Friedrich-Schiller-Universität Jena
Physikalisch-Astronomische Fakultät

Design and Implementation of Pseudorandom Number Generators for Vector Processors and their Application to Simulation in Physics

MASTER'S THESIS

for obtaining the academic degree

Master of Science (M.Sc.) in Physics

submitted by Markus Pawellek

born on May 7th, 1995 in Meiningen
Student Number: 144645

Primary Reviewer:	Bernd Brügmann
Primary Supervisor:	Joachim Gießen

Jena, July 14, 2019

Abstract

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Acknowledgements

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

List of Figures

List of Abbreviations

Abbreviation	Definition
RNG	Random Number Generator
PRNG	Pseudorandom Number Generator
LCG	Linear Congruential Generator
MT	Mersenne Twister
MT19937	Mersenne Twister with period $2^{19937} - 1$
PCG	Permuted Linear Congruential Generator
CPU	Central Processing Unit
GPU	Graphics Processing Unit
SIMD	Single Instruction, Multiple Data
SSE	Streaming SIMD Extensions
AVX	Advanced Vector Extensions

Symbol Table

Symbol	Definition
$x \in A$	x ist ein Element der Menge A.
$A \subset B$	A ist eine Teilmenge von B.
$A \cap B$	$\{x \mid x \in A$ und $x \in B\}$ für Mengen A, B — Mengenschnitt
$A \cup B$	$\{x \mid x \in A$ oder $x \in B\}$ für Mengen A, B — Mengenvereinigung
$A \setminus B$	$\{x \in A \mid x \notin B\}$ für Mengen A, B — Differenzmenge
$A \times B$	$\{(x, y) \mid x \in A, y \in B\}$ für Mengen A und B — kartesisches Produkt
$\emptyset$	$\{\}$ — leere Menge
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^n$	Menge der n-dimensionalen Vektoren
$\mathbb{R}^{n \times n}$	Menge der $n \times n$-Matrizen
$f \colon X \to Y$	f ist eine Funktion mit Definitionsbereich X und Wertebereich Y
$\partial\Omega$	Rand einer Teilmenge $\Omega \subset \mathbb{R}^n$
σ	Oberflächenmaß
λ	Lebesgue-Maß
$\int_\Omega f \, \mathrm{d}\lambda$	Lebesgue-Integral von f über der Menge Ω
$\int_{\partial\Omega} f \, \mathrm{d}\sigma$	Oberflächen-Integral von f über der Menge $\partial\Omega$
∂_i	Partielle Ableitung nach der i. Koordinate
∂_t	Partielle Ableitung nach der Zeitkoordinate
∂_i^2	Zweite partielle Ableitung nach i
∇	$\begin{pmatrix} \partial_1 & \partial_2 \end{pmatrix}^\mathrm{T}$ — Nabla-Operator
Δ	$\partial_1^2 + \partial_2^2$ — Laplace-Operator
$\mathrm{C}^k(\Omega)$	Menge der k-mal stetig differenzierbaren Funktion auf Ω
$\mathrm{L}^2(\Omega)$	Menge der quadrat-integrierbaren Funktionen auf Ω
$\mathrm{H}^1(\Omega)$	Sobolevraum
$f\|_{\partial\Omega}$	Einschränkung der Funktion f auf $\partial\Omega$
$\langle x, y \rangle$	Euklidisches Skalarprodukt
$[a, b]$	$\{x \in \mathbb{R} \mid a \leq x \leq b\}$
(a, b)	$\{x \in \mathbb{R} \mid a < x < b\}$
$[a, b)$	$\{x \in \mathbb{R} \mid a \leq x < b\}$
$u(\cdot, t)$	Funktion $\tilde{u}$ mit $\tilde{u}(x) = u(x, t)$
A^T	Transponierte der Matrix A
id	Identitätsabbildung
$a := b$	a wird durch b definiert
$f \circ g$	Komposition der Funktionen f und g
$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$	Determinante der angegeben Matrix
$\mathrm{span}\,\{\ldots\}$	Lineare Hülle der angegebenen Menge
$\|A\|$	Anzahl der Elemente in der Menge A

