до того как начать процесс дешифрования, проводит локальное генерирование выходных данных РСЛОС. Параллельная генерация выходных битов РСЛОС позволяет сократить время генерации, а также время шифрования и дешифрования.

Выходные биты РСЛОС могут быть сгенерированы последовательно и параллельно. В статье сгенерированы выходные данные обоими способами, а также рассчитано и сравнено время генерации.

Ключевые слова: регистр сдвига с линейной обратной связью, параллельная и последовательная генерации, Java, шифрование, CDMA.