Efficient Multicore Implementation of An Advanced Generator of Discrete Chaotic Sequences

Karol Desnos, Safwan El Assad, Aurore Arlicot, Maxime Pelcat, Daniel Ménard

Efficient Multicore Implementation of An Advanced Generator of Discrete Chaotic Sequences

Karol Desnos*, Safwan El Assad**, Aurore Arlicot**, Maxime Pelcat*, Daniel Menard*
* IETR, INSA Rennes, CNRS UMR 6164, Rennes, France
** IETR, Université de Nantes, CNRS UMR 6164, Nantes, France

Abstract— **This paper details the design and implementation performances of an efficient generator of chaotic discrete integer valued sequences. The generator exhibits orbits having very large lengths compared to those given in the literature. It is implemented in C language and parallelized using the Parameterized and Interfaced Synchronous Dataflow Model of Computation (PiSDF MoC). The proposed structure is shown to be scalable, parallel and time efficient. The resulting implementation combines a very long minimal chaotic sequence $o_{min} > 7*2^{128}$ 32-bit samples and a very high throughput of 173Mbps on 4 cores of a General Purpose Processor.**

Chaotic generator, pseudo-random number generator, multicore implementation, performance analysis, dataflow model of computation

I. INTRODUCTION

Many information hiding and security systems such as: cryptosystems, key generation, steganography and digital watermarking systems need pseudo-random number generators (PRNGs) to achieve their objectives. An unpredictable physical process such as plasmonic or resonant systems and quantum physics [1] can be used to produce a non-deterministic random number (or bit). Process-based algorithms are used to generate numbers (bits) deterministically [2][3][4]. In the last two decades, a class of PRNGs called chaotic PRNGs has attracted much attention. Indeed, this class presents some desirable properties such as: very big sensitivity to the initial conditions and the parameters of the system, ergodicity, and ease of implementation, that make them very good candidates for use in information hiding and security systems.

We propose in this paper an efficient implementation of a generator of discrete pseudo-chaotic sequences using N = 32 bits finite precision. The proposed structure integrates three techniques: recursion, disturbance and cascading to avoid the dynamical degradation. It is also very efficient in terms of robustness and data rate. The generator results from a French patent [15] and its PCT Extension.

Dataflow modeling and global multicore scheduling are used to generate a parallel executable for the chaotic random generator. Dataflow modeling consists in breaking the computation into independent processes that communicate through First-In First-Out (FIFO) data queues. Global multicore scheduling consists of choosing a core for executing each process and ordering their execution on each core. In this paper, the chaotic generator is described by a Parameterized and Interfaced Synchronous Dataflow graph (PiSDF) [5] that serves as an input for the PREESM rapid prototyping tool [6]. PREESM is an open-source Eclipse-based tool that can simulate and generate multicore code for a given topology, i.e. organization of cores, and parallelism of architecture within minutes.

The applications of such efficient generator are numerous in the field of information hiding and security. The paper is organized as follows. Related Works are presented in Section 2 and the description of the proposed chaotic generator in Section 3. Section 4 describes the proposed techniques of optimization. Experimental results are given in Section 5, before concluding in Section 6.

II. RELATED WORK

Chaos can be generated by any non-linear dynamical system. In discrete-time non-linear dynamical system the generation of chaos is governed by a set of difference equations, namely a chaotic map. In the literature, the number of well-known one, two and three dimensional chaotic maps is surprisingly limited, in spite of the fact that they are widely used in many applications related-data security. In [7], we studied some of them, namely, the Logistic map, the piecewise linear chaotic map (PWLCM), the Frey map, and we designed others, specifically, x cos(x), x exp[cos(x)], and 2-D Tmap. We demonstrated that, when used alone, these chaotic maps don't exhibit very good statistical properties and especially their cycle length is limited. These observations are generally available for the others well-known chaotic maps such as: 1-D Skew tent [8], 2D-Standrd map, 2D-Cat map, 2D- Baker map [9] used in many cryptosystems to achieve the confusion/diffusion effects, or Lozi maps [10] and 3-D Lorenz map [11] used as chaotic generator. Lian et al. [12] studied in detail the performance of the 2-D Standard, Cat and Baker maps. They demonstrated that the Cat map has the smallest key space, but the highest key sensitivity and that it is suitable for cryptosystems using a different key in every iteration (dynamic keys). In this manner the key space is enlarged and key sensitivity is increased. The key spaces for Standard map and Baker map are both larger than that of Cat map. But to keep high key sensitivity, the number of iterations should be larger than 4 for the Standard map and bigger than 12 for the Baker map. These maps are more suitable for cryptosystems in which the same key is used in different iterations.