1 Introduction

2 Background

2.1 Pseudorandom Number Generators

2.1.1 Mathematical Preliminaries

2.1.2 Random and Pseudorandom Sequences

2.1.3 Random and Pseudorandom Number Generators

2.1.4 Baseline Examples

2.2 Vector Processors

2.2.1 Architecture of Modern Central Processing Units

2.2.2 SIMD Instruction Sets and Efficiency

2.2.3 SSE, AVX, AVX512

2.3 Simulation in Physics and Mathematics

2.3.1 Mathematical and Physical Preliminaries

2.3.2 Baseline Model Problems

3 Related Work

The topic of PRNGs consists of several smaller parts. From a mathematical point of view, one has to talk about their definition and construction as well as methods on how to test their randomness. There have been a lot of publications concerning these issues. Hence, I am not able to give you a detailed overview. Instead, I will focus on the most relevant PRNGs and test suites, as well as some modern examples.

The creation of new PRNGs is sometimes understood to be black magic and can be hard since basically, one has to build a deterministic algorithm with a nearly non-deterministic output. In Kneusel 2018 one can find numerous different families of PRNGs. The most well-known ones are Linear Congruential Generators, Mersenne Twisters and Xorshift with its Variants. Whereas LCGs tend to be fast but weak generators in O'Neill 2014, one can find a further developed promising family of algorithms, called PCGs. Widynski 2019 describes another RNG based on the so-called middle square Weyl sequence. All of these generators have certain advantages and disadvantages in different areas such as security, games, and simulations.

After building a PRNG, one has to check if the generated sequence of random numbers fulfills certain properties. In general, these properties will somehow measure the randomness of our RNG. Typically, there are a lot of tests bundled inside a test suite such as TestU01 and Dieharder.

4 Methodology

4.1 Randomness Tests for RNGs

4.2 Performance of RNGs

There are 7 or 8 standard methods to parallelize RNGs. Explain the methods and their pros and cons. Say something about if it is better to use them vectorized or in multiprocessor.

4.2.1 Vectorization

4.2.2 Parallelization

4.2.3 Aliasing, Caching and Memory

4.3 Assembly Language

4.4 High-level Language

4.5 Intel Intrinsics

Naming scheme direct map to assembler instructions references: intel intrinsic guide advantages and disadvantages library abstraction through xsimd or agner fogs vector library

4.6 Libraries

4.7 C++ Concepts for Random Number Generators

Advantages and Disadvantages Better ideas taking best of all worlds

5 Implementation

5.1 Project Structure

5.2 Tools and Libraries

godbolt google benchmark intel vtune amplifier testu01 dieharder practrand cache miss measure by agner fog different compilers

5.3 Interface

What do we want from the interface of our RNG? It should make testing with given frameworks like TestU01, dieharder, ent and PractRand easy. Benchmarking should be possible as well. Therefore we need a good API and a good application interface. Most of the time we want to generate uniform distributed real or integer numbers. We need two helper functions. So we see that the concept of a distribution makes things complicated. We cannot specialize distributions for certain RNGs. We cannot use lambda expressions as distributions. Therefore we want to use only helper functions as distributions and not member functions. So we do not have to specify a specialization and instead use the given standard but we are able to do it. Therefore functors and old-distributions are distributions as well and hence we are compatible to the standard.

Additionally, we have to be more specific about the concept of a random number engine. The output of a random number engine of the current concept is magical unsigned integer which should be uniformly distributed in the interval [min,max]. But these magic numbers can result in certain problems if used the wrong way, see Melissa O'Neill Seeding Surprises. Therefore the general idea is to always use the helper functions as new distributions which define min and max explicitly and make sure you really get those values. This is also a good idea for the standard. And it is compatible with the current standard.

