Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Non-standard pseudo random number generators revisited for GPUs

Christoph Riesinger[a,*], Tobias Neckel[a], Florian Rupp[b]

[a] Department of Informatics, Technical University of Munich, Munich, Germany

[b] Department of Mathematics and Science, German University of Technology in Oman, Muscat, Oman

HIGHLIGHTS

- Three methods to generate normally distributed pseudo random numbers which have properties making them interesting for an implementation on the GPU.
- Established GPU random number libraries are outperformed by a factor of up to 4.53, CPU libraries by a factor of up to 2.61.
- One of the three methods has never been considered for GPUs before but delivers best performance on many benchmarked GPU architectures.

ARTICLE INFO

Article history:
Received 26 February 2016
Received in revised form
5 October 2016
Accepted 17 December 2016
Available online xxxx

Keywords:
Fine-grain parallelism and architectures
GPU
Pseudo random number generation
Ziggurat method
Rational polynomials
Wallace method

ABSTRACT

Pseudo random number generators are intensively used in many computational applications, e.g. the treatment of uncertainty quantification problems. For this reason, the right selection of such generators and their optimization for various hardware architectures is of big interest.

In this paper, we analyze three different pseudo random number generators for normally distributed random numbers: The Ziggurat method, rational polynomials to approximate the inverse cumulative distribution function of the normal distribution, and the Wallace method. These uncommon generators are typically not the first choice when it comes to generation of normally distributed random numbers. We investigate the properties of these three generators and show how their properties can be used for an efficient high-performance implementation on GPUs making these generators a good alternative on this type hardware architecture.

Various benchmark results show that our implementations outperform well established normal pseudo random number generators on GPUs by factors up to 4.5, depending on the utilized GPU architecture. We achieve generation rates of up to 4.4 billion normally distributed random numbers per second per GPU. In addition, we show that our GPU implementations are competitive against state-of-the-art normal pseudo random number generators on CPUs by being up to 2.6 times faster than an OpenMP parallelized and vectorized code.

1. Introduction

Random numbers play a vital role in numerous areas of Computational Science and Engineering (CSE). They are frequently used in methods for uncertainty quantification (such as Monte Carlo sampling, realization and approximation of stochastic processes), in performance modeling but also for other applications such as cryptography. One popular way to generate random numbers on computers are pseudo random number generators (PRNGs) [1]. In contrast to real random number generators which rely on some physical process (e.g. radioactive decay), PRNGs follow a deterministic rule to generate random numbers [2,3]. Thus, such random numbers are not entirely random but fulfill certain statistical criteria [4].

There are many PRNGs for different statistical distributions (uniform, normal, exponential, etc.) with different capabilities, properties, and characteristics [5–7]. In general, PRNGs produce uniformly distributed random numbers (in the following called uniform random numbers and analogously normal, exponential, etc. random numbers). If a different distribution is required, uniform random numbers have to be altered to the desired target distribution by special transformation operations. Most of such transformation operations try to approximate the inverse cumulative distribution function (CDF) of the aimed distribution. Such combinations of a uniform PRNG and a transformation function are also

* Corresponding author.
E-mail addresses: riesinge@in.tum.de (C. Riesinger), neckel@in.tum.de (T. Neckel), florian.rupp@gutech.edu.om (F. Rupp).

http://dx.doi.org/10.1016/j.future.2016.12.018
0167-739X/© 2016 Elsevier B.V. All rights reserved.

called PRNG, even if it is a two stage process; but exceptions exist where a PRNG directly produces random numbers of, e.g., normal distribution avoiding the generation of uniform random numbers.

On CPUs, there is already much experience with PRNGs of different distributions available in literature, in particular with respect to the performance. In recent years, new processor architectures such as GPUs, the Intel Xeon Phi, or the PEZY accelerator[1] gained huge relevance in HPC. Occasionally, such accelerators have very different performance values compared to CPUs such as degree of parallelism, bytes per flop ratio, memory bandwidth, instruction throughput, context switching overhead, and many more. Thus, experiences made for CPUs cannot be directly generalized to accelerators, in particular not to GPUs. In this paper, we focus on three PRNGs for non-uniform distribution which are not the first choice on CPUs but which have interesting properties for GPUs. Reasons why such non-standard generators are not attractive for CPUs can be hard implementations, high complexity, or just bad performance due to immense computational intensity. Such properties do not have to be disadvantages on GPUs or can even be advantageous. The three PRNGs discussed in this paper are the *Ziggurat method*, rational polynomials approximating the inverse CDF (in the following just called *rational polynomials*), and the *Wallace method*. We investigate the characteristics of these three methods with respect to suitability on GPUs.