The authors wish to thank the French National Research Agency ANR, the AtlanSTIC-CNRS FR2819 Federation and the SATT organization. This work is carried out within the framework of the research project ACSCOM (Apport du Chaos dans la Sécurité des systèmes Communicants Optiques et Mobiles), the AtlanSTIC research cluster and the GenSeq project)

However, to keep a good confusion property, the average distance change in the whole image must be greater than 40%, and then, the number of iterations of the Cat map must be no smaller than 6.

To enhance the statistical properties of the generated sequences, the main ideas are based on the introduction of a technique of disturbance of the pseudo-chaotic orbit and also on the mixing of different components.

In [8], René Lozi introduced new models of very weakly coupled logistic and symmetric tent maps, based on a matrix of disturbance and using single or double precision numbers. He demonstrated that the 3-coupled tent maps with small value of perturbation can be used as a generator of pseudo-chaotic numbers with a uniform distribution over the interval [-1, +1]. The generated sequences have orbits of very long period, greater than 109 and less than 1012.

In [13], Thomas E. Tkacik proposed a 32-bit hardware random number generator based on a Linear Feedback Shift Register (LFSR), and a Cellular Automata Shift Register (CASR). The LFSR uses a primitive polynomial of degree equal to 43 and then gives a cycle length of 243-1. The CASR is based on 37-bit with a CA150 at cell site 28, and CA90s at all other cell sites. So, it has a maximal length of 237-1. The output of the generator is formed by 32 bits, selected and permuted from the LFSR and the CASR, and then xored together. The cycle length of the generator is close to 280. The bit rate of such system depends on the maximum oscillators frequencies that can be used, but the authors don't give any information about this important question.

Gerd Dirscherl et al. in their patent [14], proposed a pseudorandom generator including a first non-linear feedback shift register (NLFSR 1) with R memory cells combined with a second one with S memory cells by a multiplication operator. The result is then xored with a third NLFSR with T memory cells to obtain a final signal representing a pseudorandom number. The period length of the output sequences is equal to (2R-1) (2S-1) (2T-1), but there is no information related to the bit rate performance.

Most of chaos-based generators of the literature rely on floating-point data operations. This property leads to a problem when the resolution or the rounding method is different on the sender and on the receiver side. In a cryptographic system the chaotic sequences generated by the emitter and by the receiver will then be different. To avoid this problem, the generator presented in this paper works on a fixed finite precision of N bits. However, with a finite precision N, the chaotic dynamics are degraded and short cycles can appear.

III. DESCRIPTION OF THE PROPOSED CHAOTIC GENERATOR

A. Architecture of the proposed chaotic generator

The architecture of the proposed chaotic generator is given in Figure 1. It consists of: 28 basic chaotic generators, 4 analogical multiplexers 8 to 1, 3 XORs; a linear feedback shift register (LFSR), an operation of elimination of a percentage of samples and a block of quantification on Nq < N.

The proposed structure is modular, scalable, generic, and it produces a very long orbit with a large secret key.

Every basic chaotic generator of Figure 1 integrates a perturbation technique, based on linear feedback shift register (LFSR), of the chaotic orbit which increases its cycle lengths by a factor (2k-1), where k is the degree of the primitive polynomial of the LFSR.