Now think of vector registers and multiprocessors. The random number engine should provide ways to fill a range with random numbers such that it can perform generation more efficiently. Think about the execution policies in C++17. They should be provided as well.

6 Results

godbolt google benchmark intel vtune amplifier testu01 dieharder

7 Conclusion

References

Barash, L. Yu., Maria S. Guskova, and Lev. N. Shchur: *Employing AVX Vectorization to Improve the Performance of Random Number Generators*. In: *Programming and Computer Software* 43.3 (2017), pp. 145–160.

Barash, L. Yu. and Lev. N. Shchur: *PRAND: GPU Accelerated Parallel Random Number Generation Library: Using Most Reliable Algorithms and Applying Parallelism of Modern GPUs and CPUs*. In: *Computer physics communications* 185.4 (2014), pp. 1343–1353.

— *RNGSSELIB: Program Library for Random Number Generation. More Generators, Parallel Streams of Random Numbers and Fortran Compatibility*. In: *Computer Physics Communications* 184.10 (2013), pp. 2367–2369.

— *RNGSSELIB: Program Library for Random Number Generation, SSE2 Realization*. In: *Computer Physics Communications* 182.7 (2011), pp. 1518–1527.

Bauke, Heiko and Stephan Mertens: *Random numbers for large-scale distributed Monte Carlo simulations*. In: *Physical Review E* 75.6 (2007), p. 066701.

Demirag, Yigit: *Vectorization Studies of Random Number Generators on Intel's Haswell Architecture*. CERN openlab, 2014.

Fog, Agner: *Calling Conventions for Different C++ Compilers and Operating Systems*. Agner Fog, 2018. URL: https://www.agner.org/optimize/calling_conventions.pdf.

— *Optimizing Software in C++: An Optimization Guide for Windows, Linux and Mac Platforms*. Agner Fog, 2018. URL: https://www.agner.org/optimize/optimizing_cpp.pdf.

— *Pseudo-Random Number Generators for Vector Processors and Multicore Processors*. In: *Journal of Modern Applied Statistical Methods* 14.23 (2015). URL: http://digitalcommons.wayne.edu/jmasm/vol14/iss1/13.

Guskova, Maria S., L. Yu. Barash, and Lev. N. Shchur: *RNGAVXLIB: Program Library for Random Number Generation, AVX Realization*. In: *Computer Physics Communications* 200 (2016), pp. 402–405.

Kneusel, Ronald T.: *Random Numbers and Computers*. Springer, 2018. ISBN: 978-3-319-776696-5.

L'Ecuyer, Pierre: "Random number generation with multiple streams for sequential and parallel computing". In: *2015 Winter Simulation Conference (WSC)*. IEEE. 2015, pp. 31–44.

L'Ecuyer, Pierre and Richard Simard: *TestU01: AC Library for Empirical Testing of Random Number Generators*. In: *ACM Transactions on Mathematical Software (TOMS)* 33.4 (2007), p. 22.

Marsaglia, George et al.: *Xorshift RNGs*. In: *Journal of Statistical Software* 8.14 (2003), pp. 1–6.

Matsumoto, Makoto and Takuji Nishimura: *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30.

O'Neill, Melissa E.: *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. In: *ACM Transactions on Mathematical Software* (2014).

Saito, Mutsuo and Makoto Matsumoto: *SIMD-Oriented Fast Mersenne Twister: A 128-bit Pseudorandom Number Generator*. In: *Monte Carlo and Quasi-Monte Carlo Methods 2006*. 2008, pp. 607–622.

Widynski, Bernard: *Middle Square Weyl Sequence RNG*. In: *arXiv preprint arXiv:1704.00358v3* (2019).

A Build System

Fog 2018b

Statutory Declaration

I declare that I have developed and written the enclosed Master's thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master's thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.
On the part of the author, there are no objections to the provision of this Master's thesis for public use.

Jena, July 14, 2019

Markus Pawellek