Since we need normal random numbers for our application (solving random ordinary differential equations (RODEs) [8,9]), we focus on this particular distribution. Nevertheless, the three non-standard PRNGs which are subject to this paper are not limited to normal distribution but can also generate random numbers of alternative distributions. When the non-standard methods are introduced in detail, we also briefly depict how different distributions can be achieved. Statistical properties of PRNGs can be experimentally checked by test batteries like Diehard [10] or TestU01 [11,12]. Investigating them is, however, not within the scope of this paper. Instead, we concentrate on implementation aspects, optimization for GPUs, and exploitation PRNGs' properties for better performance. This paper uses CUDA terminology (thread, block, grid, warp, occupancy, throughput) when it comes to GPU aspects. In addition, only CUDA-capable GPUs are used for performance measurements. But, all ideas presented in this paper also work with OpenCL and GPUs from vendors other than NVIDIA without any limitations.

The remainder of this paper is structured as follows: Section 2 lists relevant literature in the context of the non-standard PRNGs and implementation approaches on different hardware architectures. An introduction to the non-standard PRNGs together with ways how to exploit their properties for GPUs is given in Section 3. In Section 4, benchmark and profiling results are presented as well as a comparison with state-of-the-art CPU and GPU libraries for random numbers. Section 5 concludes this paper by summarizing the main achievements of our GPU implementations.

2. Related work

In this section, we briefly present literature which introduces, discusses, and adapts (e.g. for special hardware architectures) the three non-standard PRNGs used in this paper. A high-performance implementation of uniform generators, especially for GPUs, can be found in [13].

The Ziggurat method was first introduced in [14]. Over time, it was improved in terms of simplicity and performance which led to the most recent version in [15]. This is also the version that we use for our GPU implementation. Detailed discussions concerning the statistical properties of the Ziggurat method can be found in [16,17]. There are several attempts to implement the Ziggurat method on special purpose hardware, mainly on Field Programmable Gate Arrays (FPGAs). Examples can be found in [18–20]. All of these attempts realize a straight-forward implementation neglecting the special properties of the Ziggurat method. An extensive survey on several, also massively parallel architectures is done by Thomas et al. [21] where the Ziggurat method turns out to be the best choice on CPUs but not on GPUs. This result is not in accordance with our results presented in this paper. The idea of a runtime/memory trade-off as suggested in Section 3.1 is also seized by Buchmann et al. [22] where it is used for their cryptosystem application. Their implementation is limited to a normal distribution for integers.

Numerous examples exist to directly approximate the inverse normal CDF, e.g. in [23,24]. Wichura [25] suggests a set of coefficients for an approximating rational polynomial which we also use in this paper. An alternative coefficient set with different properties in terms of error bounds for specific regions of $\mathbb{R}$ is proposed by Beasley et al. [26].

The Wallace method was first presented in [27] with some very useful comments in [28]. The basis for our GPU implementation originates from the vectorized version of Brent [29] provided in the library *rannw* [30]. There is also a FPGA implementation of the Wallace method developed by Lee et al. [31].

In a former paper, we discussed and compared PRNG implementations on GPUs [32] where we focused on popular PRNGs for CPUs such as Mersenne Twister [33] or the Box/Muller method [34] which are not necessarily the best choice for GPUs. The Ziggurat method for GPUs was already discussed intensively by us in [35]. This paper augments our previous work by extended explanations how the Ziggurat is set up and by additional non-standard PRNGs with interesting properties especially for GPUs.

3. Methods

In this section, we present three non-standard PRNGs. The Ziggurat method in Section 3.1 and rational polynomials in Section 3.2 are transformation functions which approximate the inverse normal CDF. Hence, they require input from an uniform PRNG. The Wallace method in Section 3.3 directly generates normal random numbers and does not depend on any other PRNG.

3.1. Ziggurat method

The Ziggurat method is a rejection method which realizes the transformation from uniform to normal distribution in the best case with only one table lookup and one multiplication. It approximates the area under the normal probability density function (PDF) $f(x) = e^{-\frac{x^2}{2}}$ for $x \geq 0$ using N vertically stacked *strips* where N can be an arbitrary number ≥ 2. An approximation for $N = 8$ is depicted in Fig. 1. $N-1$ of the strips have rectangular shape. These *rectangles* R_i, $i = 0, \ldots, N-2$ have upper-left corners $(0, y_i)$ and lower-right corners (x_{i+1}, y_{i+1}), where $f(x_i) = y_i$ and $0 = x_0 < x_1 < \cdots < x_{N-1} = r$. The last strip $R_{N-1} = R_B$ is the *base strip* and is hatched from bottom left to top right in Fig. 1. It does not have rectangular shape but is bounded at $x = 0$ from the left, by $f(x)$ from the right, at $y = 0$ from the bottom, and at $y = y_{N-1}$ from the top. Each strip (the rectangles and the base strip) has the same area v. Every rectangle R_i, $i = 0, \ldots, N-2$ is further subdivided in three subregions: *central region*, *tail region*, and *cap region*. While central regions are not hatched, tail regions are hatched with diagonal crosses, and cap regions are hatched from bottom right to top left in Fig. 1. Central regions lie completely

[1] http://www.pezy.co.jp/en/index.html.