Novel PRNG Schemes (EC-based and/or parallel) and their Automated Testing

S. Varrette, V.Plugaru and P. Bouvry
<Firstname.Lastname@uni.lu>

Version 1.2 *

Context

Random numbers are used in many computational science fields (e.g. Monte Carlo simulations in physics, chemistry, bioinformatics and finance) and are essential to cryptography. Generating good random numbers is a difficult process, with the generators falling in two categories: Hardware-based Random Number Generators (HRNG) that are non-deterministic and Pseudo-Random Number Generators (PRNG) that are deterministic. The quality of the latter is generally tested by checking that: (1) the generated numbers are uniformly distributed in all dimensions, (2) there is no correlation between successive numbers, (3) the PRNG has a very long period for all seed states.

This project will focus on developing an automated framework that allows to test the randomness of a PRNG (treated as a black-box) using well-known statistical tests. Another objective of this project is to review the state-of-the-art as regards random number generator and their parallelization. Indeed, Monte Carlo simulation methods always require *massive* random numbers and the serial approach (consisting of dedicating a single process to the exclusive task of generating the random numbers) does not scale. Yet parallel PRNG requires some special attention, typically to ensure that the individual seeds have different sequence numbers in each processor. In general, one of the basic requirements for parallel random number streams is their mutual independence and lack of intercorrelation[1]. Three methods exists and should be reviewed:

1. Using different parameter sets [4, 5].
2. *Block-splitting* (or skipping-ahead) techniques: split the original sequence into k non-overlapping blocks, where k is the number of independent streams.
3. Leap-frogging methods, where the original sequence is split into k disjoint subsequences, where k is the number of independent streams, in

*Compilation time: 2017-12-15 10:01

[1]More precisely, even if you want random number samplings to be correlated, such correlation should be controllable – see also this page for more details.

such a way that the i-th stream would generate the random numbers $x_i, x_{k+i}, x_{2k+i}, x_{3k+i}, \ldots$.

The analysis aims at covering reference PRNG, as well as several known techniques but also cutting-edge ones allowing for a distributed executions in an heterogeneous context (CPU, GPU etc.)

Objectives

Theory: State-of-the-art literature review of statistical randomness testing, with emphasis on the NIST test suite [2] and Dieharder tests [1]. State-of-the-art literature on PRNG and their parallelization, with a special emphasis on Linear Congruential Generator (LCG) and their specialization [7], : LâĂŹEcuyer's multiple recursive generator MRG32k3a [3], Mersenne Twister MT19937 [6] (see also the `boost::random` list of generator).

Practice: Development of the automated framework for randomness testing, that can launch jobs on the UL HPC Platform (see `hpc.uni.lu`). Validation of the developed framework on top of the analysed PRNG. Implementation of Massively parallel random generator (SPRNG and specific nVidia GPU implementations). Provision of Easybuild ReciPY for all frameworks (if not available).

Technical Details

- Development of a framework that allows the automated job submission to the UL HPC platform, performing the NIST/Dieharder statistical tests of a PRNG.
- Experiments on the UL HPC clusters, with multi-node executions.
- Statistical analysis of the results and plotting with GnuPlot, R, Python/-matplotlib

General Organization

- All work is to be done in groups of up to 3 people.
- You are expected to prepare a report under the form of a research article, in LaTeX IEEE Conference style – use for instance the collaborative environment such as Overleaf or Git – we will provide a template for the source code. Details of the report:
  - 8 pages long, IEEE Conference style, including references.

  - Title: Novel PRNG Schemes (EC-based and/or parallel) and their Automated Testing
  - Authors: Student 1, [Student 2 ...], S. Varrette, V.Plugaru and P. Bouvry

- The group report (in PDF format) and all accompanying data (input files, code, launchers, result files) should be uploaded as an archive to the Moodle interface (and Gitlab) as proof of your work.
- LaTeX templates for the report, the slides etc. can be found on Gitlab @ Uni.lu – see `https://gitlab.uni.lu/MICS/public/parallel_computing`.
- Use the following search engines and digital libraries for scientific and academic papers in order to support your work:
  - `http://www.findit.lu`
  - `http://scholar.google.com`
  - `http://citeseerx.ist.psu.edu`
  - `http://www.sciencedirect.com`

References

[1] Dieharder: A random number test suite. `http://www.phy.duke.edu/~rgb/General/dieharder.php`.

[2] Lawrence E. Bassham, III, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Gaithersburg, MD, United States, 2010.

[3] Pierre L'Ecuyer. Combined multiple recursive random number generators. *Operations Research*, 44(5):816–822, 1996.

[4] Michael Mascagni and Ashok Srinivasan. Algorithm 806: Sprng: A scalable library for pseudorandom number generation. *ACM Trans. Math. Softw.*, 26(3):436–461, September 2000.

[5] Makoto Matsumoto and Takuji Nishimura. Dynamic creation of pseudorandom number generators. *Monte Carlo and Quasi-Monte Carlo Methods*, 2000:56–69, 1998.

[6] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998.

[7] S. K. Park and K. W. Miller. Random number generators: Good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, October 1988.