B. Structure of the basic chaotic generator

Figure 2 shows the recursive structure of each basic chaotic generator (P-G) in the case of third order recursive filters. The discrete Skew-tent map (NLF1) and the discrete piecewise linear chaotic map (PWLCM, NLF2) are used as a non-linear functions. The output (integer pseudo-chaotic values) of each basic chaotic generator $X(n)$ is given by the following equations.

$$X(n) = Xc1(n) \oplus Xc2(n) \quad , \tag{1}$$

$$Xc1(n) = skewtent\{F1(n-1), P1\} \oplus Q1$$

$$F1(n-1) = \mathrm{mod}\left[In1 + \sum_{i=1}^{3} [c_{1i} \times Xc1(n-i)], 2^N \right], \tag{2}$$

$$Xc2(n) = pwlcm\{F2(n-1), P2\} \oplus Q2$$

$$F2(n-1) = \mathrm{mod}\left[In2 + \sum_{i=1}^{3} [c_{2i} \times Xc2(n-i)], 2^N \right] \tag{3}$$

$$Xc1(n),\ Xc2(n)\ and\ X(n) \in \left[1, 2^N - 1\right],$$

Where $P1$ and $P2$ are the control parameters for the Skew tent map, ranging from 1 to $2^N - 1$ and for the PWLCM map ranging from are and 1 to $2^{N-1} - 1$ respectively. $Q1$ and $Q2$ are the perturbing signals produced by the LFSRs, ranging respectively from 1 to $2^{ks} - 1$ and 1 to $2^{kp} - 1$, where ks and kp are the degrees of the primitive polynomials of the LFSRs used to disturb the skew tent map and the PWLMC map. The coefficients $c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}$ and the inputs $In1$ and $In2$ are within the interval $[1, 2^N - 1]$.

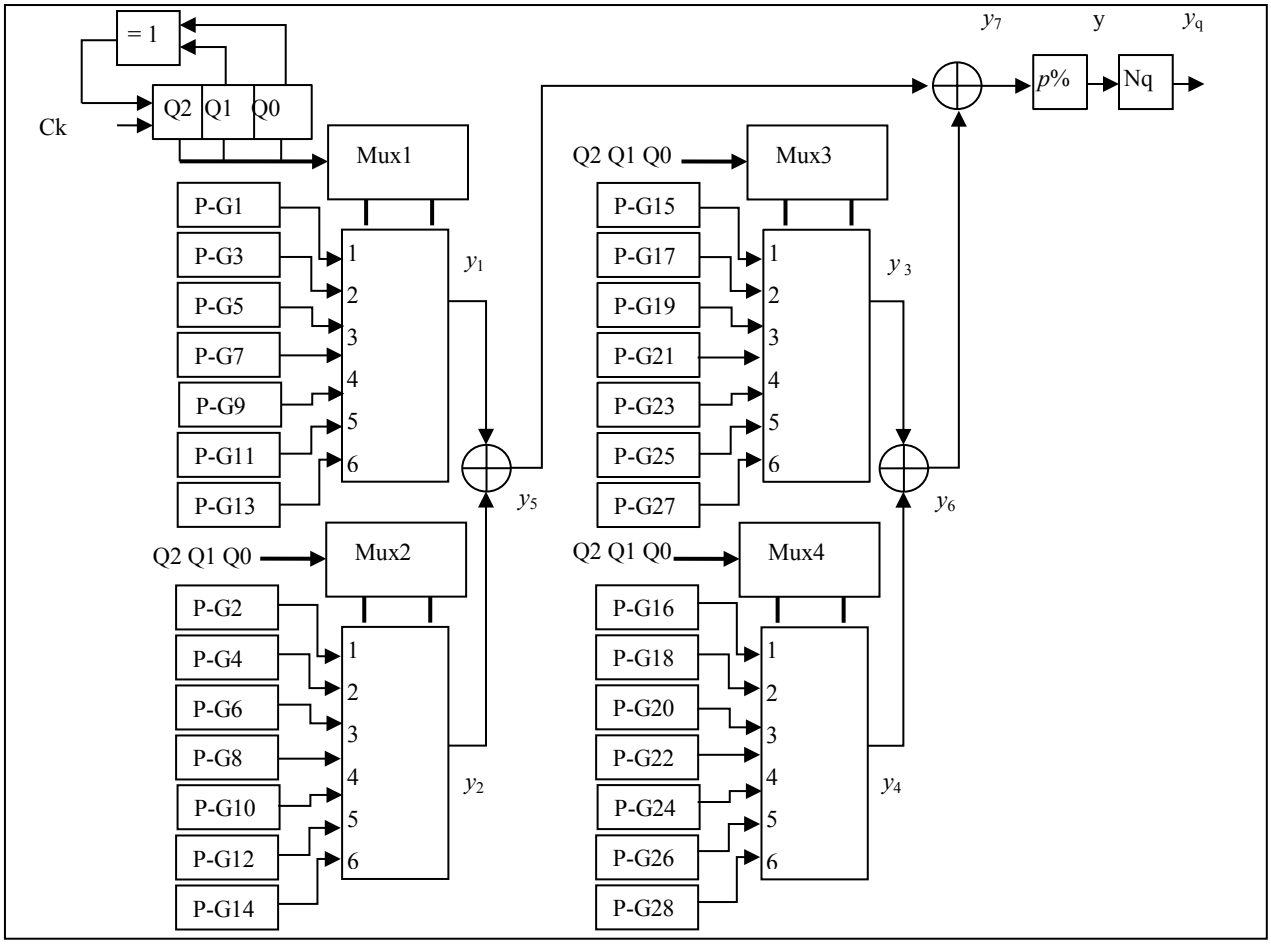

Figure 1. General architecture of the proposed chaotic generator

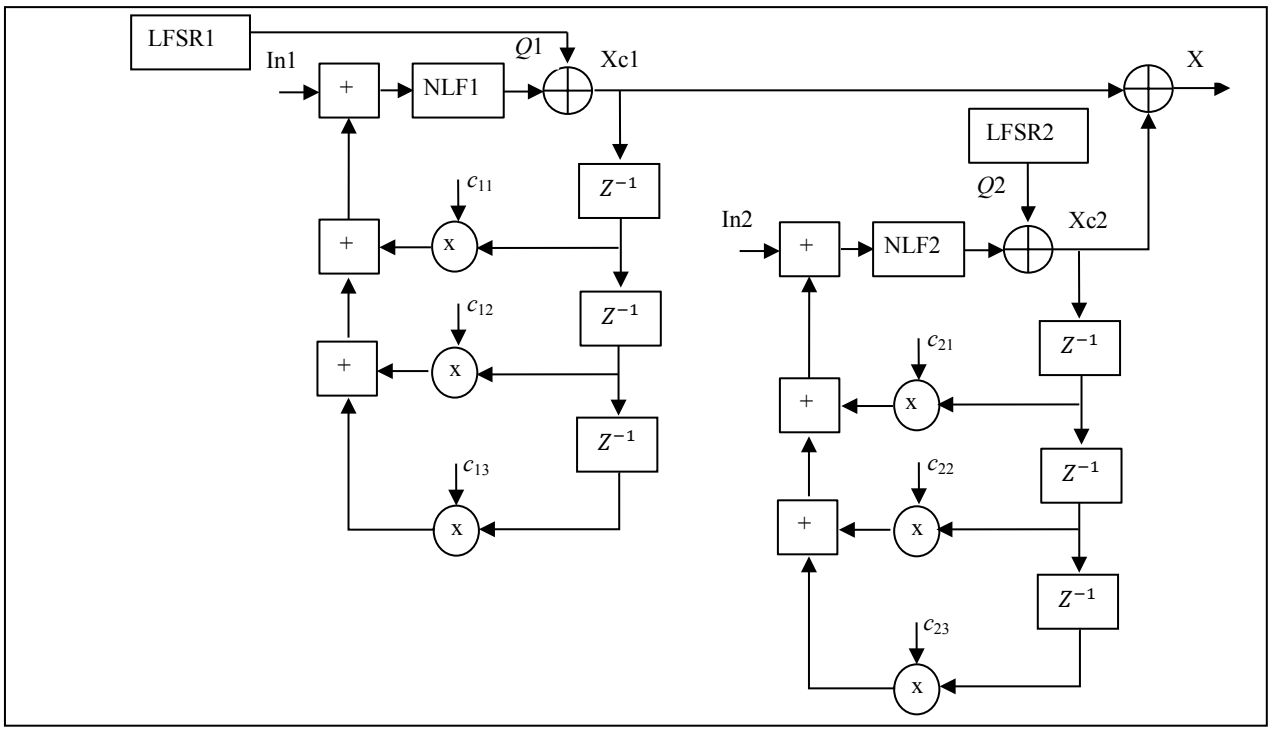

Figure 2. Structure of a single basic chaotic generator (P-G)

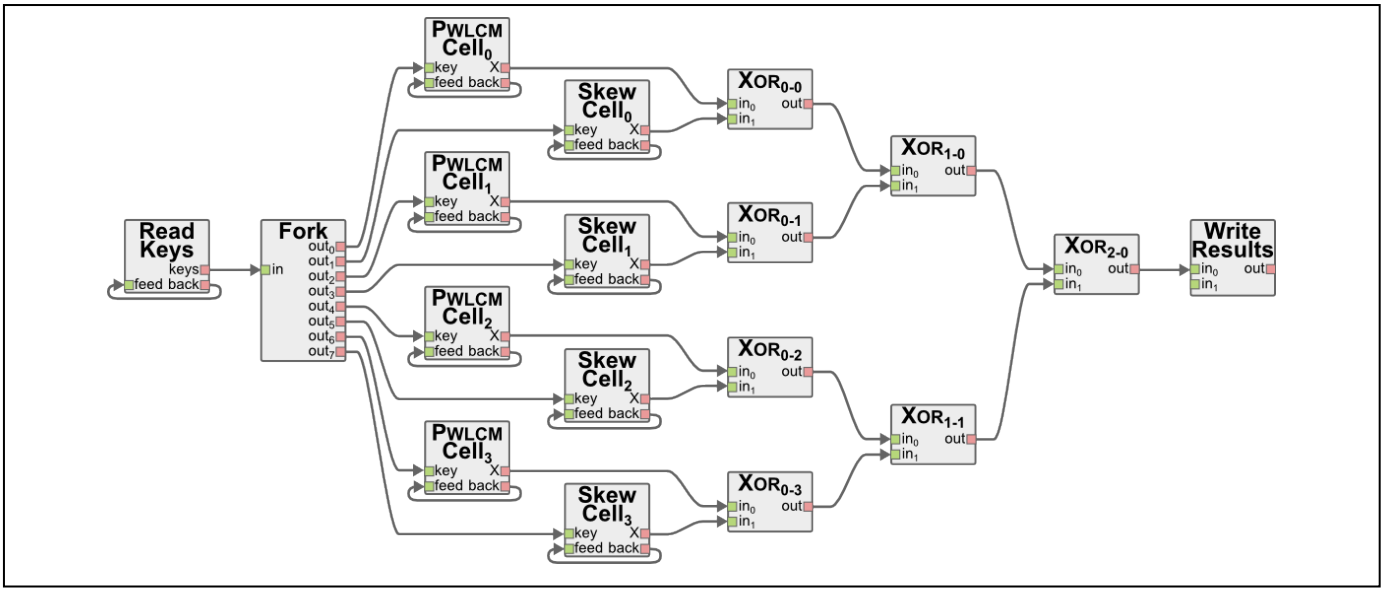

Figure 3. Single-rate dataflow graph exposing the parallelism of the random number generator

The discrete equations of the Skew tent and the PWLCM maps are given by [16][17]. The secret key of each basic generator is formed by : 6 initial conditions *Xc1(n-1), Xc1(n-2), Xc1(n-3), Xc2(n-1), Xc2(n-2), Xc2(n-3)* (6N bits); 2 initial conditions of the LFSRs (*ks* + *kp* bits) 6 parameters of the recursive cells $c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}$ (6N bits); 2 parameters of the skew tent and pwlcm maps (*P1* + *P2* bits) and 2 inputs *In1*, In2 (2N bits). With $N = 32$, $ks = 23$, $kp = 21$, the size of the secret key of the basic chaotic generator is 555 bits. The size of the secret key of the generating system of figure 1 is therefore:

$$|K| \cong 28 \times 555 = 15540 \; bits$$

C. Sequence of execution of the chaotic generator

The proposed generator of Figure 1 is running as follows:

1) Initialization of the twenty eight perturbed generators and the LFSR.

2) In every state $j = 1, 2, ..., 7$ of the LFSR (implemented by the following primitive polynomial: $g(x) = x^3 + x + 1$ and commended by the clock Ck), the length of the chaotic sequence at output y_7 is given by the least common multiple (lcm) of several secondary outputs. For example:

$$o_{j\min 5} = lcm\left[o_{j1}, o_{j2}\right] \tag{4}$$

is the length of the chaotic sequence at output y_5 where:

$$o_{j1} = lcm\left\{\left[2^{ks_{(2j-1)}} - 1\right] \times \Delta_{ks_{(2j-1)}}, \left[2^{kp_{(2j-1)}} - 1\right] \times \Delta_{kp_{(2j-1)}}\right\} \tag{5}$$

is the length of the chaotic sequence at output y_1 and

$$o_{j2} = lcm\left\{\left[2^{ks_{(2j)}} - 1\right] \times \Delta_{ks_{(2j)}}, \left[2^{kp_{(2j)}} - 1\right] \times \Delta_{kp_{(2j)}}\right\} \tag{6}$$

is the length of the chaotic sequence at output y_2 and so on.

$\Delta_{ks_m}, \Delta_{kp_m}$ (m = 1 to 28) are respectively the nominal periods of the first and the second cells of the basic chaotic generator without perturbation. The period of the clock Ck of the LFSR is given by:

$$o_{Ck} = Min\left[o_{j\min 7}{}_{j=1,2,\cdots 7}\right] \tag{7}$$

The minimal length of the chaotic sequence at output y is:

$$o_{\min} = 7 \times o_{Ck}\left[1 - p\%\right] \tag{8}$$

It is extremely long.

IV. Proposed Methods for Optimizing Implementation

To obtain an efficient implementation of the chaotic generator, parallelization at different levels is carried out. A data flow description is used to parallelize the different actors and the actor execution time is minimized by optimizing the actor C code.

A. Dataflow description

A dataflow description is used to model the application as a directed graph. Dataflow descriptions provide advanced semantics for expressing parallelism in an algorithm regardless of the targeted hardware architecture. The directed edges of the description model the flow of the data and the actors, corresponding to the graph nodes, model the transformations applied to the data. The hierarchical PiSDF description of the algorithm is transformed by PREESM into a single-rate graph that exposes the maximum parallelism from the graph. This graph is displayed in Figure 3. It contains actors for PWLCM and SkewTent nonlinear functions, XOR operations as well as a key source actor "ReadKeys" and a sequence sink actor "WriteResults". The "Fork" actor is a generated actor that automatically distributes data tokens to make the right data available for each actor. In case of large generated sequences,

the processing time is dominated by the PWLCM and SkewTent nonlinear functions. In the studied setup, 4 PWLCM and 4 SkewTent functions can be executed in parallel, bringing useful parallelism to the execution. In our experiments, the large amount of data to send between processes limits this parallelism to 2.24 on 4 cores (see Section 5.C). The parallel code generation process is described in [18]. One C thread is generated for each core and synchronized with the other threads. Communications are implemented via shared memory and synchronization.

B. *Actor C code optimization*

To minimize each actor execution time, the actor C code is optimized. After a code profiling analysis, the optimizations focus on repetitive structures (loops) and mathematical functions, which consume a significant part of the total execution time. Loop parameters are set to constant values at the compile–time to benefit from compiler optimization dedicated to loop. Loop unswitching is carried-out to remove conditional structures from loop kernels. This optimization eliminates over-cost due to tests and conditional branching. Instead of using functions from the mathematical library, functions, like modulo, are rewritten and simplified for our specific use case. These different code optimizations combined with compiler optimizations allow reducing the actor execution time of one order of magnitude.

V. EXPERIMENTAL RESULTS

A. *NIST test and mapping*

NIST test consists of a battery of 188 tests (globally 15 different tests) to conclude regarding the randomness or non-randomness of binary sequences [19]. We generated 100 sequences each with a different secret key and containing one million bits, and then we performed on them the NIST test. The proposed generator passes all the tests and therefore, it is robust against statistical attacks.

B. *Bit rate Performance*

TABLE I gives the measured mean bit rate (in Mbits/s) of the complete generator, with delay equals to 3, for the sequential implementation and the parallel implementation. The tests are made on a computer with quad-core Intel Xeon E31225 @ 3.1GHz. A very high bit rate of 173 Mbit/s is obtained on 4 cores.

TABLE I. BIT RATE PERFORMANCE

Implementation	Bit rate (Mbit/s)
Sequential implementation	77
2-core implementation	148
4-core implementation	173

C. *Details of the Multicore Execution Performances*

Figure 4 shows the bit rate of the random generator for core numbers between 1 and 4 and different sequence lengths ranging from 1 to 32768 32-bit samples. A sequence is a list of samples generated by a single iteration of the algorithm. One may remark that a fair speedup of up to 2.24 is obtained between sequential execution and execution on 4 cores. The performance obtained on 3 cores is very equivalent to the one on 2 cores due to unbalanced core loads. A minimal sequence length is required in order to obtain enough parallelism for the multicore execution. In our use case, a generated sequence of 4K samples = 128Kbits provides a near maximum parallelism. For short sequences, the amount of communication dominates the computation in terms of time.

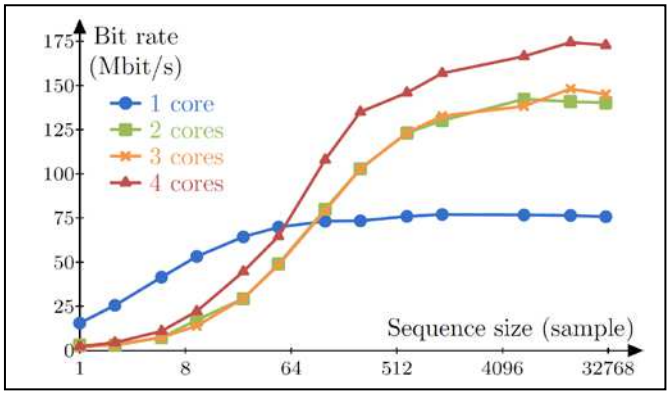

Figure 4. Bit rate performance (in bit/s) versus the size of the generated sequence (in 32-bit samples)

Figure 5 displays the memory necessary to compute the random sequence generator. For generated sequences over 256 samples, the memory needs grow linearly at a rate of about 128 Bytes/sample. The used dataflow parallelization method provides a quasi-constant memory requirement when the number of cores increases.

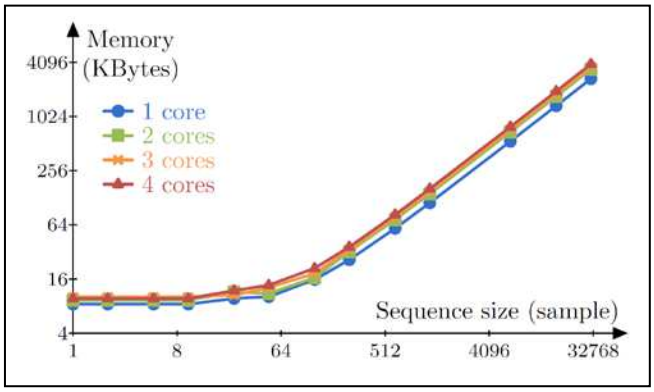

Figure 5. Necessary memory (in Bytes) versus the size of the generated sequence (in 32-bit samples)

VI. CONCLUSION

We presented in this paper a generator of discrete chaotic sequences and its efficient multicore implementation. The resulting implementation combines a very long minimal chaotic sequence $o_{\min} > 7 \times 2^{128}$ samples and a very high throughput of 173Mbps on 4 cores. A possible future work is the extension of the algorithm to obtain higher parallelism for equivalent cycle length properties.

VII. REFERENCES

[1] A. K. Hartmann. “Practical guide to computer simulations”, World Scientific, 2009.

[2] L. Blum, M. Blum, M. Shub. “A simple unpredictable pseudo random number generator”, SIAM J. Comput. 1986, 15, pp. 364-383.

[3] P. L'Ecuyer, Random numbers for Simulation”, Communications of the ACM, 1990, vol. 33, n°. 10, pp. 85-97.

[4] B. Elaine, K. John. ”Recommendation for random Number Generation using deterministic random bit generators”, Technical report. NIST SP 800-90 Rev A. 2012.

[5] K. Desnos , M. Pelcat, J.-F. Nezan, S. S. Bhattacharyya, S. Aridhi, “PiMM: Parameterized and Interfaced Dataflow Meta-Model for MPSoCs Runtime Reconfiguration”, SAMOS XIII, 2013

[6] Available online: http://preesm.sourceforge.net

[7] S. El Assad, H. Noura, I. Taralova. «Design and analyses of efficient chaotic generators for crypto-systems», Advances in Electrical and Electronics Engineering- IAENG Special Edition of the World Congress on Engineering and Computer Science 2008, vol. I, pp. 3-12, ISBN: 978-0-7695-3555-5.

[8] M. Hasler, Y. L. Maistrenko. ”An introduction to the synchronization of chaotic systems: coupled Skew tent maps”, IEEE Trans on Circuits and Systems, part I: Fundamental, theory and applications, vol. 44. N0. 10, October 1997, pp. 856-866.

[9] J. Fridrich, "Symmetric Ciphers Based no Two-Dimensional Chaotic Maps," International Journal of Bifurcation and Chaos, vol. 8, no. 6, 1998, pp. 1259-1284.

[10] R. Lozi, “Giga-periodic orbits for weakly coupled Tent and Logistic discretized maps”, Proc. Conf. Intern. On Industrial and Appl. Math., New Delhi, India, Dec. 2004, Invited conference, pp. 1-45.

[11] A. Senouci, I. Benkhaddra, A. Boukabou, K. Busawon. ”Implementation and evaluation of an hyperchaos-based PRNG”. The Fifth International Conference on Communications and Electronics, ICCE, 2014, Da Nang, Vietna, July, 6 pages.

[12] S. Lian, J. Sun, Z. Wang. “Security analysis of a chaos-based image encryption algorithm”, Elsevier, Physica A, 351, 2005, pp. 645-661.

[13] T. E. Tkacik. “A hardware random number generator”, International Workshop on Cryptographic hardware and embedded systems, August, 2002, pp. 450-453.

[14] G. Dirscherl, G. Markt, R. Gottfert. “Pseudo random number generator”, Patent US 2005/0097153 A1, May 5, 2005.

[15] S. El Assad, H. Noura. "Generator of chaotic Sequences and corresponding generating system" WO Patent WO/2011/121,218, 2011.

[16] N. Masuda, G. Jakimoski, K. Aihara, and L. Kocarev, "Chaotic block ciphers: from theory to practical algorithms," IEEE Transaction on Circuits and Systems, vol. 53, no. 6, pp. 1341-1352, 2006.

[17] S. Lian, J. Sun, J. Wang, Z. Wang, "A chaotic stream cipher and the usage in video protection "Chaos, Solitons & Fractals, 2007, vol. 34, Issue 3, pp. 851-859.

[18] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi (2014), PREESM: A Dataflow-Based Rapid Prototyping Framework for Simplifying Multicore DSP Programming. EDERC 2014, Milan, Italy.

[19] A. L. Rukhin, J. Soto, J. R. Nechvatal, M. Smid, E. B. Barker, S. Leigh, M. Levenson, M. Vengel, D. Banks, A. Heckert, J. Dray, S. Vo, 2008. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic applications”. Technical report. NIST SP 800-22 Rev. 1.